DISSERTATION
submitted
to the
Combined Faculties for the Natural Sciences and for Mathematics
of the
Ruperto-Carola University of Heidelberg, Germany
for the degree of
Doctor of Natural Sciences

Put forward by
Diplom-Physicist: Jorrit Fahlke
Born in: Witzwort, Germany
Oral examination:

Discontinuous Galerkin Methods for Parallel Simulation
of Ground-Penetrating Radar in 3D

Advisors: Prof. Dr. Peter Bastian, Prof. Dr. Kurt Roth

Abstract

In this work we examine different numerical methods for the simulation of Maxwell's equations in 3D with the application to ground-penetrating radar. In particular we consider an edge-based finite element and a discontinuous Galerkin method, both in the time domain. We implement these methods using the finite element framework Dune and the discretization module `dune-pdelab` and test the implementations using two example problems. Finally, we apply them to a ground-penetrating radar problem derived from the ASSESS-GPR test site and compare the results to actual measurements made on the site.

Zusammenfassung

In dieser Arbeit untersuchen wir verschiedene numerische Methoden zur Simulation der Maxwellgleichungen in 3D mit Anwendung auf Bodenradar. Insbesondere betrachten wir eine kantenbasierte Finite-Elemente-Methode und eine Discontinuous-Galerkin-Methode im Zeitbereich. Diese beiden Methoden implementieren wir mit Hilfe von Dune und dem Diskretisierungsmodul `dune-pdelab` und testen diese Implementierungen anhand von zwei Beispielproblemen. Zuletzt wenden wir sie auf ein Bodenradarproblem an, dass sich vom ASSESS-GPR-Versuchsaufbau herleitet, und vergleichen die Ergebnisse mit einer Messung, die an diesem Versuchsaufbau gemacht wurde.

Contents

1. Introduction

In this work we are concerned with the numerical simulation of ground-penetrating radar. The main aim is to develop efficient and accurate numerical methods for 3D simulations. These are needed ultimately to do inversion of GPR data on soils that do not have translational symmetry in one direction.

1.1. GPR and related methods

Ground-penetrating radar (GPR) has many application. It is used in archaeology for surveys of historical sites, to quickly find interesting features warranting a closer inspection. It is also used to study materials of historical artifacts and to identify different materials for the purpose of restoration. In engineering it is used to examine roads, bridges, tunnels, buildings and other man-made structures for the purpose of quality control and to discover faults due to wear and tear. Military applications include the detection of tunnels and buried objects such as mines.

The focus of this work is on the use of GPR in soil physics and hydrology. In these fields GPR is used to map the ground structure and the soil water content. Here its main advantage is that it operates non-destructively and can cover larger areas relatively quickly. Closely related are application in planetology and the examination of glaciers.

The basic principle of GPR is as follows. A sending antenna emits an electromagnetic pulse, ideally directed towards the ground and a receiving antenna records the returned signal. This results in a *trace*, consisting of a number of *samples* for different points in time. Then the position of the sending and/or receiving antenna is changed and the process is repeated. This results in a *line*, a collection of many traces for different antenna positions.

The most common measurement technique is common offset (CO): the distance between sending and receiving antenna is kept constant, e.g. by building them side-by-side into the same case. This makes it very suited to obtain data on a larger site: usually one operator drags the antenna-assembly over the ground while another walks alongside recording the data on a laptop. A small wheel attached to the antenna measures the distance and triggers the measurement of one sample every few centimeters. This is very similar to what is done by engineers to examine roads: there, multiple transmitting-receiving antenna assemblies are mounted on a vehicle, which also contains the recording electronics, and the operator simply has to drive on the road.

An alternative technique is wide angle reflection and refraction (WARR). One antenna is stationary and the other is pulled away, again triggering a trace every few centimeters. Very similar is the common midpoint (CMP) technique. Here, receiving and sending

antenna are both moved away from a common midpoint. Both techniques can be used to directly measure the refractive index of the topmost soil layer by comparing the travel time of the air wave to that of the ground wave. Especially CMP is also well suited to determine both the depth of a particular reflector and the refractive index of the ground above that reflector, since it gathers information from different angles.

Finally, multi-channel GPR integrates ideas from CMP and WARR into CO measurements: at least two transmitting-receiving antenna assemblies (T1,R1) and (T2,R2) are chained together, separated by a little distance. These are dragged across the ground with a constant distance and triggered in regular intervals, just as for CO. This provides now four channels for measurements: the two internal channels T1–R1 and T2–R2, but also the short cross-box channel T2–R1 and the long cross-box channel T1–R2. Just like CMP and WARR this means the same region in the ground is examined from different angles, and both reflector depth and refractive index can be determined. Most multi-channel GPR systems use actually more than just two antenna assemblies.

Lines are commonly displayed in the form of a *radargram*, an image with the position of the trace on the horizontal axis, the time of each sample on the vertical axis (with later times at the bottom), and the sample value determining the gray level. This can already look deceptively like a cross section of the ground. However, this is usually misleading and several post-processing steps and assumptions (or multi-channel measurements) are required to get an actual approximate representation of the ground.

GPR works in the frequency range 10MHz–5GHz. Its penetration depth is usually in the order of a few meters, but can be tens of meters in arid regions due to the lower water content. In the extreme the penetration depth in ice can be in the order of kilometers[14]. On the other hand, when the ground has high contents of iron, clay, salt or water, GPR can be completely inapplicable. The penetration depth is of course better for lower frequencies, on the other hand higher frequencies provide better resolution.

Due to the limited penetration depth GPR can normally not be used to explore natural resources. There is however a related technique by the name magnetotellurics. It operates with frequencies in the range of 0.1mHz–10kHz and reaches penetration depths in the order of 15km. When modeling magnetotellurics, the wave nature of the electromagnetic field is often neglected resulting in a purely diffusive approximation.

An older technique for exploring natural resources is seismic imaging. Despite being based on different physical principles, seismic imaging and GPR can both be modeled by wave equations. Thus from a numerical point of view, they share a much closer relation than GPR and magnetotellurics do.

1.2. Modeling of GPR

The interpretation of the data from GPR is a non-trivial task. While irregularities are easy to spot, which make this tool suitable for archaeology and military applications, doing a quantitative analysis is a much more difficult task.

To get a better handle on the problem, various people have approached it with the help of analytical and computational models. In [14] Gerhards examined the problem

using ray-based, plane-wave and Green's function-based approaches, resulting in the development of multi-channel GPR techniques. Lambot *et al.* model the ground as a series of purely horizontal layers[24]. This allows for very efficient modeling, even taking the 3D nature of the electromagnetic wavefront into account. It does however neglect horizontal variability of the ground structure. Leidenberger in [25] implemented a FETD simulator (*hades3d*) to investigate the influence of dispersion, but was limited to around 300 000 degrees of freedom. Buchner in [7] conducted inversion of the ground structure and water content of a site with known ground structure based on GPR data. He achieved good agreement, but was limited to using 2D simulations.

1.3. Computational Challenges

Doing a full simulation of Maxwell's equations is quite challenging. Assuming a frequency of 200MHz and maximum relative permittivity of 5 the minimum wavelength will be 30cm. At the same time, the size of the domain is some meters – for the sample problem we will use later it is approximately 6m $\times$ 4m $\times$ 2m. With a structured mesh this would lead to a mesh of about 10 000 tetrahedra. Due to the complex internal layering meshes generated by a mesh generator have much more tetrahedra and the minimum element size will be much smaller than the maximum. For the sample problem, a mesh generated by Gmsh had $\approx$ 150 000 tetrahedra, with the smallest element size around 3cm, see table 7.1. To observe the sampling limit, the maximum element size must be half the minimum wavelength, which already means a factor of eight in the number of elements and thus memory requirement. For the computational effort the increase is a factor of 16, since the number of time steps has to double as well due to the CFL condition. To get good accuracy we need to increase the mesh resolution further, leading to further increase in the fourth order of the mesh resolution.

To solve problems of such sizes, we need to employ parallel computers. This brings its own challenges. The components of sequential PDE solvers must be adapted or even replaced by different schemes to work in a parallel setting. The work must be distributed evenly among the available processors, or part of the available computing time would be wasted. It must be organized in a way to minimize communication between processors, since communication is expensive.

There are several approaches and numerical schemes to choose from: frequency domain schemes solve an elliptic problem for many frequencies and then transform the result into the time domain. They have the disadvantage that they lead to a difficult to solve matrix. In addition a final Fourier transform must be applied to obtain the result in the time domain. In contrast, for time domain discretizations people often use explicit schemes. In the case of discontinuous Galerkin time-domain (DGTD) and finite-element time-domain (FETD), this means that a mass matrix needs to be solved. For FETD, this matrix is very benign and can be solved with a simple iterative solver in a few iterations. Since each step of a linear solver requires a communication, a this is advantageous in a parallel setting. For DGTD the mass matrix can be solved even more easily: it is (block) diagonal, and it is sufficient to invert each block individually. With (block)

diagonal matrices the advantage in the parallel case is even greater, since the number of communications required to solve the matrix is reduced to one. In fact, even storing the matrix can be avoided by applying the matrix on the fly in an assembly run.

Finite difference time-domain (FDTD) schemes such as the one by Yee [40] are well-known, and solvers implementing them, both commercial and open source (e.g. MEEP [32]), are highly optimized. They are however limited to second order accuracy. They employ a regular grid that cannot easily be adapted to irregular geometries of materials inside the domain.

Finite-element time-domain schemes work on unstructured meshes which can readily be fitted to irregular geometries. In addition, higher order basis functions can be used that lead to higher order schemes. However, even when used with an explicit time scheme, they lead to a non-diagonal mass matrix, making an iterative solver necessary.

Discontinuous Galerkin time domain schemes combine the advantages of finite difference and finite element schemes. They operate on unstructured meshes and allow higher order basis functions, but still avoid the use of an iterative solver: when used with an explicit time scheme, the mass matrix is block-diagonal with one block for each mesh element. These blocks can easily be inverted using a direct solver. With the appropriate basis, even the blocks can be diagonal. Discontinuous Galerkin schemes are easy to extend to higher order, which enables them to efficiently use modern processors even in a sequential setting.

1.4. Outline

Chapter 2 lays the theoretical groundwork. The underlying Maxwell's equations are described as well as some of the consequences. We present to forms of Maxwell's equations: a first order system, useful to the DGTD scheme, and a second order wave equation, useful for the FETD scheme. On the topic of soil physics we introduce the REV and use it do derive the CRIM formula, which may be used to describe the dielectric permittivity of mixed materials. Finally we describe aspects of computer science such as measures to assess algorithms, both sequential and parallel.

Chapter 3 describes the algorithms used to discretize the domain spatially: a novel discontinuous Galerkin time-domain scheme and as a reference a finite element time domain scheme.

Chapter 4 describes the schemes used for the discretization in time.

Chapter 5 describes the linear algebra solvers used for the equation systems resulting from the temporal discretization.

Chapter 6 describes two test problems that were used to test the implementation. The first test case is a superposition of plane waves traveling through the computational domain. The second is a dipole antenna in the center of a spherical domain transmitting a pulse. Both test problems have analytic solutions, and the first

is used to compare the accuracy of the DG scheme at various orders to the DG scheme.

Chapter 7 describes application of these methods to a GPR measurement on an man-made test side with known geometry.

2. Theory

Solving large PDEs, as required by the GPR, requires the of sufficiently potent computers. Since the clock frequency of individual processors cannot be raised much further, such computers have a parallel architecture, and more computing power is gained by adding processors. This makes it necessary to adapt programs to such an architecture – the program consists of many threads of execution, each running on its own processor. These threads of execution must coordinate, which entails communication overhead. In section 2.1 we describe how to assess parallel programs, and we look at the architecture of parallel computers. setting of parallel computing.

The theoretical groundwork needed for the treatment of ground-penetrating radar includes foremost Maxwell's equations. These are the fundamental physical description of electromagnetic waves and other electromagnetic phenomena. They will be the subject of section 2.2.

But the ultimate goal of GPR is to detect water contents, in particular the sharp discontinuities in water content which are related to ground structure. Therefore a relation between the dielectric permittivity, which is the relevant property as far as electromagnetics is concerned, and water content is needed. This is given by the CRIM formula, which is described in section 2.4, along with the concept of the REV which is fundamental in soil physics.

2.1. Parallel computing

Today's computers employ a high level of parallelism. One example is instruction level parallelism, where either the processor dynamically or the compiler statically resolves data dependencies between instructions and schedules them in such a way that as few units as possible of the processor remain unused at any given time. This is however mostly transparent to the programmer and the user of the program, so I will not consider this further here.

2.1.1. Flynn's Taxonomy

Let me introduce the classification of parallel programming models by Flynn [13]. Flynn classifies machines based on how many data and instruction streams they are able to process simultaneously. The four classes are:

Single instruction-stream single data-stream (SISD) This describes a traditional sequential architecture.

Single instruction-stream multiple data-stream (SIMD) Here a single instruction controls several processing units operating on different data. Modern processors implement this concept in their vector units, and graphic cards are heavily based on it.

Multiple instruction-stream single data-stream (MISD) This is a very unusual architecture, although it is not totally unheard of. Flynn himself gives an example in his paper.

Multiple instruction-stream multiple data-stream (MIMD) This is essentially a collection of more or less independent machines of one of the other types. Each instruction stream operates on one or more data streams. An example would be a modern multiprocessor, or a cluster.

With the exception of MISD modern computers incorporate elements of any of the classes: vector units are ubiquitous are members of the SIMD class. Even uniprocessor machines usually have a graphics card or some other auxiliary processor which makes them MIMD, strictly speaking.

2.1.2. Shared vs. Distributed Memory

Another useful classification for a parallel machine is according to whether all processors share a common address space or not. When a common address space exists the computer is called a *shared memory* machine, otherwise it is called *distributed memory*. A single computer with multiple cores is usually a shared memory machine. A cluster is an example of a distributed memory machine: multiple compute nodes, each a more-or-less standalone computer, joined by a fast network.

Shared memory offers the advantage that all processes of a running program can access the same data structures, which avoids the need to store data needed by multiple processes more than once. Programs written for such a machine commonly use *multithreading* as a programming model, and are supported by libraries such as pthreads or the C++11 standard library, or language extensions such as OpenMP. Shared memory has the disadvantage that machine of this type become difficult to build when the number of processors becomes large. Uniform memory access (UMA) systems, where all processors have equal access to all memory, become impractical at around 8 to 16 processors. The next step are non-uniform memory access (NUMA) systems. These are essentially a collection of UMA systems (called nodes in the NUMA context) which are joint at the lowest caches by a fast interconnect, see figure 2.1. Access of a processor to memory on the same node works as usual. However, if some processor accesses memory on a different node, the cache on the requesting node has to communicate with the cache on the node containing the memory through the interconnect. This makes access to some memory slower, and it is often beneficial to keep memory data structures and processors used by one process on the same node.

NUMA has its own limits, of course, and to build even larger systems the concept of a shared address space is usually given up, resulting is distributed memory machines. In

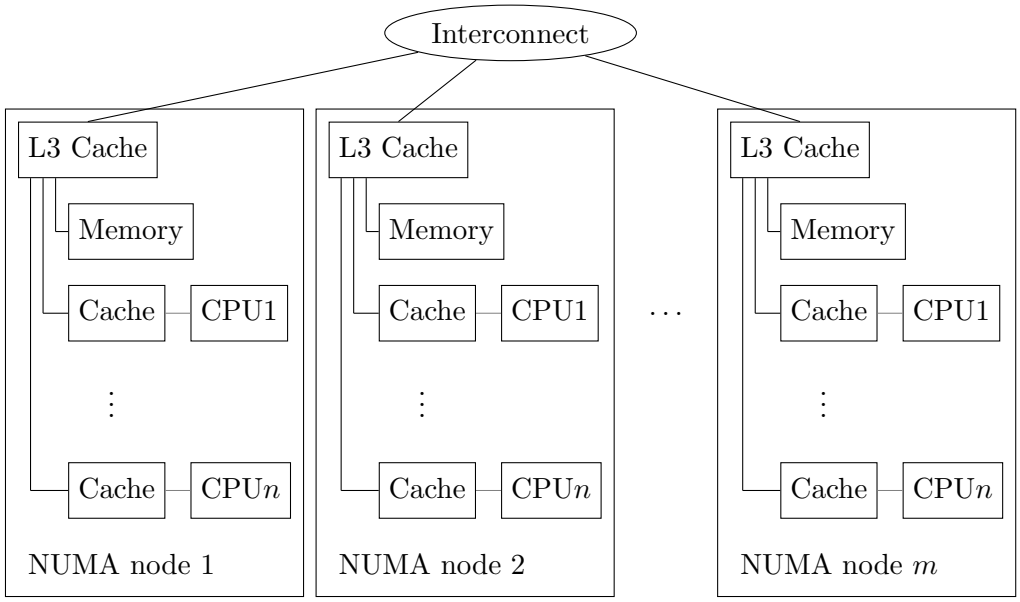


Figure 2.1.: Typical structure of a NUMA machine.

practice this i usually a cluster of UMA or NUMA machines joined by a fast network. Since different processors can no longer transparently access each other memory, the processes of a program have to handle communication explicitly, giving rise to the *message passing* programming model. This is supported by the *message passing interface* (MPI), a standard implemented by many competing libraries, providing the fundamental building blocks such as sending and receiving messages.

For NUMA systems it is often advantageous to program them using message passing instead of multithreading. Then the processes of the program do not share memory, which makes it possible to keep each process running on the NUMA node where it's memory is located.

2.1.3. Scalability

Given a problem, an algorithm to solve it and a parallel computer, ideally I would expect that the time to complete the algorithm is halved when I use twice as many processors. This is however nearly never the case. One reason is that many algorithms have portions that cannot be executed in parallel, and thus do not benefit from the additional processors. Another reason is that the intermediate results of one process of the parallel program is often needed by another process. This means that this data need to be communicated, which takes time in itself.

Let $T(N, P)$ be the runtime of an algorithm A on input data of size N on P processors, and let $T^{\text{best}}(P)$ be the runtime of the best sequential algorithm B available for the same

problem. Then define the speedup of A as

$$S(N,P) = \frac{T^{\text{best}}(P)}{T(N,P)} \tag{2.1}$$

and the efficiency as

$$E(N,P) = \frac{T^{\text{best}}(P)}{PT(N,P)}. \tag{2.2}$$

These measures can be used in different ways. A *strong scalability* study plots the speedup $S(N,P)$ (or the efficiency $E(N,P)$) over P for constant N. For an algorithm that scales ideally the speedup graph should be a diagonal line of slope 1, and the efficiency should be a constant 1. Based on this, Amdahl gave a rather pessimistic outlook on parallel computing in 1967[1]. His argument was that that every algorithm can be split into a port that can be parallelized and a part that is inherently sequential. The sequential part is always present and non-vanishing (i.e. program setup time). In a strong scaling study the speedup is limited by this sequential component, at some point it does not make sense to add more processors since it does not reduce the computation time any further. This has become known as *Amdahl's law*.

The problem with Amdahl's law is that it answers the question "what is the shortest time needed to solve a given problem?" However, for many practical problems the more interesting question is "given a limited time, how big can I make N while still being able to run my algorithm on a given computer?" This was pointed out by Gustafson in [17]. To answer this question a *weak scalability* study is useful: instead of keeping N constant, N is scaled with P. The inherently sequential part is mostly independent of the problem size, so that good scalability can be achieved even for very large P.

2.1.4. Scalability in the Context of PDEs

In this section I will consider aspects of parallelization of particular interest when solving partial differential equations with mesh-based methods. Consider a regular mesh with 9×9 elements covering the computational domain. This can be partitioned onto nine processes using a 3×3 pattern, see figure 2.2. What is interesting here is that communication happens at the boundary between processes. This can be either via overlap mesh elements as shown in the figure, or via nodes shared by several processes. The important point here is that the amount of data exchanged scales with the boundary, i.e. $\Theta(\sqrt{N/P})$ in the example, while the computational effort scales with the volume (at the least), i.e. $\Omega(N/P)$. Thus for weak scaling $N \propto P$ the proportion of the communication in the overall runtime will stay constant.

2.2. Electromagnetic Theory

In 1865 Maxwell presented his "Dynamical Theory of the Electromagnetic Field"[29] to the Royal Society. This treatment included a set of equations, from which Heaviside later

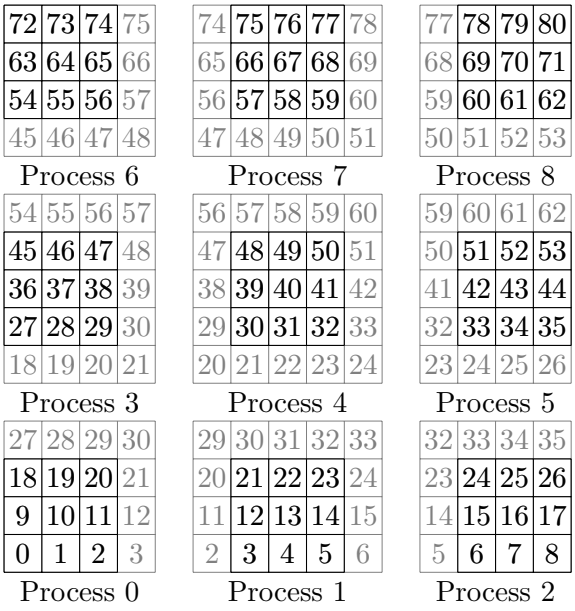


Figure 2.2.: Example of a 9×9 mesh partitioned onto nine processes. The numbers are the indices of the grid elements, overlap elements on a given process are in gray.

selected the four vector equations known today as Maxwell's equations. Individually these equations predate Maxwell's treatment, which is reflected in their names. In today's notation, the equations are:

$$\partial_t \mathbf{B} + \nabla \times \mathbf{E} = 0 \tag{2.3a}$$
$$\nabla \cdot \mathbf{D} = \rho \tag{2.3b}$$
$$\partial_t \mathbf{D} - \nabla \times \mathbf{H} = -\mathbf{J} \tag{2.3c}$$
$$\nabla \cdot \mathbf{B} = 0 \tag{2.3d}$$

(2.3b) is known as *Gauss's law*. It relates the electrical displacement to charges. (2.3d) does the same for the magnetic field, with the twist that there are no magnetic charges. It is thus known as *Gauss's law for magnetism*. (2.3c) is *Ampère's circuital law*. In its original form it just relates the magnetizing field to the current density. The correction by Maxwell, which is included here, extends this to displacement currents. Finally, (2.3a) is the *Maxwell-Faraday equation*, a generalization of Faraday's law of induction.

Maxwell's equations are connected by a set of constitutive relations:

$$\mathbf{D} = \varepsilon \mathbf{E} \tag{2.3e}$$
$$\mathbf{B} = \mu \mathbf{H} \tag{2.3f}$$
$$\mathbf{J} = \sigma \mathbf{E} + \mathbf{J}^s \tag{2.3g}$$

Symbol	Unit (SI)	Description
$\mathbf{E}$	$\mathrm{V/m} = \mathrm{N/C}$	Electric field
$\mathbf{B}$	$\mathrm{T} = \mathrm{N\,s/C\,m}$	Magnetic field
$\mathbf{D}$	$\mathrm{C/m^2}$	Electric displacement field
$\mathbf{H}$	$\mathrm{A/m}$	Magnetizing field
$\mathbf{J}$	$\mathrm{A/m^2}$	Total current density
$\mathbf{J}^s$	$\mathrm{A/m^2}$	External current density
ρ	$\mathrm{C/m^3}$	Charge density
ε	$\mathrm{F/m}$	Dielectric permittivity
μ	$\mathrm{V\,s/A\,m}$	Permeability
σ	$\mathrm{S/m} = 1/\Omega\,\mathrm{m}$	Electric conductivity

Table 2.1.: Quantities appearing in Maxwell's equations.

Although not usually considered part of Maxwell's equations, the continuity of charges is closely related:

$$\partial_t \rho + \nabla \cdot \mathbf{J} = 0 \tag{2.3h}$$

The quantities used in Maxwell's equations are listed in table 2.1.

Notice that if (2.3d) is fulfilled at any point in time, it follows from (2.3a) by taking the divergences that (2.3d) will be fulfilled for all points in time. For (2.3b) and (2.3c) the situation is a little bit more complicated: given the divergence of (2.3c) either (2.3b) or the continuity equation (2.3h) follows from the other.

2.2.1. Field Continuity at Boundaries

To examine the behavior of the fields at material boundaries, it is useful to consider Maxwell's equations in *integral form*:

$$\oint_{\partial\Sigma} d\mathbf{r}\, \mathbf{E} \cdot \hat{\mathbf{t}} = -\int_{\Sigma} d\mathbf{r}\, \partial_t \mathbf{B} \cdot \hat{\mathbf{n}} \tag{2.4}$$

$$\oint_{\partial\Omega} d\mathbf{r}\, \mathbf{D} \cdot \hat{\mathbf{n}} = \int_{\Omega} d\mathbf{r}\, \rho \tag{2.5}$$

$$\oint_{\partial\Sigma} d\mathbf{r}\, \mathbf{H} \cdot \hat{\mathbf{t}} = \int_{\Sigma} d\mathbf{r}\, (\mathbf{J} + \partial_t \mathbf{D}) \cdot \hat{\mathbf{n}} \tag{2.6}$$

$$\oint_{\partial\Omega} d\mathbf{r}\, \mathbf{B} \cdot \hat{\mathbf{n}} = 0 \tag{2.7}$$

Here, $\Omega \in \mathbb{R}^3$ is some arbitrary connected domain and $\partial\Omega$ is its boundary, with $\hat{\mathbf{n}}$ the outer unit normal to $\partial\Omega$. Σ is some arbitrary area and $\hat{\mathbf{n}}$ is a unit normal to Σ; $\partial\Sigma$ is the boundary of Σ with $\hat{\mathbf{t}}$ being a unit vector tangential to $\partial\Sigma$. $\hat{\mathbf{n}}$ and $\hat{\mathbf{t}}$ are smooth on their respective domains and related by the right hand rule: $\hat{\mathbf{t}} \times \hat{\mathbf{n}}$ is a unit vector on and outward to $\partial\Sigma$.

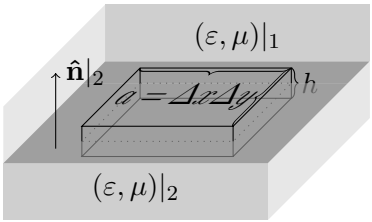


Figure 2.3.: Integration volume Ω for the divergence equations.

There are general relations regarding the continuity of the electromagnetic field at material discontinuities. To investigate this, let the space be filled with two different materials:

$$\varepsilon(\mathbf{r}) = \begin{cases} \varepsilon^1 & \mathbf{r}\cdot\hat{\mathbf{n}}|_2 < 0 \\ \varepsilon^2 & \mathbf{r}\cdot\hat{\mathbf{n}}|_2 > 0 \end{cases} \tag{2.8}$$

$$\mu(\mathbf{r}) = \begin{cases} \mu^1 & \mathbf{r}\cdot\hat{\mathbf{n}}|_2 < 0 \\ \mu^2 & \mathbf{r}\cdot\hat{\mathbf{n}}|_2 > 0 \end{cases} \tag{2.9}$$

Here, $\hat{\mathbf{n}}|_2$ is a unit vector normal to the interface between the two materials, pointing into material 1. Consider a volume Ω with flat sides of area a parallel to the material interface and on either side of it, and side walls perpendicular to the interface and of height h. I'll allow a surface charge $\rho(\mathbf{r}) = \delta(\mathbf{r}\cdot\hat{\mathbf{n}}|_2)\rho^f(\mathbf{r})$ on the interface, all other fields are assumed to be constant in each region. Then I can compute the integrals in (2.5)

$$\oint_{\partial\Omega} d\mathbf{r}\, \mathbf{D}\cdot\hat{\mathbf{n}} = a(\mathbf{D}|_1 - \mathbf{D}|_2)\cdot\hat{\mathbf{n}}|_2 + O(h) \tag{2.10}$$

$$\int_\Omega d\mathbf{r}\, \rho = a\rho^f \tag{2.11}$$

Taking the limes $h \to 0$ yields

$$(\mathbf{D}|_1 - \mathbf{D}|_2)\cdot\hat{\mathbf{n}}|_2 = \rho^f. \tag{2.12}$$

For (2.7) the argument is the same, except that there are no magnetic charges and thus the right hand side is immediately 0.

$$(\mathbf{B}|_1 - \mathbf{B}|_2)\cdot\hat{\mathbf{n}}|_2 = 0 \tag{2.13}$$

For the other two equation I consider a rectangular area Σ with two sides of length l parallel to the interface, one inside either material, and two sides of length h across the interface and perpendicular to it. I'll allow a surface current $\mathbf{J}(\mathbf{r}) = \delta(\mathbf{r}\cdot\hat{\mathbf{n}}|_2)\mathbf{J}^f(\mathbf{r})$ on the interface with $\mathbf{J}^f\cdot\hat{\mathbf{n}}|_2 = 0$, all other fields are assumed to be constant in each region.

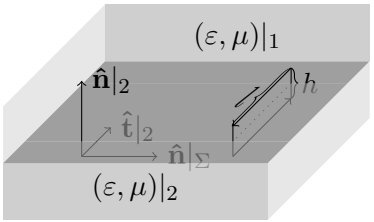


Figure 2.4.: Integration area Σ for the curl equations.

Then I can compute the integrals in (2.6)

$$\oint_{\partial\Sigma} d\mathbf{r}\, \mathbf{H}\cdot\hat{\mathbf{t}} = l(\mathbf{H}|_2 - \mathbf{H}|_1)\cdot\hat{\mathbf{t}}|_2 + O(h), \tag{2.14}$$

$$\int_{\Sigma} d\mathbf{r}\, (\mathbf{J} + \partial_t \mathbf{D})\cdot\hat{\mathbf{n}}|_\Sigma = l\mathbf{J}^f\cdot\hat{\mathbf{n}}_\Sigma + O(h). \tag{2.15}$$

$\hat{\mathbf{t}}|_2$ is the tangential vector along the side of Σ that lies inside material 2, and $\hat{\mathbf{n}}|_\Sigma$ is the normal on Σ. They are related with the material interface normal by $\hat{\mathbf{t}}|_2 = \hat{\mathbf{n}}|_2 \times \hat{\mathbf{n}}|_\Sigma$ and all three are mutually perpendicular. Using this, shifting the terms in the resulting box product, and letting again $h \to 0$ results in

$$((\mathbf{H}|_2 - \mathbf{H}|_1)\times\hat{\mathbf{n}}|_2)\cdot\hat{\mathbf{n}}|_\Sigma = \mathbf{J}^f\cdot\hat{\mathbf{n}}_\Sigma \tag{2.16}$$

I did not specify a particular orientation of $\hat{\mathbf{n}}|_\Sigma$, except that it must be perpendicular to $\hat{\mathbf{n}}|_2$, so this must hold for all orientations of $\hat{\mathbf{n}}|_\Sigma$ that are in the plane of the material interface, and I can identify the terms on the left side of the scalar product

$$(\mathbf{H}|_2 - \mathbf{H}|_1)\times\hat{\mathbf{n}}|_2 = \mathbf{J}^f. \tag{2.17}$$

For (2.4) the argument is the same, except that there is no surface current, and I obtain

$$(\mathbf{E}|_2 - \mathbf{E}|_1)\times\hat{\mathbf{n}}|_2 = 0. \tag{2.18}$$

In the absence of surface charges this implies continuity of the normal component of $\mathbf{D}$ and $\mathbf{B}$ across arbitrary interfaces, and similar continuity of the tangential components of $\mathbf{E}$ and $\mathbf{H}$ in the absence of surface currents.

This can be used to derive often used boundary conditions: consider the boundary of a *perfect electric conductor* with arbitrarily large conductivity $\lim\sigma = \infty$. Any field electric field $\mathbf{E} \neq 0$ in this material would lead to arbitrarily large currents $\lim\mathbf{J} = \mathbf{J}^s + \mathbf{E}\lim\sigma = \infty$, thus the electric fields can only be $\mathbf{E} = 0$. Using (2.18) I can show that the limes of the tangential electric field towards a perfectly conducting wall must go to zero

$$\mathbf{E}\times\hat{\mathbf{n}} = 0 \qquad \text{on } \Gamma^{\mathrm{PEC}}. \tag{2.19}$$

The corresponding boundary condition for the **H** field is called in analogy a *perfect magnetic conductor* boundary condition (PMC) although there are, of course, no magnetic monopoles.

$$\mathbf{H} \times \hat{\mathbf{n}} = 0 \qquad \text{on } \Gamma^{\text{PMC}} \tag{2.20}$$

The boundary conditions lead to perfect reflection of waves incident on the boundary. For the **E** field PEC corresponds to constrained mechanical boundary, while for **H** it corresponds to an open one. For PMC it is the other way around: for **H** it corresponds to a constrained and for **E** to an open mechanical boundary.

2.2.2. Time Harmonic Maxwell's equations

When all fields are periodic in time with a periodicity $\omega/(2\pi)$, the fields can be written as

$$\mathbf{E}^t = \Re\mathbf{E}^\omega e^{-i\omega t} \qquad \mathbf{H}^t = \Re\mathbf{H}^\omega e^{-i\omega t} \tag{2.21}$$

$$\mathbf{D}^t = \Re\mathbf{D}^\omega e^{-i\omega t} \qquad \mathbf{B}^t = \Re\mathbf{B}^\omega e^{-i\omega t} \tag{2.22}$$

$$\mathbf{J}^t = \Re\mathbf{J}^\omega e^{-i\omega t} \qquad \rho^t = \Re\rho^\omega e^{-i\omega t} \tag{2.23}$$

I've made it explicit here whether a quantity is considered to be in the time domain or the frequency domain by using superscripts t and ω, respectively. Normally, when such a superscript is missing, the quantity is in the time domain.

Inserting this into Maxwell's equations makes it possible to conduct the temporal derivatives

$$-i\omega\mathbf{B}^\omega + \nabla \times \mathbf{E}^\omega = 0 \tag{2.24}$$

$$\nabla \cdot \mathbf{D}^\omega = \rho \tag{2.25}$$

$$-i\omega\mathbf{D}^\omega - \nabla \times \mathbf{H}^\omega = -\mathbf{J}^\omega \tag{2.26}$$

$$\nabla \cdot \mathbf{B}^\omega = 0. \tag{2.27}$$

Similarly, the equation of continuity becomes

$$-i\omega\rho^\omega + \nabla\mathbf{J}^\omega = 0 \tag{2.28}$$

It is implicitly understood that only the real part of the fields corresponds to any physical quantity. For a given ω the complex argument of these fields has the meaning of a phase angle. In addition the time-dependent term $e^{-i\omega t}$ is identical for all fields and normally dropped.

The time-harmonic and the time-dependent Maxwell's equations are related by a Fourier transform. Some problems are easier solved in the frequency domain, and a solution in the time domain can be obtained by a backtransformation.

2.2.3. Plane Waves

A general plane wave is any electromagnetic field of the form

$$\mathbf{E}^t = \mathbf{E}^0 f(\mathbf{k}\cdot\mathbf{r} - \omega t), \tag{2.29}$$

$$\mathbf{H}^t = \mathbf{H}^0 g(\mathbf{k}\cdot\mathbf{r} - \omega t). \tag{2.30}$$

More commonly however, *plane wave* denotes a fundamental solution of Maxwell's equations in homogeneous isotropic charge-free space. Plane waves are useful since any solution to Maxwell's equations can be decomposed as a superposition of plane waves.

In the frequency domain, plane waves have the form (for the electric field)

$$\mathbf{E}^\omega = \mathbf{E}^0 e^{i\mathbf{k}\cdot\mathbf{r}}. \tag{2.31}$$

$\mathbf{E}^0$ is the amplitude of the electric field. It can include a complex part which corresponds to a phase shift. As I shall show, $\mathbf{E}^0$ cannot be chosen completely arbitrarily, it must be perpendicular to the wave vector $\mathbf{k}$.

Inserting (2.31) into (2.24) and using $\mathbf{B} = \mu\mathbf{H}$ yields

$$\mathbf{H}^\omega = \frac{1}{\omega\mu}\mathbf{k}\times\mathbf{E}^0 e^{i\mathbf{k}\cdot\mathbf{r}} =: \mathbf{H}^0 e^{i\mathbf{k}\cdot\mathbf{r}}, \tag{2.32}$$

the magnetic field of the plane wave. Note that the amplitude $\mathbf{H}^0$ of the magnetic field must be perpendicular both to the amplitude of the electric $\mathbf{E}^0$ field and the wave vector $\mathbf{k}$.

Inserting (2.32) into (2.26) and using $\mathbf{D} = \varepsilon\mathbf{E}$ yields:

$$\mathbf{E}^\omega = -\frac{1}{\omega\varepsilon}\mathbf{k}\times\mathbf{H}^0 e^{i\mathbf{k}\cdot\mathbf{r}} = \mathbf{E}^0 e^{i\mathbf{k}\cdot\mathbf{r}} \tag{2.33}$$

The identity to the right is the ansatz (2.31). This shows that for any solution of Maxwell's equations of the form (2.31) the amplitude of the electric field $\mathbf{E}^0$ must be perpendicular to the wave vector $\mathbf{k}$, and I find that $\mathbf{k}$, $\mathbf{E}^0$ and $\mathbf{H}^0$ are mutually perpendicular and it follows that plane electromagnetic waves are transverse.

I can combine (2.33) and (2.32) and use $\varepsilon\mu c^2 = 1$

$$\mathbf{E}^0 = -\frac{1}{\omega^2 c^2}\mathbf{k}\times\mathbf{k}\times\mathbf{E}^0 = \frac{1}{\omega^2 c^2}(\mathbf{k}\cdot\mathbf{k})\mathbf{E}^0 \tag{2.34}$$

This results in a relation between $\mathbf{k}$ and ω:

$$c^2\omega^2 = \mathbf{k}^2 \tag{2.35}$$

For a given wave number $\mathbf{k}$ $\mathbf{E}^0$ has still two degrees of freedom. Without loss of generality, assume $\mathbf{k} = k\hat{\mathbf{e}}^x$ pointing in direction of the x-axis. Then there are two possible linear polarizations: $\mathbf{E}^{0,h} = E^{0,h}\hat{\mathbf{e}}^y$ and $\mathbf{E}^{0,v} = E^{0,v}\hat{\mathbf{e}}^z$, commonly denoted *horizontal* and *vertical*. (This naming convention obviously does not make sense for all

possible orientation of $\mathbf{k}$.) These two linear polarizations can be combined to obtain two circular polarizations:

$$\mathbf{E}^{0,r} = \mathbf{E}^{0,h} - i\mathbf{E}^{0,v} \tag{2.36}$$

$$\mathbf{E}^{0,l} = \mathbf{E}^{0,h} + i\mathbf{E}^{0,v} \tag{2.37}$$

At any given time on any line parallel to $\mathbf{k}$ the $\mathbf{E}^r$ field vector traces a the threads of a right handed screw through space while the $\mathbf{E}^l$ field vector traces the threads of a left handed screw.

Returning to the time domain is as simple as reintroducing the time-dependent term and taking the real part:

$$\mathbf{E}^t = \mathbf{E}^0 e^{i\mathbf{k}\cdot\mathbf{r}-i\omega t} \tag{2.38}$$

$$\mathbf{H}^t = \mathbf{H}^0 e^{i\mathbf{k}\cdot\mathbf{r}-i\omega t} \tag{2.39}$$

2.3. Numerical formulation

Maxwell's equations can be solved in many ways numerically. There are *frequency domain* and *time domain solvers*. There a solvers that work with the fields directly and solvers that work with the scalar and vector potentials. There are solvers that use Maxwell's equation as a first order *system* and solvers that transform it into a second order *wave equation* first. For scattering analysis it is often useful to split the field into *incident* and *scattered field*, in contrast to the *total field* that is usually used otherwise. Primary variables can be the fields $\mathbf{E}$ and $\mathbf{H}$ or the fluxes $\mathbf{D}$ and $\mathbf{B}$. In this section I shall describe some of the aspects of these formulations, sufficient for the numerical schemes introduced in a later chapter.

2.3.1. System Formulation

The system formulation is almost identical to the two Maxwell equations (2.3a) and (2.3c). The only difference is the insertion of the constitutive relations such that only primary variables are present:

$$\partial_t \varepsilon \mathbf{E} - \nabla \times \mathbf{H} + \sigma \mathbf{E} = -\mathbf{J}^s \tag{2.40}$$

$$\partial_t \mu \mathbf{H} + \nabla \times \mathbf{E} \qquad = 0 \tag{2.41}$$

This is the formulation used by the famous finite difference scheme of Yee[40].

I shall use the system formulation for the discontinuous Galerkin scheme as well. However, I shall use $\mathbf{D}$ and $\mathbf{B}$ as primary variables:

$$\begin{aligned} \partial_t \mathbf{D} - \nabla \times \mu^{-1}\mathbf{B} + \frac{\sigma}{\varepsilon}\mathbf{D} &= -\mathbf{J}^s \\ \partial_t \mathbf{B} + \nabla \times \varepsilon^{-1}\mathbf{D} \qquad &= 0 \end{aligned} \tag{2.42}$$

This will make it slightly simpler to show hyperbolicity.

In both cases, the divergence equations $\nabla \cdot \mathbf{D} = \rho$ and $\nabla \cdot \mathbf{B} = 0$ were not used. They are a compatibility requirement on the initial conditions; if they are fulfilled once the other two of Maxwell's equation (2.3a) and (2.3c) ensure that they are always fulfilled.

2.3.2. Wave Formulation

The finite element method will use a wave formulation as described in [23]. This can be obtained by combining the two equation in the system form (2.40) and keeping $\mathbf{E}$ as the primary variable:

$$\partial_t^2 \varepsilon \mathbf{E} + \nabla \times (\mu^{-1} \nabla \times \mathbf{E}) + \partial_t \sigma \mathbf{E} = -\partial_t \mathbf{J}^s \tag{2.43}$$

Here, the compatibility requirement on the initial conditions is slightly modified. $\nabla \cdot \mathbf{B} = 0$ becomes irrelevant, since the magnetic field is only present implicitly as the curl of $\mathbf{E}$ insuring that it is automatically fulfilled. Applying the divergence operation to (2.43) an using the equation of continuity results in

$$\partial_t^2 \nabla \cdot \mathbf{D} = \partial_t^2 \rho \tag{2.44}$$

i.e. there are now two integration constants to fix to ensure that $\nabla \cdot \mathbf{D} = \rho$ is automatically fulfilled. This can be done either by requiring that $\nabla \cdot \mathbf{D} = \rho$ be fulfilled for two distinct points in time

$$\nabla \cdot \mathbf{D} = \rho \qquad \text{at } t^1 \text{ and } t^2 \text{ with } t^1 \neq t^2 \tag{2.45}$$

or by requiring that

$$\nabla \cdot \mathbf{D} = \rho \qquad \text{at } t^1 \text{ and} \tag{2.46}$$

$$\partial_t \nabla \cdot \mathbf{D} = \partial_t \rho \qquad \text{at } t^2 \tag{2.47}$$

with t^1 and t^2 possibly, but not necessarily equal.

2.4. Theory of GPR

Soil usually consists of several closely intermixed components. The most obvious is the soil matrix, the solid part. What is left over is a network of pores, the pore space. The pore space is often (partially) filled with water. Finally, the rest of the pore space is filled with air, which must not be neglected. In situations such as contaminated soil there may be additional components such as oil that don't mix with the other components, but I will not consider these here.

The exact geometry of the individual pores, the microstructure, is of little interest scientifically. Knowing the geometry on the *micro scale* would enable us to solve the relevant molecular-scale equations, but that would be prohibitively expensive and would provide no abstraction: the exact pore geometry of every soil sample is different, and we would be unable to apply conclusion obtained from one soil sample to another similar soil sample.

What we are interested in then are averaged parameters on a larger scale, the *macro scale* or *continuum scale*. I'll show the averaging process using the volume fraction θ as an example. I'll mark quantities on the micro scale by a superscript $\cdot^{\mu}$. Quantities on the macro scale will be left unmarked.

The volume fraction can be computed by

$$\theta^i(\mathbf{r}) = \langle \chi^{\mu,i} \rangle(\mathbf{r}) = \int_{\mathbb{R}^3} d\mathbf{r}' \, \chi^{\mu,i}(\mathbf{r} + \mathbf{r}') \kappa(\mathbf{r}') \tag{2.48}$$

$\chi^{\mu,i} : \mathbb{R}^3 \to \{0, 1\}$ is the *indicator function* for phase i:

$$\chi^i(\mathbf{r}) := \begin{cases} 1 & \mathbf{r} \in \text{ phase } i, \\ 0 & \mathbf{r} \notin \text{ phase } i. \end{cases} \tag{2.49}$$

It is of course a microscopic quantity. $\kappa : \mathbb{R}^3 \to \mathbb{R}^+$ is the weight function. It must fulfill two properties: the weights should be positive $\kappa(\mathbf{r}) \geq 0 \forall \mathbf{r}$ and the total of all weights should be one $\int_{\mathbb{R}^3} d\mathbf{r} \, \kappa(\mathbf{r}) = 1$ so as not to distort the derived macroscopic quantities. To obtain a traditional averaging volume, use κ that is zero outside a certain radius σ and constant inside. Another choice would be a suitably scaled Gaussian with standard deviation σ, which has the advantage that the derived macroscopic quantities are smooth (but the disadvantage of an infinite support). κ is called the *representative elementary volume* (REV). The name is used even though κ is not always actually a "volume", i.e. in the case of the Gaussian.

One very useful quantity that can be derived from the volume fractions is the porosity. It is the volume fraction of the pore space, or equivalently $\phi(\mathbf{r}) = 1 - \theta^m(\mathbf{r})$, where "$m$" denotes the matrix. Other useful quantities are the volumetric water content $\theta^w(\mathbf{r})$ (with "w" denoting water), and the saturation $\Theta^w = \theta^w / \phi$

While the choice of the shape of κ is mostly one of convenience, the extend σ of κ is more restricted. This is the question what length scales are characteristic for the micro- and the macroscale, respectively. Lets consider the typical size of the pores, for instance. If we chose σ to be roughly of this size, $\theta^{m,i}$ will vary wildly with position and will provide no abstraction compared to $\chi^{\mu,i}$. If we make σ larger the variations will become smaller. This is illustrated in figure 2.5. Some soils may contain macroscopic heterogeneities. When σ reaches the size of these heterogeneities, the variation of the porosity may actually increase again.

2.4.1. Water Content and Dielectric Permittivity

I am interested to relate the dielectric permittivity to the water content in the soil. The velocity of light in some material i can be written as

$$c^i = \frac{c^0}{n^i} \tag{2.50}$$

with c^0 the velocity of light in vacuum and n^i the refractive index of material i. The velocity of light in material is lower than in vacuum; from the point of view of a photon it appears as if the traveled distance is longer in material. This leads to the notion of the optical path length, which is given by

$$l^i := l n^i \tag{2.51}$$

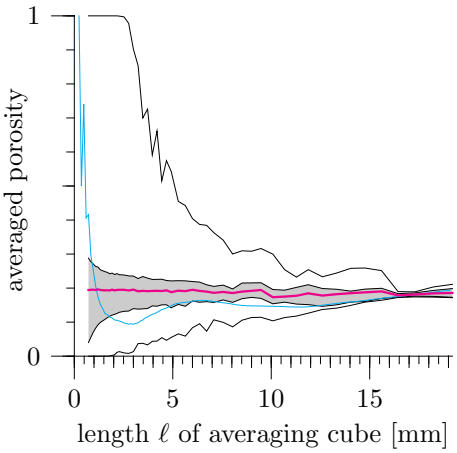


Figure 2.5.: Estimated porosity of a soil sample with known microscale geometry and uniform macrostructure as a function of the averaging cube's length. The cyan curve represents a particular location. The other curves represent the ensemble of all cubes: average (magenta), minimum and maximum and the two quartiles. Half of all values are within the gray band. The linear extend of a reasonable REV would be some 17mm. Figure taken from *Soil Physics* [35] and used with permission.

in material i, i.e. it is just the optical path length in vacuum scaled by the index of refraction. Using the procedure above we can compute a macroscopic refractive index for the composite material:

$$n(\mathbf{r}) = \int_{\mathbb{R}^3} d\mathbf{r}' \sum_i n^i \chi^{\mu,i}(\mathbf{r}+\mathbf{r}')\kappa(\mathbf{r}') \tag{2.52}$$

$$= \sum_i n^i \int_{\mathbb{R}^3} d\mathbf{r}' \, \chi^{\mu,i}(\mathbf{r}+\mathbf{r}')\kappa(\mathbf{r}') \tag{2.53}$$

$$= \sum_i n^i \theta^i(\mathbf{r}) \tag{2.54}$$

The magnetic permeability is equal to the vacuum permeability $\mu^i = \mu^0$ for water, air and many matrix materials, and I can use $n^i = c^0\sqrt{\mu^i\varepsilon^i}$ to obtain for the permittivities

$$\sqrt{\varepsilon(\mathbf{r})} = \sum_i \theta^i(\mathbf{r})\sqrt{\varepsilon^i} \tag{2.55}$$

This is known as the *complex refractive index model* or *CRIM formula* for short. It was first derived in [4] for the case of a complex permittivity $\varepsilon = \varepsilon' + i\varepsilon''$ by considering the effect on the actual electric field. It is actually just one of a class of more general models of the form

$$\varepsilon(\mathbf{r})^\alpha = \sum_i \theta^i(\mathbf{r})(\varepsilon^i)^\alpha \tag{2.56}$$

which were examined by Brown [5]. He found, among other situations, for layered material $\alpha = 1$ if the electric field is parallel to the layers and $\alpha = -1$ if the electric field is perpendicular to the layers. Roth *et al.* examined the situation in soils more closely in [36], and found $\alpha = 0.46$. This is not too far of from the CRIM, so that is the one usually used.

3. Spatial Discretization

Solving Maxwell's equations in the time domain using numerical simulations reaches back as far as 1966 when Yee published a finite difference scheme using a pair of staggered grids for electric and magnetic fields[40]. Nearly fifty years later, this scheme is still heavily used in many applications, and is often sufficient. It is however no longer the only standard scheme.

Finite element schemes offer the advantage of working on unstructured grids and promise higher order convergence. Initially they suffered from so called *spurious solutions*, leading to results that did not make sense physically. The reason was the use of nodal elements, and with the use of edge-based elements these problems have been overcome. Today, finite element schemes can be considered standard too for solving electromagnetic problems. They do however have the drawback of a non-diagonal mass matrix, making it necessary to solve a matrix in each time step.

Discontinuous Galerkin schemes have a successful history in the area of hyperbolic conservation laws, e. g. [9]. They were first introduced for neutron transport by Reed and Hill in 1973[33] and subsequently analyzed by Lesaint and Raviart in 1974[26]. Shortly afterwards they were extended to parabolic and elliptic problems, see e. g. [11, 3, 38]. These methods were developed mostly independent from the hyperbolic methods and became known as interior penalty methods.

Hesthaven and Warburton in 2002 [18] and 2004 [19, 20] introduced a discontinuous Galerkin method for Maxwell's equations, based on the formulation as a conservation law. At the same time, Cockburn, Li, and Shu examined a DG scheme using locally a locally divergence-free basis[8]. In 2006 Lou and Jin proposed[28] a dual-field element-level decomposition method based on the second order wave equation. In [16] Grote, Schneebeli and Schötzau proposed an interior penalty scheme which solves the second order wave equation.

We shall describe the standard finite element method for electromagnetics as well as a discontinuous Galerkin method. We use the method of lines: we first do a semi-discretization in space, and solve the resulting system of ordinary differential equations (ODEs) with an appropriate time stepper. We describe the spatial discretization in the current chapter and the temporal discretization in the next chapter. Finally, in chapter 5 we describe the solvers used to solve the linear equation systems resulting from the temporal discretization.

3.1. Finite Elements

Electromagnetic problems are often posed in an infinite domain, with the exception of cavity problems. However, computation power is finite, when discretizing the problem

the computational domain employed is usually finite, too, and special boundary conditions are needed to approximate the infinite domain of the original problem. These boundary conditions must ensure that any wave leaving the computational domain must be able to do so with minimum disturbance. In addition, for scattering problems, the boundary condition should make it possible to prescribe incoming waves.

3.1.1. Boundary Conditions

One way to implement the truncation is to use boundary integrals, leading to a hybrid finite element-boundary integral method. This has the advantage that there are no approximations involved in the boundary condition. However, the assembly of the linear system is computationally expensive, scaling with the square of the size of the boundary.

We will use an approximate boundary condition derived from the Sommerfeld radiation condition. Boundary conditions of this type are known as *radiating boundary conditions* (RBC) or *absorbing boundary conditions* (ABC). Although there can be notable reflections at the boundary, boundary conditions of this type are relatively simple to implement, have little computational overhead and are often sufficient.

A third possibility are *perfectly matched layers* (PML). Here the computational domain is surrounded by a layer with special non-physical material parameters that dampen any waves traveling through it but still avoid reflections back into the domain of interest. This perfectly matched layer can be backed on its outer boundary by a reflecting boundary condition, or, for added accuracy, by an ABC.

Let Ω be the domain of interest with $\partial\Omega$ its boundary. All sources and sinks must be contained in Ω. Then the time-harmonic field in the region outside of Ω must fulfill the vector wave equation

$$\nabla \times \nabla \times \mathbf{E}(\mathbf{r}, t) - k^2 \mathbf{E}(\mathbf{r}, t) = 0 \qquad \text{in } \mathbb{R}^3 \setminus \Omega \tag{3.1}$$

and the Sommerfeld radiation condition

$$\lim_{r \to \infty} r[\nabla \times \mathbf{E}(\mathbf{r}, \omega) + jk\hat{\mathbf{r}} \times \mathbf{E}(\mathbf{r}, \omega)] = 0. \tag{3.2}$$

Such a field can be described by an infinite series, and a matching series of operators can be constructed on the domain boundary, each annihilating terms of the field up to a certain order. Each operator yields an ABC accurate to a certain order. However, we'll just note that the operator of the lowest order happens to be the Sommerfeld radiation condition applied at $\partial\Omega$ instead of infinity with $\hat{\mathbf{r}}\times$ applied:

$$\hat{\mathbf{r}} \times \nabla \times \mathbf{E}(\mathbf{x}, \omega) + jk\hat{\mathbf{r}} \times \hat{\mathbf{r}} \times \mathbf{E}(\mathbf{x}, \omega) = 0 \tag{3.3}$$

Using $k = \omega/c = \omega\sqrt{\varepsilon\mu}$, the intrinsic admittance $Y = \sqrt{\varepsilon/\mu}$ and the Fourier transform of derivatives of some function $f(t)$

$$\mathcal{F}[\partial_t^n f(t)] = (j\omega)^n \mathcal{F}[f(t)] \tag{3.4}$$

we obtain the first order ABC in the time domain:

$$\hat{\mathbf{r}} \times [\mu^{-1} \nabla \times \mathbf{E}(\mathbf{r}, t)] + Y \partial_t [\hat{\mathbf{r}} \times \hat{\mathbf{r}} \times \mathbf{E}(\mathbf{r}, t)] = 0 \tag{3.5}$$

There are some more hidden approximations here: First, in this derivation, the domain Ω containing all sources is usually assumed to be spherical such that $\|\mathbf{r}\|$ is constant along $\partial\Omega$. When doing numerical computations, especially for GPR, a spherical domain is usually not feasible. So we replace the direction of the position vector $\hat{\mathbf{r}}$ by the outer normal $\hat{\mathbf{n}}$ of $\partial\Omega$. This will lead to errors in the corners of the domain.

Second, the derivation assumes a uniform outer medium; in particular homogeneous ε in $\mathbb{R}^3 \setminus \Omega$. This is not true for GPR: here we have different layers in the ground and a layer of air on top, extending at least to the domain boundary. Assuming that these stop at the domain boundary (i. e. $\varepsilon = \varepsilon_0$ on $\partial\Omega$) would lead to reflections. The other possibility, the one we have chosen, is to extend ε from Ω to $\partial\Omega$. This on the other hand violates the precondition that ε is uniform outside Ω, and will lead to reflections at the domain boundary where ε changes.

3.1.2. Weak Formulation

We will use the wave formulation (2.43) as a starting point:

$$\nabla \times (\mu^{-1}\nabla \times \mathbf{E}(\mathbf{r},t)) + \varepsilon\partial_t^2\mathbf{E}(\mathbf{r},t) + \sigma\partial_t\mathbf{E}(\mathbf{r},t) = -\partial_t\mathbf{J}_i(\mathbf{r},t) \qquad \text{in } \Omega \tag{3.6}$$

On the boundary I impose a mixed boundary condition sufficient to implement an absorbing boundary condition

$$\hat{\mathbf{n}} \times (\mu^{-1}\nabla \times \mathbf{E}(\mathbf{r},t)) + Y\partial_t(\hat{\mathbf{n}} \times \hat{\mathbf{n}} \times \mathbf{E}(\mathbf{r},t)) = \mathbf{U}(\mathbf{r},t) \qquad \text{on } \partial\Omega \tag{3.7}$$

This closely follows [23].

We call the space of electric fields U such that $\mathbf{E} \in U$. To derive the weak formulation we demand that (3.6) is fulfilled in a weak sense: we take the L^2 scalar product with some test function $\mathbf{F} \in U$ and the result must hold for all $\mathbf{F}$.

$$\begin{aligned}\int_\Omega d\mathbf{r}\left\{\nabla \times (\mu^{-1}\nabla \times \mathbf{E}(\mathbf{r},t))\right\} \cdot \mathbf{F}(\mathbf{r}) + \partial_t^2 \int_\Omega d\mathbf{r}\varepsilon\mathbf{E}(\mathbf{r},t)\cdot\mathbf{F}(\mathbf{r}) \\ + \partial_t \int_\Omega d\mathbf{r}\sigma\mathbf{E}(\mathbf{r},t)\cdot\mathbf{F}(\mathbf{r}) = -\int_\Omega d\mathbf{r}\partial_t\mathbf{J}_i(\mathbf{r},t)\cdot\mathbf{F}(\mathbf{r}) \qquad \forall\mathbf{F}\in U\end{aligned} \tag{3.8}$$

In this approach both the *ansatz functions* $\mathbf{E}$ and the *test functions* $\mathbf{F}$ are from the same space U. This is called the Galerkin approach, it is used by finite element and discontinuous Galerkin schemes. One notable class of schemes where ansatz and test functions are from different spaces if finite volumes.

The double $\nabla\times$ operator would be problematic when discretizing with lowest order base function, so we reformulate the first term using Green's first vector theorem. The negative sign in the resulting surface integral vanishes when reordering the integrand.

$$\begin{aligned}\int_\Omega d\mathbf{r}\mu^{-1}(\nabla \times \mathbf{E}(\mathbf{r}))\cdot(\nabla\times\mathbf{F}(\mathbf{r})) + \oint_{\partial\Omega} d\mathbf{r}(\hat{\mathbf{n}}\times\mu^{-1}\nabla\times\mathbf{E}(\mathbf{r},t))\cdot\mathbf{F}(\mathbf{r}) \\ + \partial_t^2\int_\Omega d\mathbf{r}\varepsilon\mathbf{E}(\mathbf{r},t)\cdot\mathbf{F}(\mathbf{r}) + \partial_t\int_\Omega d\mathbf{r}\sigma\mathbf{E}(\mathbf{r},t)\cdot\mathbf{F}(\mathbf{r}) \\ = -\int_\Omega d\mathbf{r}\partial_t\mathbf{J}_i(\mathbf{r},t)\cdot\mathbf{F}(\mathbf{r}) \qquad \forall\mathbf{F}\in U\end{aligned} \tag{3.9}$$

We can now apply (3.7) on the boundary:

$$\begin{aligned}\int_\Omega d\mathbf{r}\mu^{-1}(\nabla\times\mathbf{E}(\mathbf{r}))\cdot(\nabla\times\mathbf{F}(\mathbf{r}))+\partial_t\oint_{\partial\Omega}d\mathbf{r}Y(\hat{\mathbf{n}}\times\mathbf{E}(\mathbf{r},t))\cdot(\hat{\mathbf{n}}\times\mathbf{F}(\mathbf{r}))\\ +\partial_t^2\int_\Omega d\mathbf{r}\varepsilon\mathbf{E}(\mathbf{r},t)\cdot\mathbf{F}(\mathbf{r})+\partial_t\int_\Omega d\mathbf{r}\sigma\mathbf{E}(\mathbf{r},t)\cdot\mathbf{F}(\mathbf{r})\\ =-\int_\Omega d\mathbf{r}\partial_t\mathbf{J}_i(\mathbf{r},t)\cdot\mathbf{F}(\mathbf{r})-\oint_{\partial\Omega}d\mathbf{r}\mathbf{U}(\mathbf{r},t)\cdot\mathbf{F}(\mathbf{r})\qquad\forall\mathbf{F}\in U\end{aligned}\tag{3.10}$$

We approximate U by a suitable finite element space U^h with the base $\{\mathbf{N}^{(i)}\}$ and substitute $\mathbf{F}$ by its approximation

$$\mathbf{F}(\mathbf{r})\approx\mathbf{F}^h(\mathbf{r})=\sum_j F_j\mathbf{N}^{(j)}(\mathbf{r})\tag{3.11}$$

Since (3.10) is linear in $\mathbf{F}^h$, demanding that it holds for any $\mathbf{F}^h\in U^h$ is equivalent to demanding that it is fulfilled for any base function $\mathbf{N}^{(j)}$.

$$\begin{aligned}\int_\Omega d\mathbf{r}\mu^{-1}(\nabla\times\mathbf{E}(\mathbf{r}))\cdot(\nabla\times\mathbf{N}^{(j)}(\mathbf{r}))+\partial_t\oint_{\partial\Omega}d\mathbf{r}Y(\hat{\mathbf{n}}\times\mathbf{E}(\mathbf{r},t))\cdot(\hat{\mathbf{n}}\times\mathbf{N}^{(j)}(\mathbf{r}))\\ +\partial_t^2\int_\Omega d\mathbf{r}\varepsilon\mathbf{E}(\mathbf{r},t)\cdot\mathbf{N}^{(j)}(\mathbf{r})+\partial_t\int_\Omega d\mathbf{r}\sigma\mathbf{E}(\mathbf{r},t)\cdot\mathbf{N}^{(j)}(\mathbf{r})\\ =-\int_\Omega d\mathbf{r}\partial_t\mathbf{J}_i(\mathbf{r},t)\cdot\mathbf{N}^{(j)}(\mathbf{r})-\oint_{\partial\Omega}d\mathbf{r}\mathbf{U}(\mathbf{r},t)\cdot\mathbf{N}^{(j)}(\mathbf{r})\qquad\forall j\end{aligned}\tag{3.12}$$

We approximate $\mathbf{E}$ in the same manner by $\mathbf{E}^h$

$$\mathbf{E}(\mathbf{r},t)\approx\mathbf{E}^h(\mathbf{r},t)=\sum_i E_i(t)\mathbf{N}^{(i)}(\mathbf{r}).\tag{3.13}$$

This way, we have a separation of time and space; the base functions $\mathbf{N}^{(i)}$ depend only on the spatial coordinates while the coefficients E_i depend only on time. This results in the matrix equation

$$\mathsf{T}\partial_t^2E(t)+(\mathsf{R}+\mathsf{Q})\partial_tE(t)+\mathsf{S}E(t)+f(t)=0\tag{3.14}$$

with the matrices given by

$$\mathsf{T}_{ij}=\int_\Omega d\mathbf{r}\varepsilon\mathbf{N}^{(i)}(\mathbf{r})\cdot\mathbf{N}^{(j)}(\mathbf{r})\tag{3.15}$$

$$\mathsf{R}_{ij}=\int_\Omega d\mathbf{r}\sigma\mathbf{N}^{(i)}(\mathbf{r})\cdot\mathbf{N}^{(j)}(\mathbf{r})\tag{3.16}$$

$$\mathsf{Q}_{ij}=\oint_{\partial\Omega}d\mathbf{r}Y(\hat{\mathbf{n}}\times\mathbf{N}^{(i)}(\mathbf{r}))\cdot(\hat{\mathbf{n}}\times\mathbf{N}^{(j)}(\mathbf{r}))\tag{3.17}$$

$$\mathsf{S}_{ij}=\int_\Omega d\mathbf{r}\mu^{-1}(\nabla\times\mathbf{N}^{(i)}(\mathbf{r}))\cdot(\nabla\times\mathbf{N}^{(j)}(\mathbf{r}))\tag{3.18}$$

and the vector f by

$$f_j=\int_\Omega d\mathbf{r}\partial_t\mathbf{J}_i\cdot\mathbf{N}^{(j)}(\mathbf{r})+\oint_{\partial\Omega}d\mathbf{r}\mathbf{U}\cdot\mathbf{N}^{(j)}(\mathbf{r}).\tag{3.19}$$

3.1.3. $H(\mathrm{curl})$-conforming Base Functions

The definition of the $H(\mathrm{curl})$-conforming base functions was taken from [23]. I'm going to reiterate the most important aspects here, in particular using a notation appropriate for Dune.

These elements are commonly known under a number of names: *Whitney elements* after a book by H. Whitney from 1957[39], *Nedelec elements* after a paper by J. C. Nedelec from 1980[31], *edge elements* since in the lowest order the degrees of freedom are located on the mesh element edges, *tangential (vector) finite elements* since they provide for tangential continuity across mesh element boundaries, sometimes even simply *vector finite elements* since they are vector-valued and cannot be decomposed into scalar-valued base functions (although there are other finite elements that share this property), and finally $H(\mathrm{curl})$*-conforming finite elements* since they are conforming in $H(\mathrm{curl})$.

There are also incompatible nomenclatures regarding the order of these basis functions. [23] calls the lowest order the *zeroth order*, [31] calls the lowest order the *first order*. Some people even compromise on calling the lowest order the *0.5th order* (e.g. [2]). We are dealing with lowest order basis functions only. Where the order matters, we will call them 0.5^{th} order, following the compromise, since that provides the least potential for confusion.

For the purpose of this work we implemented these elements in the `dune-localfunctions` module for both 2D and 3D.

Given the P^1 basis functions $\{L^{(i)}\}$ with the property $L^{(i)}(\mathbf{x}^{(j)}) = \delta_{ij}$ for the mesh element's vertices $\mathbf{x}^{(i)}$, the edge shape functions are

$$\mathbf{N}^{(i)} = \frac{f^{(i)}}{\ell^{(i)}}(L^{(i^{(0)})}\nabla L^{(i^{(1)})} - L^{(i^{(1)})}\nabla L^{(i^{(0)})}) \tag{3.20}$$

Here, i is an index for the edge within the mesh element, $\ell^{(i)}$ is the length of edge i and $i^{(0)}$ and $i^{(1)}$ are the indices of the vertices of edge i within the mesh element. $f^{(i)} \in \{-1, 1\}$ is the orientation of the edge; this is needed so we can ensure tangential continuity when constructing basis functions out of the shape functions on neighboring mesh elements. For a given edge in some mesh element we determine $f^{(i)}$ by comparing the ordering of edge's vertices within the mesh element to the global ordering of the same vertices: if they match then $f^{(i)} = 1$, otherwise $f^{(i)} = -1$.

Interpolation using lowest order edge elements converges with h in the $H(\mathrm{curl})$ norm[31] and thus in the L^2 norm.

3.1.4. A Note on Spurious Modes

The most common way to solve a cavity problem is to use an eigenvalue solver: The eigenmodes are the fundamental solutions, and the corresponding eigenvalues are the squared frequencies of these modes. The result is then a spectrum of eigenvalues, giving the frequencies supported by the cavity.

In the earliest node-based finite element models this approach led to so called *spurious modes*, observed as nonphysical frequencies in the spectrum. The introduction of edge

finite element methods resolved the problem of the spurious modes. In [37] Webb provided a good explanation why node-based elements suffer from spurious modes and edge based elements do not: in essence the $\nabla \times \nabla \times$ operator has a null subspace of modes of the form $-\nabla\Phi$ for some scalar field Φ. Analytically, these modes have frequencies $\omega = 0$. However, with node-based finite elements these modes are approximated badly, leading to $\omega > 0$, i.e. spurious frequencies in the spectrum. Edge elements spend a far greater part of the finite element space on modeling the null space and are better at approximating these modes, leading to $\omega = 0$, so these modes don't appear in the spectrum.

That explains the situation for eigenvalue solvers. For time-domain solvers this has the following consequences: Any excitation, whether by currents or by initial conditions will likely also excite modes from the null space of the form $-\nabla\Phi$. These modes are unaffected by the wave equation (2.43), meaning they will stay constant over time. If $\sigma = 0$ the situation is even worse: then $-\partial_t \nabla\Phi$ will be unaffected too, leading to a linear increase of the maximum/minimum field values over time, and to a quadratic increase of the energy density $(\mathbf{E} \cdot \mathbf{D} + \mathbf{B} \cdot \mathbf{H})/2$.

This can be avoided by choosing the excitations carefully. We'll give here some thumb rules that excitations should satisfy to avoid trouble. They are somewhat stricter than actually necessary, but most of the time this does not pose problems.

An excitation by current sources should have the following properties:

$$\int_{t^{\text{start}}}^{t^{\text{end}}} \mathbf{J} \approx 0 \tag{3.21}$$

$$\int_{t^{\text{start}}}^{t^{\text{end}}} \partial_t \mathbf{J} \approx 0 \tag{3.22}$$

Violation of the first property means that charges are left in the domain after t^{end}. Actually, since this has a physical meaning, this may sometimes be desired. Violation of the second property means that a current is left in the domain after t^{end}. This will lead to the linear increase in field values described above. An excitation with a Gaussian temporal shape violates the first property and is thus seldom used. Note that the integrals are discrete ones as used by the time stepper—a numerical error here may become relevant for long enough computations. So in addition it may be necessary to choose the time step size carefully.

For a problem without charges, initial conditions should fulfill the compatibility condition

$$\nabla \cdot \varepsilon \mathbf{E} = 0 \tag{3.23}$$

More important however is the second condition

$$\nabla \cdot \varepsilon \partial_t \mathbf{E} = 0 \tag{3.24}$$

If this is violated, then there is a current in the domain, which will, again, lead to the maximum field values increasing linearly in time. Keep in mind that the initial condition need to be evaluated at $t^{(0)} - \Delta t$ in addition to $t^{(0)}$. The easiest way to fulfill

this condition is to start with $\mathbf{E}(t^{(0)} - \Delta t) = \mathbf{E}(t^{(0)}) = 0$ and this is most often done. However, then excitation must happen via other means. Another possibility is to use $\mathbf{E}(t^{(0)} - \Delta t) = \mathbf{E}(t^{(0)})$. This avoids any currents in the domain, but charges may still be present—these are however much less problematic.

3.2. Discontinuous Galerkin

For the discontinuous Galerkin scheme we chose to use the system formulation, since this can easily be written as a hyperbolic conservation law. The choice of $\mathbf{D}$ and $\mathbf{B}$ as primary variables makes it easier to show hyperbolicity.

The DG scheme is built up as follows: First we derive a general DG scheme for hyperbolic conservation laws of fields $u(\mathbf{r}, t)$ of m components – e.g. for scalars we would have $m = 1$, for spatial vector fields we would have $m = 3$, and for Maxwell we will have $m = 6$ (since we collect the components of $\mathbf{D}$ and $\mathbf{B}$ into one 6-component vector).

This scheme will require numerical upwind fluxes which we will derive in a second step. In the scalar (and linear) case this decision is straightforward, but for the case of a hyperbolic system the upwind side differs for different components of the solution. Here, it helps to consider the Riemann problem. The usual method for a hyperbolic system is to transform the solution into a basis of *characteristic variables* and doing an upwind decision for each component of the transformed variable individually. However, this is only valid for homogeneous material. To still be able to derive an upwind flux, we need to consider the *Rankine-Hugoniot conditions* of the problem.

It should be noted that the same expression for the numerical flux can also be derived by considering the physics of interfaces in the context of Maxwell's equation. This has been done e.g. by Mohammadian, Shankar and Hall in [30] for finite volumes schemes.

3.2.1. DG for Hyperbolic Conservation Laws

Consider a hyperbolic conservation law

$$\partial_t u + \sum_{j=1}^{n} \partial_j F^{(j)}(u) = q \qquad \text{in } \Omega \times \Sigma \tag{3.25}$$

with the unknown function $u = u(\mathbf{r}, t) \in \mathbb{R}^m$, the fluxes $F^{(j)}(u) = F^{(j)}(u(\mathbf{r}, t), \mathbf{r}, t) \in \mathbb{R}^m$, $j = 1, \ldots, n$ and the source term $q = q(\mathbf{r}, t) \in \mathbb{R}^m$. $\Omega \subset \mathbb{R}^n$ is the spatial domain and $\Sigma = (t^{(0)}, \infty)$ is the temporal domain. We will usually omit explicitly writing the dependency on space and time. Initial and boundary conditions are specified as

$$u(\mathbf{r}, t^{(0)}) = u^{(0)}(\mathbf{r}) \qquad \text{in } \Omega, \tag{3.26}$$

$$u(\mathbf{r}, t) = g(\mathbf{r}, t) \qquad \text{on } \partial\Omega. \tag{3.27}$$

(3.25) can be written in a more compact form as

$$\partial_t u + \operatorname{div} \mathsf{F}(u) = q \qquad \text{in } \Omega \times \Sigma \tag{3.28}$$

if we define $\mathsf{F}_{ji} = F_i^{(j)}$ and use $\operatorname{div} \mathsf{F}(u) = (\nabla \cdot \mathsf{F}(u))^T$.

To derive a discontinuous Galerkin scheme for this problem, let us first introduce some notation. Let $\mathcal{T}^h$ be a decomposition of Ω into elements (a mesh): $\bar{\Omega} = \bigcup_{T \in \mathcal{T}^h} \bar{T}$ and $T^{(1)} \cap T^{(2)} = \emptyset$ for $T^{(1)} \neq T^{(2)}$ and $T^{(1)}, T^{(2)} \in \mathcal{T}^h$. Let $\Gamma = \bigcup_{T \in \mathcal{T}^h} \partial T$ be the skeleton of $\mathcal{T}^h$ and let $\Gamma^{(0)} = \Gamma \setminus \partial\Omega$ be the interior skeleton. For every mesh element $T \in \mathcal{T}^h$ and every point $\mathbf{r} \in \partial T$, let $\hat{\mathbf{n}}|_T$ denote a unit outer normal to T in $\mathbf{r}$. For every point $\mathbf{r} \in \Gamma$, let $T^- \in \mathcal{T}^h$ denote an element such that $\mathbf{r} \in \partial T^-$. Note: this choice of T^- is only necessary for the sake of notation. The final result does not depend on the particular choice. For every point $\mathbf{r} \in \Gamma^{(0)}$, let $T^+ \in \mathcal{T}^h$ denote an element such that $\mathbf{r} \in \partial T^+$ and $\hat{\mathbf{n}}|_{T^-} = -\hat{\mathbf{n}}|_{T^+}$. When the element is T^-, we will usually simply write $\hat{\mathbf{n}}$ instead of $\hat{\mathbf{n}}|_{T^-}$. Let $(\,\cdot\,,\,\cdot\,)_\omega$ denote the bilinear form

$$(u, v)_\omega := \int_\omega d\mathbf{r}\, u \cdot v = \int_\omega d\mathbf{r} \sum_{i=1,\dots,m} u_i v_i \qquad u, v : \omega \to \mathbb{R}^m \tag{3.29}$$

for vector-valued functions and

$$(\mathsf{U}, \mathsf{V})_\omega := \int_\omega d\mathbf{r}\, \mathsf{U} : \mathsf{V} = \int_\omega d\mathbf{r} \sum_{\substack{i=1,\dots,m \\ j=1,\dots,n}} \mathsf{U}_{ij} \mathsf{V}_{ij} \qquad \mathsf{F}, \mathsf{G} : \omega \to \mathbb{R}^{m \times n} \tag{3.30}$$

for matrix-valued functions. Here, ":" denotes the Frobenius inner product. When both arguments are from the same function space, this bilinear form is just the L^2 scalar product over a domain ω.

On any mesh element $T \in \mathcal{T}^h$ we require that the conservation law is fulfilled in a weak sense: for (possibly) vector-valued test functions v we require that

$$\partial_t (u, v)_T + (\operatorname{div} \mathsf{F}(u), v)_T = (q, v)_T \qquad \forall v \in V(T). \tag{3.31}$$

Using $\operatorname{grad} v = \nabla v^T$ and applying integration by parts yields

$$\partial_t (u, v)_T - (\mathsf{F}(u), \operatorname{grad} v)_T + ((\hat{\mathbf{n}} \cdot \mathsf{F}(u))^T, v)_{\partial T} = (q_t, v)_T \qquad \forall v \in V(T) \tag{3.32}$$

Next we combine this for all mesh elements to obtain a weak formulation for the whole domain. We do this by asserting for any two mesh elements $T^{(1)}$ and $T^{(2)}$, that whatever flux leaves one element via the common interface $I := \bar{T}^{(1)} \cup \bar{T}^{(2)}$ must enter the other element – otherwise there would be sources or sinks at mesh element boundaries. This means that the normal flux must be continuous across I: $\hat{\mathbf{n}} \cdot [\mathsf{F}(u)]_{T^{(1)}} = \hat{\mathbf{n}} \cdot [\mathsf{F}(u)]_{T^{(2)}}$. This requirement is what makes this scheme conservative. We require the test functions v and ansatz function u to be differentiable in each mesh element T, but allow them to be discontinuous across mesh element boundaries.

We call $[\![v]\!] := v|_{T^+} - v|_{T^-}$ and $[\![u]\!] := u|_{T^+} - u|_{T^-}$ the jump of v and u respectively on $\Gamma^{(0)}$, and for convenience we extend $v|_{T^+} = 0$ and $u|_{T^+} = g$ on $\partial\Omega$ such that the jump is even defined on the domain boundary.

Note that the jump changes sign depending on the choice of T^-. The flux function F depends on the value of the ansatz function u, which may be discontinuous, but the

normal flux value needs to be continuous. We resolve this by replacing the normal flux with the numerical flux $\hat{F}$, which depends on both values of u

$$(\hat{\mathbf{n}} \cdot \mathsf{F}(u))^T \to \hat{F}(u|_{T^-}, u|_{T^+}) \qquad \text{on } \Gamma. \tag{3.33}$$

For the scheme to be consistent the numerical flux needs to equal the normal flux when the normal flux has no jump: for u such that

$$f := (\hat{\mathbf{n}} \cdot [\mathsf{F}(u)]_{T^-})^T = (\hat{\mathbf{n}} \cdot [\mathsf{F}(u)]_{T^+})^T \tag{3.34}$$

it must also hold that

$$\hat{F}(u|_{T^-}, u|_{T^+}) = f \tag{3.35}$$

for all $\mathbf{r} \in \Gamma$.

Then the weak formulation for the whole domain is

$$\partial_t (u, v)_\Omega - (\mathsf{F}(u), \operatorname{grad} v)_\Omega + (\hat{F}(u|_{T^-}, u|_{T^-}), [\![v]\!])_\Gamma = (q, v)_\Omega \qquad \forall v \in V(\Omega) \tag{3.36}$$

Note that, since the choice of T^- and T^+ is arbitrary, and the jump $[\![\cdot]\!]$ changes sign when their meaning is exchanged, $\hat{F}$ must change it sign too so the value of $(\hat{F}(u|_{T^-}, u|_{T^-}), [\![v]\!])_\Gamma$ is independent of this choice.

To complete the spatial discretization, we need to approximate u and v in some basis

$$u(t, \mathbf{r}) \approx \sum_i u_i^h(t) \phi^{(i)}(\mathbf{r}) \qquad \operatorname{span}\{\phi^{(i)}\} = U^h \approx U \tag{3.37}$$

$$v(\mathbf{r}) \approx \sum_j v_j^h \psi^{(j)}(\mathbf{r}) \qquad \operatorname{span}\{\psi^{(j)}\} = V^h \approx V \tag{3.38}$$

and the weak formulation becomes

$$\mathsf{M} \partial_t u^h(t) + \mathsf{A} u^h(t) = f^h(t) \tag{3.39}$$

with

$$\mathsf{M}_{ji} = (\phi^{(i)}, \psi^{(j)})_\Omega, \tag{3.40}$$

$$\mathsf{A}_{ji} = -(\mathsf{F}(\phi^{(i)}), \operatorname{grad} \psi^{(j)})_\Omega + \left(\hat{F}(\phi^{(i)}|_{T^-}, \phi^{(i)}|_{T^+}), \left[\!\left[\psi^{(j)} \right]\!\right] \right)_\Gamma, \tag{3.41}$$

$$f_j^h(t) = (q_t, \psi^{(j)})_\Omega. \tag{3.42}$$

Since this is a Galerkin scheme, ansatz and test space are identical $U = V$ and are approximated in the same way $U^h = V^h$. For convenience we also assume that the base functions $\phi^{(i)} = \psi^{(i)}$ are identical.

What is left now is to derive a suitable numerical flux $\hat{F}$.

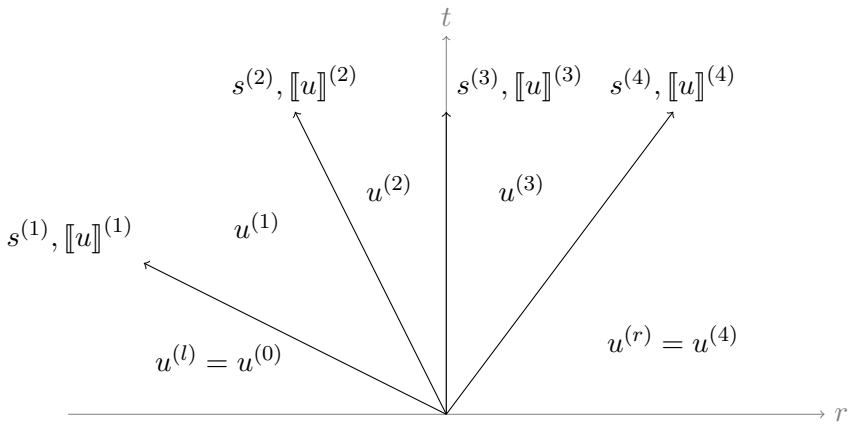


Figure 3.1.: Example of a general solution to the linear Riemann problem. In this example there are four discontinuities, each with a jump $[\![u]\!]^{(i)} = u^{(i)} - u^{(i-1)}$ in u and traveling with a speed $s^{(i)}$. $u^{(i)}$, $i = 0, \dots, 4$ are the values of the solution constant in between the waves. $u^{(l)}$ and $u^{(r)}$ are the values of the initial condition.

3.2.2. Numerical Flux

To solve a conservation law using a DG scheme we need to integrate expressions of the form

$$\hat{\mathbf{n}} \cdot \mathsf{F}(u, \mathbf{r}) v \qquad \text{on } \partial T \tag{3.43}$$

on the mesh cell surfaces ∂T, for any test function v. The interesting part here is the value of the normal flux $\hat{\mathbf{n}} \cdot \mathsf{F}(u, \mathbf{r})$; since this is a DG scheme u may have a discontinuity at ∂T. It is wrong to simply use the value u in T, since that would lead to a different value if the normal flux is evaluated instead in the neighboring mesh element T'. The normal flux needs to be continuous across mesh cell interfaces – otherwise there would be sources at the interfaces.

We solve this by substituting the numerical flux $\hat{F}(u|_{T^-}, u|_{T^+})$ for the normal flux $(\hat{\mathbf{n}} \cdot \mathsf{F}(u))^T$. We require the value of the numerical flux to be the same as the normal flux, i.e. $\hat{F}(u, u) = (\hat{\mathbf{n}} \cdot \mathsf{F}(u))^T$, when there is no jump in the value of u.

Riemann-Problem

One way to derive a numerical flux is to consider the Riemann problem. This is simply a conservation law

$$\partial_t u + \partial_r f(u) = 0 \qquad \text{in } L \times \Sigma = \mathbb{R} \times (t^{(0)}, \infty) \tag{3.44}$$

on a line L (with a coordinate denoted by r), together with a step as initial condition to model the discontinuity.

$$u(r, 0) := \begin{cases} u^{(l)} & r < 0, \\ u^{(r)} & r > 0. \end{cases} \tag{3.45}$$

As before in (3.28), the flux function f may also directly depend on r and t, even though we don't explicitly write it.

A key component in solving the Riemann problem is the condition of Rankine and Hugoniot, which puts restriction on the kind of discontinuities that may be present in a solution. Given a solution u for the conservation law (3.44) and a curve $C = \{(\sigma(t), t)\}$ that divides the domain $L \times \Sigma$ into a left part $N^{(l)}$ and a right part $N^{(r)}$, then the Rankine-Hugoniot condition requires that

$$s^{(C)} [\![u]\!]^{(C)} = [\![f(u)]\!]^{(C)} \qquad \forall (s,t) \in C \cap L \times \Sigma. \tag{3.46}$$

Here we use the notation

$$[\![x]\!]^{(C)} = \lim_{\substack{r' \to r \\ r' > r}} x|_{(r',t)} - \lim_{\substack{r' \to r \\ r' < r}} x|_{(r',t)} \qquad (r,t) \in C \tag{3.47}$$

to denote the jump in x across C, and $s^{(C)} = d\sigma/dt$ to denote the speed of C. A derivation of this condition is given e.g. in chapter 3 of [34].

For Maxwell it is sufficient to only consider linear fluxes $f(u) = \mathsf{A}(r)u$. For the derivation of a numerical flux it is sufficient to consider a piecewise constant coefficient matrix of the form

$$\mathsf{A}(r) = \begin{cases} \mathsf{A}^{(l)} & r < 0, \\ \mathsf{A}^{(r)} & r > 0. \end{cases} \tag{3.48}$$

In this case the solution of the problem for $t > 0$ consists of a set of waves, of discontinuities in u, see figure 3.1 for a general example. Each wave is traveling with a constant speed left or right or is stationary at $r = 0$. Each wave has a certain constant amplitude, which is just the jump in u. In between these waves, u remains at a constant value, depending neither on r nor t.

A solution to the Riemann problem will yield an expression for the flux $\hat{f} = f(u)|_{r=0}$ across $r = 0$ for all times $t > t^{(0)}$, even for discontinuous initial conditions. This can be used to obtain a numerical flux for a conservation law in arbitrary dimensions: each time we need a numerical flux in a point $\mathbf{P}$ at a cell boundary, we consider the Riemann problem at a line L in direction $\hat{\mathbf{n}}$ perpendicular to that cell boundary, with a coordinate $r = (\mathbf{r} - \mathbf{P}) \cdot \hat{\mathbf{n}}$ such that $\mathbf{P}$ is at $r = 0$. The two values of u inside and outside of the cell, $u|_{T^-}$ and $u|_{T^+}$ are used as the values $u^{(l)}$ and $u^{(r)}$ for the initial conditions of the Riemann problem, and the value of the numerical flux will be $\hat{F}(u|_{T^-}, u|_{T^+}) = \hat{f}$.

It is instructive to consider the scalar case of the general linear Riemann problem first, and the system case later. Together with the variable- vs. constant-coefficient classification we obtain a total of four cases, which we will consider in turn.

Scalar Constant-Coefficients Case

Here the conservation law takes the form

$$\partial_t u + \partial_r (au) = 0, \tag{3.49}$$

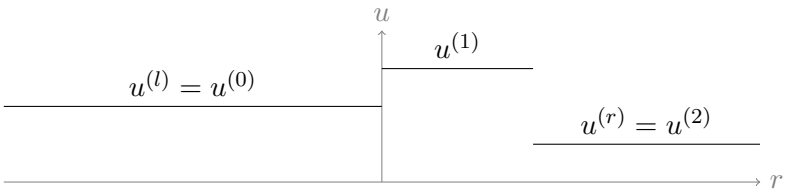


Figure 3.2.: Solution of the scalar variable-coefficients Riemann problem for $t > 0$ and $a^{(l)} > a^{(r)} > 0$.

and the initial condition is

$$u(r,0) = \begin{cases} u^{(l)} & r < 0, \\ u^{(r)} & r > 0. \end{cases} \tag{3.50}$$

Any Rankine-Hugoniot condition (3.46) takes the form

$$v \left[\!\left[u\right]\!\right] = a \left[\!\left[u\right]\!\right]. \tag{3.51}$$

Since the jump $[\![u]\!]$ is arbitrary, this means we have a single discontinuity, traveling at constant speed $v = a$. To the left and to the right of the discontinuity the solution is constant in r and t:

$$u(r,t) = \begin{cases} u^{(l)} & r < at, \\ u^{(r)} & r > at. \end{cases} \tag{3.52}$$

For our DG scheme we are interested in the flux at $r = 0$, which is

$$\hat{f} = f(u)|_{r=0} = \begin{cases} au^{(l)} & a > 0, \\ au^{(r)} & a < 0. \end{cases} \tag{3.53}$$

Scalar Variable-Coefficients Case

The conservation law now takes the form

$$\partial_t u + \partial_r(a(r)u) = 0. \tag{3.54}$$

For our application it is sufficient to consider piecewise constant coefficients of the form

$$a(r) = \begin{cases} a^{(l)} & r < 0, \\ a^{(r)} & r > 0, \end{cases} \tag{3.55}$$

corresponding the coefficients inside and outside of T from the nD problem. Let $a^{(l)}, a^{(r)} > 0$; then the case of $a^{(l)}, a^{(r)} < 0$ can be obtained by replacing $r \to -r$. We don't consider the case of $a^{(l)}$ and $a^{(r)}$ having different sign or being zero since we don't need it for our purpose.

Since both $a^{(l)}$ and $a^{(r)}$ are positive, any discontinuity in u to either side of $r = 0$ must travel rightward; this follows from the constant-coefficient case. For the region

$r < 0$ this means that there can be no discontinuity since there is no discontinuity in the initial condition to begin with. For the region $r > 0$ there is potentially a discontinuity originating from $(r, t) = (0, 0)$; it must adhere to the Rankine-Hugoniot condition

$$a^{(r)}(u^{(r)} - u^{(*)}) = f^{(r)}(u^{(r)}) - f^{(r)}(u^{(*)}) \tag{3.56}$$

Unfortunately, this does not yield any information about the jump at the discontinuity, since this also follows exactly from the definition of the fluxes.

The last possibility for a discontinuity is a stationary one at $r = 0$. Here the Rankine-Hugoniot condition yields that the flux must be continuous

$$a^{(l)}u^{(l)} = a^{(r)}u^{(*)}, \tag{3.57}$$

which lets us calculate the value of u between the two waves:

$$u^{(*)} = \frac{a^{(l)}}{a^{(r)}}u^{(0)}. \tag{3.58}$$

The complete solution in the case $a^{(l)}, a^{(r)} > 0$ is then

$$u(r,t) = \begin{cases} u^{(l)} & r < 0, \\ \frac{a^{(l)}}{a^{(r)}}u^{(l)} & 0 < r < a^{(r)}t, \\ u^{(r)} & a^{(r)}t < r. \end{cases} \tag{3.59}$$

This is illustrated in figure 3.2. By symmetry, in the case $a^{(l)}, a^{(r)} < 0$ the solution is

$$u(r,t) = \begin{cases} u^{(l)} & r < a^{(l)}t, \\ \frac{a^{(r)}}{a^{(l)}}u^{(r)} & a^{(l)}t < r < 0, \\ u^{(r)} & 0 < r. \end{cases} \tag{3.60}$$

In both cases the flux at $r = 0$ is

$$\hat{f} = f(u)|_{r=0} = \begin{cases} a^{(l)}u^{(l)} & a^{(l)}, a^{(r)} > 0, \\ a^{(r)}u^{(u)} & a^{(l)}, a^{(r)} < 0. \end{cases} \tag{3.61}$$

This case is important because it shows that a discontinuity in the solution can arise from a discontinuity in the material parameters, even if the initial conditions are chosen such that $u^{(l)} = u^{(r)}$.

Constant-Coefficients System Case

In this case the conservation law takes the form

$$\partial_t u + \partial_r(\mathsf{A}u) = 0. \tag{3.62}$$

We're only considering hyperbolic conservation laws, so the coefficient matrix A must be real diagonalizable, i.e. $\mathsf{A} = \mathsf{R}\mathsf{D}\mathsf{R}^{-1}$, where R can be constructed such that its columns

are eigenvectors $e^{(i)}$ of A, and D_{ii} are the eigenvalues of A. By defining the characteristic variables $\tilde{u} = \mathsf{R}^{-1}u$ and left-multiplying the conservation law with R we obtain the diagonal form

$$\partial_t \tilde{u} + \partial_r(\mathsf{D}\tilde{u}) = 0. \tag{3.63}$$

This however is just a set of independent conservation laws, of which each can be solved as in the scalar constant-coefficients case. By splitting $\mathsf{D} = \mathsf{D}^+ + \mathsf{D}^-$ such that

$$\mathsf{D}^+_{ii} := \begin{cases} \mathsf{D}_{ii} & \mathsf{D}_{ii} > 0, \\ 0 & \mathsf{D}_{ii} \leq 0, \end{cases} \qquad \mathsf{D}^-_{ii} := \begin{cases} \mathsf{D}_{ii} & \mathsf{D}_{ii} < 0, \\ 0 & \mathsf{D}_{ii} \geq 0, \end{cases} \tag{3.64}$$

and transforming the initial conditions into characteristic variables

$$\tilde{u}^{(l)} = \mathsf{R}^{-1}u^{(l)} \qquad \tilde{u}^{(r)} = \mathsf{R}^{-1}u^{(r)} \tag{3.65}$$

we are able to write the flux $\tilde{f}$ in characteristic variables at $r = 0$ as

$$\tilde{f}(\tilde{u})|_{r=0} = \mathsf{D}^+\tilde{u}^{(l)} + \mathsf{D}^-\tilde{u}^{(r)}. \tag{3.66}$$

It is now useful to define a splitting of $\mathsf{A} = \mathsf{A}^+ + \mathsf{A}^-$ with $\mathsf{A}^\pm = \mathsf{R}\mathsf{D}^\pm\mathsf{R}^{-1}$. We can back-transform this flux into original coordinates

$$\hat{f} = f(u)|_{r=0} = \mathsf{R}\tilde{f}(\tilde{u})|_{r=0} = \mathsf{A}^+u^{(l)} + \mathsf{A}^-u^{(r)}. \tag{3.67}$$

From (3.63) it follows that the actual solution $u(r,t)$ contains one discontinuity for each component (sets of degenerate eigenvalues will yield only one discontinuity). The jump at the discontinuity is a multiple of an eigenvector of A, and the discontinuity travels with a speed determined by the corresponding eigenvalue. Using the Rankine-Hugoniot condition (3.46) at each discontinuity, we can connect the left and the right state by

$$[\![\mathsf{A}u]\!] = \mathsf{A}u^{(r)} - \mathsf{A}u^{(l)} = \mathsf{A}u^{(m)} - \mathsf{A}u^{(0)} = \sum_{i=1}^{m} \mathsf{D}_{ii}(u^{(i)} - u^{(i-1)}). \tag{3.68}$$

This suggests an alternate way to determine the intermediate states, which shall be useful in the case of a variable-coefficient system.

Variable-Coefficients System Case

In this case the conservation law takes the form

$$\partial_t u + \partial_r(\mathsf{A}(r)u) = 0. \tag{3.69}$$

As in the scalar case we'll assume piecewise constant coefficients

$$\mathsf{A}(r) = \begin{cases} \mathsf{A}^{(l)} & r < 0, \\ \mathsf{A}^{(r)} & r > 0, \end{cases} \tag{3.70}$$

Unfortunately, we cannot transform the system into the diagonal form anymore. We can do so just for the left side, or just for the right side, but in general (and in particular for Maxwell) the eigenvectors of $\mathsf{A}^{(l)}$ will be not match those of $\mathsf{A}^{(r)}$. It is still possible to find solutions and fluxes, following [27], although in that paper only waves traveling left and right are needed and the flux at $r = 0$ is only considered tangentially.

The idea to the solution is to connect the left and the right state by a chain Rankine-Hugoniot conditions derived from (3.46) to determine the intermediate states. These Rankine-Hugoniot conditions have the following form:

$$s^{(i)} \left[\!\left[u\right]\!\right]^{(i)} = \mathsf{A}^{(l)} \left[\!\left[u\right]\!\right]^{(i)} \qquad i = 1, \ldots, \theta - 1 \tag{3.71}$$

$$0 = s^{(\theta)} \left[\!\left[u\right]\!\right]^{(\theta)} = \mathsf{A}^{(r)} u^{(\theta)} - \mathsf{A}^{(l)} u^{(\theta-1)} \tag{3.72}$$

$$s^{(i)} \left[\!\left[u\right]\!\right]^{(i)} = \mathsf{A}^{(r)} \left[\!\left[u\right]\!\right]^{(i)} \qquad i = \theta + 1, \ldots, d \tag{3.73}$$

This means that for leftward traveling discontinuities the jump $\left[\!\left[u\right]\!\right]^{(i)}$ at the discontinuity is an eigenvector of $\mathsf{A}^{(l)}$ and the speed of the discontinuity is the corresponding eigenvalue. Likewise for right traveling waves, however the jump and the speed are now an eigenpair of $\mathsf{A}^{(r)}$. We give any stationary discontinuity the index θ and count the discontinuities from fastest left-traveling (index 1) to fastest right-traveling (index d).

Since $s^{(\theta)} = 0$, the Rankine-Hugoniot condition (3.72) does not directly restrict the jump $\left[\!\left[u\right]\!\right]^{(\theta)}$ in the solution, it only requires that the flux at $r = 0$ is continuous. We can use either $f(0,t) = \mathsf{A}^{(l)} u^{(\theta-1)}$ or $f(0,t) = \mathsf{A}^{(r)} u^{(\theta)}$ to compute it. By inserting the Rankine-Hugoniot conditions for the corresponding side, this gives us

$$f(0,t) = \mathsf{A}^{(l)} u^{(l)} + \sum_{i=1}^{\theta-1} s^{(i)} \left[\!\left[u\right]\!\right]^{(i)} \tag{3.74}$$

or

$$f(0,t) = \mathsf{A}^{(r)} u^{(r)} - \sum_{i=\theta+1}^{d} v^{(i)} \left[\!\left[u\right]\!\right]^{(i)} . \tag{3.75}$$

To use these expressions we need to determine the magnitude of each $s^{(i)} \left[\!\left[u\right]\!\right]^{(i)}$.

To do this we construct a base $\{r^{(j)}\}$ from the eigenvectors to the negative eigenvalues of $\mathsf{A}^{(l)}$ and the eigenvectors to the positive eigenvalues of $\mathsf{A}^{(r)}$. We call the eigenvalue corresponding to $r^{(j)}$ $\lambda(r^{(j)})$. We can now express $s^{(i)} \left[\!\left[u\right]\!\right]^{(i)}$ in terms of this basis as

$$s^{(i)} \left[\!\left[u\right]\!\right]^{(i)} = \sum_{j:\lambda(r^{(j)})=s^{(i)}} w_j r^{(j)} \qquad i \neq \theta \tag{3.76}$$

This yields the expression

$$\hat{f} = f(u)|_{r=0} = \mathsf{A}^{(l)} u^{(l)} + \sum_{j:\lambda(r^{(j)})<0} w_j r^{(j)} \tag{3.77}$$

which computes the flux across the interface starting from the left and

$$\hat{f} = f(u)|_{r=0} = \mathsf{A}^{(r)}u^{(r)} - \sum_{j:\lambda(r^{(j)})>0} w_j r^{(j)}, \tag{3.78}$$

which computes the flux across the interface starting from the right. These can be combined by eliminating $\hat{f}$ to

$$\sum_j w_j r^{(j)} = \mathsf{A}^{(r)}u^{(r)} - \mathsf{A}^{(l)}u^{(l)}. \tag{3.79}$$

Introducing M as the matrix which has the basis vectors $r^{(i)}$ as its columns we see that this is a linear equation system that needs to be solved for the coefficients w.

$$\mathsf{M}w = [\![\mathsf{A}u]\!] \tag{3.80}$$

It should be noted that the matrix M is in general not square, since it does not include columns for eigenvectors to 0-eigenvalues of $\mathsf{A}^{(l)}$ or $\mathsf{A}^{(r)}$, so it may be more convenient to solve the system

$$\mathsf{M}^T\mathsf{M}w = \mathsf{M}^T [\![\mathsf{A}u]\!] \tag{3.81}$$

instead.

Once w has been determined, the flux can be obtained with the help of either of (3.77) or (3.78).

3.2.3. Application to Maxwell

Now that the general framework is in place we can apply it to Maxwell's equations. We use the system form of Maxwell's equations in $\mathbf{D}$ and $\mathbf{B}$ (2.42). For $\sigma = 0$ it may be written as

$$\partial_t u + \partial_x(\mathsf{A}^{(x)}u) + \partial_y(\mathsf{A}^{(y)}u) + \partial_z(\mathsf{A}^{(z)}u) = j \tag{3.82}$$

with

$$\mathsf{A}^{(x)} = \begin{pmatrix} 0 & -\mu^{-1}\mathsf{C}(\hat{\mathbf{e}}^{(x)}) \\ \varepsilon^{-1}\mathsf{C}(\hat{\mathbf{e}}^{(x)}) & 0 \end{pmatrix}, \tag{3.83a}$$

$$\mathsf{A}^{(y)} = \begin{pmatrix} 0 & -\mu^{-1}\mathsf{C}(\hat{\mathbf{e}}^{(y)}) \\ \varepsilon^{-1}\mathsf{C}(\hat{\mathbf{e}}^{(y)}) & 0 \end{pmatrix}, \tag{3.83b}$$

$$\mathsf{A}^{(z)} = \begin{pmatrix} 0 & -\mu^{-1}\mathsf{C}(\hat{\mathbf{e}}^{(z)}) \\ \varepsilon^{-1}\mathsf{C}(\hat{\mathbf{e}}^{(z)}) & 0 \end{pmatrix}, \tag{3.83c}$$

and

$$j = \begin{pmatrix} -\mathbf{J}^s \\ 0 \end{pmatrix}, \qquad u = \begin{pmatrix} \mathbf{D} \\ \mathbf{B} \end{pmatrix}. \tag{3.83d}$$

$\mathsf{C}(\mathbf{r})$ is a 3×3 matrix that has the same effect as a cross product with $\mathbf{r}$ from the left: $\mathsf{C}(\mathbf{r})\mathbf{r}' = \mathbf{r} \times \mathbf{r}'$. It has the form

$$\mathsf{C}(\mathbf{r}) = \begin{pmatrix} 0 & -\mathbf{r}_z & \mathbf{r}_y \\ \mathbf{r}_z & 0 & -\mathbf{r}_x \\ -\mathbf{r}_y & \mathbf{r}_x & 0 \end{pmatrix}. \tag{3.84}$$

We use the remainder of this section to derive the numerical flux using the methodology developed in section 3.2.2. We begin by restricting (3.82) to a line. This results a single coefficient matrix $\mathsf{A}(\hat{\mathbf{n}})$, and we continue by deriving a set of eigenvectors for it. We then use the eigenvectors to assemble the matrix $\mathsf{M}^T\mathsf{M}$ needed for (3.81). Inversion of $\mathsf{M}^T\mathsf{M}$ and insertion of the resulting coefficients w is done in appendix A. In this section we simply conclude by giving the resulting flux, and by comparing it to the flux we would derive for constant coefficients.

Riemann Problem for Maxwell

To compute the numerical flux, we restrict (3.82) to a line L perpendicular to the interface. Then $\hat{\mathbf{n}}$ is a unit vector along this line, and we let r denote a coordinate on this line, such that $\partial_r = \hat{\mathbf{n}} \cdot \operatorname{div} = \hat{\mathbf{n}}_x\partial_x + \hat{\mathbf{n}}_y\partial_y + \hat{\mathbf{n}}_z\partial_z$. Then (3.82) becomes

$$\partial_t u + \partial_r(\mathsf{A}u) = j \qquad \text{on } L, \tag{3.85}$$

where $\mathsf{A} = \hat{\mathbf{n}}_x\mathsf{A}^{(x)} + \hat{\mathbf{n}}_y\mathsf{A}^{(y)} + \hat{\mathbf{n}}_z\mathsf{A}^{(z)}$ or, with $\hat{\mathsf{C}} = \mathsf{C}(\hat{\mathbf{n}})$,

$$\mathsf{A} = \begin{pmatrix} 0 & -\mu^{-1}\hat{\mathsf{C}} \\ \varepsilon^{-1}\hat{\mathsf{C}} & 0 \end{pmatrix}. \tag{3.86}$$

We now need to determine the eigenvalues and eigenvectors of A.

Eigenvectors of the Flux Matrix

Using the similarity transformation

$$\mathsf{T} = \operatorname{diag}(\sqrt{\varepsilon^{-1}}, \sqrt{\varepsilon^{-1}}, \sqrt{\varepsilon^{-1}}, \sqrt{\mu^{-1}}, \sqrt{\mu^{-1}}, \sqrt{\mu^{-1}}) \tag{3.87}$$

we obtain the matrix

$$\mathsf{N} := \frac{1}{c}\mathsf{T}\mathsf{A}\mathsf{T}^{-1} = \begin{pmatrix} 0 & -\hat{\mathsf{C}} \\ \hat{\mathsf{C}} & 0 \end{pmatrix} \tag{3.88}$$

with $c = 1/\sqrt{\varepsilon\mu}$ denoting the local speed of light. Since $\hat{\mathsf{C}}$ is antisymmetric, N is symmetric. The eigenvectors and eigenvalues of N are

$$\kappa^{(1)} = \kappa^{(2)} = 0 \qquad \nu^{(1)} = \begin{pmatrix} \hat{\mathbf{n}} \\ 0 \end{pmatrix} \qquad \nu^{(2)} = \begin{pmatrix} 0 \\ \hat{\mathbf{n}} \end{pmatrix} \tag{3.89}$$

$$\kappa^{(3)} = \kappa^{(4)} = 1 \qquad \nu^{(3)} = \begin{pmatrix} \hat{\boldsymbol{\alpha}} \\ \hat{\boldsymbol{\beta}} \end{pmatrix} \qquad \nu^{(4)} = \begin{pmatrix} -\hat{\boldsymbol{\beta}} \\ \hat{\boldsymbol{\alpha}} \end{pmatrix} \tag{3.90}$$

$$\kappa^{(5)} = \kappa^{(6)} = -1 \qquad \nu^{(5)} = \begin{pmatrix} \hat{\boldsymbol{\beta}} \\ \hat{\boldsymbol{\alpha}} \end{pmatrix} \qquad \nu^{(6)} = \begin{pmatrix} \hat{\boldsymbol{\alpha}} \\ -\hat{\boldsymbol{\beta}} \end{pmatrix}. \tag{3.91}$$

$\hat{\boldsymbol{\alpha}}$ and $\hat{\boldsymbol{\beta}}$ can chosen such that the eigenvalue equation is fulfilled for e.g. $\kappa^{(3)}$ and $\nu^{(3)}$, which yields the two conditions

$$\hat{\mathsf{C}}\hat{\boldsymbol{\alpha}} = \hat{\mathbf{n}} \times \hat{\boldsymbol{\alpha}} = \hat{\boldsymbol{\beta}}, \qquad -\hat{\mathsf{C}}\hat{\boldsymbol{\beta}} = -\hat{\mathbf{n}} \times \hat{\boldsymbol{\beta}} = \hat{\boldsymbol{\alpha}}. \tag{3.92}$$

This shows that $\hat{\mathbf{n}}$, $\hat{\boldsymbol{\alpha}}$, and $\hat{\boldsymbol{\beta}}$ must be mutually orthogonal. We also require $\hat{\boldsymbol{\alpha}}$ and $\hat{\boldsymbol{\beta}}$ to be unit vectors. This is not really necessary, but it will make later computations easier by avoiding scaling factors. One possible choice of $\hat{\boldsymbol{\alpha}}$ and $\hat{\boldsymbol{\beta}}$ results from choosing one of the axis-parallel unit vectors

$$\hat{\mathbf{e}} := \begin{cases} \hat{\mathbf{e}}^{(z)} & \text{if } |\hat{\mathbf{e}}^{(z)} \times \hat{\mathbf{n}}| \geq \frac{1}{2}, \\ \hat{\mathbf{e}}^{(y)} & \text{else,} \end{cases} \tag{3.93}$$

and then defining $\hat{\boldsymbol{\alpha}}$ and $\hat{\boldsymbol{\beta}}$ as

$$\hat{\boldsymbol{\beta}} := \frac{\hat{\mathbf{e}} \times \hat{\mathbf{n}}}{|\hat{\mathbf{e}} \times \hat{\mathbf{n}}|}, \qquad \hat{\boldsymbol{\alpha}} := \hat{\boldsymbol{\beta}} \times \hat{\mathbf{n}}. \tag{3.94}$$

If (κ, ν) is an eigenpair of N, then $(\lambda, e) = (c\kappa, \mathsf{T}^{-1}\nu)$ is an eigenpair of A. Thus the eigenvectors and eigenvalues of A are:

$$\lambda^{(1)} = \lambda^{(2)} = 0 \qquad e^{(1)} = \begin{pmatrix} \sqrt{\varepsilon}\hat{\mathbf{n}} \\ 0 \end{pmatrix} \qquad e^{(2)} = \begin{pmatrix} 0 \\ \sqrt{\mu}\hat{\mathbf{n}} \end{pmatrix} \tag{3.95}$$

$$\lambda^{(+)} = \lambda^{(3)} = \lambda^{(4)} = c \qquad e^{(3)} = \begin{pmatrix} \sqrt{\varepsilon}\hat{\boldsymbol{\alpha}} \\ \sqrt{\mu}\hat{\boldsymbol{\beta}} \end{pmatrix} \qquad e^{(4)} = \begin{pmatrix} -\sqrt{\varepsilon}\hat{\boldsymbol{\beta}} \\ \sqrt{\mu}\hat{\boldsymbol{\alpha}} \end{pmatrix} \tag{3.96}$$

$$\lambda^{(-)} = \lambda^{(5)} = \lambda^{(6)} = -c \qquad e^{(5)} = \begin{pmatrix} \sqrt{\varepsilon}\hat{\boldsymbol{\beta}} \\ \sqrt{\mu}\hat{\boldsymbol{\alpha}} \end{pmatrix} \qquad e^{(6)} = \begin{pmatrix} \sqrt{\varepsilon}\hat{\boldsymbol{\alpha}} \\ -\sqrt{\mu}\hat{\boldsymbol{\beta}} \end{pmatrix}. \tag{3.97}$$

Note that the condition for hyperbolicity of (3.82) is just that it is possible to determine real eigenvalues and a full set of eigenvectors for A for any $\hat{\mathbf{n}}$. Since we were able to do just that, we have proven that the Maxwell conservation law (3.82) is hyperbolic.

Two of A's eigenvalues are zero $\lambda^{(1)} = \lambda^{(2)} = 0$. The corresponding eigenvectors represent components of $\mathbf{D}$ and $\mathbf{B}$ parallel to $\hat{\mathbf{n}}$ and perpendicular to the interface. Since the eigenvalues are zero this means that those components are not transported across the interface. There are two positive eigenvalues $\lambda^{(3)} = \lambda^{(4)} = c$; their eigenvectors represent plane waves traveling in the direction of $\hat{\mathbf{n}}$. Plane waves come in two polarizations, thus the twofold degeneration. The last two eigenvalues are negative $\lambda^{(5)} = \lambda^{(6)} = -c$. They represent plane waves traveling in direction $-\hat{\mathbf{n}}$, and there are again two polarizations.

Determining the Amplitude of the Discontinuities

For the Riemann problem it is sufficient to only consider piecewise constant parameters

$$\varepsilon = \begin{cases} \varepsilon^{(l)} & r < 0, \\ \varepsilon^{(r)} & r > 0, \end{cases} \qquad \mu = \begin{cases} \mu^{(l)} & r < 0, \\ \mu^{(r)} & r > 0. \end{cases} \tag{3.98}$$

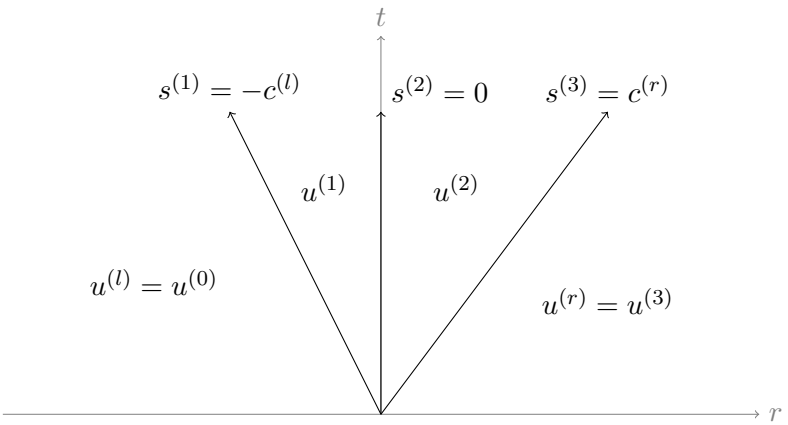


Figure 3.3.: Sketch of the Riemann problem for the variable-coefficient Maxwell problem.

This in turn leads to a left and right speed of light $c^{(l)}$ and $c^{(r)}$ and to a left and right coefficient matrix $\mathsf{A}^{(l)}$ and $\mathsf{A}^{(r)}$ each with their own set of eigenpairs $\{(\lambda^{(l,i)}, e^{(l,i)})\}$ and $\{(\lambda^{(r,i)}, e^{(r,i)})\}$. There are three discontinuities that appear in the solution of the Riemann problem: one for the negative eigenvalues $\lambda^{(l,5)} = \lambda^{(l,6)} = -c^{(l)}$ in the region $r < 0$, one for the positive eigenvalues $\lambda^{(r,3)} = \lambda^{(r,4)} = c^{(r)}$ in the domain $r > 0$, and one at $r = 0$, both for the zero eigenvalues $\lambda^{(l,1)} = \lambda^{(l,2)} = \lambda^{(r,1)} = \lambda^{(r,2)} = 0$ and for the discontinuity in the material parameters. We will call the intermediate solutions between the discontinuities $u^{(1)}$ and $u^{(2)}$. This is illustrated in figure 3.3.

This yields three Rankine-Hugoniot conditions:

$$-c^{(l)} \left[\!\left[u\right]\!\right]^{(1)} = \mathsf{A}^{(l)} \left[\!\left[u\right]\!\right]^{(1)}, \tag{3.99}$$

$$c^{(r)} \left[\!\left[u\right]\!\right]^{(3)} = \mathsf{A}^{(r)} \left[\!\left[u\right]\!\right]^{(3)}, \tag{3.100}$$

$$0 = \mathsf{A}^{(r)} u^{(2)} - \mathsf{A}^{(l)} u^{(1)}. \tag{3.101}$$

The jumps across these discontinuities are in spaces spanned by eigenvectors of the respective coefficient matrix

$$\left[\!\left[u\right]\!\right]^{(1)} \in \operatorname{span}\{e^{(l,5)}, e^{(l,6)}\}, \tag{3.102}$$

$$\left[\!\left[u\right]\!\right]^{(3)} \in \operatorname{span}\{e^{(r,3)}, e^{(r,4)}\}, \tag{3.103}$$

and we can use these eigenvectors to define the columns of the matrix

$$\mathsf{M} = \begin{pmatrix} e^{(l,5)} & e^{(l,6)} & e^{(r,3)} & e^{(r,4)} \end{pmatrix}. \tag{3.104}$$

Now that we have M, we can solve (3.81) and plug the result into (3.78) to obtain an expression for the flux across the interface. This is a tedious calculation, which we have pushed into appendix A.

The Riemann Flux for Maxwell

To give the result here we need to introduce

$$Y := \sqrt{\frac{\varepsilon}{\mu}} = c\varepsilon \qquad \text{(local admittance)}, \tag{3.105}$$

$$Z := \sqrt{\frac{\mu}{\varepsilon}} = c\mu \qquad \text{(local impedance)}. \tag{3.106}$$

These exists in a left and right variety: $Y^{(l)}$, $Z^{(l)}$ and $Y^{(r)}$, $Z^{(r)}$. It is convenient to introduce an intermediate admittance and impedance

$$\frac{1}{Y^{(*)}} := \frac{1}{Y^{(l)}} + \frac{1}{Y^{(r)}}, \qquad \frac{1}{Z^{(*)}} := \frac{1}{Z^{(l)}} + \frac{1}{Z^{(r)}}. \tag{3.107}$$

Note that $Y^{(*)}Z^{(*)} \neq 1$. Finally, it is convenient to collect the quantities in matrices

$$\mathsf{X}^{(l)} := \operatorname{diag}(Y^{(l)}, Y^{(l)}, Y^{(l)}, Z^{(l)}, Z^{(l)}, Z^{(l)}), \tag{3.108}$$

$$\mathsf{X}^{(*)} := \operatorname{diag}(Y^{(*)}, Y^{(*)}, Y^{(*)}, Z^{(*)}, Z^{(*)}, Z^{(*)}). \tag{3.109}$$

With these preparations in place the flux can be given as

$$\hat{f} = f(u)|_{r=0} = \mathsf{A}^{(r)}u^{(r)} - \mathsf{X}^{(*)}\mathsf{N}(\mathbb{1} + \mathsf{X}^{(l)}\mathsf{N})\,[\![\mathsf{A}u]\!] \tag{3.110}$$

Finally it is interesting to examine how this expression simplifies for constant coefficients $\varepsilon := \varepsilon^{(l)} = \varepsilon^{(r)}$ and $\mu := \mu^{(l)} = \mu^{(r)}$. In that case we have the following identities

$$\mathsf{X} := \mathsf{X}^{(l)} \qquad \frac{1}{2}\mathsf{X} = \mathsf{X}^{(*)} \tag{3.111}$$

$$c := c^{(l)} = c^{(r)} \qquad \mathsf{A} := \mathsf{A}^{(l)} = \mathsf{A}^{(r)} = \frac{1}{c}\mathsf{X}\mathsf{N} \tag{3.112}$$

and we can simplify the flux as

$$\hat{f}(u^{(l)}, u^{(r)}) = \mathsf{A}u^{(r)} - \frac{1}{2c}\mathsf{A}^2(\mathbb{1} + \frac{1}{c}\mathsf{A})\,[\![u]\!]\,. \tag{3.113}$$

Decomposing

$$u^{(l)} := \sum \gamma_i^{(l)} e^{(i)}, \qquad u^{(r)} := \sum \gamma_i^{(r)} e^{(i)} \tag{3.114}$$

we can further simplify this to

$$\hat{f} = c(\gamma_3^{(l)}e^{(3)} + \gamma_4^{(l)}e^{(4)}) - c(\gamma_5^{(r)}e^{(5)} + \gamma_6^{(r)}e^{(6)}) \tag{3.115}$$

$$= \mathsf{A}^{+}u^{(l)} + \mathsf{A}^{-}u^{(r)}, \tag{3.116}$$

which is exactly the expression we get from considering characteristic variables.

3.2.4. Boundary Conditions

On the boundary we must incorporate the boundary condition g. We will assume that the material parameters are continuous across the domain boundary. The boundary conditions are then incorporated into the numerical flux on the boundary

$$\hat{F}(u|_{T^-}, u|_{T^+}) = \hat{F}(u|_{T^-}, g) \qquad \text{on } \partial\Omega, \tag{3.117}$$

since $u|_{T^+} = g$ on $\partial\Omega$.

Many boundary conditions popular in electromagnetics can be implemented by different specifications of g. When considering (3.116) it becomes obvious that only the component of g that represents an incoming wave matters, i.e. four components may be freely chosen. Similarly, the outgoing flux is completely determined by the relevant components of $u^{(l)}$ and we cannot prescribe it. This has the effect of always applying an outflow boundary condition for these components, which is exactly what we want.

Radiation Boundary Conditions For DG, this is the natural boundary condition. By setting the outer field to zero

$$g := 0 \qquad \text{on } \Gamma^{\text{RBC}} \tag{3.118}$$

the incoming flux will be zero. For waves leaving the domain, i.e. the outgoing flux, the outer field does not matter.

This boundary condition sets all outgoing characteristics to zero, and is thus identical to the first approximation for systems of Engquist and Majda[12]. Engquist and Majda also state that this boundary condition is equivalent to their first approximation for the wave equation, which we use for the FE scheme.

Incident Waves Any known incident wave can be applied directly as the values of g. In fact, the radiation boundary condition is a special case of this with no incoming wave.

Perfect Electric Conductor On the surface of a perfect electric conductor the tangential electric field must be $\mathbf{E}^{\tan} = 0$, which implies $\mathbf{D}^{\tan} = 0$ for isotropic ε and μ. The idea is now to chose g dependent on u such that outgoing and incoming waves cancel each other at the boundary and yield a zero tangential electric field. Let u and g be represented in characteristic variables

$$u = \sum_j \tilde{u}_j e^{(j)} \qquad\qquad g = \sum_j \tilde{g}_j e^{(j)}. \tag{3.119}$$

then we need to choose g such that

$$(\tilde{u}_3 e^{(3)} + \tilde{u}_4 e^{(4)} + \tilde{g}_5 e^{(5)} + \tilde{g}_6 e^{(6)})_j = 0 \qquad\qquad j = 1, \ldots, 3 \tag{3.120}$$

from this choice and the definition of the eigenvectors of A it follows that

$$(\tilde{u}_3 e^{(3)} + \tilde{u}_4 e^{(4)} - \tilde{g}_5 e^{(5)} - \tilde{g}_6 e^{(6)})_j = 0 \qquad\qquad j = 4, \ldots, 6 \tag{3.121}$$

Given a solution $u = (\mathbf{D}, \mathbf{B})^T$ an easy way to achieve this is to set

$$g := \begin{pmatrix} -\mathbf{D} \\ \mathbf{B} \end{pmatrix} \qquad \text{on } \Gamma^{\text{PEC}} \tag{3.122}$$

Note that what actually happens here is that the electric component of the flux becomes zero. This only ensures that the electric field does not change over time; to actually make is zero, it must already be zero in the initial conditions.

Perfect Magnetic Conductor This follows the same argument as for the PEC boundary condition; but exchanges $\mathbf{D}$ and $\mathbf{B}$:

$$g := \begin{pmatrix} \mathbf{D} \\ -\mathbf{B} \end{pmatrix} \qquad \text{on } \Gamma^{\text{PMC}} \tag{3.123}$$

3.3. Other considerations

3.3.1. Probe Evaluations

To generate radargrams from the computed fields, and sometimes to compare the solution to a reference solution we use point probes. This is simply an arbitrary position where the field is evaluated and stored. A point probe that is placed right at the boundary of two or more mesh elements evaluates to the mean field value of all adjacent elements.

The alternative, arbitrarily picking one element to evaluate the probe in, has disadvantages when comparing computations on multiple refinement levels. The mesh element that is picked for evaluation on the finer level is not necessarily a child of the mesh element that is picked on the coarser level, which may make the convergence seem worse than it actually is near that probe.

Additionally, when picking an element arbitrarily, we still need to ensure that all processes in a parallel computation agree on the same element. This voids the advantage of simplicity that this approach has in the sequential case.

3.3.2. Point Sources

Point sources are externally imposed currents in the form of a δ-distribution, i. e.

$$\mathbf{J}_{\text{ext}}(\mathbf{x}, t) = \mathbf{J}^0_{\text{ext}}(t)\delta(\mathbf{x} - \mathbf{x}^0). \tag{3.124}$$

Mathematically they are treated just like any other imposed current. However, when plugged into the weak formulation they lead to terms of the form

$$\int_{\Omega_e} d\mathbf{x}\, \mathbf{w}(t)\delta(\mathbf{x} - \mathbf{x}^0)\mathbf{v}(\mathbf{x}) \tag{3.125}$$

where $\mathbf{v}(\mathbf{x})$ is the test function. These terms cannot be assembled using the standard quadrature rules, but they are of course trivially assembled using the rules for δ-distributions.

As for the point probes it may happen that a point source is located right at the boundary of one or more elements.[1] We are using the same method here to resolve ambiguities: the point probe is assembled in each adjacent cells, but the result is divided by the number of adjacent cells such that the effect of the point probe is distributed evenly among them. Let A be the set of cells adjacent to the point probe, then (3.125) becomes

$$\sum_{e \in A} \int_{\Omega_e} d\mathbf{x} \, |A|^{-1} \mathbf{w}(t) \delta(\mathbf{x} - \mathbf{x}^0) \mathbf{v}(\mathbf{x}) \tag{3.126}$$

[1] This was the case for the dipole test problem which had a spherical domain with the point source located exactly in the middle, precisely where Gmsh placed a mesh vertex.

4. Temporal Discretization

After spatial discretization, we are left with a system of linear second order ordinary differential equations (ODEs) of the form

$$F^{(2)}(d_t^2 u(t), d_t u(t), u(t); t) = 0 \tag{4.1}$$

for the finite element scheme and a system of linear first order ODEs of the form

$$F^{(1)}(d_t u(t), u(t); t) = 0 \tag{4.2}$$

for the DG scheme. It should be noted that it is always possible to rewrite a linear second order ODE in the form of a linear first order ODE by replacing u with $(u, d_t u)^T$:

$$F^{(2)}(d_t^2 u(t), d_t u(t), u(t); t) = 0 \qquad \rightarrow \qquad F^{(1)}\left(d_t \begin{pmatrix} u(t) \\ d_t u(t) \end{pmatrix}, \begin{pmatrix} u(t) \\ d_t u(t) \end{pmatrix}; t\right) = 0 \tag{4.3}$$

This is however not always advisable since it may lead to bigger matrices and vectors.

4.1. Time Stepper for FE

For the FE scheme we need to solve the second order ODE (3.14). We do not rewrite it as a first order ODE, but use the central differences scheme as described by [23] to solve the second order ODE directly.

This scheme can be derived by expanding $u(t - \Delta t)$ and $u(t + \Delta t)$ as a Tailor series around t up to second order:

$$u(t - \Delta t) = u(t) - \Delta t\, d_t u(t) + \frac{(\Delta t)^2}{2} d_t^2 u(t) + O((\Delta t)^3), \tag{4.4}$$

$$u(t + \Delta t) = u(t) + \Delta t\, d_t u(t) + \frac{(\Delta t)^2}{2} d_t^2 u(t) + O((\Delta t)^3). \tag{4.5}$$

By subtracting and adding these two equations we obtain approximations for the first and the second derivative:

$$d_t u(t) = \frac{u(t + \Delta t) - u(t - \Delta t)}{2\Delta t} + O((\Delta t)^2), \tag{4.6}$$

$$d_t^2 u(t) = \frac{u(t + \Delta t) - 2u(t) + u(t - \Delta t)}{(\Delta t)^2} + O(\Delta t). \tag{4.7}$$

Introducing time steps $t^{(n)} := t^{(0)} + n\Delta t$ at regular intervals, approximating

$$u(t^{(n)}) \approx u^{(n)} \tag{4.8}$$

$$d_t u(t^{(n)}) \approx \dot{u}^{(n)} := \frac{u^{(n+1)} - u^{(n-1)}}{2\Delta t} \tag{4.9}$$

$$d_t^2 u(t^{(n)}) \approx \ddot{u}^{(n)} := \frac{u^{(n+1)} - 2u^{(n)} + u^{(n-1)}}{(\Delta t)^2} \tag{4.10}$$

according to the tailor expansion, and inserting into (4.1) yields the discrete relation

$$F^{(2)}\left(\frac{u^{(n+1)} - 2u^{(n)} + u^{(n-1)}}{(\Delta t)^2}, \frac{u^{(n+1)} - u^{(n-1)}}{2\Delta t}, u^{(n)}, t\right) = 0. \tag{4.11}$$

Given initial values $u^{(n-1)}$ and $u^{(n)}$ this can be used to compute a new approximation $u^{(n+1)}$.

For the finite element scheme we identify $u := E$ and

$$F^{(2)}(d_t^2 u(t), d_t u(t), u(t), t) := \mathsf{T} d_t^2 u(t) + (\mathsf{R} + \mathsf{Q}) d_t u(t) + \mathsf{S} u(t) + f(t). \tag{4.12}$$

Applying the same approximations as for (4.11) and bringing the terms that depend on $u^{(n+1)}$ to the left hand side yields

$$\left(\frac{\mathsf{R} + \mathsf{Q}}{2\Delta t} + \frac{\mathsf{T}}{(\Delta t)^2}\right) u^{(n+1)} = \left(\frac{2\mathsf{T}}{(\Delta t)^2} - \mathsf{S}\right) u^{(n)} + \left(\frac{\mathsf{R} + \mathsf{Q}}{2\Delta t} - \frac{\mathsf{T}}{(\Delta t)^2}\right) u^{(n-1)} - f(t^{(n)}) \tag{4.13}$$

This scheme is explicit since the stiffness matrix S appears only on the right hand side. It can be shown to be stable for $\Delta t \leq 2/\sqrt{\rho(\mathsf{T}^{-1}\mathsf{S})}$. This however requires determination of the largest eigenvalue $\rho(\,\cdot\,)$ of $\mathsf{T}^{-1}\mathsf{S}$. While this can be done efficiently, I did not implement this. Instead I simply determined the time step size by trial and error.

`Dune-pdelab` did only contain one-step time stepping schemes, i.e. schemes where the new solution only depends on one old solution. For the central difference scheme as described here, two old solutions are required to implement the new solution, making it necessary for me to implement a new time stepper.

4.2. Time Steppers for the DG scheme

The DG discretization yields a linear first order ODE of the form (4.2). There are many time stepping schemes that can solve this kind of ODE. In our case it is important to select a scheme which yields an accuracy comparable to the spatial discretization and to use explicit schemes to make the resulting linear system easy to solve. We will use explicit Euler, Heun's method and Shu's third order scheme [10] for first, second and third order accurate spatial discretizations, respectively. `Dune-pdelab` provides a framework for one-step multi-stage methods which can be used to implement all of these methods.

We rewrite the first-order ODE (4.2) employing its linearity as[1]

$$F^{(1)}(d_t u(t), u(t); t) = d_t m(u(t); t) + r(u(t); t) = 0. \tag{4.14}$$

`Dune-pdelab` solves this ODE in multiple stages. Each stage $r = 1, \ldots, s$ has an associated time $t^{(n,r)} := t^{(n)} + d^{(r)}\Delta t^{(n)}$ within the time step and an approximate solution $u^{(n,r)}$. For convenience the initial time and solution of the time step are available as $t^{(n,0)} := t^{(n)}$ and $u^{(n,0)} := u^{(n)}$. Each stage r then consists of finding $u^{(n,r)}$ such that

$$\sum_{i=0}^{r} \left\{ a^{(r,i)} m(u^{(n,i)}; t^{(n,i)}) + \Delta t^{(n)} b^{(r,i)} r(u^{(n,i)}; t^{(n,i)}) \right\} = 0. \tag{4.15}$$

Finally, the approximate solution of the time step is that of the last stage $u^{(n+1)} := u^{(n,s)}$.

The coefficients $a^{(r,i)}$ and $b^{(r,i)}$ together with the fractional stage times $d^{(r)}$ and the number of stages s completely determine the particular scheme. Both explicit schemes and diagonally implicit schemes may be described in this way. For explicit schemes the coefficients $b^{(r,r)}$ are zero. This makes it sufficient to solve $m(u^{(n,r)}; t^{(n,r)})$ in (4.15) for $u^{(n,r)}$. In contrast, for implicit schemes a superposition of $m(u^{(n,r)}; t^{(n,r)})$ and $r(u^{(n,r)}; t^{(n,r)})$ needs to be solved. Without loss of generality we shall restrict ourselves to normalized schemes, those with $a^{(r,r)} = 1$ for $r = 1, \ldots, s$.

For explicit schemes we assume that $m(u; t)$ is strictly linear in u, i. e. $m(u; t) = \mathsf{M}(t)u$. This allows us to reformulate (4.15) for explicit normalized schemes as

$$\mathsf{M}(t^{(n,r)})u^{(n,r)} = -\sum_{i=0}^{r-1} a^{(r,i)} m(u^{(n,i)}; t^{(n,i)}) - \Delta t^{(n)} \sum_{i=0}^{r-1} b^{(r,i)} r(u^{(n,i)}; t^{(n,i)}), \tag{4.16}$$

which can be solved by a suitable linear solver.

This algorithm and parameterization is a generalization of the one described in [10], but can handle implicit schemes and non-autonomous $m(u; t)$, too. Parameters for the schemes used here are given in table 4.2.

[1] To see that this expression indeed depends only linearly on $d_t u(t)$ and $u(t)$ conduct the total derivative.

scheme	explicit Euler	Heun's method	Shu's 3^{rd} order scheme
$(a^{(r,i)})$	$\begin{pmatrix} -1 & 1 \end{pmatrix}$	$\begin{pmatrix} -1 & 1 & \\ -1 & 0 & 1 \end{pmatrix}$	$\begin{pmatrix} -1 & 1 & & \\ -\frac{3}{4} & -\frac{1}{4} & 1 & \\ -\frac{1}{3} & 0 & -\frac{2}{3} & 1 \end{pmatrix}$
$(b^{(r,i)})$	$\begin{pmatrix} 1 & 0 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & \\ \frac{1}{2} & \frac{1}{2} & 0 \end{pmatrix}$	$\begin{pmatrix} 1 & 0 & & \\ 0 & \frac{1}{4} & 0 & \\ 0 & 0 & \frac{2}{3} & 0 \end{pmatrix}$
$(d^{(i)})^T$	$\begin{pmatrix} 0 & 1 \end{pmatrix}$	$\begin{pmatrix} 0 & 1 & 1 \end{pmatrix}$	$\begin{pmatrix} 0 & 1 & \frac{1}{2} & 1 \end{pmatrix}$

Table 4.1.: Coefficients for the time stepping schemes.

5. Linear Equation Solvers

After discretization we are left with one or more systems of equations of the form

$$\mathsf{A}x = b \tag{5.1}$$

with the known matrix A and right-hand side b. The system is to be solved for x, which consists either of the unknown coefficients themselves (E or u) or an update to them, depending on the formulation. In the case of a stationary problem or for first-order ODE discretizations there will only be one such system to be solved, however, higher-order ODE discretizations may require multiple stages and thus multiple systems of the form (5.1) must be solved.

For many discretization schemes, including FE and DG, A is a sparse matrix and thus can be stored efficiently in a compressed format. The sparsity pattern, i. e. the set of matrix entries that may possibly hold non-zero values, is fully determined by the mesh and the finite element basis. A simple mesh and the corresponding sparsity patterns for both FE and DG is shown in figure 5.1.

In the case of GPR we have neglected dependencies of the material parameters ε, μ and σ on the fields $\mathbf{E}$ and $\mathbf{H}$, thus A is independent of x and the system (5.1) is linear. In addition, for our problem the material parameters and the mesh do not change over time, resulting in A independent of t.

5.1. Finite Elements

For the finite-element scheme, the linear system to be solved is given in (4.13). To write it in the form (5.1) we set

$$\mathsf{A} := \frac{\mathsf{R}+\mathsf{Q}}{2\Delta t} + \frac{\mathsf{T}}{(\Delta t)^2}, \tag{5.2}$$

$$x := E^{n+1}, \tag{5.3}$$

and

$$b := \left(\frac{2\mathsf{T}}{(\Delta t)^2} - \mathsf{S}\right) E^n + \left(\frac{\mathsf{R}+\mathsf{Q}}{2\Delta t} - \frac{\mathsf{T}}{(\Delta t)^2}\right) E^{n-1} - f^n. \tag{5.4}$$

The matrix A is a sparse matrix with approximately 20 entries per row for 3D.

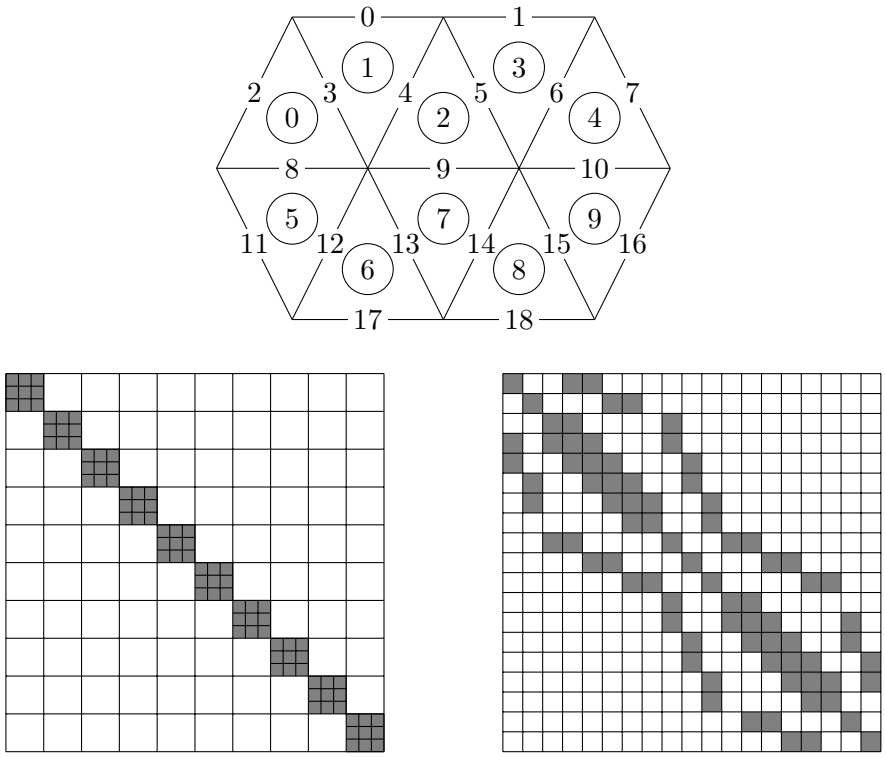


Figure 5.1.: Matrix structure for DG (left) and FE (right). Non-zero entries are shown in gray. For DG, each entry is a dense block corresponding to one mesh element. For FE, each entry is a scalar corresponding to one mesh edge. The corresponding mesh is shown on top, with numbering for edges (plain) and elements (circled). This example uses a 2D domain to keep it comprehensible.

The matrix A results from the form

$$\mathsf{A}_{ij} := \frac{1}{(\Delta t)^2} \int_\Omega d\mathbf{r}\, \varepsilon\, \mathbf{N}^{(i)} \cdot \mathbf{N}^{(j)} \tag{5.5a}$$

$$+ \frac{1}{2\Delta t} \int_\Omega d\mathbf{r}\, \sigma\, \mathbf{N}^{(i)} \cdot \mathbf{N}^{(j)} \tag{5.5b}$$

$$+ \frac{1}{2\Delta t} \oint_{\partial\Omega} d\mathbf{r}\, Y \left(\hat{\mathbf{n}} \times \mathbf{N}^{(i)}\right) \cdot \left(\hat{\mathbf{n}} \times \mathbf{N}^{(j)}\right). \tag{5.5c}$$

This form consists of three terms. ε is always positive, so (5.5a) is positive definite. σ is positive or zero, so (5.5b) is positive semidefinite. (5.5c) positive semidefinite too; Y is always positive, but the integral is zero for base functions that are zero on the domain boundary. As a result, A is positive definite and we can use the Conjugate Gradient method to solve it.

When using Conjugate Gradients to solve the finite element system, the number of iterations required to achieve a reduction of the initial error of ρ is

$$i \leq \left\lceil \frac{\sqrt{\kappa}}{2} \ln \frac{2}{\rho} \right\rceil \qquad\qquad \kappa = \frac{\lambda_{\max}(\mathsf{A})}{\lambda_{\min}(\mathsf{A})} \tag{5.6}$$

This depends primarily on condition number of the matrix κ. But there is also another way to get quicker convergence: all error components corresponding do a degenerate eigenvalue are reduced together. Even if eigenvalues aren't perfectly degenerate, clustered eigenvalues still improve convergence.

Conjugate gradients can be derived from the method *conjugate directions*, which constructs a set of search directions, and cancels the error in each direction independently. The search directions must be conjugate with respect to A and are constructed via Gram-Schmidt conjugation from a set of linearly independent vectors, e.g. the coordinate axes. In general, Gram-Schmidt conjugation has the disadvantage that all previous search directions must be kept in memory, so new search directions can be made conjugate to them. This is solved by Conjugate Gradients by using the residual as basis vectors—as it turns out, the residual is already conjugate to all but the most recent search direction, so only that needs to be kept around.

Applying n iterations of conjugate gradients to a problem with n degrees of freedom yields the exact solution (in perfect arithmetic), since the error has been canceled in all n directions. Unfortunately, n iterations is usually far too costly. Also, arithmetic in real-world computers isn't perfect, so the search directions will slowly loose conjugacy. Conjugate gradients is still effective as an iterative solver: For evenly distributed eigenvalues the energy norm of the error can be proven to converge as

$$\|e^{(i)}\|_{\mathsf{A}} \leq 2 \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right)^i \|e^{(0)}\|_{\mathsf{A}} \tag{5.7}$$

and (5.6) follows.

Preconditioning can be applied to reduce the condition number κ or yield better clustering of eigenvalues or both. Note that even for a symmetric and positive definite preconditioner M^{-1} that approximates A^{-1}, the matrix $\mathsf{M}^{-1}\mathsf{A}$ is not necessarily symmetric

nor positive definite, and thus cannot be solved using conjugate gradients. Instead M is split such that $\mathsf{E}\mathsf{E}^T = \mathsf{M}$ and $\mathsf{E}^{-1}\mathsf{A}\mathsf{E}^{-T}\mathsf{E}^T x = \mathsf{E}^{-1}b$ is solved for $y = \mathsf{E}^T x$ using conjugate gradients. $\mathsf{E}^{-1}\mathsf{A}\mathsf{E}^{-T}$ has the same eigenvalues as $\mathsf{M}^{-1}\mathsf{A}$, so any improvement in to the eigenvalues by the preconditioner still applies. The conjugate gradients algorithm can be refactored in terms of M^{-1} to eliminate any use of E completely, which also avoids the need to solve $\mathsf{E}^T x = y$ separately.

We use a Jacobi preconditioner for our CG solver, i.e.

$$\mathsf{M} = \mathrm{diag}(\mathsf{A}). \tag{5.8}$$

This choice has the advantage that the parallel preconditioner is exactly equivalent to the sequential preconditioner, and thus the only deviations in the result stem from reordering of arithmetic operations.

5.1.1. Parallel Implementation

The Conjugate Gradient solver can be parallelized simply by supplying a parallel implementation a few operations: the scalar product of two vectors, the norm of a vector and the application of the preconditioner to a vector. These operations are responsible for communicating; the implementation of the CG method is then identical in the parallel and the sequential case.

The parallel implementation of the scalar product $x \cdot y$ must take into account that the entries of the vectors are distributed across the processors in an inconsistent representation. In particular, base functions that cross processor boundaries have one vector entry in each adjacent processor. Assembly happens independently on each processor, so each such vector entry only has a partial result, and the vectors must be made consistent by summing up the entries for each base function across the processors. Then one processor must be chosen for each base function that is responsible for computing the product for that base functions. Finally all sum up their results in a global communication, and the sum becomes the result of the scalar product, returned to the conjugate gradient method on each processor.

The norm $\|x\| = \sqrt{x \cdot x}$ could be implemented in terms of the scalar product. We do however use a specialized implementation, since only one vector needs to be made consistent and we can avoid some communication.

Parallelizing the application of the preconditioner to a vector $\mathsf{M}^{-1}x$ must be done for each preconditioner individually. For many preconditioners this is a difficult problem, and often the preconditioner can only be applied on each processor individually, resulting in an increased number of iterations compared to the sequential case. The Jacobi preconditioner is however simple enough to truly parallelize: it just consists of the diagonal of A. We can interpret the diagonal of a as a vector, make that vector consistent across all processors in a preprocessing step, and invert each entry. Application of the preprocessor to a vector can then be done without communication on each processor individually.

5.2. Discontinuous Galerkin

For the discontinuous Galerkin scheme there are multiple systems of the form (4.13), one for each stage r of the time stepping scheme:

$$\mathsf{A}^{(r)} x^{(r)} = b^{(r)}. \tag{5.9}$$

The definitions of the quantities follow from (4.16) as

$$\mathsf{A}^{(r)}_{jk} := a_{rr} m(\phi^{(k)}, \psi^{(j)}; t^{(r)}), \tag{5.10}$$

$$x^{(r)}_k := u^{(r)}_k, \tag{5.11}$$

and

$$b^{(r)}_j := \sum_{i=0}^{r-1} \left\{ a_{ri} m(u^{(i)}, \psi^{(j)}; t^{(i)}) - b_{ri} \Delta t \, r(u^{(i)}, \psi^{(j)}; t^{(i)}) \right\}. \tag{5.12}$$

Since $m(\phi, \psi; t)$ only couples base functions on the same mesh element, the resulting $\mathsf{A}^{(r)}$ is block-diagonal, with one block per mesh element. This make it feasible to solve the system by directly inverting $\mathsf{A}^{(r)}$, since only each block needs to be inverted. This is just what happens when we apply one step of a a Jacobi preconditioner.

What still needs to be shown is that each block on the diagonal of $\mathsf{A}^{(r)}$ is actually invertible. This can be seen by observing that $\langle \phi, \psi \rangle = m(\phi, \psi; t)$ is an inner product. Thus the $\mathsf{A}^{(r)}$ are a Gram matrices, thus positive definite, thus invertible.

5.2.1. Parallel implementation

For the DG scheme, in contrast to the FE scheme, each degrees of freedom is uniquely owned by one processor. However, each processor has an overlap layer of one mesh element, the vector entries associated with these overlap elements are copied from the processor that owns that degree of freedom. Assembly of the residual results in an inconsistent vector, where the entries in the overlap only contain partial values. To apply the Jacobi preconditioner, we simply apply a sequential Jacobi preconditioner on each processor, and then copy the values in the overlap from the processor that owns them in a communication.

The resulting vector is consistent, and contains the solution for that stage.

6. Testing

To check the implementation of both methods, I consider two test problems. Both test problems have known analytic solutions. For the first test problem, the solution is smooth, consisting of a superposition of circular polarized plane waves, which are coupled into the domain via Dirichlet boundary conditions. This test problem is also used to compare the solution quality of these two schemes.

The other test problem consists of an initially empty domain, which is excited by a point current. This is a test for excitation by a point current, point probes and the radiating boundary conditions.

6.1. Smooth Problem

The setup for the smooth test problem is a region of free space with the computational domain $\Omega = (0,1)^3$. The domain is terminated with time-dependent Dirichlet boundary conditions

$$\left.\begin{aligned} \hat{\mathbf{n}} \times \mathbf{E} &= \hat{\mathbf{n}} \times \mathbf{E}^* \\ \hat{\mathbf{n}} \times \mathbf{H} &= \hat{\mathbf{n}} \times \mathbf{H}^* \end{aligned}\right\} \quad \text{on } \partial\Omega \tag{6.1}$$

where $\mathbf{E}^*$ and $\mathbf{H}^*$ denote the chosen analytic solution (the second condition applies only to DG). Initial conditions are

$$\mathbf{E} = \mathbf{E}^* \qquad \text{at } t \in \{t_0 - \Delta t, t_0\} \tag{6.2}$$

for FE and

$$\left.\begin{aligned} \mathbf{E} &= \mathbf{E}^* \\ \mathbf{H} &= \mathbf{H}^* \end{aligned}\right\} \quad \text{at } t = t_0 \tag{6.3}$$

for DG.

Consider the following set of fields:

$$\mathbf{E}^1(\mathbf{x},t) = \begin{pmatrix} 0 \\ -\cos\alpha x_0 \sin\alpha t \\ \sin\alpha x_0 \cos\alpha t \end{pmatrix} \qquad \mathbf{H}^1(\mathbf{x},t) = \begin{pmatrix} 0 \\ \cos\alpha x_0 \sin\alpha t \\ \sin\alpha x_0 \cos\alpha t \end{pmatrix} \tag{6.4a}$$

$$\mathbf{E}^2(\mathbf{x},t) = \begin{pmatrix} \cos\alpha x_1 \sin\alpha t \\ 0 \\ \cos\alpha x_1 \cos\alpha t \end{pmatrix} \qquad \mathbf{H}^2(\mathbf{x},t) = \begin{pmatrix} \sin\alpha x_1 \sin\alpha t \\ 0 \\ \sin\alpha x_1 \cos\alpha t \end{pmatrix} \tag{6.4b}$$

$$\mathbf{E}^3(\mathbf{x},t) = \begin{pmatrix} -\cos\alpha x_2 \sin\alpha t \\ -\cos\alpha x_2 \cos\alpha t \\ 0 \end{pmatrix} \qquad \mathbf{H}^3(\mathbf{x},t) = \begin{pmatrix} -\sin\alpha x_2 \sin\alpha t \\ -\sin\alpha x_2 \cos\alpha t \\ 0 \end{pmatrix} \tag{6.4c}$$

Each pair $(\mathbf{E}^i, \mathbf{H}^i)$ is a plane wave with circular polarization traveling along one of the coordinate axes. They are solutions for Maxwell's equations for $\varepsilon = \mu = 1$ and $\sigma = 0$.

In addition to plane waves, fields of the form $\mathbf{E} = -\nabla\Phi$ and $\mathbf{H} = -\nabla\Phi^M$ for scalar fields Φ and Φ^M are also solutions. Physically these solutions are governed by Gauss' law $\nabla \cdot \mathbf{D} = \rho$ and the absence of magnetic charges $\nabla \cdot \mathbf{B} = 0$. (2.42) and (2.43) don't use these equations, so as far as the numerical schemes are concerned these are valid solutions.

The actual solution used for this test was obtained by summing solutions (6.4)

$$\mathbf{E}^* = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} + \sum_{i=1}^{3} \mathbf{E}^i \qquad \text{(for FE)} \tag{6.5}$$

$$(\mathbf{E}^*, \mathbf{H}^*) = \sum_{i=1}^{3} (\mathbf{E}^i, \mathbf{H}^i) \qquad \text{(for DG)} \tag{6.6}$$

One particular feature of these solutions is that they require non-zero and time-dependent Dirichlet boundary conditions.

For FE the expected convergence rate is 1 during h-refinement. For DG with order k the expected convergence rate is $k + 1$.

6.1.1. Dirichlet Boundary Conditions for FE

The finite element scheme was constructed with absorbing boundary conditions. However, in this test the domain is truncated with time-dependent Dirichlet boundary conditions. I implemented them in the standard `dune-pdelab` way: in each time step the boundary condition is interpolated into the degrees of freedom on $\partial\Omega$ and these degrees of freedom are fixed, prohibiting any changes from the linear solver. This effectively overrides any absorbing boundary condition.

6.1.2. Time Step Size

I determined time step size experimentally for each method. The test problem was simulated with $\alpha = 2\pi$ in the time interval $[0, T]$ with the end-time T dependent on h and the discretization as shown in table 6.1. To determine stability, the maximum norm over the entries of the DoF vector was computed at time T and the result was plotted for different time step sizes k. The resulting graph is almost constant for stable k, but grows dramatically if k enters the unstable region, the transition is marked be a sharp kink.

Time step sizes determined this way are problematic. A larger T will generally lead to a smaller k for an otherwise unchanged setup. It is even possible to find stable values for k that are smaller than some unstable ones; this happened in a case where the end-time T was an integer multiple of the frequency α/π which in turn was an integer multiple of k. Thus convergence computations below were often found to be unstable with the time step sizes from table 6.1, and I had to successively decreased k until the computation was stable.

		preliminary			final
method	h	T $[10^{-3}]$	steps	k $[10^{-3}]$	k $[10^{-3}]$
DG0	1	8400.0	41	204.9	142.9
	0.5	4200.0	47	89.4	71.4
	0.25	2100.0	48	43.8	35.7
	0.125	1050.0	47	22.3	17.9
	0.0625	525.0	46	11.4	8.9
	0.03125	262.5	45	5.8	4.5
	0.015625				2.2
	0.0078125				1.1
DG1	1	4200.0	35	120.0	90.9
	0.5	2100.0	41	51.2	45.5
	0.25	1050.0	41	25.6	22.7
	0.125	525.0	41	12.8	11.4
	0.0625	262.5	41	6.4	5.7
	0.03125				2.8
	0.015625				1.4
DG2	1	2100.0	24	87.5	71.4
	0.5	1050.0	26	40.4	35.7
	0.25	525.0	26	20.2	17.9
	0.125	262.5	25	10.5	8.9
	0.0625				4.5
	0.03125				2.2
FE	1	4200.0	11	381.8	200.0
	0.5	2100.0	16	131.2	100.0
	0.25	1050.0	18	58.3	50.0
	0.125	525.0	18	29.2	25.0
	0.0625	262.5	18	14.6	12.5
	0.03125				6.2
	0.015625				3.1

Table 6.1.: Experimentally determined time step sizes k for the smooth test problem. The columns marked *preliminary* show the time steps sizes determined using a special test problem for this purpose. The column marked *final* shows the time step sizes that were actually used later.

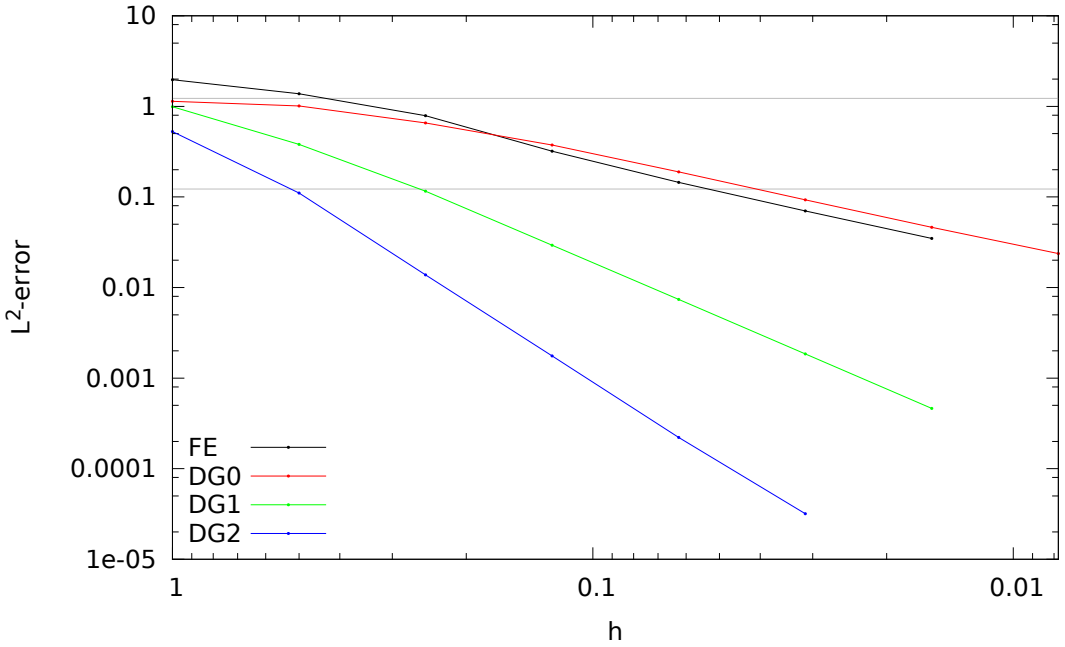


Figure 6.1.: Convergence graphs for the smooth problem in the L^2-error of $\mathbf{E}$ at time $t = 1$ for $\alpha = 2\pi$ ($\lambda = 1$). For comparison, the upper gray line shows the L^2-error of $\mathbf{E} = 0$ (≈ 1.25) and the lower gray line shows one tenth of that.

Also, for the purpose of convergence computations k has to be scaled in the same way as h. This leads to much smaller time step sizes then necessary for the coarser meshes. The time step sizes that were actually used for computations have been noted in the rightmost column table 6.1.

6.1.3. Convergence

For the convergence computations the simulation was run in the interval $[0, 1]$. The L^2-error was computed at $t = 1$, using the analytic solution as the reference. Initial values and Dirichlet boundary condition values were taken from the analytic solution as well. The errors are plotted in figure 6.1.

This setup also allows to investigate what mesh resolution is needed to resolve waves of a certain wavelength. Figure 6.2 shows the L^2-error for the case $\alpha = 8\pi$ ($\lambda = 1/4$). As can be seen, the usual rule $\lambda \geq 10h$ isn't much better than simply assuming $\mathbf{E} = 0$ for FE and DG0. As expected, DG1 and DG2 are much better in this regard, however.

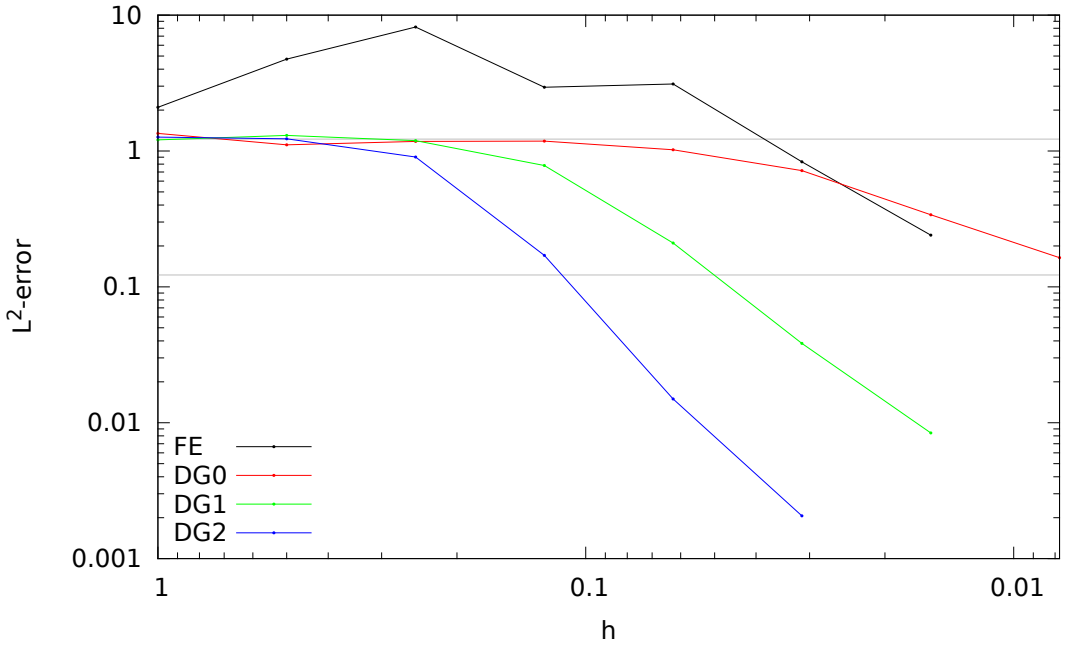


Figure 6.2.: Convergence graphs for the smooth problem in the L^2-error of $\mathbf{E}$ at time $t = 1$ for $\alpha = 8\pi$ ($\lambda = 1/4$). For comparison, the upper gray line shows the L^2-error of $\mathbf{E} = 0$ (≈ 1.25) and the lower gray line shows one tenth of that.

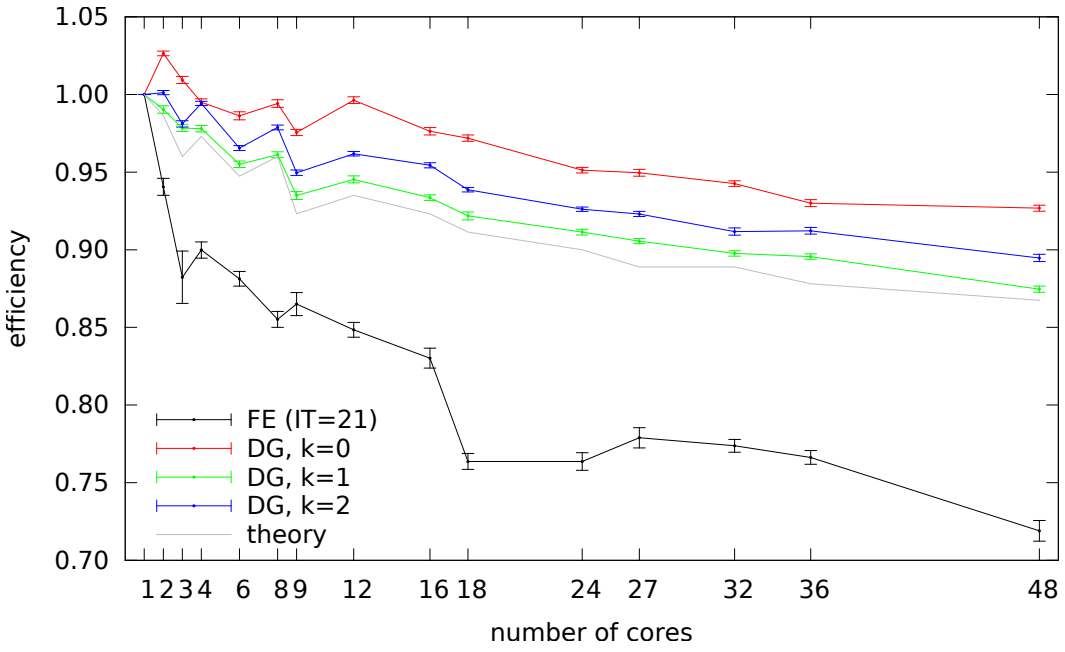


Figure 6.3.: Per-timestep weak efficiency of the DG code on a NUMA system with 48 cores (fiona). The test used the smooth problem with $\alpha = 2\pi$. Computations were done on a structured tetrahedral grid with with $6 \cdot 48^3$ cells and prescribed optimal partitioning. The gray line shows the efficiency that can be expected for DG when considering the effect one layer of ghost cells for the given partitioning. The black line shows the efficiency of the FE scheme. Here, only time steps with 21 iterations in the linear solver were considered.

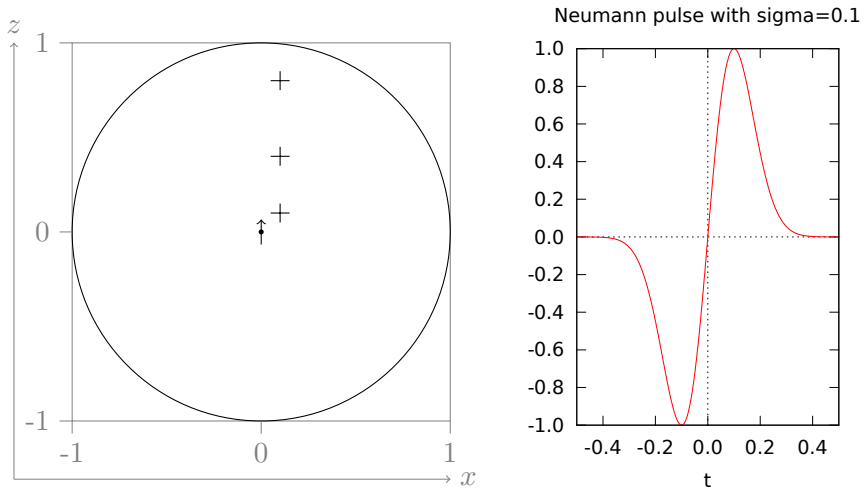


Figure 6.4.: (left) Setup for the dipole problem. The little arrow in the center represents the position and polarization of the point source; the crosses represent the positions where the fields are evaluated (the evaluation positions are also offset by 0.1 into the y-direction which cannot be shown in this 2D representation.) (right) Neumann pulse, the temporal shape used for the excitation. It is normalized such that its maximum is 1.

6.1.4. Scalability

We did some strong scalability computations on the shared-memory system fiona[1] The resulting efficiency graphs are shown in figure 6.3. All DG methods (up to the maximum tested order 2) scale better than the FE method. For DG we are able to computed an expected efficiency based on the size of the overlap regions in relation to the interior regions an each processor. This does not take actual communication into account, only computation overhead due to the need to duplicate data for the purpose of parallelization. As the graph shows, the actual efficiency of the DG schemes is actually better than this expected efficiency. This is due to effects of caching: as the number of processors grows, the per-processor size of the problem shrinks. The size of the level 1 and 2 caches however is constant per processor. This is also true for the level 3 cache for up to eight processes[2]This means that the caches have a better chance to hold the problem the more processes are used.

6.2. Infinitely Small Dipole Antenna

The dipole problem is computed in a spherical domain of radius 1. In the origin there is an infinitely small dipole antenna exciting the setup with a current pulse polarized along the z-axis. The current pulse will create a circular wave traveling outwards to the domain boundary, where it leaves the domain through an absorbing boundary condition (ABC). This test problem is a check for the point source and the ABC.

For this test we only use the analytic solution for $\mathbf{E}$, since we can't test against $\mathbf{H}$ for the FE scheme. The solution for $\mathbf{E}$ for a Neumann pulse with width σ emitted at $\mathbf{x} = 0$ and centered temporally around $t = 0$ is

$$\begin{aligned}\mathbf{E}(\mathbf{x}, t) = \frac{\mu c\sqrt{e}}{4\pi r^2} \exp\left(-\frac{(r-ct)^2}{2c^2\sigma^2}\right) \\ \cdot \left\{-2\hat{\mathbf{e}}_r \cos\theta \left[\frac{r-ct}{c\sigma} + \frac{c\sigma}{r}\right] - \hat{\mathbf{e}}_\theta \sin\theta \left[\frac{r(r-ct)^2}{c^3\sigma^3} - \frac{t}{\sigma} + \frac{c\sigma}{r}\right]\right\} \quad (6.7)\end{aligned}$$

The polar unit vectors can be computed from the Cartesian unit vectors by

$$\begin{aligned}\hat{\mathbf{e}}_\phi &= -\hat{\mathbf{e}}_x \sin\phi + \hat{\mathbf{e}}_y \cos\phi \\ \hat{\mathbf{e}}_\theta &= \hat{\mathbf{e}}_x \cos\theta\cos\phi + \hat{\mathbf{e}}_y \cos\theta \sin\phi - \hat{\mathbf{e}}_z \sin\theta,\end{aligned}$$

The analytic solution is derived in appendix B.

6.2.1. Convergence

This problem has two peculiarities: The singularity at the center and the radiation boundary conditions.

For the radiation boundary conditions the reflection coefficient depends on the angle of incidence, see the book of Jin[23] figure 9.2. It is zero only for 0° (normal incidence). As the angle of incidence tends towards 90°, the reflection coefficient tends toward 1. We use a piecewise linear boundary approximation and do not refine towards the mesh boundary, so in the far field the worst angle of incidence is determined by the coarsest mesh; it is approximately 11°. Thus we expect a certain proportion of the wave to be reflected, and this proportion does not depend on the mesh resolution. Effectively this will mean that our scheme will converge to a solution slightly different from our analytic reference.

The singularity at the center can reduce the convergence order when it is included in the error measure. Excluding a region around that singularity is difficult:

- One could build a small fictitious spherical boundary around the center into the geometry, but that would force the mesh generator to produce lower-quality meshes.

[1] See appendix C for machine details.

[2] The limitation to eight processes stems from the fact that the machine only had eight level 3 caches. Beyond eight processes there was no increase in total available level 3 cache. And even that required careful placement of the processes.

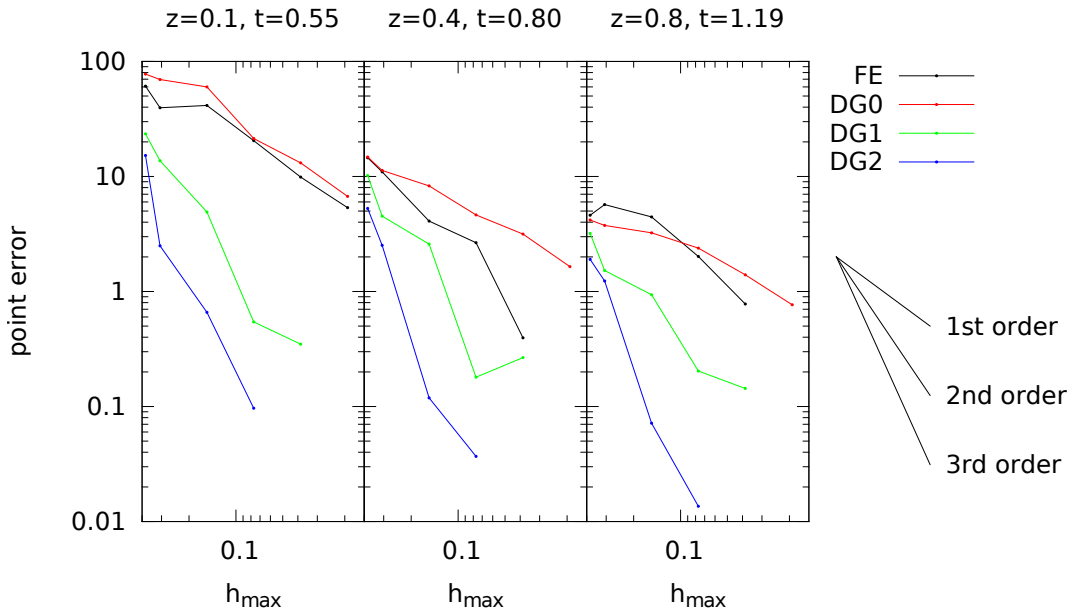


Figure 6.5.: Convergence of the dipole problem at pulse center for wavelength $\lambda = 0.5$ (center frequency $\nu = 2$). This is the error at three different points as shown by the crosses in figure 6.4: $(0.1, 0.1, 0.1)$ (left), $(0.1, 0.1, 0.4)$ (center) and $(0.1, 0.1, 0.8)$ (right) at the time when the center of the wave passes that point.

- One could exclude mesh elements below a certain distance from the center from the calculation of the error, but that would result in a very irregular shape of the excluded region.

- One could use a cut-cell approach to integrate only certain parts of cells near the center, but this is quite costly to implement.

Also, choosing the size of the excluded region is somewhat arbitrary and will influence the number of refinements needed until asymptotic behavior is reached.

For this reasons we have opted to evaluate the error only at certain probe locations chosen to represent different parts of the domain: one near the center at $(0.1, 0.1, 0.1)$, one near the boundary at $(0.1, 0.1, 0.8)$ and one in the intermediate region at $(0.1, 0.1, 0.4)$.

Another consideration is the time when to evaluate the error. The most interesting time is when the pulse passes a certain probe location, so this is the time that we used. For each probe i, we computed the time when the center of the pulse is expected to pass as

$$t_p^{(i)} = t_0 + \frac{\|\mathbf{x}^{(i)}\|}{c} \tag{6.8}$$

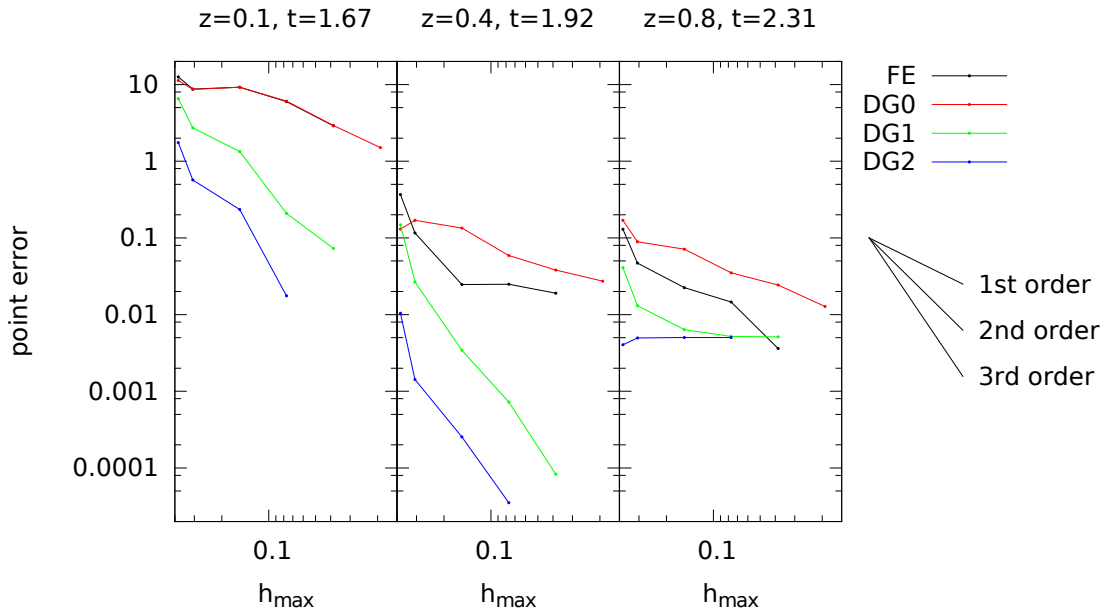


Figure 6.6.: Convergence of the dipole problem at pulse center for wavelength $\lambda = 2$ (center frequency $\nu = 0.5$). This is the error at three different points as shown by the crosses in figure 6.4: $(0.1, 0.1, 0.1)$ (left), $(0.1, 0.1, 0.4)$ (center) and $(0.1, 0.1, 0.8)$ (right) at the time when the center of the wave passes that point.

The errors were then computed as

$$e_h^{(i)} = \|\mathbf{E}_h(\mathbf{x}^{(i)}, t_p^{(i)}) - \mathbf{E}_{\text{ref}}(\mathbf{x}^{(i)}, t_p^{(i)})\| \tag{6.9}$$

Figure 6.5 shows the errors for a wavelength $\lambda = 0.5$ and figure 6.6 for $\lambda = 2$. For DG, in the latter case the probe near the boundary is strongly influenced by a wave reflected from the boundary; this is expected due to the approximate absorbing boundary condition. For the FE scheme we do not observe this problem, although the code can barely reach a similar accuracy due to constraints in computing power.

For the shorter wavelength $\lambda = 0.5$, this problem does not occur, since the boundary is no longer in the near field.

7. Application to Ground-Penetrating Radar

In this final chapter I apply the developed methods to ground penetrating radar. The setup is that of an existing man-made test site for which appropriate GPR measurements were available. For this setup I first do a comparison of the FETD method with the DG method at comparable accuracy at a frequency of 200MHz. This allows to compare the computational effort of these methods. Second, I do a second order DG simulation with an exciting pulse of 400MHz and compare that to the measured data.

7.1. Setup: ASSESS-GPR

ASSESS-GPR is a man-made test site for GPR measurements operated by the group of Kurt Roth near Heidelberg. Geometry, boundary conditions and hydraulic parameters are known by construction, and regular GPR measurements are performed. A schematic of the site is shown in figure 7.1 (top). The site has been built into a bunker silo roughly 19m long, 4m wide and 2m tall. The setup consists of some layers of different sands on top of a layer of gravel. Together with a tube installed through the sand layers, the gravel layer can be used to control the hydraulic head. In a 2001 paper[6] Buchner *et al.* give a more detailed description of the site.

A portion of this test site as shown in figure 7.1 (bottom) has been chosen for the computations based on a number of criteria: availability of WARR measurements for comparison, abilities of the mesh generator influencing the quality of the resulting mesh and available computing power limiting the size of the portion and trace duration. The transmitter was placed to the right at $(17.68\text{m}, 2\text{m}, 1.85\text{m})$, roughly 5.5cm above ground. The excitation was done using a point source to inject a current oriented in y-direction. The exciting pulse had a center frequency of 200MHz, corresponding to wavelength in air of 1.5m; its center was at $t = 4.7\text{ns}$. Point-probes measuring the electric field were placed along the line at $y = 2\text{m}, z = 1.85\text{m}$ at distances of 5cm; only the E_y component of the measured electric field was used. An artificial permittivity distribution was used for the simulation.

7.2. Material Parameters

Magnetic permeability is of no importance in the ASSESS-GPR setup, thus we used $\mu_r = 1$. Conductivity can be of importance but was neglected in this case, i.e. $\sigma = 0$.

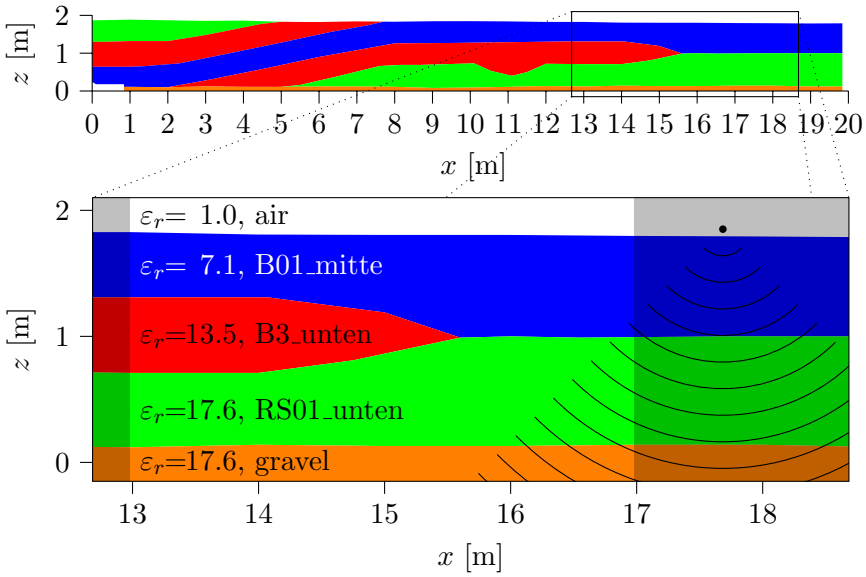


Figure 7.1.: (top) Schematic setup of the ASSESS site. (bottom) Part used for the computations. The dot and the expanding waves represent the sending antenna. The shaded areas are not shown in the measured radargrams.

Since good water content data wasn't available I used some made-up data that was reasonably within expectations. For each material a constant permittivity was chosen as shown in figure 7.1 (bottom). Note that the gravel layer and the RS01_unten sand layer have practically the same permittivity.

Radiating boundary conditions were chosen on the whole boundary. This is arguably not completely correct for the bottom. On the other hand any response from the bottom would be highly attenuated anyway due to the high dielectric permittivity at the bottom.

7.3. Mesh Generation

I modeled the mesh geometry in Gmsh[15], including the layer boundaries. The biggest challenge during mesh generation was to generate a mesh of good quality, usable for both FE and DG schemes. For DG, the CFL number and thus the time step size is limited not only by the edge lengths, but also by the shape of the mesh elements; smaller angles lead to smaller time steps. This seems to be less of a problem for FE.

For thin volumes in the model geometry gmsh tends to generate at least two layers of cells. This became a problem for the gravel layer, since it led to smaller-than-necessary cells and thus smaller-than-necessary time steps. I circumvented this problem by making the gravel layer thicker than in the actual test site; this is no problem since I don't expect so see any meaningful reflection from the bottom of the gravel layer anyway.

The meshing of the geometry was done using Gmsh' Delaunay algorithm both for the

basic mesh	coarse	fine	
Characteristic length	136mm	64mm	
global refines	0	0	1
Number of elements	148 667	1 362 749	10 901 992
Number of edges	182 629	1 631 471	12 880 626
$\min h$	30.8mm	15.6mm	7.8mm
$\max h$	295 mm	150 mm	112 mm
$\min h / \max h$	0.104	0.104	0.070
$\min \Gamma$	0.227	0.212	0.135

Table 7.1.: Properties of the meshes used for calculation. The left and middle were directly generated by Gmsh, the right mesh is the same as the middle mesh with one level of global refinement applied.

2D and the 3D mesh. The mesh was then optimized with `-optimize_netgen`. Meshes needed to stay smaller than $\approx 10^6$ elements, otherwise distributing them to hundreds of cluster nodes became impossible due to limitations of the ALUGrid library.[1] When I needed a finer mesh, I had to obtain it by applying global refinement after distribution. However, I avoided global refinement as far as possible since it can severely impact mesh quality, at least in the first three refinement level where it may introduce elements dissimilar to already existing ones.

I constructed two basic meshes by trying different values of Gmsh characteristic length parameter in the ranges of 0.06m to 0.08m and 0.12m to 0.14m (21 candidates in each range). From these candidates I selected the *fine basic mesh* with a characteristic length of 0.064m and the *coarse basic mesh* with a characteristic length of 0.136m. Selection criteria were the quality of the mesh and the compatibility of the two meshes, i.e. the coarse mesh was chosen such that it had roughly half the resolution of the fine mesh and roughly $1/8^{\text{th}}$ of the elements.

I judged the mesh quality by considering the following properties:

The ratio of the smallest and the largest edge length $\min h / \max h$**:** Since $\min h$ determines the time step size and $\max h$ determines the resolution of grid their ration should be as close to 1 as possible.

Worst element anisotropy $\min \Gamma$**:** Γ is an element quality measure used by Gmsh. It is described as proportional to the ratio of inscribed radius and circumscribed radius. In fact, in the case of a tetrahedron T, it is

$$\Gamma_T := 2\frac{r_i}{r_o}$$

where r_i is indeed the inscribed radius T, but r_o is something different: let l be the length of the longest edge in T and let R be the regular tetrahedron[2]with edge

[1] These limitations have since been lifted.

[2] A regular tetrahedron is a tetrahedron whose edges are all of the same length.

scheme	FE	1st order DG
mesh	fine, refined once	coarse, unrefined
#dofs	12 880 626	3 568 008
#steps	7 301	3 400
Δt	13.7ps	27.8ps
t_{end}	100 ns	100 ns
#CPU threads	512	64
walltime	7h 15min	22min

Table 7.2.: Properties and performance of the numerical schemes.

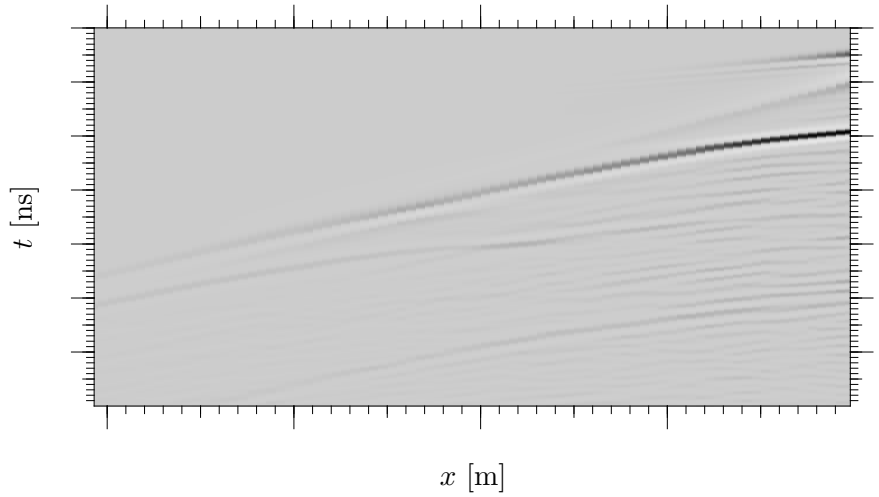


Figure 7.2.: WARR simulation using FE and $h_{\max} \approx 11\text{cm}$

length l, then r_o is the circumscribed radius of R. The factor 2 in the definition of Γ_T makes sure that the maximum possible value for Γ_T will be 1, this is exactly the case when T is a regular tetrahedron.

Three meshes were used in the following calculation: the coarse and fine basic meshes, and the fine basic mesh globally refined once. Their properties are listed in table 7.1.

7.4. Comparison of FE and First Order DG

Here I compare the computational effort of the FE scheme to that of the first order DG scheme. From figure 6.1 I determined the mesh resolutions that should give me a similar L^2 error for both schemes. For the first order DG scheme I used the coarse basic mesh as is, and for the FE scheme I used the fine basic mesh with one level of global refinement applied.

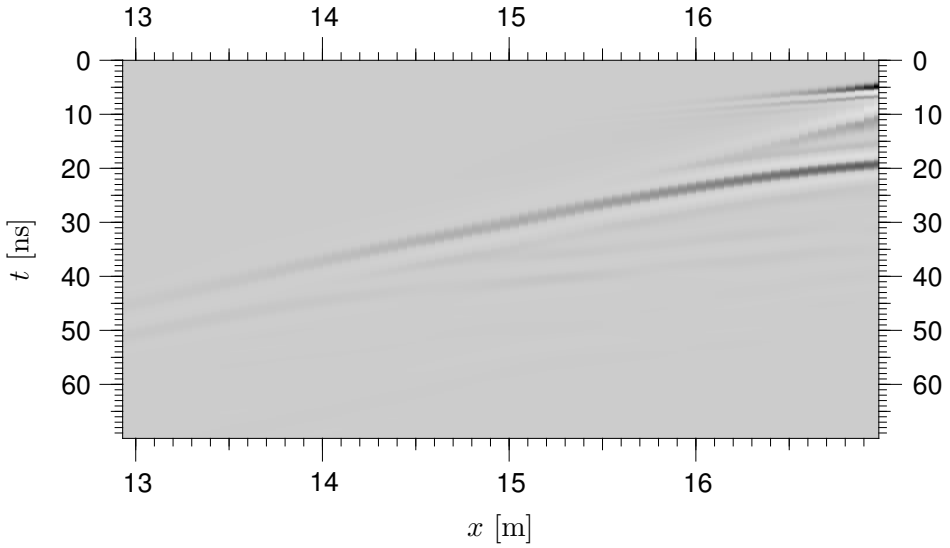


Figure 7.3.: WARR simulation using 1st order DG and $h_{\max} \approx 30$cm

The second order DG scheme has been excluded from the comparison, because for the coarsest mesh I can generate that still represents the geometry of the domain this scheme is still expected to yield a much better L^2 error than any I can achieve with the FE scheme given the computation power I had available. The zeroth order DG scheme has been excluded because with the finest mesh I can apply it to given my limits of computing power, it would be completely unable to resolve the waves and make the computation meaningless.

The setting used was sufficient to obtain a WARR radargram: one sending antenna represented by a point source, and many receiving antennas represented by a line of point probes. The signal emitted by the point source was polarized in the direction of the y-axis, had the shape of a Neumann pulse and a mean frequency of 200MHz. The machine used for computation was helics3a (see appendix C for its properties). The performance and effort for both schemes is shown in table 7.2. The DG scheme required roughly a quarter of the number of degrees of freedom, half the time steps, and thus required only $\sim 0.6\%$ of the computational effort of the FE scheme as measured in used CPU threads and time spent computing.

The radargram in figure 7.2 is the result of the FE simulation; the one in figure 7.3 is from the DG simulation. The FE radargram shows a lot of clutter but gives a much sharper impression when compared to the DG radargram. There are some sampling artifacts in the DG radargram, e.g. in the trace at $x = 16.1$m.

The sharper appearance of the FE radargram isn't just due to the different mesh resolutions. When comparing radargrams obtained with similar mesh resolutions (not shown here) the FE radargram is still sharper, although a higher resolution does improve the sharpness in the case of DG.

basic mesh	coarse	fine	
global refines	0	0	1
time steps	3 465	6 720	19 404
time step size [ps]	20.2	10.4	3.6
#DoFs	8 920 020	81 764 940	654 119 520
#CPU threads	64	256	512
execution time [h]	2.38	11.14	135.31
cost [CPU-days]	6.3	118.9	2886.5

Table 7.3.: Properties of the second order computations. Only the simulated time up to 70ns was considered. #DoFs does not include overlap.

7.5. Comparison of Second Order DG to a Measured Radargram

The radargram in figure 7.4 is a result of a real measurement, the one in figure 7.5 is from a second order DG simulation. The measurement was done with a 400MHz IDS antenna, consequently the simulation was done with a center frequency of 400MHz too. Both radargrams look approximately how one would expect them to look given the geometry of the setup.

In the measured radargram the air wave is pretty weak; in the simulated radargram it is very strong and shows multiple ripples. This difference is to be expected; in the measurement there is shielding between antenna and air and the geometry of geometry of the antenna is that of a bow-tie, not an infinitesimal point leading to quite different radiation characteristics. This also explains why the ground wave appears different in the measured and the simulated radargram.

In the measured radargram the next reflection is at 14ns (at $x = 17.98$m). This reflection does not appear to be present in the simulated radargram. There are three possible explanation:

The reflection originates from the capillary fringe.
The capillary fringe is not a sharp boundary like the boundaries between different layers of sand. Both the 14ns and the 20ns reflection are too narrow to be originating from the capillary fringe.

The estimated permittivity used for the simulation is too far of.
In this case the 14ns reflection in the measured radargram actually corresponds to the 20ns reflection in the simulated data and originates at the boundary between the B01_mitte and RS01_unten layers. Then the 20ns reflection in the measured radargram must originate from the boundary between RS01_unten and the gravel layer. The thickness of the RS01_unten layer between $x = 17$m and $x = 18$m is a pretty constant 0.86m, which has to be traveled twice. Assuming for a moment $\varepsilon_r = 1$ in RS01_unten, and that the wave is traveling straight down, there should be

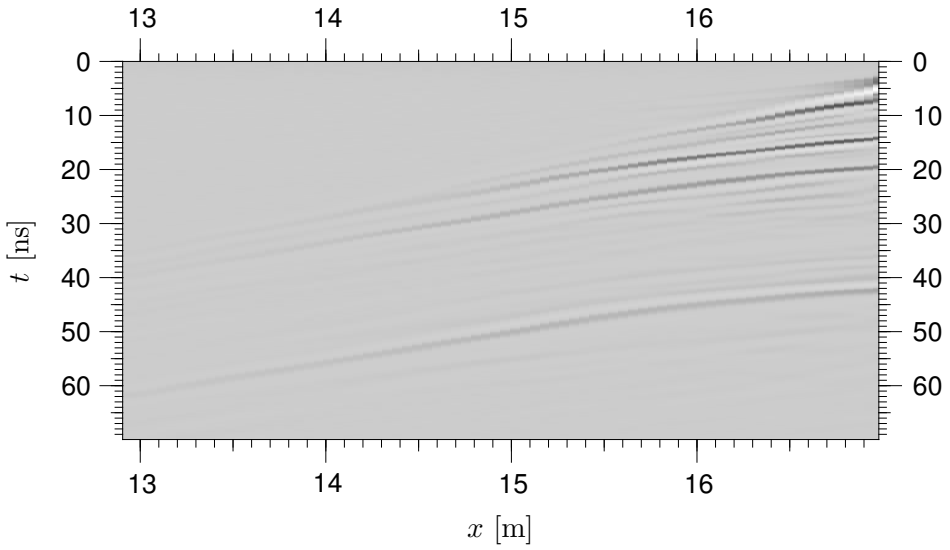


Figure 7.4.: Real-world WARR measurement with a 400MHz IDS antenna.

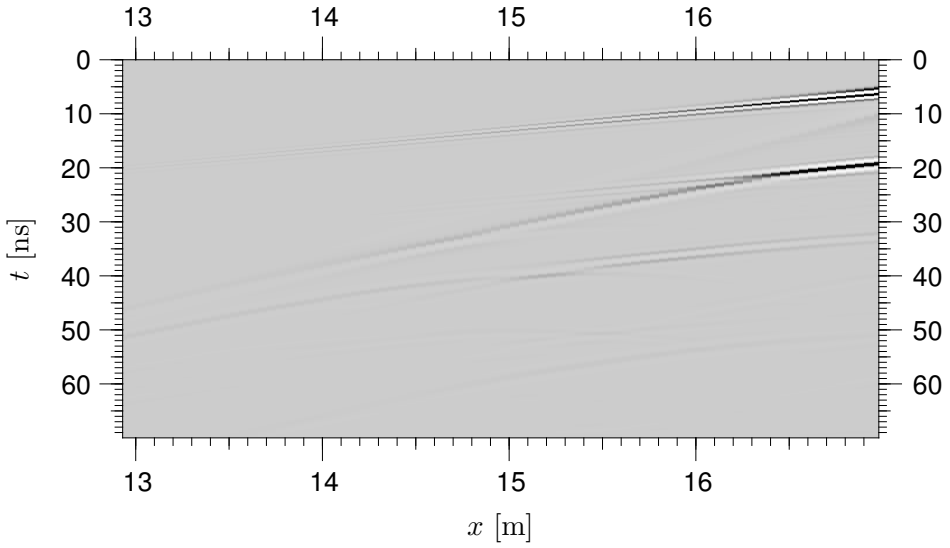


Figure 7.5.: WARR radargram from a second order DG simulation using the fine base mesh without refinement.

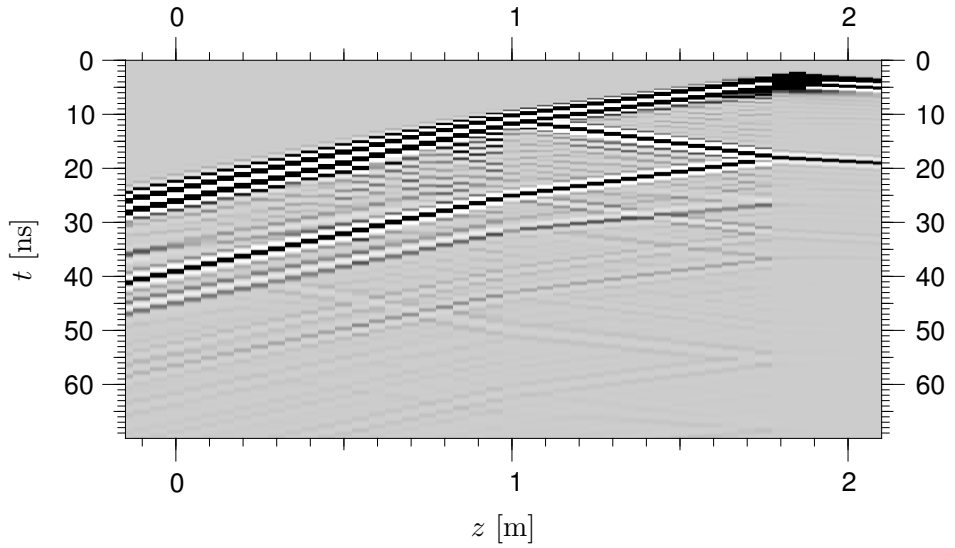


Figure 7.6.: Simulated radargram taken on a vertical line through the sending antenna. This is from the same simulation as figure 7.5.

a distance of 5.7ns between the reflections, which is pretty much what is observed. However the wave is not traveling vertically, and ε_r will be much larger than 1 in RS01_unten. Correcting these bad assumptions will lead to a longer interval between both reflections.

There is an unexpected reflector in the experimental setup.

Indeed, as it turns out, during construction of the ASSESS site the sand had to be compressed, which introduced artificial compression layers.

Therefore I conclude that the 14ns reflection in the measured radargram originates at a compaction boundary.

In the simulated radargram there is a reflection at 33ns. This reflection is actually a multiple of the 20ns reflection. To make sure this is the case I put probes on a vertical line through the sending antenna and used that data to generate a radargram, see figure 7.6. This kind of radargram is of course only possible to do in a simulation.

Finally, there is the reflection at 42ns in the measured data. It is definitely not a multiple of the 20ns reflection, it occurs clearly too late for that. It probably originates either at the upper or the lower boundary of the gravel layer. However, I cannot rule out the possibility that it originates at another compression boundary, or the water table, or even the capillary fringe.

Another difference is the slope of the reflections: the reflection that crosses the right boundary at 20ns crosses the left boundary at 40ns in the measured data but at 45ns in the simulated data. This is probably due to the water content distribution that was used for the simulation and that doesn't quite correspond to the actual water content distribution at the time of the measurement.

8. Conclusions

In this work we were concerned with simulation of ground-penetrating radar. Our method of choice was a full-scale 3D simulation of Maxwell's equations for best accuracy. The typical layered structure of the ground required a method that can handle unstructured meshes. The large problem size required a method that is both efficient and has good scalability on parallel systems.

We developed a novel explicit discontinuous Galerkin scheme based on schemes for hyperbolic conservation laws. We implemented it with the help of the Dune-framework and `dune-pdelab` using message-passing parallelization. Strong scalability for the DG scheme was excellent. In fact, at 48 nodes the second order scheme was at an efficiency of 89%, even better than the expected 87%. This is due to the effect of caching; as the number of nodes goes up, the size of the per-node problem decreases and is more likely to fit into the various caches. Experimental convergence of the DG scheme in the L^2 norm was as expected: $k+1$ for the k^{th} order scheme. This was tested for schemes of order zero, one and two using to homogeneous test problems with known analytic solutions. The tests showed a great benefit in accuracy of the higher order schemes. The second order DG schemes proves almost three orders of magnitude more accurate in the L^2 norm than both the zeroth order DG scheme and the FE scheme, using far less resources.

As a further check, we also implemented a standard finite element scheme based on lowest order $H(\text{curl})$-conforming finite elements. Comparison was difficult since the finite element scheme had trouble keeping up the DG scheme in accuracy.

The results were compared with real-world measurements. These were taken at the man-made ASSESS-GPR test site with a known ground structure. These measurements showed good agreement with the simulation feature-wise, even though the permeability field had to be guessed.

These results show that it is possible to conduct 3D GPR simulations with current computers. This opens the possibilities for a far greater understanding of ground-penetrating radar systems.

8.1. Outlook

These simulation were limited to WARR measurement. Other interesting measurement modes are common offset (CO) and common midpoint (CMP) measurements. Both of them have in common that the sending antenna moves during the measurement. Since one full simulation is necessary per sending antenna position, this means the computational effort increases by another order of magnitude. To meet this challenge the scheme needs to be optimized further.

One approach is to use even higher order shape functions. This is however limited by the fact that the mesh cells cannot become arbitrarily large; they still must be able to resolve the ground structure. Another approach is to reduce the number of mesh elements by creating the mesh adapted to the permeability field. This is possible since the wavelength limits the size of the mesh elements, and the wavelength for any given frequency is itself determined by the local permeability. A third approach may be an improved time stepping scheme. One possibility are *exponential integrators*[21]: they promise to eliminate the need to observe a CFL condition for the sake of stability; the remaining limitation for the time step is only due to accuracy. This is particularly promising since the location of the smallest mesh element are mostly determined by the meshing process and not where the highest accuracy is required.

The influence of conductivity has been neglected here. Conductivity mostly leads to a quicker attenuation of waves, so it was not considered of great importance. However, jumps in conductivity can also give rise to reflections, and a conductive medium can alter the shape of a traveling wave, delaying the passage of the waves maximum compared to an unaltered wave.

The addition of conductivity should just lead to an additional term $(\frac{\sigma}{\varepsilon}u, v)$ in the residual. With regard to the theory, only superficial additions should be necessary.

Open issues are the reflections at the boundary, that turned out to be worse than expected. These particularly seem to arise when a discontinuity in the permittivity meets the domain boundary. If this cannot be resolved, a perfectly matched layer (PML) may help mitigate the problem.

A. Derivation of the Maxwell Flux

Let us first consider the basis $\{\hat{\mathbf{n}}, \hat{\boldsymbol{\alpha}}, \hat{\boldsymbol{\beta}}\}$ of the $\mathbb{R}^3$ that is used to express the eigenvectors. Due to the requirements (3.92), this basis is orthogonal

$$\hat{\mathbf{n}} \cdot \hat{\boldsymbol{\alpha}} = \hat{\mathbf{n}} \cdot \hat{\boldsymbol{\beta}} = \hat{\boldsymbol{\alpha}} \cdot \hat{\boldsymbol{\beta}} = 0. \tag{A.1}$$

Since we also required $\hat{\boldsymbol{\alpha}}$ and $\hat{\boldsymbol{\beta}}$ to be unit vectors, and $\hat{\mathbf{n}}$ is a unit vector anyway, we have

$$\hat{\mathbf{n}} \cdot \hat{\mathbf{n}} = \hat{\boldsymbol{\alpha}} \cdot \hat{\boldsymbol{\alpha}} = \hat{\boldsymbol{\beta}} \cdot \hat{\boldsymbol{\beta}} = 1, \tag{A.2}$$

i.e. the basis is also orthonormal. (3.92) can also be used to recognize the terms $\hat{\boldsymbol{\beta}}\hat{\boldsymbol{\alpha}}^T - \hat{\boldsymbol{\alpha}}\hat{\boldsymbol{\beta}}^T$ and $\hat{\boldsymbol{\alpha}}\hat{\boldsymbol{\alpha}}^T + \hat{\boldsymbol{\beta}}\hat{\boldsymbol{\beta}}^T = \mathbb{1} - \hat{\mathbf{n}}\hat{\mathbf{n}}^T$ as the cross product with $\hat{\mathbf{n}}$ applied once or twice:

$$\begin{aligned}
\hat{\boldsymbol{\beta}}\hat{\boldsymbol{\alpha}}^T - \hat{\boldsymbol{\alpha}}\hat{\boldsymbol{\beta}}^T &= \hat{\mathsf{C}}\hat{\boldsymbol{\alpha}}\hat{\boldsymbol{\alpha}}^T + \hat{\mathsf{C}}\hat{\boldsymbol{\beta}}\hat{\boldsymbol{\beta}}^T \\
&= \hat{\mathsf{C}}(\hat{\mathbf{n}}\hat{\mathbf{n}}^T + \hat{\boldsymbol{\alpha}}\hat{\boldsymbol{\alpha}}^T + \hat{\boldsymbol{\beta}}\hat{\boldsymbol{\beta}}^T) \qquad \text{since } \hat{\mathsf{C}}\hat{\mathbf{n}} = \hat{\mathbf{n}} \times \hat{\mathbf{n}} = 0 \\
&= \hat{\mathsf{C}} \qquad \text{since } \hat{\mathbf{n}}\hat{\mathbf{n}}^T + \hat{\boldsymbol{\alpha}}\hat{\boldsymbol{\alpha}}^T + \hat{\boldsymbol{\beta}}\hat{\boldsymbol{\beta}}^T = \mathbb{1} \qquad \text{(A.3)} \\
\hat{\boldsymbol{\alpha}}\hat{\boldsymbol{\alpha}}^T + \hat{\boldsymbol{\beta}}\hat{\boldsymbol{\beta}}^T &= -\hat{\mathsf{C}}\hat{\boldsymbol{\beta}}\hat{\boldsymbol{\alpha}}^T + \hat{\mathsf{C}}\hat{\boldsymbol{\alpha}}\hat{\boldsymbol{\beta}}^T \\
&= -\hat{\mathsf{C}}(\hat{\boldsymbol{\beta}}\hat{\boldsymbol{\alpha}}^T - \hat{\boldsymbol{\alpha}}\hat{\boldsymbol{\beta}}^T) \\
&= -\hat{\mathsf{C}}^2 \qquad \text{(A.4)}
\end{aligned}$$

These identities will be useful in the following calculation.

To compute the numerical flux for the DG scheme, we need to solve (3.81) for w, and then insert w into (3.78). From (3.104) we compute $\mathsf{M}^T\mathsf{M}$ with the help of (A.1) and (A.2):

$$\begin{aligned}
\mathsf{M}^T\mathsf{M} &= \begin{pmatrix} \sqrt{\varepsilon^{(l)}}\hat{\boldsymbol{\beta}}^T & \sqrt{\mu^{(l)}}\hat{\boldsymbol{\alpha}}^T \\ \sqrt{\varepsilon^{(l)}}\hat{\boldsymbol{\alpha}}^T & -\sqrt{\mu^{(l)}}\hat{\boldsymbol{\beta}}^T \\ \sqrt{\varepsilon^{(r)}}\hat{\boldsymbol{\alpha}}^T & \sqrt{\mu^{(r)}}\hat{\boldsymbol{\beta}}^T \\ -\sqrt{\varepsilon^{(r)}}\hat{\boldsymbol{\beta}}^T & \sqrt{\mu^{(r)}}\hat{\boldsymbol{\alpha}}^T \end{pmatrix} \begin{pmatrix} \sqrt{\varepsilon^{(l)}}\hat{\boldsymbol{\beta}} & \sqrt{\varepsilon^{(l)}}\hat{\boldsymbol{\alpha}} & \sqrt{\varepsilon^{(r)}}\hat{\boldsymbol{\alpha}} & -\sqrt{\varepsilon^{(r)}}\hat{\boldsymbol{\beta}} \\ \sqrt{\mu^{(l)}}\hat{\boldsymbol{\alpha}} & -\sqrt{\mu^{(l)}}\hat{\boldsymbol{\beta}} & \sqrt{\mu^{(r)}}\hat{\boldsymbol{\beta}} & \sqrt{\mu^{(r)}}\hat{\boldsymbol{\alpha}} \end{pmatrix} \\
&= \begin{pmatrix} q^{(l)} & & & -q^{(*)} \\ & q^{(l)} & q^{(*)} & \\ & q^{(*)} & q^{(r)} & \\ -q^{(*)} & & & q^{(r)} \end{pmatrix} \qquad \text{(A.5)}
\end{aligned}$$

Here we have used the abbreviations

$$q^{(l)} := \varepsilon^{(l)} + \mu^{(l)}, \qquad q^{(r)} := \varepsilon^{(r)} + \mu^{(r)}, \qquad q^{(*)} := \sqrt{\varepsilon^{(l)}\varepsilon^{(r)}} - \sqrt{\mu^{(l)}\mu^{(r)}}. \tag{A.6}$$

We observe that $\mathsf{M}^T\mathsf{M}$ consists of two independent 2×2 matrices that can be inverted separately by

$$\begin{pmatrix} \alpha & \beta \\ \gamma & \delta \end{pmatrix}^{-1} = \frac{1}{\alpha\delta - \beta\gamma} \begin{pmatrix} \delta & -\beta \\ -\gamma & \alpha \end{pmatrix}. \tag{A.7}$$

The denominator is the same for both submatrices, and we obtain for the inverse

$$(\mathsf{M}^T\mathsf{M})^{-1} = \frac{1}{q} \begin{pmatrix} q^{(r)} & & & q^{(*)} \\ & q^{(r)} & -q^{(*)} & \\ & -q^{(*)} & q^{(l)} & \\ q^{(*)} & & & q^{(l)} \end{pmatrix}, \tag{A.8}$$

$$q := q^{(l)}q^{(r)} - (q^{(*)})^2 = \left(\sqrt{\varepsilon^{(l)}\mu^{(r)}} + \sqrt{\mu^{(l)}\varepsilon^{(r)}} \right)^2. \tag{A.9}$$

For the right hand side we introduce the notation $b := \mathsf{M}^T [\![\mathsf{A}u]\!]$. This lets us write for the entries of w

$$w = (\mathsf{M}^T\mathsf{M})^{-1}b = \frac{1}{q} \begin{pmatrix} q^{(r)}b_1 + q^{(*)}b_4 \\ q^{(r)}b_2 - q^{(*)}b_3 \\ q^{(l)}b_3 - q^{(*)}b_2 \\ q^{(l)}b_4 + q^{(*)}b_1 \end{pmatrix}. \tag{A.10}$$

Finally we split the flux difference $[\![\mathsf{A}u]\!]$ into an upper part $\mathbf{f}^{(D)}$ and a lower part $\mathbf{f}^{(B)}$. This lets us express the right hand side in terms of the eigenvectors as

$$b = \mathsf{M}^T [\![\mathsf{A}u]\!] = \mathsf{M}^T \begin{pmatrix} \mathbf{f}^{(D)} \\ \mathbf{f}^{(B)} \end{pmatrix} = \begin{pmatrix} \sqrt{\varepsilon^{(l)}}\hat{\boldsymbol{\beta}}^T\mathbf{f}^{(D)} + \sqrt{\mu^{(l)}}\hat{\boldsymbol{\alpha}}^T\mathbf{f}^{(B)} \\ \sqrt{\varepsilon^{(l)}}\hat{\boldsymbol{\alpha}}^T\mathbf{f}^{(D)} - \sqrt{\mu^{(l)}}\hat{\boldsymbol{\beta}}^T\mathbf{f}^{(B)} \\ \sqrt{\varepsilon^{(r)}}\hat{\boldsymbol{\alpha}}^T\mathbf{f}^{(D)} + \sqrt{\mu^{(r)}}\hat{\boldsymbol{\beta}}^T\mathbf{f}^{(B)} \\ -\sqrt{\varepsilon^{(r)}}\hat{\boldsymbol{\beta}}^T\mathbf{f}^{(D)} + \sqrt{\mu^{(r)}}\hat{\boldsymbol{\alpha}}^T\mathbf{f}^{(B)} \end{pmatrix}. \tag{A.11}$$

For Maxwell (3.78) takes the form

$$\hat{f} = f(u)|_{r=0} = \mathsf{A}^{(r)}u^{(r)} - w_3e^{(r,3)} - w_4e^{(r,4)}. \tag{A.12}$$

By inserting (A.10) and reordering we obtain

$$\hat{f} = \mathsf{A}^{(r)}u^{(r)} - \frac{1}{q}\left(q^{(l)}[e^{(r,3)}b_3 + e^{(r,4)}b_4] + q^{(*)}[-e^{(r,3)}b_2 + e^{(r,4)}b_1] \right). \tag{A.13}$$

Now we have to compute the two terms in the square brackets:

$$\begin{aligned} & e^{(r,3)}b_3 + e^{(r,4)}b_4 \\ & \quad = \begin{pmatrix} \varepsilon^{(r)}(\hat{\boldsymbol{\alpha}}\hat{\boldsymbol{\alpha}}^T + \hat{\boldsymbol{\beta}}\hat{\boldsymbol{\beta}}^T)\mathbf{f}^{(D)} + \sqrt{\varepsilon^{(r)}\mu^{(r)}}(\hat{\boldsymbol{\alpha}}\hat{\boldsymbol{\beta}}^T - \hat{\boldsymbol{\beta}}\hat{\boldsymbol{\alpha}}^T)\mathbf{f}^{(B)} \\ \sqrt{\varepsilon^{(r)}\mu^{(r)}}(\hat{\boldsymbol{\beta}}\hat{\boldsymbol{\alpha}}^T - \hat{\boldsymbol{\alpha}}\hat{\boldsymbol{\beta}}^T)\mathbf{f}^{(D)} + \mu^{(r)}(\hat{\boldsymbol{\alpha}}\hat{\boldsymbol{\alpha}}^T + \hat{\boldsymbol{\beta}}\hat{\boldsymbol{\beta}}^T)\mathbf{f}^{(B)} \end{pmatrix} \end{aligned} \tag{A.14}$$

$$\begin{aligned} & -e^{(r,3)}b_2 + e^{(r,4)}b_1 \\ & \quad = \begin{pmatrix} -\sqrt{\varepsilon^{(l)}\varepsilon^{(r)}}(\hat{\boldsymbol{\alpha}}\hat{\boldsymbol{\alpha}}^T + \hat{\boldsymbol{\beta}}\hat{\boldsymbol{\beta}}^T)\mathbf{f}^{(D)} + \sqrt{\mu^{(l)}\varepsilon^{(r)}}(\hat{\boldsymbol{\alpha}}\hat{\boldsymbol{\beta}}^T - \hat{\boldsymbol{\beta}}\hat{\boldsymbol{\alpha}}^T)\mathbf{f}^{(B)} \\ \sqrt{\varepsilon^{(l)}\mu^{(r)}}(\hat{\boldsymbol{\alpha}}\hat{\boldsymbol{\beta}}^T - \hat{\boldsymbol{\beta}}\hat{\boldsymbol{\alpha}}^T)\mathbf{f}^{(D)} + \sqrt{\mu^{(l)}\mu^{(r)}}(\hat{\boldsymbol{\alpha}}\hat{\boldsymbol{\alpha}}^T + \hat{\boldsymbol{\beta}}\hat{\boldsymbol{\beta}}^T)\mathbf{f}^{(B)} \end{pmatrix} \end{aligned} \tag{A.15}$$

By using (A.3) and (A.4), this can be simplified to

$$e^{(r,3)}b_3 + e^{(r,4)}b_4 = \begin{pmatrix} -\varepsilon^{(r)}\hat{\mathsf{C}}^2\mathbf{f}^{(D)} - \sqrt{\varepsilon^{(r)}\mu^{(r)}}\hat{\mathsf{C}}\mathbf{f}^{(B)} \\ \sqrt{\varepsilon^{(r)}\mu^{(r)}}\hat{\mathsf{C}}\mathbf{f}^{(D)} - \mu^{(r)}\hat{\mathsf{C}}^2\mathbf{f}^{(B)} \end{pmatrix} \tag{A.16}$$

$$-e^{(r,3)}b_2 + e^{(r,4)}b_1 = \begin{pmatrix} \sqrt{\varepsilon^{(l)}\varepsilon^{(r)}}\hat{\mathsf{C}}^2\mathbf{f}^{(D)} - \sqrt{\mu^{(l)}\varepsilon^{(r)}}\hat{\mathsf{C}}\mathbf{f}^{(B)} \\ -\sqrt{\varepsilon^{(l)}\mu^{(r)}}\hat{\mathsf{C}}\mathbf{f}^{(D)} - \sqrt{\mu^{(l)}\mu^{(r)}}\hat{\mathsf{C}}^2\mathbf{f}^{(B)} \end{pmatrix} \tag{A.17}$$

This is now inserted into (A.13).

$$\begin{aligned} \hat{f} = \mathsf{A}^{(r)}u^{(r)} & - \frac{1}{q}\begin{pmatrix} \left(-q^{(l)}\varepsilon^{(r)} + q^{(*)}\sqrt{\varepsilon^{(l)}\varepsilon^{(r)}}\right)\hat{\mathsf{C}}^2\mathbf{f}^{(D)} \\ \left(-q^{(l)}\mu^{(r)} - q^{(*)}\sqrt{\mu^{(l)}\mu^{(r)}}\right)\hat{\mathsf{C}}^2\mathbf{f}^{(B)} \end{pmatrix} \\ & - \frac{1}{q}\begin{pmatrix} \left(-q^{(l)}\sqrt{\varepsilon^{(r)}\mu^{(r)}} - q^{(*)}\sqrt{\mu^{(l)}\varepsilon^{(r)}}\right)\hat{\mathsf{C}}\mathbf{f}^{(B)} \\ \left(q^{(l)}\sqrt{\varepsilon^{(r)}\mu^{(r)}} - q^{(*)}\sqrt{\varepsilon^{(l)}\mu^{(r)}}\right)\hat{\mathsf{C}}\mathbf{f}^{(D)} \end{pmatrix} \end{aligned} \tag{A.18}$$

We now expand the abbreviations q, $q^{(l)}$ and $q^{(*)}$. It is most convenient to write the resulting coefficients in terms of the admittance (3.105) and impedance (3.106), as well as the intermediate admittance and impedance (3.107). The coefficients then take the form

$$\frac{1}{q}\left(-q^{(l)}\varepsilon^{(r)} + q^{(*)}\sqrt{\varepsilon^{(l)}\varepsilon^{(r)}}\right) = -Z^{(l)}Y^{(*)}, \tag{A.19}$$

$$\frac{1}{q}\left(-q^{(l)}\mu^{(r)} - q^{(*)}\sqrt{\mu^{(l)}\mu^{(r)}}\right) = -Y^{(l)}Z^{(*)}, \tag{A.20}$$

$$\frac{1}{q}\left(-q^{(l)}\sqrt{\varepsilon^{(r)}\mu^{(r)}} - q^{(*)}\sqrt{\mu^{(l)}\varepsilon^{(r)}}\right) = -Y^{(*)}, \tag{A.21}$$

$$\frac{1}{q}\left(q^{(l)}\sqrt{\varepsilon^{(r)}\mu^{(r)}} - q^{(*)}\sqrt{\varepsilon^{(l)}\mu^{(r)}}\right) = Z^{(*)}, \tag{A.22}$$

and the flux can be written as

$$\hat{f} = \mathsf{A}^{(r)}u^{(r)} + \begin{pmatrix} Z^{(l)}Y^{(*)}\hat{\mathsf{C}}^2\mathbf{f}^{(D)} \\ Y^{(l)}Z^{(*)}\hat{\mathsf{C}}^2\mathbf{f}^{(B)} \end{pmatrix} + \begin{pmatrix} Y^{(*)}\hat{\mathsf{C}}\mathbf{f}^{(B)} \\ -Z^{(*)}\hat{\mathsf{C}}\mathbf{f}^{(D)} \end{pmatrix}. \tag{A.23}$$

With the matrices (3.108), (3.109) and (3.88) this can be simplified to

$$\hat{f} = f(u)|_{r=0} = \mathsf{A}^{(r)}u^{(r)} - \mathsf{X}^{(*)}\mathsf{N}(\mathsf{X}^{(l)}\mathsf{N} + \mathbb{1})\left[\!\left[\mathsf{A}u\right]\!\right]. \tag{A.24}$$

B. Analytic Solution for the Dipole Problem

For the moment we'll assume that the exciting current has the form

$$\mathbf{J}_\omega(\mathbf{x},t) = \mathbf{J}_\omega(\mathbf{x})e^{-i\omega t} = \hat{\mathbf{e}}_z\delta(\mathbf{x})e^{-i\omega t}. \tag{B.1}$$

According to [22] eq. (9.3) this allows us to write the vector potential as

$$\mathbf{A}_\omega(\mathbf{x},t) = \frac{\mu}{4\pi}\int \mathbf{J}_\omega(\mathbf{x}',t)\frac{e^{ik|\mathbf{x}-\mathbf{x}'|}}{|\mathbf{x}-\mathbf{x}'|}\,d^3x'. \tag{B.2}$$

The fields can then be obtained from the vector potential via

$$\mathbf{H}_\omega(\mathbf{x},t) = \mu^{-1}\nabla\times\mathbf{A}_\omega(\mathbf{x},t) \tag{B.3}$$

$$\mathbf{E}_\omega(\mathbf{x},t) = \frac{iZ}{k}\nabla\times\mathbf{H}_\omega(\mathbf{x},t) = \frac{ic^2}{\omega}\nabla\times\nabla\times\mathbf{A}_\omega(\mathbf{x},t). \tag{B.4}$$

Here the following quantities were used:

$$k = \frac{\omega}{c} \qquad \lambda = \frac{2\pi c}{\omega} = \frac{2\pi}{k} \qquad Z = \sqrt{\frac{\mu}{\varepsilon}} \tag{B.5}$$

Inserting our current into (B.2) we obtain

$$\mathbf{A}_\omega(\mathbf{x},t) = \hat{\mathbf{e}}_z\frac{\mu e^{i(kr-\omega t)}}{4\pi r}. \tag{B.6}$$

Here and in the following we assume the following translation into spherical and cylindrical coordinates:

$$x_0 = \rho\cos\phi \qquad x_1 = \rho\sin\phi \qquad x_2 = z \tag{B.7}$$

$$\rho = r\sin\theta \qquad \phi = \phi \qquad z = r\cos\theta \tag{B.8}$$

$$x_0 = r\sin\theta\cos\phi \qquad x_1 = r\sin\theta\sin\phi \qquad x_2 = r\cos\theta \tag{B.9}$$

To continue we need

$$\nabla\times\hat{\mathbf{e}}_z\frac{e^{ikr}}{r} = \hat{\mathbf{e}}_\phi\frac{\sin\theta}{r^2}(1-ikr)e^{ikr} \tag{B.10}$$

$$\nabla\times\nabla\times\hat{\mathbf{e}}_z\frac{e^{ikr}}{r} = \hat{\mathbf{e}}_r\frac{2\cos\theta}{r^3}(1-ikr)e^{ikr} - \hat{\mathbf{e}}_\theta\frac{\sin\theta}{r^3}(k^2r^2+ikr-1)e^{ikr} \tag{B.11}$$

The first was obtained using the curl in cylindrical coordinates

$$\nabla \times \mathbf{F} = \hat{\mathbf{e}}_\rho \left[\frac{1}{\rho}\partial_\phi F_z - \partial_z F_\phi\right] + \hat{\mathbf{e}}_\phi[\partial_z F_\rho - \partial_\rho F_z] + \hat{\mathbf{e}}_z \frac{1}{\rho}[\partial_\rho(\rho F_\phi) - \partial_\phi F_\rho] \tag{B.12}$$

$$= -\hat{\mathbf{e}}_\phi \partial_\rho F_z \qquad \text{for } \mathbf{F}||\hat{\mathbf{e}}_z, \mathbf{F} = \mathbf{F}(r) \tag{B.13}$$

$$= -\hat{\mathbf{e}}_\phi (\partial_r F_z)\partial_\rho r = -\hat{\mathbf{e}}_\phi \frac{\rho}{r}\partial_r F_z = -\hat{\mathbf{e}}_\phi \sin\theta\, \partial_r F_z, \tag{B.14}$$

the second by using the curl in spherical coordinates

$$\nabla \times \mathbf{G} = \hat{\mathbf{e}}_r \frac{1}{r\sin\theta}[\partial_\theta(\sin\theta G_\phi) - \partial_\phi G_\theta]$$
$$+ \hat{\mathbf{e}}_\theta \left[\frac{1}{r\sin\theta}\partial_\phi G_r - \frac{1}{r}\partial_r(rG_\phi)\right] + \hat{\mathbf{e}}_\phi \frac{1}{r}[\partial_r(rG_\theta) - \partial_\theta G_r] \tag{B.15}$$

$$= \hat{\mathbf{e}}_r \frac{1}{r\sin\theta}\partial_\theta(\sin\theta\, G_\phi) - \hat{\mathbf{e}}_\theta \frac{1}{r}\partial_r(rG_\phi) \qquad \text{for } \mathbf{G}||\hat{\mathbf{e}}_\phi, \mathbf{G} = \mathbf{G}(r,\theta) \tag{B.16}$$

Now we can write the magnetic and the electric fields

$$\mathbf{H}_\omega(\mathbf{x},t) = \frac{e^{i(kr-\omega t)}}{4\pi r^2}\hat{\mathbf{e}}_\phi \sin\theta(1-ikr) \tag{B.17}$$

$$\mathbf{E}_\omega(\mathbf{x},t) = \frac{ie^{i(kr-\omega t)}}{4\pi\varepsilon\omega r^3}[2\hat{\mathbf{e}}_r\cos\theta(1-ikr) - \hat{\mathbf{e}}_\theta\sin\theta(k^2r^2+ikr-1)] \tag{B.18}$$

Up to now everything was done for one particular frequency ω. To cover the whole frequency range we need a weighted integral over all frequencies:

$$\mathbf{J}(\mathbf{x},t) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty} d\omega J(\omega)\mathbf{J}_\omega(\mathbf{x},t) = \frac{1}{\sqrt{2\pi}}\hat{\mathbf{e}}_z\delta(\mathbf{x})\int_{-\infty}^{\infty} d\omega J(\omega)e^{-i\omega t} \tag{B.19}$$

By the principle of superposition, the same integral has to be applied to the fields:

$$\mathbf{H}(\mathbf{x},t) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty} d\omega J(\omega)\mathbf{H}_\omega(\mathbf{x},t) \tag{B.20}$$

$$\mathbf{E}(\mathbf{x},t) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty} d\omega J(\omega)\mathbf{E}_\omega(\mathbf{x},t) \tag{B.21}$$

We define the temporal dependency of the exciting current

$$J(t) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty} d\omega\, J(\omega)e^{-i\omega t} \tag{B.22}$$

such that $\mathbf{J}(\mathbf{x},t) = \hat{\mathbf{e}}_z\delta(\mathbf{x})J(t)$. This is just the Fourier-transform – to obtain $J(\omega)$ we need the corresponding back-transform

$$J(\omega) = \frac{1}{\sqrt{2\pi}}\int_{-\infty}^{\infty} dt\, J(t)e^{i\omega t}. \tag{B.23}$$

Now we have to commit to one particular temporal form of for the excitation. We use a Neumann-pulse normalized such that $\max_t J(t) = 1$:

$$J(t) = \frac{\sqrt{e}}{\sigma} t \exp\left(-\frac{t^2}{2\sigma^2}\right) \tag{B.24}$$

$$\begin{aligned} J(\omega) &= \frac{\sqrt{e}}{\sqrt{2\pi}\sigma} \int_{-\infty}^{\infty} dt\, t \exp\left(-\frac{t^2}{2\sigma^2}\right) e^{i\omega t} \\ &= i\omega\sigma^2\sqrt{e} \exp\left(-\frac{\omega^2\sigma^2}{2}\right) \end{aligned} \tag{B.25}$$

Inserting this into (B.20) and (B.21) leads to

$$\mathbf{H}(\mathbf{x},t) = \frac{\sqrt{e}}{4\pi r^2} \exp\left(-\frac{(r-ct)^2}{2c^2\sigma^2}\right) \hat{\mathbf{e}}_\phi \sin\theta \left[\frac{t}{\sigma} - \frac{r(r-ct)^2}{c^3\sigma^3}\right] \tag{B.26}$$

$$\begin{aligned} \mathbf{E}(\mathbf{x},t) = \frac{\mu c\sqrt{e}}{4\pi r^2} \exp\left(-\frac{(r-ct)^2}{2c^2\sigma^2}\right) \Bigg\{ &- 2\hat{\mathbf{e}}_r \cos\theta \left[\frac{r-ct}{c\sigma} + \frac{c\sigma}{r}\right] \\ &- \hat{\mathbf{e}}_\theta \sin\theta \left[\frac{r(r-ct)^2}{c^3\sigma^3} - \frac{t}{\sigma} + \frac{c\sigma}{r}\right] \Bigg\} \end{aligned} \tag{B.27}$$

To return to Cartesian coordinates, we use

$$\hat{\mathbf{e}}_r = \begin{pmatrix} \sin\theta\cos\phi \\ \sin\theta\sin\phi \\ \cos\theta \end{pmatrix} \qquad \hat{\mathbf{e}}_\phi = \begin{pmatrix} -\sin\phi \\ \cos\phi \\ 0 \end{pmatrix} \qquad \hat{\mathbf{e}}_\theta = \begin{pmatrix} \cos\theta\cos\phi \\ \cos\theta\sin\phi \\ -\sin\theta \end{pmatrix} \tag{B.28}$$

C. Machines used for Computations

	fiona	helics3a
machine type	shared-memory	cluster
CPU model	AMD Opteron™ 6172	AMD Opteron™ 6212
cluster interconnect	-/-	InfiniBand
cluster nodes	1	32
total RAM	128GiB	4TiB
total CPU threads	48	1024
per cluster node...		
├─ sockets	4	4
├─ NUMA nodes	8	8
├─ CPU threads	48	32
├─ caches		
│ ├─ L3	8 × 5MiB	8 × 6MiB
│ ├─ L2	48 × 512KiB	16 × 2MiB
│ ├─ L1 instruction	48 × 64KiB	16 × 64KiB
│ └─ L1 data	48 × 64KiB	32 × 16KiB
└─ RAM	128GiB	128GiB

D. Global-valued Finite Elements in `Dune-localfunctions`

In the course of implementing the finite element scheme for electromagnetics, it became necessary to design a more widely applicable interface for global-valued finite elements.

Formerly only local-valued finite element were available – shape functions where both domain and range a defined in the coordinate system of a reference element. User code (or a discretization module) was expected to apply the correct transformation into global coordinates. This proved successful as long as only scalar and normal-conforming vector finite elements were used – the discretization modules were easily able to chose the correct transformation.

Edge elements however require a different transformation. Either this transformation must be communicated to the user of the finite element, or the range of the finite element must be shifted from the reference element into global space. The latter choice nicely abstracts the details of the transformation and keeps them to the finite element, so that is what I chose to do.

To evaluate a function represented by a finite element basis on a particular grid element T with geometry μ we can use the following formula:

$$u(x) = \sum_{i=0}^{N_T-1} c_i P_{T,i} \varphi_i(\mu^{-1}(x)) \qquad \forall x \in T \tag{D.1}$$

The basis function φ on the reference element is provided by the local basis. The global basis takes this local basis and applies an operator $P_{T,i}$ to the values it returns. This operator is dependent on the grid element T and the number of the basis function i. The global basis thus provides values of the global basis functions

$$\Phi_i(\hat{x}) = P_{T,i} \varphi_i(\hat{x}) \tag{D.2}$$

For the transformation P the following information about grid element is important:

1. The *geometry* μ of a grid element, which handles the transformation of coordinates from the reference element to the grid element. But *values* of the base functions and in particular their derivatives need to be transformed as well in general – the correct transformation depends on the family of the finite element, the coordinate transformation μ and the number of the base function i.

2. The *vertex ordering* τ of a grid element, which says how the grid elements vertices are globally numbered in comparison to the numbering in the reference element.

This is needed to match multiple dofs on a common sub-entity between two grid elements. Another use is to choose a consistent tangential orientation of edges for edge elements.

3. The *normal orientation* of faces of a grid element. This is useful for instance for Raviart-Thomas elements: their dofs orientation points from one of the neighbouring elements into the other. This information can generally not be extracted from the vertex ordering and geometry information alone.

This section explicitly does not deal with the following issues:

- Different geometry types for different grid elements. This will lead to different number of basis functions and must already be dealt with in the local finite element.
- p-adaptiviy. Again, this will lead to different number of basis functions and must already be dealt with in the local finite element.

D.1. Geometry

The geometry information must be provided by a class basically modelling the interface of `GenericGeometry::BasicGeometry` – that includes implementations of `Geometry`. The precise requirements are as follows:

```
struct Geometry
{
  // type information
  typedef ImplementationDefined ctype;
  // local dimension
  static const std::size_t mydimension =
     implementation_defined;
  // global dimension
  static const std::size_t coorddimension =
     implementation_defined;
  // some vector type with mydimension components of type
  // ctype
  typedef ImplementationDefined LocalCoordinate;
  // some vector type with coorddimension components of type
  // ctype
  typedef ImplementationDefined GlobalCoordinate;
  // some matrix type with coorddimension x mydimension
  // components of type ctype
  typedef ImplementationDefined JacobianInverseTransposed;
  // some matrix type with mydimension x coorddimension
  // components of type ctype
  typedef ImplementationDefined JacobianTransposed;
```

```
  // general properties of the geometry
  GeometryType type() const;
  bool affine() const;

  // access to the coordinates of the corners
  std::size_t corners() const;
  GlobalCoordinate corner(std::size_t) const;

  // local to global and inverse mapping
  GlobalCoordinate global(const LocalCoordinate&) const;
  LocalCoordinate local(const GlobalCoordinate&) const;

  // access to Jacobian of the mapping
  const JacobianTransposed&
    jacobianTransposed(const LocalCoordinate&) const;
  const JacobianInverseTransposed&
    jacobianInverseTransposed(const LocalCoordinate&) const;

  // other information
  GlobalCoordinate center() const;
  ctype integrationElement(const LocalCoordinate&) const;
  ctype volume() const;
  GlobalCoordinate normal(std::size_t face,
                          const LocalCoordinate&) const;
};
```

The coordinate types (`ctype`, `mydimension`, `coorddimension`, `LocalCoordinate`, and `GlobalCoordinate`) of a `Geometry` object provided when creating an instance of a finite element should coincide with the coordinate types of that finite element's basis class.

D.1.1. Gradient Transformation

The transformation of a scalar function from the reference element to a grid element using the geometry μ is trivial:

$$\hat{f}(\hat{x}) = f(\mu(\hat{x})) \tag{D.3}$$

The transformation of the gradient of such a function is a little bit more complicated. First we will need to employ the Jacobian, which we define for a function u as:

$$J_u(x) = \begin{pmatrix} \partial_0 u_0|_x & \dots & \partial_{n-1} u_0|_x \\ \vdots & \ddots & \vdots \\ \partial_0 u_{m-1}|_x & \dots & \partial_{n-1} u_{m-1}|_x \end{pmatrix} \tag{D.4}$$

This definition of the Jacobian lets us write a linear vector-valued function u in terms of its Jacobian J_u as $u(x) = J_u \cdot x$. For a scalar valued function f the gradient is the

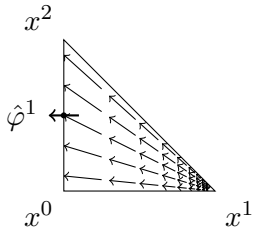


Figure D.1.: Raviart-Thomas element in 2D.

transpose of the Jacobian:

$$\nabla f|_x = \begin{pmatrix} \partial_0 f|_x \\ \vdots \\ \partial_{n-1} f|_x \end{pmatrix} = J_f^T(x) \tag{D.5}$$

To do the actual transformation we employ the chain rule

$$\hat{J}_{\hat{f}}(\hat{x}) = J_f(\mu(\hat{x})) \cdot \hat{J}_\mu(\hat{x}) \tag{D.6}$$

After transposing, left-multiplying by $\hat{J}_\mu^{-T}(\hat{x})$ and replacing the transposed Jacobians by gradient where applicable, we obtain

$$\nabla f|_{\mu(\hat{x})} = \hat{J}_\mu^{-T}(\hat{x}) \cdot \hat{\nabla} \hat{f}|_{\hat{x}} \tag{D.7}$$

D.1.2. Raviart-Thomas Elements – Piola Transformation

Raviart-Thomas elements are finite elements that ensure continuity of the normal component across grid elements. They do allow for jumps in the tangential components, however. For these elements, the degrees-of-freedom (dofs) are usually associated with the face (sub-entity of codimension 1) on which the normal component is non-zero – see figure D.1.

These elements have the following property:

$$\varphi^i(x) \cdot n^j = \delta_{ij} \quad \forall x \in \text{face } j \tag{D.8}$$

Here n^j is the outer normal unit vector to face j and δ_{ij} is the Kronecker delta. Naturally, transforming the basis should preserve that property. This is achieved by the Piola-transformation:

$$\varphi^i(\mu(\hat{x})) = \frac{\hat{J}_\mu(\hat{x})}{|\hat{J}_\mu(\hat{x})|} \hat{\varphi}^i(\hat{x}) \tag{D.9}$$

D.1.3. Edge Elements

Edge elements are used in finite element electro-magnetics. In the lowest order, their dofs are associated with edges, i.e. sub-entities of dimension 1. They can be expressed

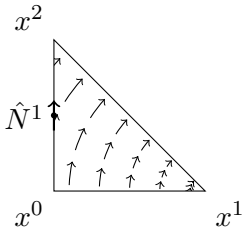


Figure D.2.: Edge-element in 2D.

in terms of first order node-based Lagrange finite elements L^i as follows:

$$N^i = \ell^i (L^{i_0} \nabla L^{i_1} - L^{i_1} \nabla L^{i_0}) \tag{D.10}$$

Here i_0 and i_1 are the indices of the nodes at the endpoints of edge i and ℓ^i is the length of edge i. See figure D.2.

Edge elements have a similar property as Raviart-Thomas elements: the tangential component is 1 on the associated edge and 0 on all other edges:

$$N^i(x) \cdot t^j = \delta_{ij} \quad \forall x \in \text{edge } j \tag{D.11}$$

For the transformation we make the ansatz

$$N^i(\mu(\hat{x})) = \alpha^i A \hat{N}^i(\hat{x}) \tag{D.12}$$

with the scalars α^i and a matrix A. We express N^i and $\hat{N}^i$ in terms of the corresponding P1 bases

$$\ell^i \{L^{i_0}(\mu(\hat{x})) \cdot \nabla L^{i_1}|_{\mu(\hat{x})} - L^{i_1}(\mu(\hat{x})) \cdot \nabla L^{i_0}|_{\mu(\hat{x})}\} = \alpha^i A \hat{\ell}^i \{\hat{L}^{i_0}(\hat{x}) \cdot \hat{\nabla} \hat{L}^{i_1}|_{\hat{x}} - \hat{L}^{i_1}(\hat{x}) \cdot \hat{\nabla} \hat{L}^{i_0}|_{\hat{x}}\} \tag{D.13}$$

By replacing the global P1 bases by the their transformations

$$L^i(\mu(\hat{x})) = \hat{L}^i(\hat{x}) \tag{D.14}$$

$$\nabla L^i|_{\mu(\hat{x})} = \hat{J}_\mu^{-T}(\hat{x}) \hat{\nabla} \hat{L}^i|_{\hat{x}} \tag{D.15}$$

we obtain

$$\ell^i \hat{J}_\mu^{-T}(\hat{x}) \{\hat{L}^{i_0}(\hat{x}) \cdot \hat{\nabla} \hat{L}^{i_1}|_{\hat{x}} - \hat{L}^{i_1}(\hat{x}) \cdot \hat{\nabla} \hat{L}^{i_0}|_{\hat{x}}\} = \alpha^i A \hat{\ell}^i \{\hat{L}^{i_0}(\hat{x}) \cdot \hat{\nabla} \hat{L}^{i_1}|_{\hat{x}} - \hat{L}^{i_1}(\hat{x}) \cdot \hat{\nabla} \hat{L}^{i_0}|_{\hat{x}}\} \tag{D.16}$$

The expression inside the curly braces on both sides is the same. We identify

$$A = \hat{J}_\mu^{-T}(\hat{x}) \tag{D.17}$$

$$\alpha^i = \ell^i / \hat{\ell}^i \tag{D.18}$$

The full transformation then looks like this:

$$N^i(\mu(\hat{x})) = \frac{\ell^i}{\hat{\ell}^i} \hat{J}_\mu^{-T}(\hat{x}) \cdot \hat{N}^i(\hat{x}) \tag{D.19}$$

Note that this transformation only works for the base functions, not for superpositions of them. Each base function N^i has a different transformation because the base multiplier α^i depends on the number of the base function.

D.1.4. Conclusions

From the examples above we can conclude that the following information is needed from a `Geometry` class. It is quite possible that the list below is incomplete since the examples above may have missed some piece of information that may be needed in general.

- The inverse transposed of the Jacobian $\hat{J}_\mu^{-T}(\hat{x})$.
- The Jacobian itself $\hat{J}_\mu(\hat{x})$.
- The determinant of the Jacobian $|\hat{J}_\mu(\hat{x})|$.
- The lengths of the edges of the grid element ℓ^i.
- The lengths of the edges of the reference element $\hat{\ell}^i$.

When local coordinates $\hat{x}$ are provided the local-to-global map $\mu(\hat{x})$ and its inverse $\mu^{-1}(x)$ as well as the corner coordinates x^i themselves are never needed. This makes the required information independent of a shift in the global coordinates and opens an optimisation possibility for regular grids.

D.2. Vertex Ordering

The vertex ordering information is based completely on the global numbering of the vertices of a grid element. To obtain it, we collect the global IDs of the vertices in a vector indexed by the indices of the vertices within the reference element:

```
void collectVertexIds(const Element& e,
                      const GlobalIdSet& idSet,
                      std::vector<GlobaIdSet::IdType>& ids)
{
  ids.resize(e.geometry().corners());
  for(int i = 0; i < ids.size(); ++ids)
    ids[i] = idSet.subId(e, i, Element::dimension);
}
```

In the next step the *ordering reduction* operation is applied: the smallest id in the array is replaced by the number 0, the second-smallest is replaced by the number 1 etc.

```
template<class InIterator, class OutIterator>
void reduceOrder(const InIterator& inBegin,
                 const InIterator& inEnd,
                 OutIterator outIt)
{
  static const std::less<
    typename std::iterator_traits<InIterator>::value_type
    > less;

  for(InIterator inIt = inBegin; inIt != inEnd;
      ++inIt, ++outIt)
    *outIt = std::count(inBegin, inEnd,
                        std::bind2nd(less, *inIt));
}
```

To obtain an actual vector of reduced indices one can use the following code:

```
std::vector<typename GlobalIdSet::IdType> ids;
collectVertexIds(elem, globalIdSet, ids);
std::vector<std::size_t> reduced_indices(ids.size());
reduceOrder(ids.begin(), ids.end(),
            reduced_indices.begin());
```

As an example, lets assume we have a quadrilateral or a tetrahedron with the global ids of the vertices being 14 for vertex 0, 27 for vertex 1, 3 for vertex 2 and 800 for vertex 3. After ordering reduction the reduced vector will contain 1, 2, 0 and 3 in that order.

When determining the vertex ordering for a sub-entity, the reduced indices corresponding to the vertices sub-entity are extracted into a smaller vector and the reduction is applied again, at least conceptually. In reality, the reduction is mostly only necessary because the type of the global ids may be a complicated non-integral struct, and we want to keep the vertex ordering information as lean as possible. The actual information is always contained in the relative ordering of the indices/ids, and the reduction preserves that.

The ordering information can always be obtained from the global ids of the vertices. However, for some grids, such as ALUGrid, using the global ids is quiet expensive. On the other hand, ALUGrid already stores a twist of the faces, which can be easily extracted and contains the same information as the vertex ordering, just encoded in a different way. Though this does not provide vertex ordering information for the whole element, this information is seldom needed.

To accommodate all sides, we define an interface class `VertexOrderingInterface`. Implementations of this interface can be used to provide vertex ordering information. Grids that store the vertex ordering internally for certain sub-entities can provide an optimised implementation. These implementations may omit vertex ordering information for sub-entities where such information is not readily available; they should throw `NotImplemented` if such information is requested anyway.

Note that the information is still required to be consistent for those sub-entities where information is available: Consider a tetrahedron and pick two triangular faces A and B with a common edge. If someone requests vertex ordering information for one of the faces and reduces that information to the edge, the result must be the same no matter whether face A or B was used or whether the ordering was requested directly for the edge itself.

The interface for the class is as follows:

```
struct VertexOrderingInterface {
  // dimension of the entity this applies to
  static const std::size_t dimension;
  // geometry type of the entity this applies to
  const GeometryType type() const;

  // iterate over some sub-entity's vertex indices; must be
  // a RandomAccess iterator, value_type may be constant
  class iterator;

  // get begin iterator for the vertex indices of some
  // sub-entity
  iterator begin(std::size_t codim,
                 std::size_t subEntity) const;
  // get end iterator for the vertex indices of some
  // sub-entity
  iterator begin(std::size_t codim,
                 std::size_t subEntity) const;

  // get reduced vertex ordering for the specified
  // sub-entity
  void getReduced(std::size_t codim, std::size_t subEntity,
                  std::vector<std::size_t>& order) const;
};
```

Information about the dimension and the geometry type is included because it determines the limits for the parameters (via the `GenericReferenceElements`). The `getReduced()` method shall resize the vector passed in the `order` parameter to the suitable size.

D.3. Matching Multiple Dofs on a Common Sub-Entity

Some finite elements have more than one dof on a given sub-entity of an element, and assign a position inside the sub-entity to that dof (i.e. Pk $k \geq 4$, Qk $k \geq 3$). For conforming schemes the ordering of the dofs on a sub-entity shared by two or more elements must match such that the dofs on the same position can be identified.

A similar situation arises with edge elements of order 1.5: For simplices they have three base functions on a face but only two of them are independent. A finite element implementation must make sure to pick the same two base functions for the face in neighbouring elements so their dofs can be identified.

Both issues can be addressed using the information provided by the ordering of the global ids of the vertices.

D.4. Flipping of Base Function Values

Some finite element families, most notably Raviart-Thomas and edge elements, assign an orientation to (some of) their dofs. That is, the value of the dof a^i is interpolated from a function u as the functions value at the dofs position x^i projected onto some unit vector e^i:

$$a^i = u(x^i) \cdot e^i \tag{D.20}$$

The direction of that unit vector is the orientation of the dof. For Raviart-Thomas e^i is the unit vector normal to the face (codimension 1 sub-entity), for edge elements e^i is the unit vector tangential to the edge (dimension 1 sub-entity) on which the dof is located.

To be continuous over element borders, elements connected to a common sub-entity must agree upon a common global orientation for that sub-entity. If their local orientation differs from the global orientation, the `Basis` must multiply the value of the corresponding basis function by -1. The tricky part is the to determine the global orientation for the common sub-entity correctly.

D.4.1. Tangential Orientation for Lines

Lines have two vertices which can be used to choose the orientation of the line: the orientation vector points from the vertex with the lower index/id to the vertex with the higher index/id. That is the geometric interpretation, we actually don't want to compare coordinates, but preferably just integers.

To obtain the local orientation, of an edge in an element, collect the indices of the edges vertices, and here we mean *indices inside the reference element* of the element:

```
unsigned local_orientation[2];
local_orientation[0] = refelem.subEntity(edge_index, dim-1,
                                         0, dim);
local_orientation[1] = refelem.subEntity(edge_index, dim-1,
                                         1, dim);
```

For the global orientation we do basically the same. However, this time we use the vertex ordering information derived from the global ids of the vertices of the element instead of vertex indices inside the reference element:

```
unsigned global_orientation[2];
global_orientation[0] = vertex_order[local_orientation[0]];
global_orientation[1] = vertex_order[local_orientation[1]];
```

The ordering of the index values determines the local and global orientation:

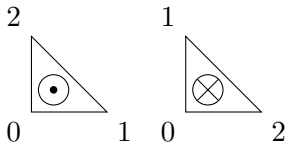
```
if((local_orientation[0] < local_orientation[1])
   == (global_orientation[0] < global_orientation[1]))
{
  // local and global orientation are identical
  // nothing to do
} else {
  // local and global orientation differ
  // flip base function value
}
```

D.4.2. Normal Orientation for Codimension 1 Sub-Entities

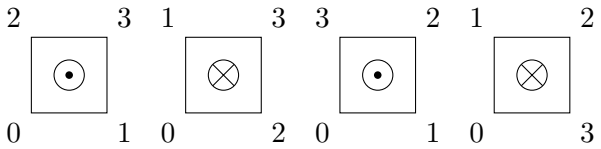
Normal orientation for sub-entities of codimension 1 is important for Raviart-Thomas elements. Normal orientation is more tricky and cannot be done using the ordering of the indices/id of the corners alone. Some additional information is needed, such as the sign of the determinant of the Jacobian of the geometry map $\mathrm{sgn}(\det(\hat{J}_\mu))$. This is however not enough for lower dimensional grids in a higher dimensional world, since then the Jacobian is no longer quadratic and has no determinant.

The reason why the information about the vertex ids is not enough is roughly that to construct the normal orientation there is alway some kind of rotation involved. In 2D the codimension 1 sub-entities are edges. We can obtain a normal orientation by rotating the tangential orientation by 90°. To get a consistent result however, this rotation must be done in the global coordinate system for the global orientation and in the respective local coordinate systems for the local orientations. Locally on the element we have only the local coordinate system available, however. If the geometric transformation μ involves mirroring, then the sense of the rotation will be different for the local and the global coordinate system. The sign of the Jacobian's determinant can tell us whether there is mirroring involved or not.

In 3D the construction of the orientation differs: for triangles we walk through the indices/ids in ascending order and determine the direction of the normal vector by the right-hand rule:



Similar for quadrilaterals, although if their indices/ids are acyclic we just have to pick and orientation (here we chose to ignore the highest index/id and determine the orientation from the remaining indices/ids as for triangles):



This is all rather tedious and in fact there is a much simpler way, which will even work in the case of lower-dimensional grids in a higher-dimensional world. Sub-entities of codimension 1 are always situated between two Elements. Choosing a normal orientation for the sub-entity means to choose a vector that points from one element into the other. The global orientation can thus be chosen by comparing the ids of the elements: it points outward in the element with the lower id and inward in the element with the higher id.

The face orientation should be passed as a bool vector:

```
typedef std::vector<bool> FaceOrientation;
```

The vector is indexed by the index of the face in the reference element. A value of `true` means the global orientation of the face is outward, `false` means it is inward.

D.5. API

The API for global-valued finite elements consists of five interface classes (`BasisInterface`, `InterpolationInterface`, `CoefficientsInterface`, `FiniteElementInterface`, and `FiniteElementFactoryInterface`) and two traits classes (`BasisTraits` and `FiniteElementTraits`). In contrast to the local interface which prefixes all its names with "`Local`" we do not use any prefix here. "`Local`" is already taken, "`Global`" would suggest that this interface is completely in global coordinates, "`GlobalValue`" is too clumsy and adds too much to the lengths of names.

D.5.1. Finite Element Interface

```
struct FiniteElementInterface
{
  // types of component objects
  struct Traits
  {
    typedef ImplementationDefined Basis;
    typedef ImplementationDefined Coefficients;
    typedef ImplementationDefined Interpolation;
  };

  // constructor arguments are implementation specific
  FiniteElementInterface(...);
  // ... except for the copy constructor
  FiniteElementInterface(const FiniteElementInterface&);
```

```
  // extract component objects
  const typename Traits::Basis& basis() const;
  const typename Traits::Coefficients& coefficients() const;
  const typename Traits::Interpolation&
         interpolation() const;
  GeometryType type() const;
};
```

The member class `Traits` may be a member typedef instead. Constructor signatures and existence is implementation-defined, except for the copy constructor, which must be present and publicly accessible. Construction is generally done by a factory class. To keep copy-construction efficient it is recommended that instances of this class are light proxy objects.

The reason to mandate copy-construction is as follows: Up to now with local finite elements `dune-pdelab` used the class `FiniteElementMap` as a kind of finite element factory. If the finite element was required in different variants for a given grid (i.e. because normal continuity was required for Raviar-Thomas elements), the `FiniteElementMap` would store all the variants internally and return a reference to the correct variant upon request. Since global-valued finite elements depend on the geometry of the grid element, this trick is no longer useful, especially if you plan to modify the finite element object by "binding" it to the geometry. The problem is that more than one finite element for different grid elements may be required at the same time (think iterating over the intersections). If the `FiniteElementMap` returns the same variant for both grid elements the user code will first bind the finite element to the inside element and later to the outside element, since both of his finite element references point to the same object. Thus when he tries to access the inside finite element, he will in reality access the outside element.

D.5.2. Finite Element Factory Interface

```
struct FiniteElementFactoryInterface
{
  // may also be an inline class
  typedef ImplementationDefined FiniteElement;

  // construction is implementation-defined
  FiniteElementFactoryInterface(...);

  // finite element object creation
  // arguments are implementation defined
  const FiniteElement make(...);
};
```

The method to create a finite element object is `make()`. The created object is returned by value (`const FiniteElement`). The factory implementation may choose to return by reference instead (`const FiniteElement&`). Because temporaries may be bound to const references in C++, this way code using the factory can always bind the returned

value to a const reference and avoid copy construction if that is not necessary:

```
const Factory::FiniteElement& fe = factory.make();
```

In any case, the returned value or reference must be valid for as long as the factory object exists.

Since each finite element family will need different information to create a finite element object tailored to a particular grid element, the actual argument of the `male()` are implementation-defined. Earlier in this section we have seen different types of information which may be needed to create a tailored finite element: geometry, vertex ordering and face orientation. If they are needed for a given finite element implementation, that finite element should require necessary items in the order given above and in the encoding given earlier. If neither geometry nor vertex ordering is required, but the geometry type is, that should be given in place of geometry and vertex ordering directly. Any extra information should be given after these arguments. The possible signatures for make thus are:

```
make(const Geometry&, const VertexOrder&, const
   FaceOrientation&, ...);
make(const Geometry&, const VertexOrder&, ...);
make(const Geometry&, const FaceOrientation&, ...);
make(const Geometry&, ...);
make(const VertexOrder&, const FaceOrientation&, ...);
make(const VertexOrder&, ...);
make(const GeometryType&, const FaceOrientation&, ...);
make(const GeometryType&, ...);
make(const FaceOrientation&, ...);
make(...);
```

Implementation must document what kind of arguments are required for `make()`.

The constructor signature is implementation-defined.

It is recommended that the factory caches as much information as possible. For instance, for regular hypercube grids the Jacobian of the geometry does not change and is the only thing needed to transform the derivatives. For this case the constructor should take a sample geometry and precompute the transformation. Whether the regular and the general case are distinguished by different constructor arguments to the same factory class, or whether there is one factory class for the regular and one for the general case is left to the implementor of the factory.

D.5.3. Basis Interface

```
struct BasisInterface
{
  struct Traits
  {
    // domain properties (local and global)
    typedef ImplementationDefined DomainField;
```

```
  static const std::size_t dimDomainLocal =
         implementation_defined;
  static const std::size_t dimDomainGlobal =
         implementation_defined;
  typedef ImplementationDefined DomainLocal;
  typedef ImplementationDefined DomainGlobal;

  // range properties (global range only)
  typedef implementation_defined RangeField;
  static const std::size_t dimRange =
         implementation_defined;
  typedef ImplementationDefined Range;

  // jacobian properties (dimRange x dimDomainGlobal
  // Matrix with components of type RangeField)
  typedef ImplementationDefined Jacobian;

  // maximum number of partial derivatives supported
  static const std::size_t diffOrder =
         implementation_defined;
};

// Number of shape functions
std::size_t size () const;
// Polynomial order of the shape functions for quadrature
std::size_t order () const;

// Evaluate all shape functions at given position
void evaluateFunction
( const typename Traits::DomainLocal& in,
  std::vector<typename Traits::Range>& out) const;

// Evaluate jacobian of all shape functions at given
// position; required for Traits::diffOrder >= 1
void evaluateJacobian
( const typename Traits::DomainLocal& in,
  std::vector<typename Traits::Jacobian>& out) const;

// Evaluate derivatives of all shape functions at given
// position;  required for Traits::diffOrder >= 2
void evaluate
( const array<std::size_t,
              Traits::dimGlobalDomain>& directions,
  const typename Traits::DomainLocal& in,
```

```
    std::vector<typename Traits::Range>& out) const;
};
```

The basis interface closely follows the local basis interface with some notable exceptions.

First there are the types in the traits class. Since coordinates are still given in the reference elements coordinate system but derivatives are done with respect to global coordinates, a distinction must be made between local and global domain. The other change is that the member types of the traits class no longer have a suffix "`Type`" since it is quite clear from the camel-case naming convention that they are types.

Second the method for general derivatives `evaluate()` is no longer a template method and its argument `directions` has different semantics. In the local basis interface, `directions` was a list of directions in which to take derivatives, i.e. `directions={0, 1, 0, 2}` for the derivative $\partial_0\partial_1\partial_0\partial_2$. This is inconvenient since it requires `directions` to be a list of variable length, making the length a template parameter, and because the order implied in the above derivative does not really exist, it can just as well be written as $\partial_0^2\partial_1\partial_2$. So in the global-value interface `directions` lists the exponents in the last expression: `direction={2, 1, 1}`. This way the length of `directions` will always be `Traits::dimDomainGlobal` and `evaluate()` no longer needs to be a template.

D.5.4. Interpolation Interface

```
struct InterpolationInterface
{
  // Export basis traits
  typedef BasisInterface::Traits Traits;

  // determine coefficients interpolating a given function
  template<typename F, typename C>
  void interpolate (const F& f, std::vector<C>& out) const;
};
```

The interface for global-value interpolation objects also has little modifications compared to local interpolation objects. Main addition is the member type `Traits` which is the same as in the corresponding basis class. This is to document the parameter types that `interpolate()` will use to evaluate the function `f`.

For the member method `evaluate()` the requirements for the function object `f` change slightly: it is still required to support the expression `f.evaluate(x, y)`, and `x` in that expression is still a local coordinate (though the type is named a little bit different: `const Traits::DomainLocal&`. The difference is that the returned value `y` is now in global coordinates and of the type `Traits::Range`.

D.5.5. Coefficients Interface

```
struct CoefficientsInterface
{
  // number of coefficients
  std::size_t size() const;
```

```
  // get i'th index
  const LocalKey& localKey(std::size_t i) const;
};
```

The interface for the coefficients class is exactly the same as for the local coefficients. If the global-valued finite elements is implemented in term of a local finite element it will often be possible to simply reuse the coefficients class of the local finite element.

If there is some dof-matching required for common sub-entities of neighbouring elements however, and this dof-matching can be done entirely by reordering the dofs on the sub-entity, then the coefficients class is the place to do it.

E. Notation

$\cdot_i$ Lower indices are used to access elements of vectors and matrices, both Cartesian vectors and degree-of-freedom vectors. E.g. $\mathbf{r}_y$, u_i, A_{ij}.

$\cdot^{(i)}$ Upper indices are used to access elements of time series $u^{(i)} = u(t^{(i)})$ and other collections like basis vectors $\hat{\mathbf{e}}^{(i)}$, eigenvalues $\lambda^{(i)}$, eigenvectors $\hat{u}^{(i)}$.

$\mathbf{r}$ Bold is used strictly for Cartesian vectors, e.g. $\mathbf{x}$, $\mathbf{E}$, $\mathbf{H}$, $\hat{\mathbf{n}}$. The six component field vector $u = \begin{pmatrix}\mathbf{E}\\ \mathbf{H}\end{pmatrix}$ is not a Cartesian vector, and is thus not bold.

$\hat{\cdot}$ The circumflex marks unit vectors, whether Cartesian or not: $\hat{\mathbf{e}}^{(x)}$, $\hat{u}^{(i)}$, $\hat{\mathbf{n}}$.

$\mathsf{U} : \mathsf{V}$ Frobenius inner product of $\mathsf{U}, \mathsf{V} \in \mathbb{R}^{m\times n}$. $\mathsf{U} : \mathsf{V} = \sum_{i=1}^{m}\sum_{j=1}^{n} \mathsf{U}_{ij}\mathsf{V}_{ij}$.

$\partial_t x$, $d_t x$ Partial and total derivative, equivalent to $\partial x/\partial t$ and dx/dt, respectively.

∇g, $\nabla_{\mathbf{r}'} g, \operatorname{grad} g, \operatorname{grad}_{\mathbf{r}'} g$ Gradient. When used with a subscript, it denotes the gradient with respect to that variable. When used without a subscript, it denotes the gradient with respect to $\mathbf{r}$. When applied to a vector $g \in \mathbb{R}^m$, $\operatorname{grad} g = \nabla g^T$ or $(\operatorname{grad} g)_{ij} = \partial_i g_j$.

$\nabla \cdot \mathbf{E}$, $\nabla_{\mathbf{r}'} \cdot \mathbf{E}$ Divergence. When used with a subscript, it denotes the divergence with respect to that variable. When used without a subscript, it denotes the divergence with respect to $\mathbf{r}$.

$\operatorname{div} \mathsf{F}$ Divergence of a matrix F. $\operatorname{div} \mathsf{F} = (\nabla \cdot \mathsf{F})^T$, or $(\operatorname{div} \mathsf{F})_j = \sum_i \partial_i \mathsf{F}_{ij}$. $\operatorname{div} \mathsf{F}$ is a column vector.

$\nabla \times \mathbf{E}$, $\nabla_{\mathbf{r}'} \times u$ Curl. When used with a subscript, it denotes the curl with respect to that variable. When used without a subscript, it denotes the curl with respect to $\mathbf{r}$. The curl may also be applied to the combined fields u, then it applies to the component fields individually:

$$\nabla_{\mathbf{r}} \times u = \nabla_{\mathbf{r}} \times \begin{pmatrix}\mathbf{E}\\ \mathbf{H}\end{pmatrix} = \begin{pmatrix}\nabla_{\mathbf{r}} \times \mathbf{E}\\ \nabla_{\mathbf{r}} \times \mathbf{H}\end{pmatrix}$$

$J_u f$ Jacobian of f with respect to u.

$[\![v]\!], [\![v]\!]^{(C)}$ Jump of a scalar or vector expression $v(\mathbf{r})$ across some discontinuity C. Formally, let the domain be separated by some continuous line C into two parts $N^{(l)}$ and $N^{(r)}$. Then

$$[\![v]\!]|_{\mathbf{r}} := \lim_{\substack{\mathbf{r}' \to \mathbf{r} \\ \mathbf{r}' \in N^{(r)}}} v|_{\mathbf{r}'} - \lim_{\substack{\mathbf{r}' \to \mathbf{r} \\ \mathbf{r}' \in N^{(l)}}} v|_{\mathbf{r}'} \qquad \forall \mathbf{r} \in C. \tag{E.1}$$

Note that the sign of the jump changes when the two parts of the domain are switched.

$\hat{\mathbf{e}}^{(x)}, \hat{\mathbf{e}}^{(y)}, \hat{\mathbf{e}}^{(z)}$ Axis-parallel unit vectors of a Cartesian coordinate system.

$\mathsf{C}(\mathbf{a})$, $\hat{\mathsf{C}}$ Left cross product with some vector represented as a matrix: $\mathsf{C}(\mathbf{a})\mathbf{b} = \mathbf{a} \times \mathbf{b}$. $\hat{\mathsf{C}}$ means cross product with the normal vector $\hat{\mathsf{C}} = \mathsf{C}(\hat{\mathbf{n}})$. The matrix $\mathsf{C}(\mathbf{a})$ is antisymmetric.

F. Bibliography

[1] Gene M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities, reprinted from the afips conference proceedings, vol. 30 (atlantic city, nj, apr. 18–20), afips press, reston, va., 1967, pp. 483–485, when dr. amdahl was at international business machines corporation, sunnyvale, california. *Solid-State Circuits Newsletter, IEEE*, 12(3):19–20, 2007.

[2] L. S. Andersen and J. L. Volakis. Hierarchical tangential vector finite elements for tetrahedra. *Microwave and Guided Wave Letters, IEEE*, 8(3):127–129, Mar 1998.

[3] Garth A. Baker. Finite element methods for elliptic equations using nonconforming elements. *Mathematics of Computation*, 31(137):–45, 1977.

[4] J.R. Birchak, C.G. Gardner, J.E. Hipp, and J.M. Victor. High dielectric constant microwave probes for sensing soil moisture. *Proceedings of the IEEE*, 62(1):93, 1974.

[5] W. F. Brown. *Dielektrika*, volume 17 of *Handbuch der Physik*. Springer, Berlin, 1956.

[6] Jens S. Buchner, Alexander Kuhne, Benny Antz, Kurt Roth, and Ute Wollschlager. Observation of volumetric water content and reflector depth with multichannel ground-penetrating radar in an artificial sand volume. In *2011 6th International Workshop on Advanced Ground Penetrating Radar (IWAGPR)*, page 1, 2011.

[7] Jens Stefan Buchner. *Constructive Inversion of Vadose Zone GPR Observations*. PhD thesis, Combined Faculties for the Natural Sciences and for Mathematics of the Ruperto-Carola University of Heidelberg, Heidelberg, Germany, November 2012.

[8] Bernardo Cockburn, Fengyan Li, and Chi-Wang Shu. Locally divergence-free discontinuous galerkin methods for the maxwell equations. *Journal of Computational Physics*, 194(2):588–610, 2004.

[9] Bernardo Cockburn and Chi-Wang Shu. Tvb runge-kutta local projection discontinuous galerkin finite element method for conservation laws ii: General framework. *Mathematics of Computation*, 52(186):–411, 1989.

[10] Bernardo Cockburn and Chi-Wang Shu. The runge–kutta discontinuous galerkin method for conservation laws v: multidimensional systems. *Journal of Computational Physics*, 141(2):199–224, 1998.

[11] Jim Douglas, Jr and Todd Dupont. Interior penalty procedures for elliptic and parabolic galerkin methods. In R. Glowinski and J. L. Lions, editors, *Computing Methods in Applied Sciences*, volume 58, chapter Lecture Notes in Physics, pages 207–216. Springer Berlin Heidelberg, 1976.

[12] Bjorn Engquist and Andrew Majda. Absorbing boundary conditions for the numerical simulation of waves. *Mathematics of Computation*, 31(139):629–651, 1977.

[13] Michael J. Flynn. Some computer organizations and their effectiveness. *Computers, IEEE Transactions on*, 100(9):948–960, 1972.

[14] Holger Gerhards. *Ground Penetrating Radar as a Quantitative Tool with Applications in Soil Hydrology*. PhD thesis, University of Heidelberg, 2008.

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

[16] Marcus J. Grote, Anna Schneebeli, and Dominik Schötzau. Interior penalty discontinuous galerkin method for maxwell's equations: Energy norm error estimates. *Journal of Computational and Applied Mathematics*, 204(2):375–386, 2007. Special Issue: The Seventh International Conference on Mathematical and Numerical Aspects of Waves (WAVES'05).

[17] John L. Gustafson. Reevaluating amdahl's law. *Commun. ACM*, 31(5):532–533, may 1988.

[18] J. S. Hesthaven and T. Warburton. Nodal high-order methods on unstructured grids: I. time-domain solution of maxwell's equations. *Journal of Computational Physics*, 181(1):186–221, 2002.

[19] J. S. Hesthaven and T. Warburton. High-order accurate methods for time-domain electromagnetics. *Computer Modeling in Engineering & Sciences*, 5(5):395–407, 2004.

[20] J.S. Hesthaven and T. Warburon. Discontinuous galerkin methods for the time-domain maxwell's equation. *ACES Newsletter*, 19(1):10–29, 2004.

[21] Marlis Hochbruck and Alexander Ostermann. Exponential integrators. *Acta Numerica*, 19:209, 2010.

[22] John David Jackson. *Klassische Elektrodynamik*. de Gruyter, Berlin, New York, 4$^{\text{th}}$ edition, 2004.

[23] Jianming Jin. *The Finite Element Method in Electromagnetics*. John Wiley & Sons, Inc., New York, 2 edition, 2002.

[24] Sébastien Lambot, Evert C. Slob, Idesbald van den Bosch, Benoit Stockbroeckx, and Marnik Vanclooster. Modeling of ground-penetrating radar for accurate characterization of subsurface electric properties. *IEEE TRANSACTIONS ON GEOSCIENCE AND REMOTE SENSING*, 42(11):2555, 2004.

[25] Patrick Leidenberger. Modeling ground penetrating radar signal propagation in subsoils with dispersive dielectric properties. Diploma thesis, Ruprecht-Karls-Universität Heidelberg, Fakultät für Physik und Astronomie, 2007.

[26] P. Lesaint and Pierre-Arnaud Raviart. On a finite element method for solving the neutron transport equation. In Carl de Boor, editor, *Mathematical aspects of finite elements in partial differential equations*, New York [u.a.], 1974. Acad. Press.

[27] Randall J LeVeque. Finite-volume methods for non-linear elasticity in heterogeneous media. *International journal for numerical methods in fluids*, 40(1-2):93–104, 2002.

[28] Zheng Lou and Jian-Ming Jin. A novel dual-field time-domain finite-element domain-decomposition method for computational electromagnetics. *IEEE Transactions on Antennas and Propagation*, 54(6):1850–1862, June 2006.

[29] J. Clerk Maxwell. A dynamical theory of the electromagnetic field. *Philosophical Transactions of the Royal Society of London*, 155:459–512, 1865.

[30] Alireza H. Mohammadian, Vijaya Shankar, and William F. Hall. Computation of electromagnetic scattering and radiation using a time-domain finite-volume discretization procedure. *Computer Physics Communications*, 68(1–3):175–196, 1991.

[31] J.C. Nedelec. Mixed finite elements in $\mathbb{R}^3$. *Numerische Mathematik*, 35(3):315–341, 1980.

[32] Ardavan F. Oskooi, David Roundy, Mihai Ibanescu, Peter Bermel, J. D. Joannopoulos, and Steven G. Johnson. Meep: A flexible free-software package for electromagnetic simulations by the fdtd method. *Computer Physics Communications*, 181:687–702, January 2010.

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

[34] Michael Renardy and Robert C Rogers. *An introduction to partial differential equations*, volume 13. Springer, 2004.

[35] K. Roth. Soil physics lecture notes. Institute of Environmental Physics, Heidelberg University, D-69120 Heidelberg, Germany, 2012. V2.2.

[36] K. Roth, R. Schulin, H. Flühler, and W. Attinger. Calibration of time domain reflectometry for water content measurement using a composite dielectric approach. *Water Resour. Res.*, 26(10):2267–2273, 1990.

[37] J. P. Webb. Edge elements and what they can do for you. *Magnetics, IEEE Transactions on*, 29(2):1460–1465, 1993.

[38] Mary Fanett Wheeler. An elliptic collocation-finite element method with interior penalties. *SIAM Journal on Numerical Analysis*, 15(1):–152, 1978.

[39] Hassler Whitney. *Geometric integration theory.* Number 21 in Princeton mathematical series ; 21 ; Princeton mathematical series. Univ. Pr., Princeton, N.J., 1957.

[40] Kane Yee. Numerical solution of initial boundary value problems involving maxwell's equations in isotropic media. *Antennas and Propagation, IEEE Transactions on*, 14(3):302–307, 1966.

Thanks

There are many people (and institutions) without whom this would have been impossible. The deserve my greatest thanks.

Peter Bastian, he gave me a chance to work on this topic, and he is the advisor for this thesis with all that entails.

My co-advisor Kurt Roth. His group built the GPR testbed and provided insight into GPR operation. I enjoyed the discussions with you very much.

Benedikt Oswald, whom I had many highly valuable discussions with. He also provided the mesh for the sphere and shared some simulation results from his HADES3D software I could check my own code against. And he invited me to visit the Paul Scherrer Institute.

Olaf Ippisch, who gave most valuable advice, and was always available for discussion.

The poeple from Kurt Roth's group. Patrik Klenk and in particular Jens Buchner, for many fruitful discussions. Ute Wollschläger, also for discussions, but more importantly, she showed me the GPR antennas.

Felix Heimann, who could always answer questions about GPR.

Christina Seegers and Florian Bogda, then students in Kurt Roth's group, who conducted the real-world measurements.

Christian Engwer, for discussions, his patience, and his prodding.

The virtual institute INVEST of the Helmholtz Association and the Deutsche Forschungsgemeinschaft (DFG) through project RO 1080/8-2 for financial support.

There are also people who I'd like to thank more generally. Their contribution was more indirect, but cannot be neglected nevertheless.

Marina Seikel, for showing me light sabers and sharks, for giving me the most useful tools ever, and for generally being cute.

My parents, Henry and Hilfe Fahlke. They are my parents, duh, there is far too much to mention.

Markus Demleitner and Sven Marnach, for financial support.

Andreas Buhr, for a shift in perspective.

The guys from the Peter Bastian's group: Peter Bastian, Olaf Ippisch, Ole Klein, Adrian Ngo, Felix Heimann, Jurgis Pods, Pavel Hron, Dan Popović, Panos Drouvelis, Valeria Malieva, Steffen Müthing, Sven Marnach, Christian Engwer, Markus Blatt, Ansgar Burchardt, Rebecca Neumann, Johannes Schönke, for being agreeable colleagues.

Peter Bastian, Markus Blatt, Andreas Dedner, Christian Engwer, Robert Klöfkorn, Martin Nolte, Mario Ohlberger and Oliver Sander, who initiated Dune. Carsten Gräser, Christoph Gersbacher and Steffe Müthing, who later joined the team. And of course the innumerable people who contributed to Dune.

Linus Torvalds, and many contributors, for Linux. RMS, and many contributors, for free software.