University of West Bohemia in Pilsen
Faculty of Electrical Engineering

Doctoral thesis

Large-scale Numerical Simulations of Magneto-Hydrodynamics Phenomena in Astrophysics

Mgr. Lukáš Korous

2018

Abstract

The objective of this Doctoral Thesis was to develop, implement and test new algorithms for the large-scale solution of nonstationary compressible MHD equations based on higher-order discontinuous Galerkin (DG) methods. The basis for the new methods will be the discontinuous Galerkin methods and adaptive mesh refinement (AMR) algorithms. The new algorithms will be implemented and tested in the framework of the open source library deal.II, and they will be applied to selected problems of MHD in astrophysics, namely the magnetic reconnection phenomena.

Keywords

numerical simulation, finite element method, MHD equations, adaptivity, discontinuous Galerkin method, astrophysics, solar flares, AMR, distributed computing

Abstract [CZ]

Záměrem této práce je navrhnout, implementovat a otestovat nové algoritmy pro rozsáhlé simulace nestacionárních jevů spadajících do oblasti stlačitelné magnetohydrodynamiky. Vytvořený software bude založen na použití nespojité Galerkinovy metody (discontinuous Galerkin, DG) s vyššími řády přesnosti. Zároveň bude použita metoda automatického zjemňování výpočetní triangulace (automatic mesh refinement, AMR). Vytvořené algoritmy budou testovány ve frameworku deal.II a budou aplikovány na skutečné problémy v astrofyzice, jmenovitě na simulaci jevu magnetické rekonexe.

Keywords [CZ]

numerická simulace, metoda konečných prvků, MHD rovnice, adaptivní algoritmy, nespojitá Galerkinova metoda, astrofyzika, sluneční erupce, AMR, distribuované výpočty

Acknowledgement

I hereby acknowledge and thank for the lead and support of my supervisor, doc. Ing. Pavel Karban, Ph.D., as well as my advisor prof. Ing. Ivo Doležel, CSc. I would also like to thank Ing. Jan Skála, Ph.D. and Mgr. Miroslav Bárta, Ph.D. to give me valuable insight into the problems of astrophysics, and I also hereby appreciate the work of the entire deal.II development team for their professional approach to software development and support.

Statement

I am presenting this doctoral thesis, created during my doctoral studies at the Faculty of Electrical Engineering of the University of West Bohemia.

I confirm having prepared this work by my own, and having listed all used sources of information in the bibliography. All license conditions of all works of software and other nature were respected.

In Pilsen, August 7, 2018, Lukáš Korous

Contents

1 Introduction **1**
- 1.1 Magnetohydrodynamics in Astrophysics . . . 3
  - 1.1.1 Magnetic flux tube model of plasma . . . 4
  - 1.1.2 Magnetic reconnection and other phenomena . . . 7
- 1.2 Aim of this work . . . 8
- 1.3 State of the art . . . 9

2 Mathematical model **11**
- 2.1 Derivation of the mathematical model . . . 11
  - 2.1.1 Initial setup . . . 11
  - 2.1.2 Euler's equations of compressible flow . . . 12
  - 2.1.3 Maxwell's equations of electromagnetism . . . 12
  - 2.1.4 Derived relations between electromagnetic quantities . . . 13
  - 2.1.5 Simplifying assumptions . . . 14
  - 2.1.6 Adding the induction equation . . . 15
  - 2.1.7 Conservative form of the MHD equations . . . 15
  - 2.1.8 Solution considerations . . . 16
- 2.2 Weak formulation of the problem . . . 17
- 2.3 Boundary conditions . . . 18
  - 2.3.1 Essential (inflow) boundary conditions . . . 18
  - 2.3.2 Outflow (do-nothing) boundary conditions . . . 18
  - 2.3.3 Periodic boundary conditions . . . 18

3 Numerical approach **19**
- 3.1 Triangulation . . . 19
  - 3.1.1 Distributed triangulation . . . 21
- 3.2 Discontinuous Galerkin method . . . 22
  - 3.2.1 Overview of the DG method . . . 22
  - 3.2.2 DG formulation of MHD equations . . . 23
  - 3.2.3 Numerical flux . . . 25
  - 3.2.4 Numerical handling of boundary conditions . . . 32
- 3.3 Divergence-free FE space . . . 34
- 3.4 Discretization in time . . . 36
  - 3.4.1 Discrete problem . . . 36
  - 3.4.2 Time step length . . . 37
- 3.5 Algebraic formulation . . . 37
- 3.6 Numerical integration . . . 39

3.7 Assembling the algebraic problem . . . 41
3.8 Slope limiting . . . 43
3.8.1 Vertex-based limiter . . . 45
3.9 Time-stepping and linearization . . . 49
3.9.1 Time-stepping . . . 49
3.10 Performance considerations . . . 50
3.10.1 Parallelization . . . 50
3.10.2 Vectorization . . . 50
3.10.3 Distribution . . . 51

4 Adaptive Mesh Refinement **53**
4.1 Overview of the AMR . . . 53
4.2 Adaptive-mesh refinement and DG . . . 57
4.2.1 Periodic boundary conditions . . . 59
4.2.2 Relationship with slope limiters . . . 60
4.3 Reference solution approach . . . 61
4.3.1 Algorithm . . . 62
4.3.2 Implementation notes . . . 65

5 Results **66**
5.1 Benchmarks . . . 66
5.1.1 Hardware specification . . . 66
5.1.2 MHD Blast . . . 67
5.1.3 Orszag-Tang vortex . . . 87
5.2 Flux tube eruption model . . . 92
5.2.1 Problem parameters . . . 92
5.2.2 Initial condition . . . 93
5.2.3 Boundary conditions . . . 94

6 Conclusion, outlook **95**
6.1 Outloook . . . 96

Bibliography **96**

1 Introduction

Notation

Overview of notation used in the text is given in table 1.1.

Symbol	Meaning
a	a scalar quantity "a"
$\boldsymbol{a}$	a 3-element vector "a"
a_i	the i-th component of a vector $\boldsymbol{a}$
$\mathbf{A}$	an 8-element vector "A"
ρ	Density
$\boldsymbol{\pi}$	Momentum
μ	Permeability
μ_0	Permeability of vacuum
t	Time variable
$\boldsymbol{x} = (x, y, z)$	Space variable
p	Pressure
$\boldsymbol{u}$	Velocity
u	Magnitude of velocity, i.e. $\lvert\boldsymbol{u}\rvert$
$\boldsymbol{B}$	Magnetic flux density
$\boldsymbol{J}$	Current density
B	Magnitude of magnetic flux density, i.e. $\lvert\boldsymbol{B}\rvert$
$\boldsymbol{E}$	Electric field
e	Internal energy density
c	Speed of light
$\boldsymbol{g}$	Gravitational acceleration
σ	Conductivity, $\sigma = \frac{1}{\eta}$
η	Electrical restisistivity, $\eta = \frac{1}{\sigma}$
c_v	Specific heat at constant volume
c_p	Specific heat at constant pressure
γ	Poisson adiabatic constant
θ	Absolute temperature
$\boldsymbol{q}$	Heat flux
q	Electric charge
$\boldsymbol{f}$	Density of force acting on fluid
$\boldsymbol{F}_L$	Lorentz force, $\boldsymbol{F}_L = q\left(\boldsymbol{E} + \boldsymbol{u} \times \boldsymbol{B}\right)$
$\boldsymbol{f}_L$	Lorentz force density, $\boldsymbol{f}_L = \rho_q \boldsymbol{E} + \boldsymbol{J} \times \boldsymbol{B}$
ε	Electrical permittivity
ε_0	Electrical permittivity of vacuum
ρ_q	Charge density
U_k	Kinetic energy
U_m	Magnetic energy
U	Total energy, $U = U_k + U_m + \rho e$
p_T	Total pressure
$\tilde{U}$	Hydrodynamic energy, $\tilde{U} = U - U_m$
$\boldsymbol{n}$	Unit outer normal

Table 1.1: Notation

Here

$$U_m(\boldsymbol{x},t) = \frac{1}{2}\sum_{i=1}^{3} \boldsymbol{B}_i(\boldsymbol{x},t) \tag{1.1}$$

is the magnetic energy,

$$U_k(\boldsymbol{x},t) = \frac{1}{2}\sum_{i=1}^{3} \boldsymbol{u}_i(\boldsymbol{x},t) \tag{1.2}$$

is the kinetic energy, and the relationship between the Total energy U and pressure p reads

$$U(\boldsymbol{x},t) = \frac{p(\boldsymbol{x},t)}{\gamma-1} + U_m(\boldsymbol{x},t) + U_k(\boldsymbol{x},t). \tag{1.3}$$

Moreover, the total pressure p_T is defined as

$$p_T(\boldsymbol{x},t) = p(\boldsymbol{x},t) + U_m(\boldsymbol{x},t). \tag{1.4}$$

Note that in the text, subscripts denoting spatial dimension $1,2,3$, and x,y,z will be used interchangeably.

The term magnetohydrodynamics (MHD) covers all physical phenomena that involve both electromagnetic (EM) field and a fluid that carries the EM field. Such phenomena are very interesting, yet very complex to study. The behavior of such a fluid is utilized in some industrial applications - liquid-metal cooling of nuclear reactors, magnetic fluid in dampers, sensors for precise measuring of angular velocities, etc. Such phenomena occur in nature as well - the most significant of which are definitely the processes that take place inside and on the surface of stars - which is the topic of the next section.

1.1 Magnetohydrodynamics in Astrophysics

There are several phenomena in the universe that we can look at as magnetohydrodynamic in nature - planets consisting of metals, interplanetary space, but mainly - stars. If we talk about the nearest star - and the only one we are able to study well enough - the Sun - these phenomena include those that occur in the Sun's photosphere (the layer of Sun that is visible): Sun spots, but also phenomena that occur above the Sun (further from the center of the Sun): in Sun's chromosphere, or even corona (solar flares) - even phenomena that originate from the Sun, but then spread through our solar system - solar winds, space weather. All these phenomena have a large impact on the lives of all of us. For example solar flares (that are often followed by ejection of mass out of the Sun - the so called coronal mass ejections - CMEs) have impact on the Earth's magnetic field which in turn has impact on the electronic communication down on Earth (because the communication satellites used for transmissions may be damaged by the disturbances in the magnetic field). Also people operating at high altitudes, both in airplanes and manned space missions are exposed to the energetic particles coming from the Sun (this term is

sometimes called *cosmic rays*). For all the above reasons, it is of great importance to understand the phenomena of space weather, and other MHD phenomena that occur in space.

1.1.1 Magnetic flux tube model of plasma

We are interested in the process of evolution of the flux tube eruption. Such an eruption was captured during observations by Kotrč et al. (2012), and the captured image is shown in figure 1.1 taken from Kotrč et al. (2012) (Figure 2 in the article).

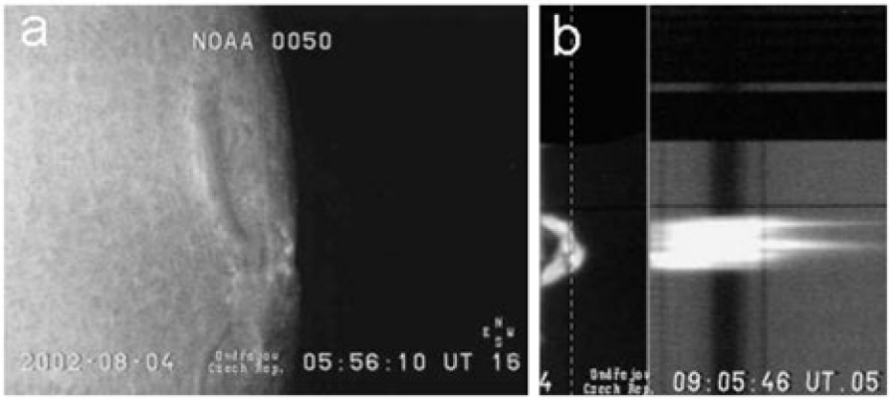


Figure 1.1: Observation of a limb event. $H\alpha$ slit-jaw is the middle (b) part, taken from Kotrč et al. (2012).

For the modeling purposes, we are interested in the so-called $H\alpha$ slit-jaw which is the side-view of the magnetic flux tube during its evolution.

In order to model this phenomena physically, and geometrically, we utilize the magnetic field model by Titov and Demoulin (Titov and Demoulin (1999)) which describes a twisted flux tube as part of a torus with minor radius a (being the radius of the tube, not shown in figure 1.2) and major radius R submerged below the photosphere of the Sun by $d < R$, oriented as in figure 1.2 (taken from Titov and Demoulin (1999)) with total current I.

The magnetic configuration is kept in a global equilibrium by the action of the Lorentz force due to the overlying magnetic field. The sources of this ambient field are modeled by a sub-photospheric line current I_0 and a pair of magnetic charges $+q, -q$ located at distance L from the center of the torus, all located at the major axis of the torus.

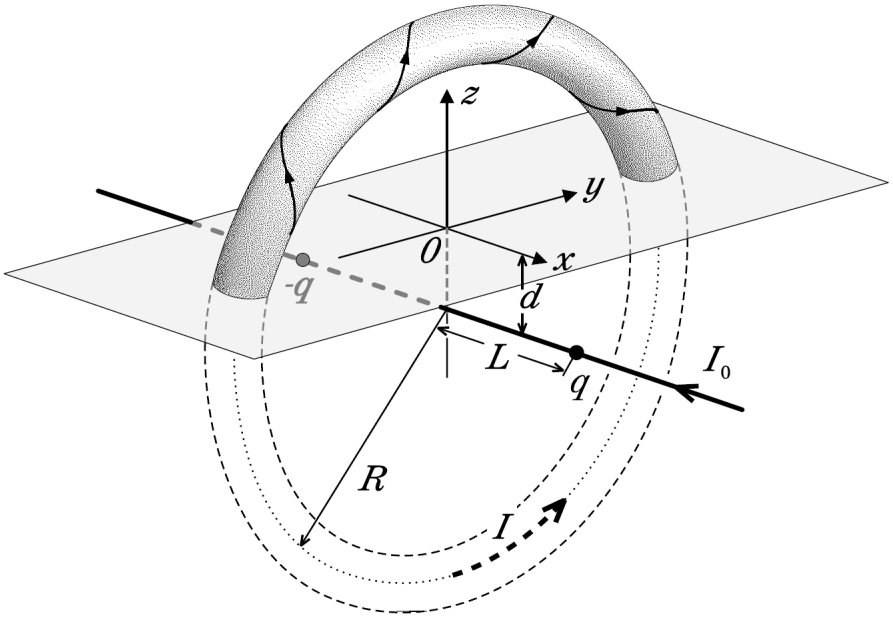


Figure 1.2: The magnetic field under study, taken from Titov and Demoulin (1999).

This geometrical/physical model needs to be properly modeled mathematically (including proper boundary conditions), then approached numerically, and finally computed using software that - in order to achieve reasonable accuracy of the solution - needs to be precisely implemented, must utilize approaches common in the high performance computing (HPC) field, and must be heavily optimized.

In the article Kotrč et al. (2012) which is a reference paper for this work, the model was set up according to observations, and numerical approach of Finite Difference Method (FDM) was used. The results obtained there are presented in figures 1.3 and 1.4, and in order to compare them with figure 1.1, also in the form in figure 1.5 - where the colors are mapped to black and white to be comparable with the observation.

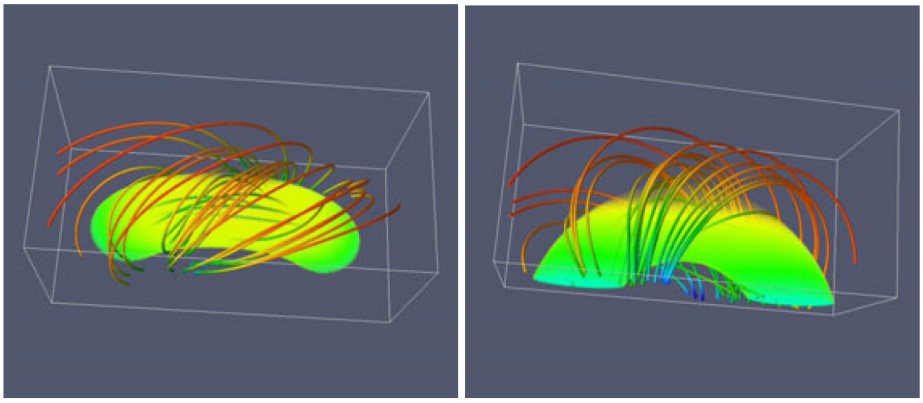


Figure 1.3: Results from Kotrč et al. (2012), initial state, density volume and magnetic field isolines - top & side view.

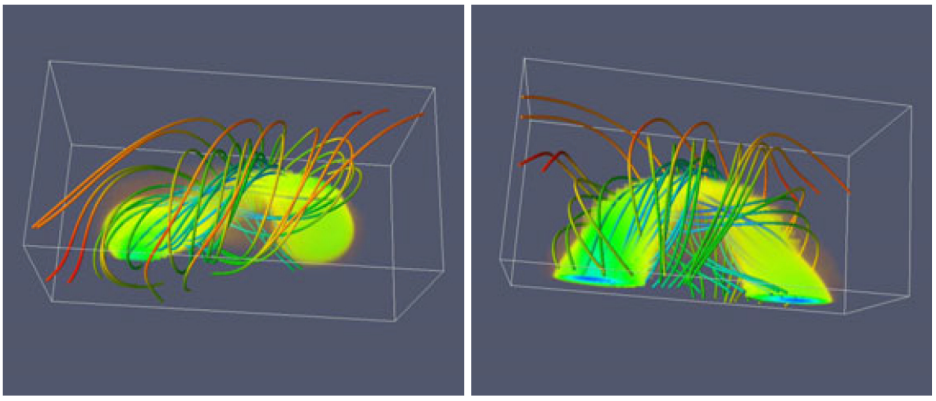


Figure 1.4: Results from Kotrč et al. (2012), state at $t = 14$, density volume and magnetic field isolines - top & side view.

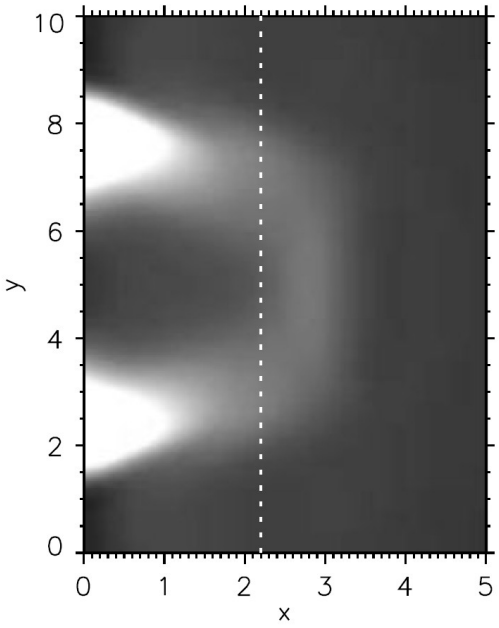


Figure 1.5: Computed approximation of the observed event (figure 1.1), presented in Kotrč et al. (2012).

The aim of this work is to be able to compute results of such problems with much higher resolution, and more importantly to implement a generic solver which can handle any similar problem with arbitrary geometric and physical parameters.

1.1.2 Magnetic reconnection and other phenomena

An additional topic, interesting from the astrophysical point of view and related to magnetohydrodynamics, is the magnetic reconnection.

Magnetic reconnection occurs within electrically charged gases called plasmas. These charged particles interact strongly with the magnetic field, but at the same time their motions modify the magnetic field. Under normal conditions, the magnetic field lines inside plasmas don't break or merge with other field lines. But sometimes, as field lines get close to each other, the entire pattern changes and everything realign into a new configuration. The amount of energy released can be formidable. Magnetic reconnection taps into the stored energy of the magnetic field, converting it into heat and kinetic energy that sends particles streaming out along the field lines. Solar flares, which are among the phenomena which are the most important to study, are driven by magnetic reconnection - and thus studying magnetic reconnection is of great importance.

From the numerical perspective, to investigate such phenomena is to bring the complexity of multiple scales present in the physical world - magnetic reconnection occurs at substantially different scales than e.g. the solar flares. To handle this numerically with adequate resolution, and reasonable computational costs, the implementation must be able to limit the number of degrees of freedom used for the discretization to such that yield the largest accuracy increase - namely, in this work, this is achieved via a state-of-the-art Adaptive Mesh Refinement approach.

1.2 Aim of this work

Main goal that was set for this work is implementation of a software package, capable of numerically solving the Magnetohydrodynamic equations with the following attributes:

- The code must implement mathematically correct and clean methods, preferably with no parameter-dependent algorithms
- The code must allow for a broad spectrum of solution attributes occuring (shocks, oscillations, high energies, ...)
- The code must be able to capture very fine details of the solution through higher resolution, in reasonable computing time
- The code must be able to run a large-scale simulations, for which a single computer is not sufficient, and utilization of modern distributed computing approach is a must
- The code must be easy to use for the physicists, must be written using industry-standard modern object-oriented programming language

In order to achieve this, the work is divided into several logically successive steps. First, the mathematical model is clearly constructed, and its formulation translated into a form suitable for the chosen numerical method (the Discontinuous Galerkin method) of solving a system of PDEs - which was chosen in order to deliver the first and second of the above requirement (handling sharp fronts, discontinuities and the like with mathematical cleanliness) - is described afterwards.

The Discontinuous Galerkin method (DG) is then described in detail, in the entire process from the integral equations down to algorithms performing basic operations. Pitfalls of the numerical solution of the MHD equations (divergence constraint, slope limiting), and chosen way of solving them is given next, while still keeping the above requirements in mind.

Lastly, the biggest challenge is the satisfaction of the requirement of both the fine resolution of the solution, and the capability of running large-scale simulations on modern distributed computing architecture - while still providing all the other attributes of the implemented code. The approach to this, which involves mainly the Adaptive Mesh Refinement technique (AMR) together with Domain Decomposition technique, is presented after. And finally the verifications, and benchmarks and actual usage on an astrophysical problem are presented.

1.3 State of the art

Only recently, the scientific computation community, due to the advances in computer and supercomputer capabilities, has started with non-trivial numerical simulations of such complex physical phenomena that the MHD model describes. Since both for industrial applications, and obviously for astrophysical application of the MHD model, it is quite expensive (or downright impossible) to perform any experiments, the benefit of being able to simulate the phenomena on a computer is very large.
There exist several available numerical simulation codes, such as (Stone et al., 2008a), (Norman et al., 1992), (Kestener et al., 1992), (Skála, J. et al., 2015). These codes have been successfully applied to a range of problems in astrophysics.
There are many numerical methods implemented in these codes, such as the finite difference method ((Skála, J. et al., 2015)), finite volume method ((Kestener et al., 1992)), and the (continous) finite element method ((Skala and Barta, 2012)).

There have been some attempts to employ also the discontinuous Galerkin method ((Rossmanith, 2013), (Mocz et al., 2014)), but so far no open-source generic software employing this method is available. Moreover, as the astrophysical interest lies in multi-scale problem, a software that can handle such problems would be much more beneficial. An approach that can achieve this capability of solving multi-scale problems is the Adaptive Mesh Refinement technique (AMR). What we understand under this term is not only a mesh refinement that is local (i.e. non-uniform), but a mesh refinement that does not originate in the problem description, and neither is invoked programatically with user input. The term 'adaptivity' means that through a predefined *refinement indicator*, which is a function operating on the set of elements of the triangulation, elements to be refined are chosen automatically. This *refinement indicator* is calculated from the solution, and in effect makes the triangulation 'adapt' to the solution - hence, AMR.

The reason for the development of a new code is two-fold. First, there is a unique collaboration between the Astronomical Institute of the Czech Academy of Sciences and the University of West Bohemia, where astrophysicists work together with electrical engineers (from theoretical and numerical modeling backgrounds), and the developed code will be usable for both simulating of astrophysical MHD phenomena, and industrial MHD applications.

Second, the newly developed code is based on locally-adaptive Discontinuous Galerkin method, which yields several advantages over the existing codes (which use e.g. finite difference, or finite volumes methods) developed at institutions of such high quality as *Princeton* - (Stone et al., 2008a), (Norman et al., 1992). The advantages are especially of performance, and automation nature - method of higher order together with AMR yields results qualitatively and quantitatively comparable to low order uniform mesh methods, but with computational cost that can easily be an order of magnitude smaller. Automation is mentioned here related to the AMR, which, without user interaction, can optimize the computational triangulation for a particular time instance in the evolution of the modeled phenomena.

Another benefit (namely over (Norman et al., 1992))of the newly created software are the use of modern object-oriented programming techniques and experience gained on creating finite element software ((Solin et al., 2014), (Ma et al., 2012), (Korous and Solin, 2012)). The implementation related to this work is written in the C++ language, with the use of existing software packages that are proven, and used by a wide community of researchers all over the world - deal.II ((Bangerth et al., 2015)), Trilinos (Heroux et al. (2005)), P4EST (Burstedde et al. (2011)), Intel Parallel Studio (Intel Corporation (2017)), UMFPACK ((Davis, 2006)), Paraview(Kitware Inc. (2017)), and others.

The state of the art of numerical simulation of magnetohydrodynamics can be summarized as a state when the mathematical theory of the equations is quite solid, but the methods to solve the equations numerically in the most optimal and fast way are still being improved. The numerical solution is not merely about theoretical convergence rates and attributes of the particular method, but also the actual implementation plays an important role - i.e. programming, hardware, and software, and execution, both before, during, and very importantly after the actual method invocation (of the so-called *postprocessing* of results). In all aspects of implementation, there is space for new approaches, new ideas, new milestones, that can expand the capabilities of today's numerical solution of magnetohydrodynamics phenomena.

2 Mathematical model

In this chapter, the mathematical model will be derived from basic equations governing the studied physical phenomena, its weak formulation will be presented, and the complete mathematical problem which is solved will be described.

2.1 Derivation of the mathematical model

In order to derive the mathematical model, we first need to establish the basics - time-space frame (section 2.1.1), assumptions (section 2.1.1), then we will start with the known Euler equations of compressible flow (section 2.1.2, derivation of which is given e.g. in Korous (2012)), and by adding the Maxwell's equations (section 2.1.3), we will derive the mathematical model for Magnetyhodrodynamics in the complexity needed for handling problems presented in the introduction.

2.1.1 Initial setup

We consider a time interval $(0, T)$ and space domain $\Omega_t \subset \mathbb{R}^3$ occupied by a fluid at time t. By $\mathcal{M}$ we denote the space-time domain in consideration:

$$\mathcal{M} = \{(\boldsymbol{x}, t)\,; \boldsymbol{x} \in \Omega_t, t \in (0, T)\}\,. \tag{2.1}$$

Moreover we assume that $\mathcal{M}$ is an open set.

Assumptions

When dealing with MHD phenomena in plasma, the following rules apply

- We assume that the fluid is inviscid.
- We assume that the fluid is compressible.
- We consider only the so-called *perfect gas* or *ideal gas* whose state variables satisfy the following *equation of state*

  $$p = R\theta\rho, \tag{2.2}$$

  where ρ denotes density, θ denotes the absolute temperature, and R is the *gas constant*, which is defined as

  $$R = c_p - c_v. \tag{2.3}$$

In the above, c_p, c_v are specific heats at constant pressure, and at constant volume.

2.1.2 Euler's equations of compressible flow

This system of equation reads

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\boldsymbol{\pi}) = 0 \tag{2.4}$$

$$\frac{\partial \boldsymbol{\pi}}{\partial t} + \nabla \cdot (\boldsymbol{\pi} \otimes \boldsymbol{u}) = \rho \boldsymbol{f} - \nabla p, \tag{2.5}$$

$$\frac{\partial \tilde{U}}{\partial t} + \nabla \cdot \left(\tilde{U}\boldsymbol{u}\right) = \rho \boldsymbol{f} \cdot \boldsymbol{u} - \nabla \cdot (p\boldsymbol{u}) + \nabla \cdot \boldsymbol{q}, \tag{2.6}$$

where $\boldsymbol{\pi}$ is momentum, p pressure, U total energy. Moreover, $\boldsymbol{u}$ denotes velocity, $\boldsymbol{f}$ density of the force acting on the fluid, and $\boldsymbol{q}$ is the heat flux. By $\otimes$, we denote the *tensor product*:

$$\boldsymbol{a} \otimes \boldsymbol{b} = \begin{pmatrix} a_1 b_1 & a_1 b_2 & a_1 b_3 \\ a_2 b_1 & a_2 b_2 & a_2 b_3 \\ a_3 b_1 & a_3 b_2 & a_3 b_3 \end{pmatrix}.$$

Moreover, the following relations hold:

$$\tilde{U} = \rho e + U_k, \tag{2.7}$$

$$p = (\gamma - 1)\left(\tilde{U} - U_k\right), \tag{2.8}$$

$$\theta = \left(\tilde{U}/\rho - |\boldsymbol{u}|^2/2\right)/c_v. \tag{2.9}$$

This system is simply called the *compressible Euler equations* for a heat-conductive perfect gas. The individual equations are called the *continuity equation*(equation (2.4)), the *Navier-Stokes equations* (equation (2.5)), and the *energy equation* (equation (2.6)).

For the force density f we assume that only the Lorentz force and gravity act upon the fluid:

$$\boldsymbol{f} = \boldsymbol{f}_L + \boldsymbol{g}.$$

2.1.3 Maxwell's equations of electromagnetism

In this work, we use Maxwell's equations with the assumption of constant electrical permittivity, and constant permeability. We take the equations describing the

electromagnetic field in vacuum. This system of equation reads

$$\nabla \times \boldsymbol{B} = \mu_0 \left(\boldsymbol{J} + \varepsilon_0 \frac{\partial \boldsymbol{E}}{\partial t} \right) \tag{2.10}$$

$$\nabla \times \boldsymbol{E} = -\frac{\partial \boldsymbol{B}}{\partial t} \tag{2.11}$$

$$\nabla \cdot \boldsymbol{E} = \frac{\rho_q}{\varepsilon_0} \tag{2.12}$$

$$\nabla \cdot \boldsymbol{B} = 0, \tag{2.13}$$

where $\boldsymbol{B}$ denotes magnetic flux density, $\boldsymbol{E}$ denotes electric field, $\boldsymbol{J}$ denotes current density, ε_0 is permittivity of vacuum, and ρ_q is electric charge density. The individual equations are known as Faraday's law (equation (2.11)), Ampere's law (equation (2.10)), and Gauss's laws (equations (2.12) and (2.13)).

2.1.4 Derived relations between electromagnetic quantities

Further relations that are useful when deriving the MHD equations are:

$$\frac{\mathrm{d}}{\mathrm{d}t} U_m = \frac{1}{\mu_0} \nabla \cdot (\boldsymbol{B} \times \boldsymbol{E}) - \boldsymbol{E} \cdot \boldsymbol{J}, \tag{2.14}$$

$$\boldsymbol{E} = -u \times \boldsymbol{B} + \frac{\eta}{\mu_0} \boldsymbol{J}, \tag{2.15}$$

$$\frac{\partial \boldsymbol{B}}{\partial t} + \nabla \times (\boldsymbol{B} \times \boldsymbol{u}) = -\frac{1}{\mu_0} \nabla \times (\eta \nabla \times \boldsymbol{B}), \tag{2.16}$$

where μ_0 denotes permeability of vacuum, η is resistivity, and U_m is magnetic energy. The equation (2.15) is the differential form of the *Ohm's law*, the equation (2.16) is the *induction equation*.

Applying now section 2.1.2 to equations (2.5) and (2.6), we obtain:

$$\frac{\partial \boldsymbol{\pi}}{\partial t} + \nabla \cdot (\boldsymbol{\pi} \otimes \boldsymbol{u}) = \rho_q \boldsymbol{E} + \boldsymbol{J} \times \boldsymbol{B} + \rho \boldsymbol{g} - \nabla p, \tag{2.17}$$

$$\frac{\partial \tilde{U}}{\partial t} + \nabla \cdot \left(\tilde{U} \boldsymbol{u} \right) = \rho_q \boldsymbol{E} \cdot \boldsymbol{u} + \boldsymbol{J} \times \boldsymbol{B} \cdot \boldsymbol{u} + \rho \boldsymbol{g} \cdot \boldsymbol{u} - \nabla \cdot (p \boldsymbol{u}) + \nabla \cdot \boldsymbol{q}. \tag{2.18}$$

The adjusted energy equation (equation (2.18)) does not include the magnetic energy U_m, which we do want to include in the MHD equations. To achieve this, we employ equation (2.14) - equation (2.16) and rearrange. After rearranging we obtain

$$\frac{\partial U}{\partial t} = -\nabla \cdot \left[\boldsymbol{\pi} \left(\frac{u^2}{2} + e \right) + p \boldsymbol{u} - \frac{1}{\mu_0} \boldsymbol{B} \times \boldsymbol{E} \right] + \rho \boldsymbol{g} \cdot \boldsymbol{u} + \nabla \cdot \boldsymbol{q}. \tag{2.19}$$

2.1.5 Simplifying assumptions

In what follows, we will make several simplifying assumptions, according to which we will get a system of equations that will adequately respect the physical model, yet will be easier to be solved.

Negligible time derivative of electric field

For the time increments that we are concerned with, the time derivative in equation (2.10) (the so-called *Maxwell's displacement current*) is very small. To estimate the minimum time increment value τ which would allow us to neglect the derivative, take the ratio of the two terms on the right hand side of equation (2.10):

$$\varepsilon_0 \frac{\frac{\partial \boldsymbol{E}}{\partial t}}{\boldsymbol{J}} \approx \frac{\frac{\varepsilon_0 \boldsymbol{E}}{\tau}}{\sigma \boldsymbol{E}} \approx \frac{\varepsilon_0}{\sigma \tau} \approx \frac{10^{-11}}{\tau}. \tag{2.20}$$

This means for time scales much greater than 10^{-11} seconds, the time derivative of $\boldsymbol{E}$ can be neglected. As a consequence, equation (2.10) can be written as:

$$\nabla \times \boldsymbol{B} = \mu_0 \boldsymbol{J}. \tag{2.21}$$

Using equation (2.21), we can write

$$\mu_0 \boldsymbol{J} \times \boldsymbol{B} = (\boldsymbol{B} \cdot \nabla)\, \boldsymbol{B} - \nabla \frac{B^2}{2} = \nabla \cdot (\boldsymbol{B}\boldsymbol{B}) - \nabla \frac{B^2}{2}, \tag{2.22}$$

where the last equality comes from equation (2.13). And using equation (2.22) we can rewrite equation (2.17) as

$$\frac{\partial \boldsymbol{\pi}}{\partial t} + \nabla \cdot (\boldsymbol{\pi} \otimes \boldsymbol{u}) = q\boldsymbol{E} + \nabla \cdot \left(\frac{1}{\mu_0} \boldsymbol{B}\boldsymbol{B} - \frac{B^2}{2} \mathrm{I} \right) + \rho \boldsymbol{g} - \nabla p \tag{2.23}$$

Negligible electric field in the Navier-Stokes equations

The magnitude of electric field is smaller than the magnetic field by the factor $\frac{u^2}{c^2}$, so we can neglect the term $\rho_q \boldsymbol{E}$ on the right-hand-side of equation (2.23).

Negligible heat fluxes

Since the heat transfer accounts for a negligible contribution to the overall energy transfer, we neglect the heat flux terms, i.e. we set $\boldsymbol{q} = \boldsymbol{0}$ in the energy equation (2.19).

2.1.6 Adding the induction equation

The induction equation (2.16) can be rearranged in the following way:

$$\frac{\partial \boldsymbol{B}}{\partial t} + \nabla \times (\boldsymbol{B} \times \boldsymbol{u}) = -\frac{1}{\mu_0} \nabla \times (\eta \nabla \times \boldsymbol{B}) \tag{2.24}$$

$$\frac{\partial \boldsymbol{B}}{\partial t} = -\nabla \times (\boldsymbol{u} \otimes \boldsymbol{B} - \boldsymbol{B} \otimes \boldsymbol{u}) + \frac{1}{\mu_0 \sigma} \left(\nabla^2 \boldsymbol{B}\right). \tag{2.25}$$

Now we can form the system of MHD equations:

$$\frac{\partial \rho}{\partial t} = -\nabla \cdot (\boldsymbol{\pi}), \tag{2.26}$$

$$\frac{\partial \boldsymbol{\pi}}{\partial t} = -\nabla \cdot (\boldsymbol{\pi} \otimes \boldsymbol{u}) + \nabla \cdot \left(\frac{1}{\mu_0} \boldsymbol{B}\boldsymbol{B} - \frac{B^2}{2} \mathrm{I}\right) + \rho \boldsymbol{g} - \nabla p, \tag{2.27}$$

$$\frac{\partial U}{\partial t} = -\nabla \cdot \left[\boldsymbol{\pi} \left(\frac{u^2}{2} + e\right) + p\boldsymbol{u} - \frac{1}{\mu_0} \boldsymbol{B} \times \boldsymbol{E}\right] + \rho \boldsymbol{g} \cdot \boldsymbol{u} \tag{2.28}$$

$$\frac{\partial \boldsymbol{B}}{\partial t} = -\nabla \times (\boldsymbol{u} \otimes \boldsymbol{B} - \boldsymbol{B} \otimes \boldsymbol{u}) + \frac{1}{\mu_0 \sigma} \left(\nabla^2 \boldsymbol{B}\right). \tag{2.29}$$

This form suggests that rewriting these equations into a more suitable (for numerical calculations) conservative form shall be possible.

2.1.7 Conservative form of the MHD equations

A conservative form of a system of equations takes the form of

$$\frac{\partial \boldsymbol{\Psi}}{\partial t} + \nabla \cdot \mathbf{F}(\boldsymbol{\Psi}) = \mathbf{S}, \tag{2.30}$$

where $\boldsymbol{\Psi}$ is the so-called *state vector*, $\mathbf{F}_i,\ i = 1, 2, 3$ are the so-called *fluxes*, and $\mathbf{S}$ is the so-called *source term*.

Rewriting the system of equations (equations (2.26) to (2.29)) to the form of equation (2.30) is fairly straightforward. We obtain the following:

$$\boldsymbol{\Psi} = \begin{pmatrix} \rho \\ \pi_1 \\ \pi_2 \\ \pi_3 \\ U \\ B_1 \\ B_2 \\ B_3 \end{pmatrix}, \tag{2.31}$$

$$\mathbf{F}_i = \begin{pmatrix} \pi_i \\ \frac{\pi_1 \pi_i}{\rho} - B_1 B_i + \frac{1}{2}\delta_{1i}\left(p + U_m\right) \\ \frac{\pi_2 \pi_i}{\rho} - B_1 B_i + \frac{1}{2}\delta_{2i}\left(p + U_m\right) \\ \frac{\pi_3 \pi_i}{\rho} - B_1 B_i + \frac{1}{2}\delta_{3i}\left(p + U_m\right) \\ \frac{\pi_i}{\rho}\left(\frac{\gamma}{\gamma - 1}p + U_k\right) + 2\eta\varepsilon_{ijk}J_j B_k + \frac{2}{\rho}\varepsilon_{ijk}\left(\pi_k B_i - \pi_i B_k\right)B_j \\ \frac{\pi_i B_1 - \pi_1 B_i}{\rho} + \eta\varepsilon_{1ij}J_j \\ \frac{\pi_i B_2 - \pi_2 B_i}{\rho} + \eta\varepsilon_{2ij}J_j \\ \frac{\pi_i B_3 - \pi_3 B_i}{\rho} + \eta\varepsilon_{3ij}J_j \end{pmatrix}, \tag{2.32}$$

$$\mathbf{S} = \begin{pmatrix} 0 \\ \rho g_1 \\ \rho g_2 \\ \rho g_3 \\ \boldsymbol{\pi} \cdot \boldsymbol{g} \\ 0 \\ 0 \\ 0 \end{pmatrix}, \tag{2.33}$$

where $J_j = (\nabla \times \boldsymbol{B})_j$, ε_{ijk} is the Levi-Civita symbol, and δ_{ij} the Kronecker delta.

2.1.8 Solution considerations

For mathematical clarity, we should state, that the solution to the equations equation (2.30) is such a function

$$\boldsymbol{\Psi} \in C^1\left((0,T), \left[C^1\left(\Omega_t\right)\right]^8\right);\ \boldsymbol{\Psi_i} \in C^1\left((0,T), C^2\left(\Omega_t\right)\right),\ i = 6,7,8, \tag{2.34}$$

for which equation (2.30) holds for all $\boldsymbol{x} \in \Omega_t,\ t \in (0,T)$. The kind of spaces we used in the definition is called the Bochner spaces.

Because such a requirement on the solution $\boldsymbol{\Psi}$ is rather strong, we shall instead look for a so-called *weak solution*, which is specified in the coming sections.

2.2 Weak formulation of the problem

The DG method is defined by first considering the weak formulation of the equations equation (2.30) obtained by multiplying the equations at every time instance t by a (vector-valued) test function $\mathbf{v} \in W_t$, where W_t is a suitable space of (vector-valued) functions $W_t = (W_1, ..., W_8)$, integrating over the space domain Ω_t, and performing integration by parts. We start with multiplying equation (2.30) by a test function:

$$\frac{\partial \mathbf{\Psi}}{\partial t}\mathbf{v} + (\nabla \cdot \mathbf{F}(\mathbf{\Psi}))\,\mathbf{v} = \mathbf{S}\mathbf{v}, \tag{2.35}$$

then we integrate over Ω_t:

$$\int_{\Omega_t} \frac{\partial \mathbf{\Psi}}{\partial t}\mathbf{v} + \int_{\Omega_t} (\nabla \cdot \mathbf{F}(\mathbf{\Psi}))\,\mathbf{v} = \int_{\Omega_t} \mathbf{S}\mathbf{v}, \tag{2.36}$$

and finally we integrate by parts:

$$\int_{\Omega_t} \frac{\partial \mathbf{\Psi}}{\partial t}\mathbf{v} - \int_{\Omega_t} \mathbf{F}(\mathbf{\Psi})\,(\nabla \cdot \mathbf{v}) + \int_{\partial\Omega_t} (\mathbf{F}(\mathbf{\Psi}) \cdot \boldsymbol{n})\,\mathbf{v} = \int_{\Omega_t} \mathbf{S}\mathbf{v}, \tag{2.37}$$

where the terms $\mathbf{F}(\mathbf{\Psi})\,(\nabla \cdot \mathbf{v})\,; (\mathbf{F}(\mathbf{\Psi}) \cdot \boldsymbol{n})\,\mathbf{v}$ are meant as a component-wise multiplication.

Now without going into detail, we can conclude, that a suitable space W would be

$$W_t = \left[H^1(\Omega_t)\right]^8, \tag{2.38}$$

and we shall look for a solution to the equation (2.37) in the same space W_t at every time instance t. We shall not relax the requirement for continuity of time-derivatives we imposed for the *hard solution* equation (2.34) for reasons discussed later, and we arrive at the following Bochner space in which we are looking for a weak solution of equation (2.30):

$$W = C^1((0,T), W_t). \tag{2.39}$$

To sum up we can define the *weak solution* $\mathbf{\Psi} = \mathbf{\Psi}((t, \boldsymbol{x}))$ of MHD equations equation (2.30) as

$\mathbf{\Psi}((t,\boldsymbol{x})) \in W$ defined in equation (2.39).
equation (2.37) holds for all $t \in (0,T)$, and all $\mathbf{v} \in W_t$.
$\mathbf{\Psi}(0,\boldsymbol{x}) = \Pi\mathbf{\Psi}^0(\boldsymbol{x})$.

Table 2.1: Notation

In table 2.1, Π is a projection of the initial condition $\mathbf{\Psi}^0$ onto W_0.

2.3 Boundary conditions

For the problem of finding the solution as described in table 2.1 to be complete, we need to specify the proper boundary conditions.

2.3.1 Essential (inflow) boundary conditions

Since the solution as described in table 2.1 of the problem specified in equation (2.37) is not influenced by its values on the boundary Ω_t, we are not able to employ standard essential boundary conditions of the form $\boldsymbol{u}(x,y,z) = \boldsymbol{u}_D(x,y,z)$ with a known $\boldsymbol{u}_D$. If such a condition is required from the physical nature of the described phenomenon (as often is the case), it is only implied by a correct definition of fluxes at the boundary - as one can see in the last integrand $(\mathbf{F}(\boldsymbol{\Psi}) \cdot \boldsymbol{n})\,\mathbf{v}$ in equation (2.37). We shall see how that is numerically handled in later chapters.

2.3.2 Outflow (do-nothing) boundary conditions

If a particular boundary is only present in the numerical model (representing e.g. an "outer" boundary that is there to limit the size of the computation to the area of interest), that is, a free boundary, that is not in any way present as an actual physical boundary or interface, this is achieved by specifying that values of $\boldsymbol{\Psi}$ do not change through the boundary in equation (2.37):

$$\frac{\partial \boldsymbol{\Psi}}{\partial \boldsymbol{n}} = 0. \tag{2.40}$$

2.3.3 Periodic boundary conditions

Periodic boundary condition is always specified on two parts Γ_1, Γ_2 of the domain boundary that share the bijection mapping between points:

$$[x_1, y_1, z_1] \leftrightarrow [x_2, y_2, z_2] \,\forall\, [x_1, y_1, z_1] \in \Gamma_1,\ \forall\, [x_2, y_2, z_2] \in \Gamma_2, \tag{2.41}$$

so that for each pair of related points $[x_1, y_1, z_1] \leftrightarrow [x_2, y_2, z_2]$, the values must be the same:

$$u([x_1, y_1, z_1]) = u([x_2, y_2, z_2]) \,\forall\, [x_1, y_1, z_1] \in \Gamma_1,\ \forall\, [x_2, y_2, z_2] \in \Gamma_2 :\ [x_1, y_1, z_1] \leftrightarrow [x_2, y_2, z_2]\,. \tag{2.42}$$

3 Numerical approach

The weak formulation of the problem we obtained in table 2.1 still posses a problematic attribute - the space defined in equation (2.39) is of infinite dimension, and therefore we would need to employ analytical methods to find the solution table 2.1 in such a space. The equation (2.37) is however rather impossible to be solved analytically, and we have to utilize some sort of numerical simulation - which in turn needs to operate on finite-dimensional spaces. But we need to make sure that the simplifying (reducing) assumptions we make on the way to the numerical model are acceptable so that the numerical solution we obtain converges (as we reduce the discretization size) to the solution defined in table 2.1.

In this chapter we shall consider that $\Omega_t = \Omega \, \forall t \in (0, T)$, i.e. the computational domain does not change with respect to time. There are approaches to numerical simulation of MHD phenomena without this condition in place, which utilize the exact same general approach described in this work plus they add additional steps in the algorithm. These are outside of the scope of this work. Also, we always take $\Omega \subset \mathbb{R}^3$.

3.1 Triangulation

We start with leaving the time-derivative untouched, and focus on the discretization in space for now - we are performing a *space semidiscretization*.

First step in the process of the discretization is to divide the computational domain $\overline{\Omega}$ into a finite number of subsets with properties described below. These subsets form the set, further denoted by T_h, called the *triangulation or mesh of the domain* Ω. The index h will be dropped when either h is irrelevant, or when dealing with mesh refinement - see chapter 4, also when needed the domain which the triangulation approximates will be noted as $T_h(\Omega)$, or $T(\Omega)$.

Please note that the terms *triangulation* and *mesh* shall be used in the text interchangeably (another synonym is a *grid*). The parameter $h > 0$ of the triangulation usually represents maximum of diameters of all elements $K \in T_h$. The elements $K \in T_h$ are in the context of the finite volume method called *finite volumes*.

Properties of T_h:

Triangulation

1. Each $K \in T_h$ is closed and connected with its interior $K^\circ \neq \emptyset$.
2. Each $K \in T_h$ has a Lipschitz boundary.
3. $\cup_{K \in T_h} K = \overline{\Omega}$
4. If $K_1, K_2 \in T_h$, $K_1 \neq K_2$, then $K_1^\circ \cap T_2^\circ = \emptyset$.

In our case of the three-dimensional problem, we assume that the domain Ω is obtained as an approximation of the original computational domain (also denoted by Ω), and the triangulation is chosen accordingly to the following attributes:

A) Each $K \in T_h$ is a closed rectangular hexahedron, possibly with curved faces.

B) For $K_1, K_2 \in T_h$, $K_1 \neq K_2$ we have either $K_1 \cap K_2 = \emptyset$ or K_1, K_2 share one face (if the shared face is a whole common face, we call the triangulation *regular*), or K_1, K_2 share one vertex, or K_1, K_2 share one face.

C) $\cup_{K \in T_h} K = \overline{\Omega}$.

Furthermore

$$T_h = \left\{ K^i, i \in I \right\}, \tag{3.1}$$

where $I \subset Z^+ = \{0, 1, 2, ...\}$ is a suitable index set.
By Γ_{ij} we denote a common face between two neighboring elements K^i and K^j. We set

$$s(i) = \left\{ j \in I; K^j \text{ is adjacent to } K^i \right\}.$$

The boundary $\partial\Omega$ is formed by a finite number of faces of elements K^i adjecent to $\partial\Omega$. We denote all these boundary faces by S_j, where $j \in I_b \subset Z^- = \{-1, -2, ...\}$. Now we set

$$\gamma(i) = \left\{ j \in I_b; S_j \text{ is a face of } K^i \in T_h \right\}$$

and

$$\Gamma_{ij} = S_j \text{ for } K^i \in T_h \text{ such that } S_j \subset \partial K^i, j \in I_b.$$

For K^i not containing any boundary face S_j we set $\gamma(i) = \emptyset$.
Obviously, $s(i) \cap \gamma(i) = \emptyset$ for all $i \in I$. If we write $S(i) = s(i) \cup \gamma(i)$, we have

$$\partial K^i = \cup_{j \in S(i)} \Gamma_{ij}, \quad \partial K^i \cap \partial\Omega = \cup_{j \in \gamma(i)} \Gamma_{ij}.$$

Furthermore we define the set of internal (i.e. not lying on the boundary $\partial\Omega$) faces as

$$\Gamma_I = \cup_{i \in I} \cup_{j \notin \gamma(i)} \Gamma_{ij}, \tag{3.2}$$

and the set of boundary (i.e. lying on the boundary $\partial\Omega$) faces as

$$\Gamma_B = \cup_{i \in I} \cup_{j \in \gamma(i)} \Gamma_{ij}. \tag{3.3}$$

Note If we were to use not $\Omega \subset \mathbb{R}^3$, but rather $\Omega \subset \mathbb{R}^4$, we may just employ the following machinery also to the time-derivative - this is not an uncommon approach. Why the approach described in this work is favored by the author is twofold:

- Data (in a general sense - e.g. algebraic systems, function bases, etc.) are smaller when using a separate handling for time-derivative
- The dependency on time and space may (and usually does) vary a lot for physical phenomena - to have a separate approach is therefore beneficial

3.1.1 Distributed triangulation

The standard approach to handle large problems that are impossible to be calculated on a single processor in mesh-based numerical simulations (such as Discontinuous Galerkin method) is to employ a *domain decomposition* method, where each of the processors on which the simulation runs holds data about a subset of elements of the mesh T. Consequently, also the matrix and vector assembly (described in algorithm 1), the linear problem solution, slope limiting, and AMR procedures are performed by all processors using data they have available. The aim here is not to go into deep technical details of distributing data, etc.

In figures 3.1 to 3.2, domain $\Omega = [0,1] \times [0,1] \times [0,1]$ was used, it was triangulated by $10 \times 10 \times 10$ mesh elements and the domain was distributed among 5 processors labelled 0..4.

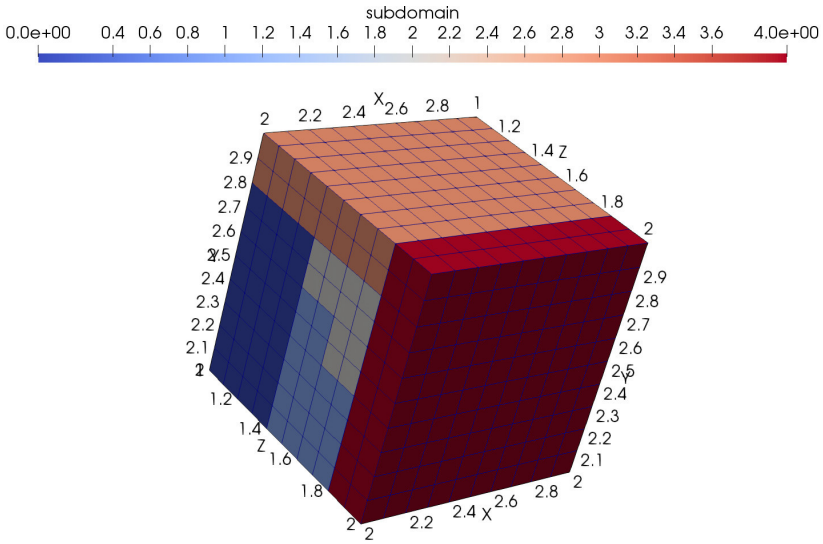


Figure 3.1: Cubical domain Ω with color-coded processor-owned elements.

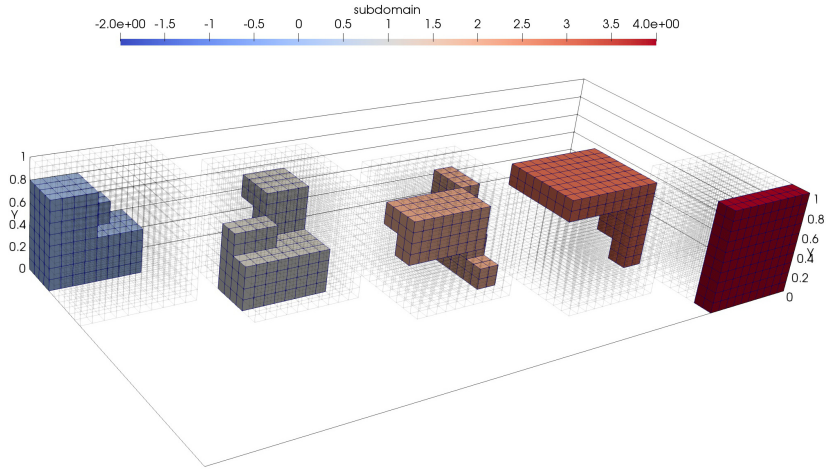


Figure 3.2: The same domain as in figure 3.1, with clearer indication of elements that belong to individual processors (0..4 left to right).

3.2 Discontinuous Galerkin method

For complex problems of compressible flow, and of course for even more complex problems of compressible MHD, there has been a number of attempts to use standard and well known Finite Element Methods that replace the spaces defined in equation (2.39) with finite-dimension spaces with bases formed by continuous piecewise polynomial functions. These attempts struggled with a common problem of spurious oscillations appearing in the solution - the origin of which is the lack of "stabilization", provided by the second-order terms in elliptic equations. Solution to these problems is the application of stabilization techniques, that usually introduce some sort of artificial diffusion (the second-order term), all of which are non-physical, and generally involve "magical" numbers - constants that are of pure computational nature (not a part of the physical description) or even worse are problem-specific.

3.2.1 Overview of the DG method

Due to this reason, there was an effort to develop methods which would not need such stabilization techniques, and would still offer reasonable resolution of shock-waves, boundary and interior layers, and steep gradients without exhibiting spurious oscillations in the approximate solutions. The approach taken here is based on the idea to combine finite volume and Finite element methods leading to the so-called *discontinuous Galerkin finite element method (DGFEM, DG)*. Here we shall derive and analyze DG for our equations. Let T_h be a triangulation of Ω. By $H^1(\Omega, T_h)$ we denote the so-called *broken Sobolev space*:

$$H^1(\Omega, T_h) = \left\{ v \in L^2(\Omega);\ v|_K \in H^1(K)\, \forall K \in T_h \right\}. \tag{3.4}$$

This space is an approximation of the space defined in equation (2.38), but it contains functions that are discontinuous on element interfaces Γ_{ij} between elements $K^i and K^j$.

For $u \in H^1(\Omega, T_h)$ we set

$$u_K^i = \text{trace of } u|_{K^i} \text{ on } \partial K^i \tag{3.5}$$

(i.e. the interior trace of u on ∂K^i). For each face $\Gamma_{ij} \subset \partial K \backslash \Gamma$ of K^i, there exists $K^j \neq K^i,\ K^j \in T_h$, adjacent to Γ_{ij} from the opposite side than K^i. Then we put

$$u_K^j = \text{trace of } u|_{K^j} \text{ on } \Gamma_{ij}. \tag{3.6}$$

In this way we obtain the exterior trace u_K^j of u on $\partial K^i \backslash \Gamma$ and define the jump of u on $\partial K \backslash \Gamma$:

$$[u]_{\Gamma_{ij}} = u_K^i - u_K^j. \tag{3.7}$$

Approximation of the broken Sobolev space

Let the domain Ω be covered with a mesh $T_h = \{K_1,\ K_2, \ldots, K_M\}$ where each element K_m carries an arbitrary polynomial degree $1 \leq p_m, \forall m = 1, 2, \ldots, M$. The broken Sobolev space $H^1(\Omega, T_h)$ will be approximated by a finite-dimensional space of piecewise-polynomial functions

$$V_h = \{v \in L^2(\Omega);\ v|_{K_m} \in P^{p_m}(K_m) \text{ for all } 1 \leq m \leq M\} \tag{3.8}$$

where P^p is defined as

$$P^p = \text{span}\{ \sum_{\substack{0 \leq i,j,k \leq p \\ i+j+k \leq p}} \alpha_i\ x_1^i\ x_2^j\ x_3^k,\quad \alpha_i \in \mathbb{R}\}.$$

3.2.2 DG formulation of MHD equations

Although the resulting system will look very similar to the weak formulation equation (2.37), the derivation makes more sense to be done starting with the equation (2.30).

As stated in section 3.1, at this point we will discretize the problem in space, and leave the time-derivative untouched. The approximate solution will be sought at each time instant t as an element of the finite-dimensional space

$$[V_h]^8, \tag{3.9}$$

where V_h is defined in equation (3.8). Functions

$$\mathbf{v}_h \in [V_h]^8 \approx \left[H^1(\Omega, T_h)\right]^8, \tag{3.10}$$

where $H^1(\Omega, T_h)$ is defined in equation (3.4), are in general discontinuous on interfaces Γ_{ij}. By $\mathbf{v}_h|_{ij}$ and $\mathbf{v}_h|_{ji}$ we denote the values of $\mathbf{v}_h$ on Γ_{ij} considered from the interior and the exterior of K^i, respectively. The symbols

$$\langle \mathbf{v}_h \rangle_{ij} = \frac{1}{2}\left(\mathbf{v}_h|_{ij} + \mathbf{v}_h|_{ji}\right), \ [\mathbf{v}_h]_{ij} = \mathbf{v}_h|_{ij} - \mathbf{v}_h|_{ji}$$

denote the average and jump of a function $\mathbf{v}_h$ on Γ_{ij}. In order to derive the discrete problem, we multiply equation (2.30) by a test function $\mathbf{v}_h \in [V_h]^8$ in a component-wise fashion, integrate over any element $K^i \in T_h$, apply Green's theorem and sum over all $i \in I$, where I is defined in equation (3.1):

$$\int_{\Omega_t} \frac{\partial \mathbf{\Psi}_h}{\partial t}\mathbf{v}_h - \sum_{K^i \in T_h} \int_{K^i} \mathbf{F}(\mathbf{\Psi}_h)(\nabla \cdot \mathbf{v}_h) + \sum_{K^i \in T_h} \sum_{j \in s_i} \int_{\Gamma_{ij}} (\mathbf{F}(\mathbf{\Psi}_h) \cdot \boldsymbol{n}_{ij})\mathbf{v}_h = \int_{\Omega_t} \mathbf{S}\mathbf{v}_h, \tag{3.11}$$

where $\boldsymbol{n}_{ij}$ is the unit outer normal to Γ_{ij}. Now, the term

$$\int_{\Gamma_{ij}} \mathbf{F}(\mathbf{\Psi}_h) \cdot \boldsymbol{n}_{ij}\mathbf{v}_h \tag{3.12}$$

is problematic, because the value of $\mathbf{\Psi}_h$ on Γ_{ij} is not unique - we have two values:

- $\mathbf{\Psi}_h|_{ij}$ - which is the value of $\mathbf{\Psi}_h$ on Γ_{ij} considered from the element K^i,
- $\mathbf{\Psi}_h|_{ji}$ - which is the value of $\mathbf{\Psi}_h$ on Γ_{ij} considered from the element K^j.

Note: This corresponds to the notation set in equations (3.5) and (3.6) - if we take K^i as the element at hand, we have

$$\mathbf{\Psi}_h|_{ij} = \mathbf{\Psi}_{h\,K^i}^{+}, \quad \mathbf{\Psi}_h|_{ji} = \mathbf{\Psi}_{h\,K^i}^{-}$$

Now, because of this non-uniqueness of the values, we replace the term equation (3.12) with the so-called *numerical flux* $\mathbf{H} = \mathbf{H}(\mathbf{v}, \mathbf{w}, \boldsymbol{n})$ in the following fashion:

$$(\mathbf{F}(\mathbf{\Psi}_h) \cdot \boldsymbol{n}_{ij})\mathbf{v}_h \approx \mathbf{H}(\mathbf{\Psi}_h|_{ij}, \mathbf{\Psi}_h|_{ji}, \boldsymbol{n}_{ij})\mathbf{v}_h. \tag{3.13}$$

We impose the following requirements on the numerical flux:

A) $\mathbf{H}(\mathbf{v}, \mathbf{w}, \boldsymbol{n})$ is defined and continuous on $\mathcal{D} \times \mathcal{D} \times \mathcal{S}_1$, where $\mathcal{D}$ is the domain of definition of the flux $\mathbf{F}$ and $\mathcal{S}_1$ is the unit sphere in $\mathbb{R}^3$.

B) $\mathbf{H}$ is *consistent*:

$$\mathbf{H}(\mathbf{v}, \mathbf{v}, \boldsymbol{n}) = \mathbf{F}(\mathbf{v})\boldsymbol{n}, \ \mathbf{v} \in \mathcal{D}, \ \boldsymbol{n} \in \mathcal{S}_1. \tag{3.14}$$

C) $\mathbf{H}$ is *conservative*:

$$\mathbf{H}\left(\mathbf{v},\mathbf{w},\boldsymbol{n}\right)=-\mathbf{H}\left(\mathbf{w},\mathbf{v},-\boldsymbol{n}\right),\ \mathbf{v},\mathbf{w}\in\mathcal{D},\ \boldsymbol{n}\in\mathcal{S}_1. \tag{3.15}$$

And using these properties of the numerical flux, we can rewrite equation (3.11) as:

$$\begin{aligned}\int_{\Omega_t}\frac{\partial\boldsymbol{\Psi}_h}{\partial t}\mathbf{v}_h-\sum_{K^i\in T_h}\int_{K^i}\mathbf{F}\left(\boldsymbol{\Psi}_h\right)\left(\nabla\cdot\mathbf{v}_h\right)\\+\sum_{\Gamma_{ij}\in\Gamma_I}\int_{\Gamma_{ij}}\mathbf{H}\left(\boldsymbol{\Psi}_h|_{ij},\boldsymbol{\Psi}_h|_{ji},\boldsymbol{n}_{ij}\right)\mathbf{v}_h=\int_{\Omega_t}\mathbf{S}\mathbf{v}_h,\end{aligned} \tag{3.16}$$

where we used the definition of internal edges (equation (3.2)).

3.2.3 Numerical flux

Generally, the numerical flux function can be a non-differentiable (or even discontinuous) function. That renders it impossible to directly employ Newton's method to solve the resulting nonlinear problem, and either linearization (e.g. a semi-implicit scheme - Dolejší and Feistauer (2015), chapter 5.1) or explicit time discretization (used in this work) needs to be used.

Another complication arising from evaluation of numerical fluxes on element interfaces exists in distributed solver, where we need to make sure that all processors have relevant data (e.g. previous solution values) from all cells that neighbor any cells assembled on the processor at hand. This issue gets worse when local mesh refinement (there are more neighbor elements of the current cell across the interface at hand), as well as if periodic boundary conditions are used (the neighbor graph is more complex).

Lax-Friedrichs numerical flux

This is the most straightforward numerical flux satisfying equation (3.14), and equation (3.15) and is defined as follows:

$$\mathbf{H}_{LF}\left(\mathbf{v},\mathbf{w},\boldsymbol{n}\right)=\frac{1}{2}\left[\mathbf{F}\left(\mathbf{v}\right)+\mathbf{F}\left(\mathbf{w}\right)\right]-\frac{\alpha}{2}\left(\mathbf{w}-\mathbf{v}\right), \tag{3.17}$$

where the parameter α is the so-called *stabilization parameter*, which in order for the solution to be stable needs to fulfill $\alpha<\frac{\Delta t}{\Delta x}$, where $\Delta t,\Delta x$ need to satisfy the CFL condition equation (3.41). Now, this numerical flux is very sensitive to the choice of α - for larger values, it tends to be very diffusive, but for lower values, it is unstable and produces oscillations in the vicinity of shocks J. and B. (2002). In this work it is primarily used for implementation verification purposes, as due to its simplicity, the risk of errors in its implementation is rather negligible.

Riemann problem for MHD

For the class of numerical fluxes, known as the *Godunov type fluxes*, the numerical flux across the element boundary is constructed as the flux $\mathbf{F}$ of a solution of the following Riemann problem (equation (3.18)) at $x = 0$:

$$\frac{\partial \mathbf{U}}{\partial t} + \frac{\partial \mathbf{F}}{\partial x} = 0, \tag{3.18}$$

where

$$\mathbf{U} = \begin{pmatrix} \rho \\ \pi_1 \\ \pi_2 \\ \pi_3 \\ U \\ B_2 \\ B_3 \end{pmatrix}, \ \mathbf{F} = \begin{pmatrix} \pi_1 \\ \frac{\pi_1^2}{\rho} - B_1^2 + \frac{1}{2}(p + U_m) \\ \frac{\pi_2 \pi_1}{\rho} - B_1 B_2 \\ \frac{\pi_3 \pi_1}{\rho} - B_1 B_3 \\ \frac{\pi_1}{\rho}\left(\frac{\gamma}{\gamma-1}p + U_k\right) + \frac{2}{\rho}(\pi_k B_1 - \pi_1 B_k) B_1 \\ \frac{\pi_1 B_2 - \pi_2 B_1}{\rho} \\ \frac{\pi_1 B_3 - \pi_3 B_1}{\rho} \end{pmatrix}, \tag{3.19}$$

with two states

$$\mathbf{U}_L = (\rho_L, \pi_{1L}, \pi_{2L}, \pi_{3L}, U_L, B_{2L}, B_{3L}) \, \mathbf{U}_R \quad = (\rho_R, \pi_{1R}, \pi_{2R}, \pi_{3R}, U_R, B_{2R}, B_{3R}), \tag{3.20}$$

and where due to the divergence free condition $\nabla \cdot \boldsymbol{B} = 0$ of the magnetic field), B_1 is given as constant.

These equations (equation (3.18)) have seven eigenvalues which correspond to two Alfve'n waves ($\lambda_{2,6}$), two slow magneto-acoustic waves ($\lambda_{3,5}$), and two fast magneto-acoustic waves ($\lambda_{1,7}$), and one entropy wave (λ_4):

$$\lambda_1 = \boldsymbol{u}_x - c_f, \tag{3.21}$$
$$\lambda_2 = \boldsymbol{u}_x - c_a, \tag{3.22}$$
$$\lambda_3 = \boldsymbol{u}_x - c_s, \tag{3.23}$$
$$\lambda_4 = \boldsymbol{u}_x, \tag{3.24}$$
$$\lambda_5 = \boldsymbol{u}_x + c_s, \tag{3.25}$$
$$\lambda_6 = \boldsymbol{u}_x + c_a, \tag{3.26}$$
$$\lambda_7 = \boldsymbol{u}_x + c_f, \tag{3.27}$$

where $c_a = \sqrt{\frac{B_1^2}{rho}}$, $c_{s,f} = \left\{ \frac{\gamma p + |B|^2 \mp \sqrt{(\gamma p + |B|^2)^{\frac{1}{2}} - 4\gamma p B_1^2}}{2\rho} \right\}^{\frac{1}{2}}$.

From this, it follows, that there are theoretically as many as 9 states (states between the characteristics equal to the eigenvalues $\lambda_1, ..., \lambda_7$ along which the solution propagates) - see figure 3.3.

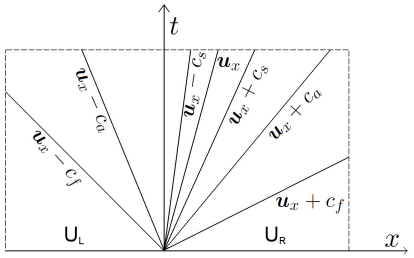


Figure 3.3: Characteristics corresponding to eigenvalues of equation (3.18).

Unfortunately, for MHD equations, there is no exact solver of the Riemann problem across the element boundary, and therefore, approximate solvers are used. One instance of the derived numerical flux based on the solution at $x = 0$ of equation (3.18) is described in the next section.

HLLD numerical flux

The abbreviation **HLLD** stands for Harten-Lax-van Leer (HLL) approximate Riemann solver, and **D** stands for Discontinuities. It is an extension of HLLC Riemann solver for the Euler equations (Batten et al. (1997)). It divides the Riemann fan into 6 states as illustrated in figure 3.4. This particular numerical flux has been introduced in Miyoshi and Kusano (2005) and has been shown to be very suitable for the studied problems. HLLD is an approximate nonlinear solution of the MHD Riemann problem, and is algebraically derived in Miyoshi and Kusano (2005) under the assumptions that the normal velocity and the background potential magnetic field in the Riemann fan are constant. It follows from these assumptions, that 4 intermediate states are sufficient to resolve the Riemann problem in adequate manner (Miyoshi and Kusano (2005)).

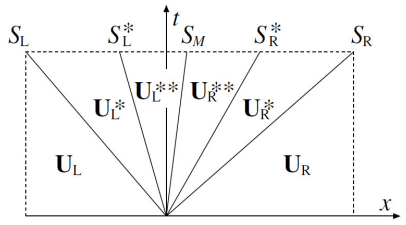


Figure 3.4: Six states considered in the definition of HLLD flux.

The HLLD flux according to its definition in Miyoshi and Kusano (2005) defines 4 intermediate (starred) states $U_L^*, U_L^{**}, U_R^*, U_R^{**}$ and 2 outer states U_L, U_R corresponding to states of U defined in equation (3.19) for the left ($x < 0$) and right

($x > 0$) parts of the time-space domain illustrated in figure 3.4. Details of these states are given in Miyoshi and Kusano (2005). The resulting fluxes are:

$$F_{HLLD} = \begin{cases} F_L \text{ if } S_L > 0, \\ F_L^* \text{ if } S_L \leq 0 \leq S_L^*, \\ F_L^{**} \text{ if } S_L^* \leq 0 \leq S_M, \\ F_R^{**} \text{ if } S_M \leq 0 \leq S_R^*, \\ F_R^* \text{ if } S_R^* \leq 0 \leq S_R, \\ F_R \text{ if } S_R < 0 \end{cases} . \tag{3.28}$$

These are derived using the assumptions mentioned earlier, that:

$$u_L^* = u_l^{**} = u_R^{**} = u_R^* = S_M \qquad \text{normal velocity constant over the Riemann fan} \tag{3.29}$$

$$p_{T_L}^* = p_{T_L}^{**} = p_{T_R}^{**} = p_{T_R}^* = p_T^* \qquad \text{total pressure constant over the Riemann fan,} \tag{3.30}$$

where p_T is the total pressure as defined in equation (1.4), and

$$S_M \frac{(S_R - u_R)\rho_R u_R - (S_L - u_L)\rho_L u_L - p_{T_R} + p_{T_L}}{(S_R - u_R)\rho_R - (S_L - u_L)\rho_L}, \tag{3.31}$$

$$S_L = \min\left[\lambda_1(U_L), \lambda_1(U_R)\right], \tag{3.32}$$

$$S_R = \max\left[\lambda_7(U_L), \lambda_7(U_R)\right], \tag{3.33}$$

$$S_L^* = S_M - \frac{|B_x|}{\sqrt{\rho_L^*}} = S_M - \frac{|B_x|}{\sqrt{\rho_L \frac{S_L - u_L}{S_L - S_M}}}, \tag{3.34}$$

$$S_R^* = S_M + \frac{|B_x|}{\sqrt{\rho_R^*}} = S_M + \frac{|B_x|}{\sqrt{\rho_R \frac{S_R - u_R}{S_R - S_M}}}. \tag{3.35}$$

Numerical fluxes comparison

For the comparison, a simple version of the benchmark described in section 5.1.2 was used. In the next figures, there are snapshots of density solution component at several time steps for the Lax-Friedrichs flux for two values of $\alpha = 0.5$, $\alpha = 1.2$, and for the HLLD flux (the higher the stabilization term α, the more stable the scheme is). On each of sections 3.2.3 to 3.2.3, the entire domain is shown on the left, and a plot over the line $y = 0$ is on the right.

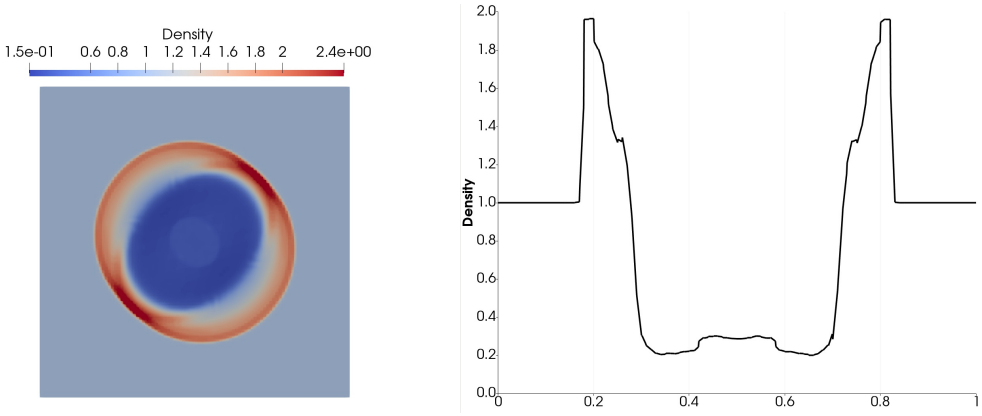


Figure 3.5: Density at time $t = 0.1$, Lax-Friedrichs flux with $\alpha = 0.5$

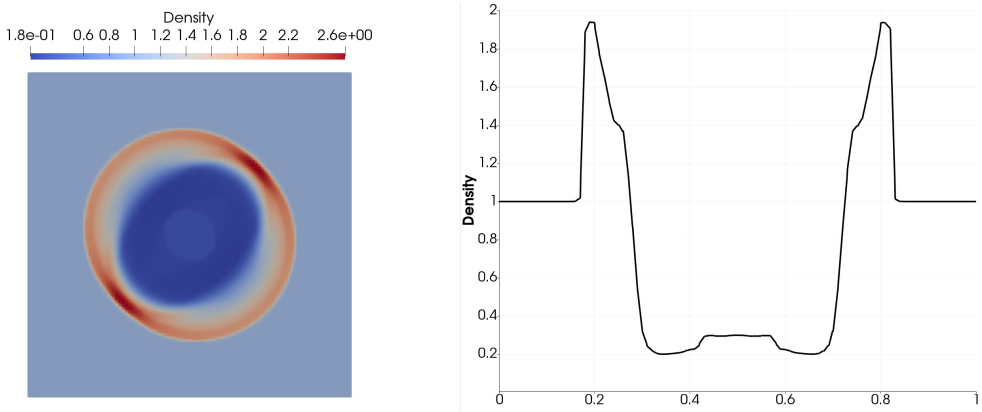


Figure 3.6: Density at time $t = 0.1$, Lax-Friedrichs flux with $\alpha = 1.2$

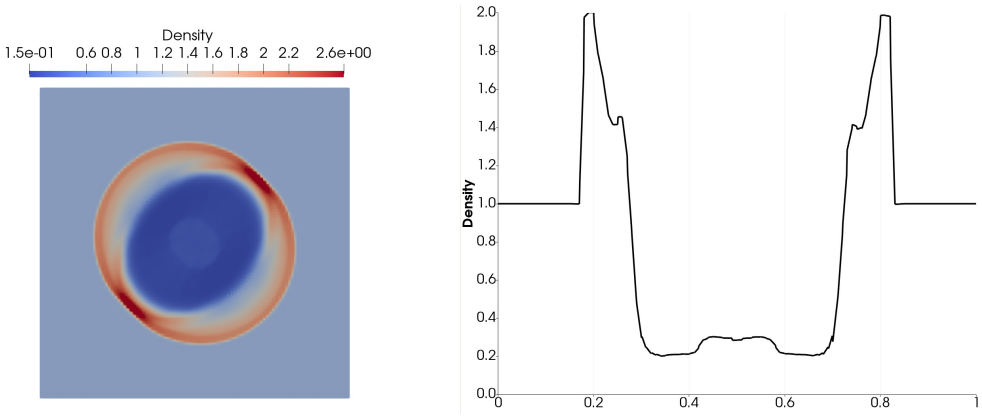


Figure 3.7: Density at time $t = 0.1$, HLLD flux

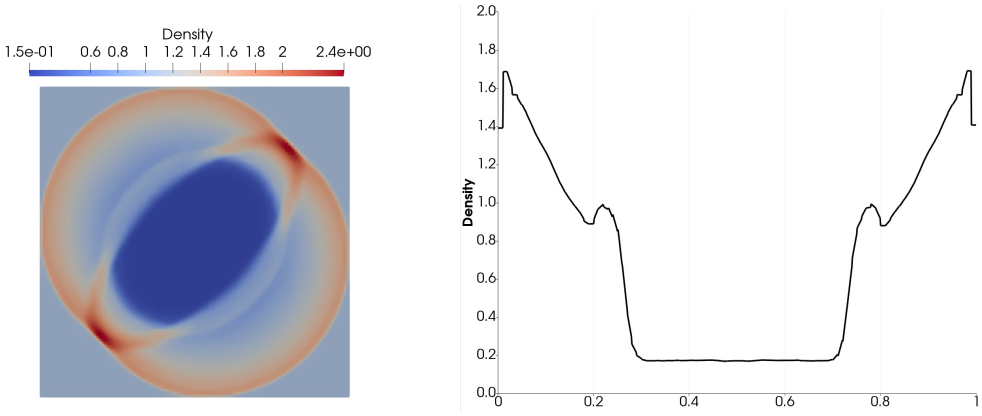


Figure 3.8: Density at time $t = 0.2$, Lax-Friedrichs flux with $\alpha = 0.5$

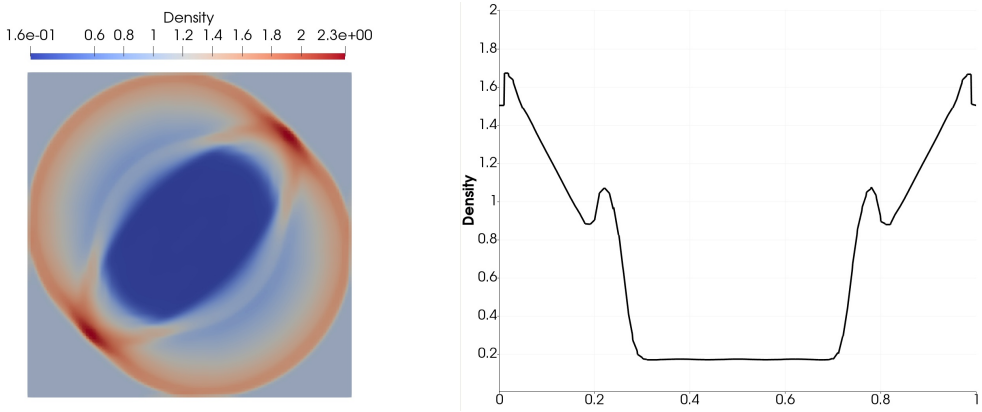


Figure 3.9: Density at time $t = 0.2$, Lax-Friedrichs flux with $\alpha = 1.2$

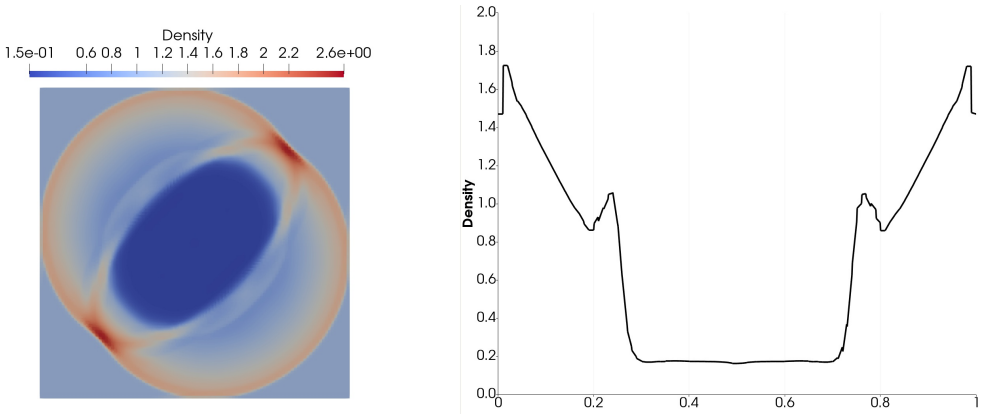


Figure 3.10: Density at time $t = 0.2$, HLLD flux

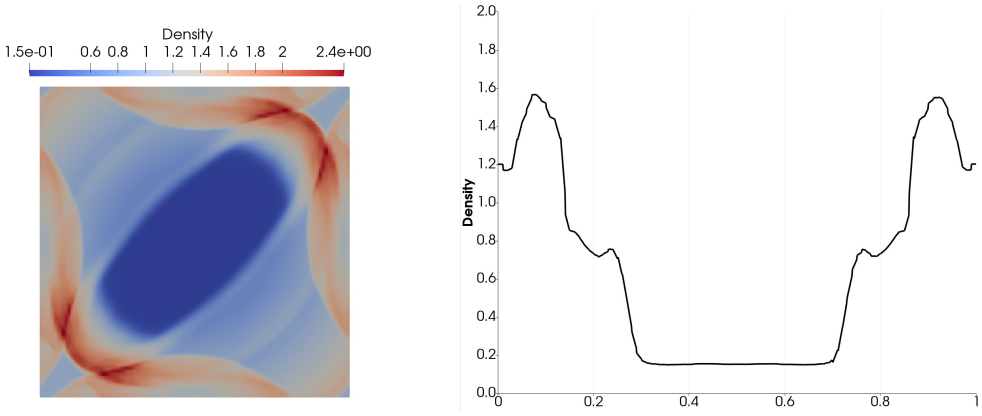


Figure 3.11: Density at time $t = 0.3$, Lax-Friedrichs flux with $\alpha = 0.5$

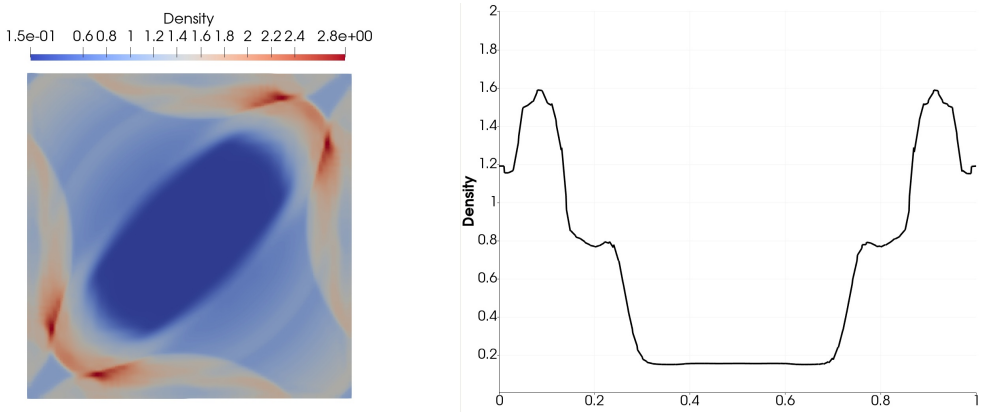


Figure 3.12: Density at time $t = 0.3$, Lax-Friedrichs flux with $\alpha = 1.2$

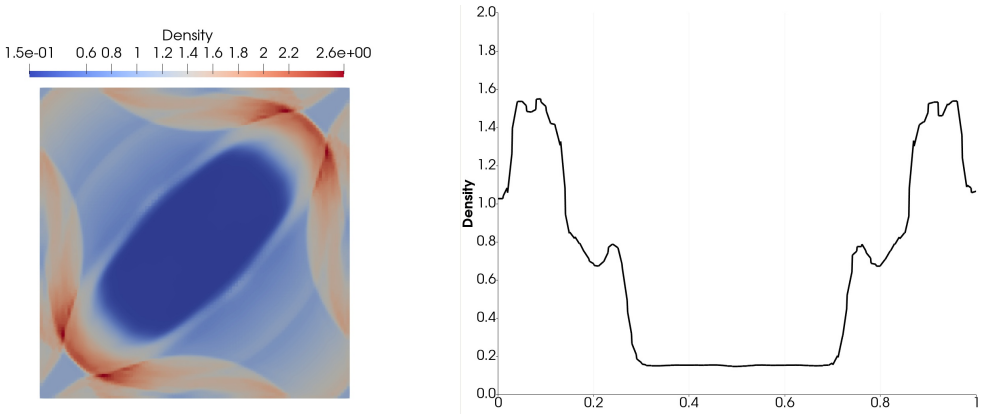


Figure 3.13: Density at time $t = 0.3$, HLLD flux

There are a few observations from all the triplets of Figures:

- The Lax-Friedrichs flux is very sensitive to the choice of α, for $\alpha = 1.2$, the solution is too diffusive,
- The 'ideal' Lax-Friedrichs flux where $\alpha = 0.5$ is quantitatively and qualitatively very close to the HLLD flux.

The second observation could lead us to the conclusion that the Lax-Friedrichs flux, being simple to implement, is superior to HLLD which is quite complex. Unfortunately, as shown e.g. in J. and B. (2002), the Lax-Friedrichs flux is not stable enough to be used for arbitrary problem setup, also when handling solution with discontinuities, it is very dissipative G. (1997).

Second reason why for practical usage of the developed code, the HLLD flux is strongly recommended is, that it is parameter-free, as opposed to Lax-Friedrichs flux having the stabilization parameter α which makes it performing differently for different problems, as per the value of α.

3.2.4 Numerical handling of boundary conditions

In what follows, we are interested in using flux-induced inflow and outflow boundary conditions (see Section section 2.3). Periodic boundary conditions are considered to be merely a special case of handling numerical fluxes across internal faces in Γ_I . To account for these boundary conditions, we need to investigate the term

$$\int_{\Gamma_{ij}} \mathbf{H}\left(\boldsymbol{\Psi}_h|_{ij}, \boldsymbol{\Psi}_h|_{ji}, \boldsymbol{n}_{ij}\right) \mathbf{v}_h$$

for $\Gamma_{ij} \in \Gamma_B$ (see equation (3.3)). This term is used in equation (3.16) for faces in Γ_I which are internal and always have 2 values connected to them - $\boldsymbol{\Psi}_h|_{ij}, \boldsymbol{\Psi}_h|_{ji}$ - which induces the notation. On a boundary face, the corresponding value to $\boldsymbol{\Psi}_h|_{ij}$ can be defined in the same way as in the case of Γ_I, but $\boldsymbol{\Psi}_h|_{ji}$ needs to be defined.

Inflow boundary condition

First, if we want to prescribe an inflow boundary condition (i.e. we know what values should the state vector $\boldsymbol{\Psi}_h$ have on $\Gamma_{ij} \in \Gamma_B$), we define

$$\overline{\boldsymbol{\Psi}_h|_{ji}} \tag{3.36}$$

to be the prescribed value.

Outflow boundary condition

If we want to model an outflow boundary condition (i.e. do nothing condition), we may use the *consistency* of the numerical flux $\mathbf{H}$ defined in equation (3.14), and

define

$$\overline{\boldsymbol{\Psi}_h|_{ji}} = \boldsymbol{\Psi}_h|_{ij}, \tag{3.37}$$

which is a suitable definition for the outflow boundary condition. It is important to mention, that setting the inflow boundary condition does not imply that solution values on this boundary equal to these prescribed value. This follows from the definition of broken Sobolev space (equation (3.4)). Moreover the values of the solution on the boundary also depend on the numerical flux used, as the values on the boundary are merely one of the input parameters for the flux (See equation (3.13)).

Periodic boundary conditions

Periodic boundary conditions come in the form described in equation (2.42), for pair of points on the boundary (equation (2.41)). The point mapping is the main complication encountered, namely in the distributed computations, and moreover if AMR is used in distributed computations (see section 4.2.1). Otherwise, from the integral evaluation perspective, an face on a periodic boundary is approached in the very same way as internal faces (we have values from both sides, and we evaluate the numerical flux in quadrature points).

DG full problem statement

Now, taking equation (3.36) and equation (3.37), we can enhance equation (3.16) with an additional term, that will add the boundary conditions into the equation:

$$\sum_{\Gamma_{ij}\in\Gamma_B} \int_{\Gamma_{ij}} \mathbf{H}\left(\boldsymbol{\Psi}_h|_{ij}, \overline{\boldsymbol{\Psi}_h|_{ji}}, \boldsymbol{n}_{ij}\right)\mathbf{v}_h,$$

so that the complete semi-discrete problem reads:

$$\begin{aligned}\int_{\Omega_t} \frac{\partial \boldsymbol{\Psi}_h}{\partial t}\mathbf{v}_h &- \sum_{K^i\in T_h}\int_{K^i}\mathbf{F}\left(\boldsymbol{\Psi}_h\right)\left(\nabla\cdot\mathbf{v}_h\right) \\ &+ \sum_{\Gamma_{ij}\in\Gamma_I}\int_{\Gamma_{ij}}\mathbf{H}\left(\boldsymbol{\Psi}_h|_{ij}, \boldsymbol{\Psi}_h|_{ji}, \boldsymbol{n}_{ij}\right)\mathbf{v}_h \\ &+ \sum_{\Gamma_{ij}\in\Gamma_B}\int_{\Gamma_{ij}}\mathbf{H}\left(\boldsymbol{\Psi}_h|_{ij}, \overline{\boldsymbol{\Psi}_h|_{ji}}, \boldsymbol{n}_{ij}\right)\mathbf{v}_h \\ &= \int_{\Omega_t}\mathbf{S}\mathbf{v}_h.\end{aligned} \tag{3.38}$$

Now we can formulate the definition of the *semi-discrete solution* $\boldsymbol{\Psi}_h = \boldsymbol{\Psi}_h\left((t,\boldsymbol{x})\right)$ *of MHD equations equation* (2.30) as

A) $\boldsymbol{\Psi}_h \in C^1\left((0,T), [V_h]^8\right)$,

B) equation (3.38) holds for all $t \in (0, T)$, and all $\mathbf{v} \in [V_h]^8$,

C) $\boldsymbol{\Psi}_h(0, \boldsymbol{x}) = \Pi_h \boldsymbol{\Psi}^0(\boldsymbol{x})$,

where Π_h is a projection of the initial condition $\boldsymbol{\Psi}^0$ onto $[V_h]^8$.

3.3 Divergence-free FE space

The divergence-free constraint of the magnetic field, $\boldsymbol{B} = 0$ (Gauss's law) is not enforced by the solution definition table 2.1. Therefore, we need to perform additional work to be sure that we do not have a non-physical solution in the sense that the constraint is not satisfied.

There are two often used approaches to handle this problem - the Constraint-Transport (CT) method, and divergence cleaning. The first one is not suitable for this work, as it constraints the triangulation in such a way, that implementing Adaptive Mesh Refinement would be very complicated, if possible at all. The second approach, the divergence cleaning methods need additional postprocessing step which may be omitted for the sake of calculation efficiency. The approach taken in this work is to replace the standard FE space equation (3.9) with basis functions equation (3.10) for the magnetic field part (B) with a vector-valued (3-dimensional) space V_h^B of functions that have exactly

$$\nabla \cdot \mathbf{v}_h^B = 0, \quad \mathbf{v}_h^B \in V_h^B, \tag{3.39}$$

where these functions are as before discontinuous on interfaces Γ_{ij}. The basis of space V_h^B for piecewise-linear functions can be selected in several ways, in this work, the following basis was selected:

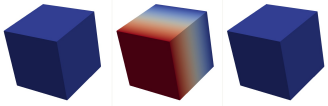
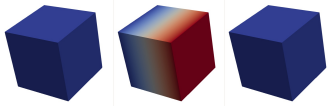
$\begin{pmatrix} B_x(x,y,z) \\ B_y(x,y,z) \\ B_z(x,y,z) \end{pmatrix}$	Visualization	$\begin{pmatrix} B_x(x,y,z) \\ B_y(x,y,z) \\ B_z(x,y,z) \end{pmatrix}$	Visualization
$\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$		$\begin{pmatrix} 0 \\ 0 \\ y \end{pmatrix}$	
$\begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$		$\begin{pmatrix} z \\ 0 \\ 0 \end{pmatrix}$	
$\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$		$\begin{pmatrix} 0 \\ z \\ 0 \end{pmatrix}$	
$\begin{pmatrix} y \\ 0 \\ 0 \end{pmatrix}$		$\begin{pmatrix} 0 \\ 0 \\ x \end{pmatrix}$	
$\begin{pmatrix} 0 \\ x \\ 0 \end{pmatrix}$		$\begin{pmatrix} x \\ -y \\ 0 \end{pmatrix}$	
$\begin{pmatrix} x \\ 0 \\ -z \end{pmatrix}$			

Table 3.1: Divergence-free space basis

In table 3.1, it is obvious that only one function actually has more nonzero components.

In what follows, the notation $[V_h]^8$ used before will be used for the space where the last three components are replaced by V_h^B. Note that there are some technicalities with respect to the usage of V_h^B needed to be handled in computation, e.g. in equation (3.66), or later in equations (3.55) and (3.57) that we do not explicitly attend to.

3.4 Discretization in time

Relations equation (3.16) represent a system of ordinary differential equations which can be solved by a suitable numerical method. Since we are interested in applying the Rothe's method (as opposed to the method of lines, which switches the order of discretization in time and space), we now want to discretize the time derivative. In order to do so, we consider a partition $0 = t_0 < t_1 < t_2 < ...$ of the time interval $(0, T)$ and set $\tau_k = t_{k+1} - t_k$. We use the notation $\boldsymbol{w}_h^k$ for the approximation of $\boldsymbol{w}_h(t_k)$.

3.4.1 Discrete problem

Then we apply a time discretization scheme, for example, the simple explicit *Euler method* and our *fully discrete problem* reads: for each $k > 0$ find $\boldsymbol{w}_h^{k+1}$ such that

A) ${\boldsymbol{\Psi}_h}^{k+1} \in [V_h]^8$,

B) For all test functions $\mathbf{v}_h \in [V_h]^8$:

$$\begin{aligned}\int_{\Omega_t} \frac{{\boldsymbol{\Psi}_h}^{k+1} - {\boldsymbol{\Psi}_h}^{k}}{\tau}\mathbf{v}_h &- \sum_{K^i \in T_h} \int_{K^i} \mathbf{F}\left(\boldsymbol{\Psi}_h^k\right)(\nabla \cdot \mathbf{v}_h) \\ &+ \sum_{\Gamma_{ij} \in \Gamma_I} \int_{\Gamma_{ij}} \mathbf{H}\left(\boldsymbol{\Psi}_h^k|_{ij}, \boldsymbol{\Psi}_h^k|_{ji}, \boldsymbol{n}_{ij}\right)\mathbf{v}_h \\ &+ \sum_{\Gamma_{ij} \in \Gamma_B} \int_{\Gamma_{ij}} \mathbf{H}\left(\boldsymbol{\Psi}_h^k|_{ij}, \overline{\boldsymbol{\Psi}_h^k|_{ji}}, \boldsymbol{n}_{ij}\right)\mathbf{v}_h \\ &= \int_{\Omega_t} \mathbf{S}\mathbf{v}_h,\end{aligned} \tag{3.40}$$

C) ${\boldsymbol{\Psi}_h}^0(\boldsymbol{x}) = \Pi_h \boldsymbol{\Psi}^0(\boldsymbol{x})$,

where Π_h is a projection of the initial condition $\boldsymbol{\Psi}^0$ onto $[V_h]^8$.

3.4.2 Time step length

Time step length is an important attribute of the discretization. If it is too small, the calculation might be taking too long to finish, with unnecessary precision with respect to time. If it is too large, we may end up with unstable calculation and obtain results with nonphysical oscillations, or without a solution whatsoever. That is why we need to take extra care to derive the proper value. From the stability perspective, we have a condition for the upper bound of the time step - this condition is called the *Courant-Friedrichs-Lewy* condition. This condition is of the following form:

$$\tau_{max} = \min\left\{\frac{\Delta_{xmin}}{c_{max}}, \frac{\Delta_{x_{min}}^2}{2\eta_{max}}\right\}, \tag{3.41}$$

where Δ_{xmin} is the smallest dimension of any element, η_{max} highest resistivity in the domain, and v_{max} highest velocity in the domain, where the following velocities are taken into account:

$$c_s = \sqrt{\frac{\gamma\left(\gamma-1\right)}{\rho}\left(U - \rho v^2 - U_B\right)}, \tag{3.42}$$

$$c_a = \sqrt{\frac{B^2}{\rho}}, \tag{3.43}$$

$$u, \tag{3.44}$$

where c_s is the speed of sound, c_a is the Alfvén speed, and u is the speed of plasma. We then take

$$c_{max} = \max\left\{c_s, c_a, u\right\}.$$

3.5 Algebraic formulation

Last step in the DG method discretization is to transform the system of equations equation (3.40) into a system of linear algebraic equations at every time step t_k and obtain the solution at this time step as the solution of this linear algebraic system.

First, we rearrange the system in the following manner:

$$\begin{aligned}\sum_{K^i \in T_h} \int_{K^i} \mathbf{v}_h {\mathbf{\Psi}_h}^{k+1} = \sum_{K^i \in T_h} \int_{K^i} \left[{\mathbf{\Psi}_h}^k + \tau \mathbf{S} + \tau \mathbf{A} \left(\mathbf{\Psi}_h^k \right) (\nabla \cdot \mathbf{v}_h) \right] \mathbf{v}_h \\ - \sum_{\Gamma_{ij} \in \Gamma_I} \int_{\Gamma_{ij}} \mathbf{H} \left(\mathbf{\Psi}_h^k|_{ij}, \mathbf{\Psi}_h^k|_{ji}, \boldsymbol{n}_{ij} \right) \mathbf{v}_h \\ - \sum_{\Gamma_{ij} \in \Gamma_B} \int_{\Gamma_{ij}} \mathbf{H} \left(\mathbf{\Psi}_h^k|_{ij}, \overline{\mathbf{\Psi}_h^k|_{ji}}, \boldsymbol{n}_{ij} \right) \mathbf{v}_h.\end{aligned} \tag{3.45}$$

We can see that the left hand side does not depend on the previous solution values, so there is no need to recalculate the matrix entries in every time step (unless we employ AMR, in which case the mesh and therefore the set of basis functions changes). Now

$${\mathbf{\Psi}_h}^{k+1} = \sum_{l=0}^{l=L} y_l \mathbf{v}_{hl}, L = \dim \left([V_h]^8 \right) \tag{3.46}$$

for some (obviously finite) basis $\{v_{h1}, ..., v_{hL}\}$ of $[V_h]^8$. Next, since $\mathbf{\Psi}_h^k$, τ, S, $\mathbf{A}$ (and the basis) are all known, we can define

$$a_{lm} = \sum_{K^i \in T_h} \int_{K^i} \mathbf{v}_{hl} \mathbf{v}_{hm}, \tag{3.47}$$

$$\begin{aligned}b_l = \sum_{K^i \in T_h} \int_{K^i} \left[{\mathbf{\Psi}_h}^k + \tau \mathbf{S} + \tau \mathbf{A} \left(\mathbf{\Psi}_h^k \right) (\nabla \cdot \mathbf{v}_{hl}) \right] \mathbf{v}_{hl} \\ - \sum_{\Gamma_{ij} \in \Gamma_I} \int_{\Gamma_{ij}} \mathbf{H} \left(\mathbf{\Psi}_h^k|_{ij}, \mathbf{\Psi}_h^k|_{ji}, \boldsymbol{n}_{ij} \right) \mathbf{v}_{hl} \\ - \sum_{\Gamma_{ij} \in \Gamma_B} \int_{\Gamma_{ij}} \mathbf{H} \left(\mathbf{\Psi}_h^k|_{ij}, \overline{\mathbf{\Psi}_h^k|_{ji}}, \boldsymbol{n}_{ij} \right) \mathbf{v}_{hl},\end{aligned} \tag{3.48}$$

$$A = \{a_{lm}\}_{l,m=1}^{l,m=L}, \tag{3.49}$$

$$b = \{b_l\}_{l=1}^{l=L}, \tag{3.50}$$

$$y = \{y_l\}_{l=1}^{l=L}, \tag{3.51}$$

and rewriting equation (3.45) using equation (3.47) - equation (3.51), we come to the *fully discrete algebraic problem at time instance* t_{k+1}:

$$Ay = b, \tag{3.52}$$

whose well-posedness, and other attributes that allow for a successful solution of this system, come from the properties of the DG method. Now if we solve the system equation (3.52), and obtain the solution vector y, we are able to reconstruct

the discrete solution ${\boldsymbol{\Psi}_h}^{k+1} \in [V_h]^8$ using the relation equation (3.46).

In the implementation, we take the elements $K \in T_h$, of the triangulation T_h to be hexahedra.

3.6 Numerical integration

Evaluation of the integral values in equations (3.47) and (3.48) is performed using the *Gaussian numerical quadrature*. A quadrature rule approximates the integral values by replacing the integral as a weighted sum of integrand values at specified points in the domain of integration. The Gaussian quadrature is constructed so that the approximation is exact for polynomials of degree $2n$ - 1 (and less). This is acceptable, as our space V_h is constructed using polynomials - see section section 3.2.1. We only need to take the value n to be corresponding to the value of p_m for the element K_m. The rule for both a 2-dimensional element face Γ, and a 3-dimensional cube K is derived from a one-dimensional approximation (where the interval $[-1, 1]$ is a convention):

$$\int_{-1}^{1} f(x)\,dx = \sum_{i=1}^{n} w_i f(x_i),$$

where the numbers $w_i > 0$ are the *quadrature weights*, and the points (numbers in this case) x_i are the *quadrature points*, in the following way:

$$\int_{\Gamma} f(\boldsymbol{x})\ dx = \int_{-1}^{1}\int_{-1}^{1} f(x_1, x_2)\ dx \approx \sum_{i=1}^{n}\sum_{j=1}^{n} w_i w_j f(x_{1i}, x_{2j}),$$

$$\int_{K} f(\boldsymbol{x})\ dx = \int_{-1}^{1}\int_{-1}^{1}\int_{-1}^{1} f(x_1, x_2, x_3)\ dx \approx \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n} w_i w_j w_k f(x_{1i}, x_{2j}, x_{3k}),$$

and transformation to a generic rectangular hexahedron is performed using the transformation in one dimension:

$$\int_a^b f(x)\,dx \approx \frac{b-a}{2}\int_{-1}^{1} f\left(\frac{b-a}{2}x + \frac{a+b}{2}\right)\,dx.$$

Applying the Gaussian quadrature rule then results in the following one-dimensional approximation:

$$\int_a^b f(x)\,dx \approx \frac{b-a}{2}\sum_{i=1}^{n} w_i f\left(\frac{b-a}{2}x_i + \frac{a+b}{2}\right).$$

And the transformations in higher dimensions follow naturally. For $\Gamma = [a_1, b_1] \times [a_2, b_2]$ we have:

$$\int_\Gamma f(x)\, d\boldsymbol{x} \approx \frac{b_2 - a_2}{2} \frac{b_1 - a_1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} w_i w_j f\left(\frac{b_1 - a_1}{2} x_i + \frac{a_1 + b_1}{2}, \frac{b_2 - a_2}{2} x_j + \frac{a_2 + b_2}{2} \right).$$

Taking now e.g. equation (3.47), and notation for quadrature points and weights, we can write (omitting the operand $\boldsymbol{x} = (x_1, x_2, x_3)$):

$$a_{lm} = \sum_{K^i \in T_h} \int_{K^i} \mathbf{v}_{hl} \mathbf{v}_{hm}, \tag{3.53}$$

$$a_{lm} := \sum_{K^i \in T_h} \int_{K^i} f\left(\mathbf{v}_{hl}, \mathbf{v}_{hm}\right), \tag{3.54}$$

$$a_{lm} \approx \sum_{K^i \in T_h} \sum_{\boldsymbol{j}=\overrightarrow{1}}^{\overrightarrow{n}} f\left(\mathbf{v}_{hl}\left(\boldsymbol{x}^i_{\boldsymbol{j}}\right), \mathbf{v}_{hm}\left(\boldsymbol{x}^i_{\boldsymbol{j}}\right)\right) w_{\boldsymbol{j}}, \tag{3.55}$$

where $\boldsymbol{j}$ is a multi-index used in sum over (volumetric) quadrature points $\boldsymbol{x}^i_{\boldsymbol{j}} \in K^i$. Similarly for the right-hand side (equation (3.48)):

$$\begin{aligned} b_l = & \sum_{K^i \in T_h} \int_{K^i} \left[{\boldsymbol{\Psi}_h}^k + \tau \mathbf{S} + \tau \mathbf{A}\left(\boldsymbol{\Psi}^k_h\right)(\nabla \cdot \mathbf{v}_{hl}) \right] \mathbf{v}_{hl} \\ & - \sum_{\Gamma_{ij} \in \Gamma_I} \int_{\Gamma_{ij}} \mathbf{H}\left(\boldsymbol{\Psi}^k_h|_{ij}, \boldsymbol{\Psi}^k_h|_{ji}, \boldsymbol{n}_{ij}\right) \mathbf{v}_{hl} \\ & - \sum_{\Gamma_{ij} \in \Gamma_B} \int_{\Gamma_{ij}} \mathbf{H}\left(\boldsymbol{\Psi}^k_h|_{ij}, \overline{\boldsymbol{\Psi}^k_h|_{ji}}, \boldsymbol{n}_{ij}\right) \mathbf{v}_{hl}, \end{aligned} \tag{3.56}$$

$$b_l := \sum_{K^i \in T_h} \int_{K^i} g\left(\mathbf{v}_{hl}\right) - \sum_{\Gamma_{ij} \in \Gamma_I} \int_{\Gamma_{ij}} g^{'}\left(\mathbf{v}_{hl}\right) - \sum_{\Gamma_{ij} \in \Gamma_B} \int_{\Gamma_{ij}} g^{''}\left(\mathbf{v}_{hl}\right) \tag{3.57}$$

$$\begin{aligned} b_l \approx & \sum_{K^i \in T_h} \sum_{\boldsymbol{j}=\overrightarrow{1}}^{\overrightarrow{n}} g\left(\mathbf{v}_{hl}\left(\boldsymbol{x}^i_{\boldsymbol{j}}\right)\right) w_{\boldsymbol{j}} \\ & - \sum_{\Gamma_{ij} \in \Gamma_I} \sum_{\boldsymbol{j}_f=\overrightarrow{1}}^{\overrightarrow{n_f}} g^{'}\left(\mathbf{v}_{hl}\left(\boldsymbol{x}^{ij}_{\boldsymbol{j}_f}\right)\right) w_{\boldsymbol{j}_f} \\ & - \sum_{\Gamma_{ij} \in \Gamma_B} \sum_{\boldsymbol{j}_f=\overrightarrow{1}}^{\overrightarrow{n_f}} g^{''}\left(\mathbf{v}_{hl}\left(\boldsymbol{x}^{ij}_{\boldsymbol{j}_f}\right)\right) w_{\boldsymbol{j}_f}, \end{aligned} \tag{3.58}$$

where in addition to $\boldsymbol{j}$ as explained before, $\boldsymbol{j}_f$ is a multi-index used sum over face quadrature points $\boldsymbol{x}^{ij}_{\boldsymbol{j}_f} \in \Gamma_{ij}$. Based on this, we can define

$$a_{lmi\boldsymbol{j}} = f\left(\mathbf{v}_{hl}\left(\boldsymbol{x}^i_{\boldsymbol{j}}\right), \mathbf{v}_{hm}\left(\boldsymbol{x}^i_{\boldsymbol{j}}\right)\right) w_{\boldsymbol{j}}, \tag{3.59}$$

$$b_{li\boldsymbol{j}} = g\left(\mathbf{v}_{hl}\left(\boldsymbol{x}^i_{\boldsymbol{j}}\right)\right) w_{\boldsymbol{j}}, \tag{3.60}$$

$$b'_{lij\boldsymbol{j}_f} = \begin{cases} g'\left(\mathbf{v}_{hl}\left(\boldsymbol{x}^{ij}_{\boldsymbol{j}_f}\right)\right) w_{\boldsymbol{j}} & \text{if } \Gamma_{ij} \in \Gamma_I \\ g''\left(\mathbf{v}_{hl}\left(\boldsymbol{x}^{ij}_{\boldsymbol{j}_f}\right)\right) w_{\boldsymbol{j}} & \text{if } \Gamma_{ij} \in \Gamma_B \end{cases}. \tag{3.61}$$

3.7 Assembling the algebraic problem

Now we have a clear expression how to evaluate the integral values equations (3.47) and (3.48) using equations (3.59) to (3.61), but we need to construct the matrix A (equation (3.49)), and the right-hand-side vector b (equation (3.50)) in an effective manner. This is generally achieved through a *element-wise* assembling of these structures. Key to this is to create a data structure that identifies for a particular element K^i all the test functions $\mathbf{v}_{hl}$ that make sense to be evaluated (have non-empty support) on K^i, i.e. we are looking for the set

$$\mathbf{v}_h\left(K^i\right) = \left\{\mathbf{v}_h \in V_h : supp\left(\mathbf{v}_h\right) \cap K^i \neq \emptyset\right\}, \tag{3.62}$$

and do the same for the faces Γ_i (both boundary, and internal):

$$\mathbf{v}_h\left(\Gamma_i\right) = \left\{\mathbf{v}_h \in V_h : supp\left(\mathbf{v}_h\right) \cap \Gamma_i \neq \emptyset\right\}. \tag{3.63}$$

Now the assembling procedure looks like this:

Algorithm 1: Assembling of the algebraic problem equation (3.52)

```
# 1 - Loop over elements
foreach K^i ∈ T_h do
    Data: Quadrature points {x_1^i, ..., x_n^i}
    Data: Jacobian of the mapping J_{K^i} mapping the reference element
          (unit cube) to the actual element
    Data: Quadrature weights {w_1, ..., w_n}
    # Loop over quadrature points
    foreach j ∈ {1, ..., n} do
        Set: (JxW)_j = J_{K^i} × w_j
        # Loop over test functions
        foreach v ∈ v_h(K^i) do
            Data: l - index of v in the global system, i.e. row in
                  equation (3.49) - equation (3.51)
            # Loop over basis functions
            foreach u ∈ v_h(K^i) do
                Data: m - index of u in the global system, i.e. column in
                      equation (3.49)
                a_lm += (JxW)_j a_lmij
            b_l += (JxW)_j b_lij
# 2 - Loop over faces
foreach Γ_ij ∈ T_h do
    Data: Quadrature points {x_1^ij, ..., x_{n_f}^ij}
    Data: Jacobian of the mapping J_{K^i} = J_{K^-} mapping the reference face
          (unit square) to the actual face, where K^i, K^j are elements
          adjacent to Γ_ij if this is an internal face, or K^i = K^j if this is a
          boundary face.
    Data: Quadrature weights {w_1, ..., w_{n_f}}
    # Loop over quadrature points
    foreach j_f ∈ {1, ..., n_f} do
        Set: (JxW)_{j_f} = J_{K^i} × w_{j_f}  # Here it does not matter if we
         choose J_{K^i} or J_{K^j}
        # Loop over test functions
        foreach v ∈ v_h(K^i) do
            Data: l - index of v in the global system, i.e. row in
                  equation (3.49) - equation (3.51)
            b_l += (JxW)_{j_f} b'_{lij j_f}
```

An important remark needs to be added here, with respect to the fact, that we aim for a distributed computation. As stated before, the domain decomposition approach leads to each of the processors $P_i \in \{P_0, ..., P_N\}$ of the total number of processors employed for the computation owning only a subset of elements $\{K \in T\}_i$.

If the processor P_i did not have any data about other elements, we would not be able to perform the evaluation involving $b^{'}$ defined in equation (3.61), or more precisely $g^{'}$ defined in equation (3.57) as the values $\boldsymbol{\Psi}_h^k|_{ji}$ for faces Γ_{ij} of elements from $\{K \in T\}_i$ will not always be there.

To amend this situation, in the distributed triangulation, each processor holds not only the mesh element data it owns, but also data for mesh elements it needs - in Discontinuous Galerkin method, for exactly the purposes of internal face integral evaluation described in the previous paragraph. This data is called *ghost elements*, or *ghost cells*, as these are only copies provided by other processors which own the particular elements. This is illustrated in figure 3.14, which is the same case of distributed triangulation as shown in figure 3.1.

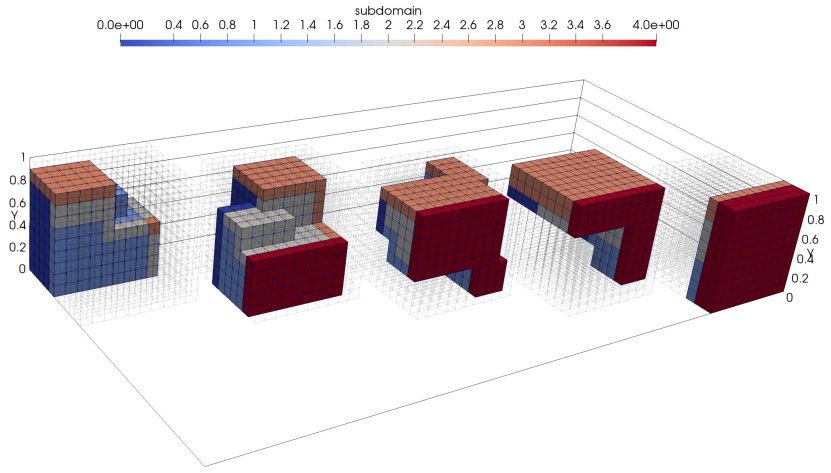


Figure 3.14: Processor-owned elements (0..4 left to right), with color-coded ghost elements from other processors.

Note that in the described algorithm, and also with respect to the remarks in the previous paragraphs, periodic boundary conditions are not specifically handled, as they are only a technically more complex case of internal faces (both from the perspective of necessity of utilizing ghost cells, and from the perspective of evaluating b_l in algorithm 1.

3.8 Slope limiting

It is well known Dolejší and Feistauer (2015) that the Discontinuous Galerkin method exhibits nonphysical spurious oscillations in the vicinity of sharp discontinuities. Noteworthy is the fact, that with continuous Finite Element spaces, the situation is even worse, as the oscillations tend to propagate through the computational domain. With the DG method, the problem is localized to a single layer of

elements bordering any sharp front. This behavior is not acceptable, and measures must be taken to eliminate such oscillations - methods aiming at solving this are usually labeled as flux limiters, slope limiters, or shock capturing schemes.

These methods can be categorized according to multiple aspects. Out of these, two are important from the perspective of this work. First categorization is whether the approach changes the equations by introducing additional term that 'smoothes' the solution near the sharp front (this may be understood as a form of *artificial viscosity / resistivity*) - such approach is proposed e.g. in Denner et al. (2017).

In this work, such an approach is not preferred, as we aim at implementing a generally usable solver, where extensive analysis of the impact of a change in the governing equations for the particular problem is not possible.

Second categorization worth mentioning is whether the particular approach is suitable for multi-scale phenomena, where the solution can exhibit large jumps in all possible configurations with respect to the (non-uniformly refined) mesh. From a pragmatic perspective, a rather simple slope and robust limiting technique that does not change the governing equations is the Barth-Jespersen limiter Barth and Jespersen (1989). The Barth-Jespersen limiter considers the piecewise-linear solution in the form

$$u_h(x) = u_c + \alpha_e (\nabla u)_c \cdot (x - x_c), \ 0 \leq \alpha_e \leq 1, \tag{3.64}$$

obtained using the linear Taylor basis functions (description of this shapeset is given in Kuzmin (2010)), where u_c is the cell average on the element K, and $(\nabla u)_c$ is the gradient of the solution on the element K.

The sought parameter α_K is the parameter (correction factor) that determines the maximum admissible slope and is defined as:

$$\alpha_K = \min_i \begin{cases} \min\left\{1, \frac{u_K^{max} - u_c}{u_i - u_c}\right\}, & if \quad u_i - u_c > 0, \\ 1, & if \ u_i - u_c = 0, \\ \min\left\{1, \frac{u_K^{min} - u_c}{u_i - u_c}\right\}, & if \quad u_i - u_c < 0, \end{cases} \tag{3.65}$$

where $u_i = u_c + (\nabla u)_c \cdot (x_i - x_c)$ is the unconstrained solution value at the vertex x_i, and u_K^{min}, u_K^{max} are the minimum and maximum cell averages of all elements sharing a common face with the element K.

However, using this limiting technique, there are two suboptimal behavior features - the bounds for the limited solution are set on one hand too tight - solution values from elements meeting at a vertex but having no common face are not taken into account (they might extend the interval for the admissible correction factor α_e, and on the other hand too loose - solution values on elements that share a common face

may extend the admissible interval for α_e based on the value at a vertex that does not belong to that particular common face.

Both these two problems are solved in the slope limiting technique chosen for this work

3.8.1 Vertex-based limiter

Introduced by D. Kuzmin in Kuzmin (2010), the Vertex-based limiter aims at being an improvement over the Barth-Jespersen limiter. It also considers the solution in the form equation (3.64), but the definition of the correction factor α_K differs:

$$\alpha_K = \min_i \begin{cases} \min\left\{1, \frac{u_i^{max}-u_c}{u_i-u_c}\right\}, & if \quad u_i - u_c > 0, \\ 1, & if \; u_i - u_c = 0, \\ \min\left\{1, \frac{u_i^{min}-u_c}{u_i-u_c}\right\}, & if \quad u_i - u_c < 0, \end{cases} \tag{3.66}$$

where in this case u_i^{min}, u_i^{min} are defined in such a way that for each of the vertices they are initialized with a small and a large constant, respectively, and then in the loop over all elements that contain the $i-$th vertex, the values are updated as:

$$u_i^{max} = \max\left\{u_c, u_i^{max}\right\}, \quad u_i^{min} = \min\left\{u_c, u_i^{min}\right\}. \tag{3.67}$$

This slope limiting technique proves to have all the required attributes from the perspective of this work. The algorithm implementing this technique looks is presented

in algorithm 2

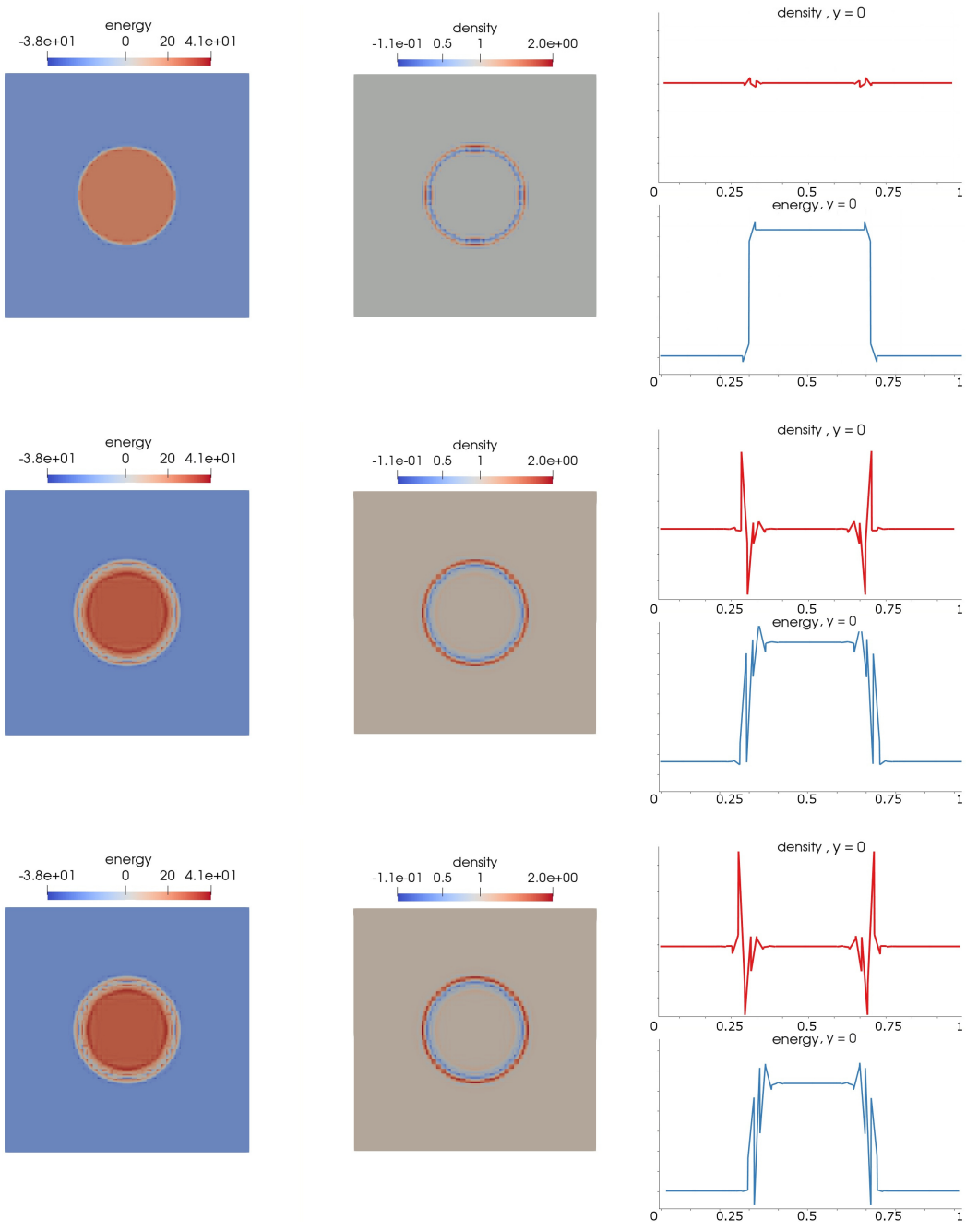
Algorithm 2: Limiting the solution using the vertex-based limiter.

```
# Loop over elements
foreach K ∈ T_h do
    # Loop over solution components
    foreach k = 1, ..., 8 do
        # In the rest of the algorithm, u will stand for u^k.
        Data: Center value of the solution u_c^K = u(x_c^K) at point x_c^K which
              is the center of K
        Data: Set of linear basis functions coefficients obtained as:
        # Loop over basis functions for the component k
        foreach
         v_1^k := v ∈ {v ∈ v_h(K), component of v = k, v piecewise linear} do
            Data: l(v) - index of v in the global system, i.e. row in
                  equation (3.49) - equation (3.51)
            Data: u_{l(v)} - solution coefficient for (v)
        Set α_K = 1
        # Loop over vertices
        foreach v_i, i = 1, ..., 8, v_i is a vertex of K do
            Data: Vertex value u_i = u(v_i)
            Data: Set of vertex neighbors {K'}_i, where v_i ∈ K' and
                  K' ≠ K
            # Loop over elements sharing the vertex v_i
            foreach K' ∈ {K'}_i do
                Data: Center value of the solution u_c^{K'} = u(x_c^{K'}) at point
                      x_c^{K'} which is the center of K'
            Calculate u_i^max = max{u_i, max{u_c^{K'}}}
            Calculate u_i^min = min{u_i, min{u_c^{K'}}}
            Evaluate equation (3.66), updating α_K
        # Loop over basis functions for the component k
        foreach v ∈ v_1^k do
            u_{l(v)} = α_K u_{l(v)}
```

Comparison of limited and unlimited solution

For illustration, the same problem as in section 3.2.3 is considered, which is suitable for illustrating the undershoots and overshoots, as the problem contains a sharp front where this behavior is clearly visible. In the next Figures, first several snapshots of an unlimited solution, and then several snapshot of a limited solution are presented. Note that the unlimited solution can't progress beyond a certain point, as the undershoots and overshoots get so large that the density values get to be

negative, therefore e.g. calculating its square root needed for the speed of sound evaluation is impossible. Even before that, we get to non-physical regime, and e.g. the energy values start growing beyond limit.



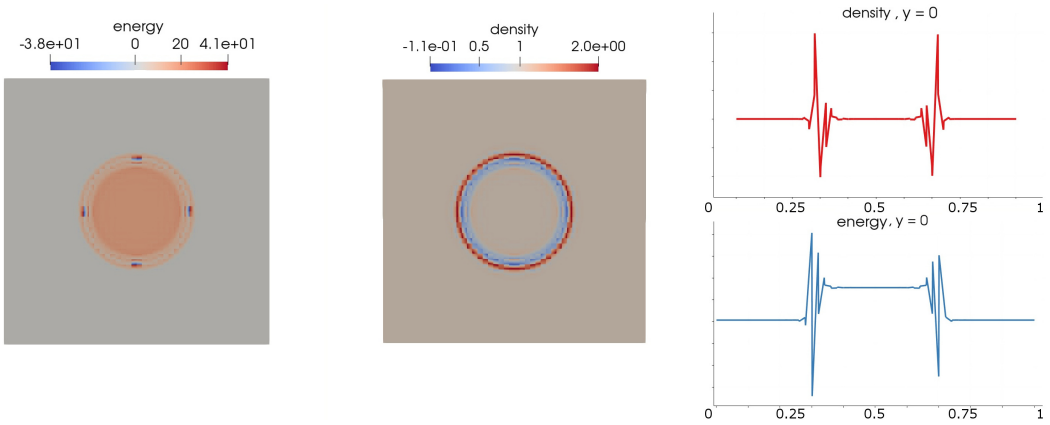
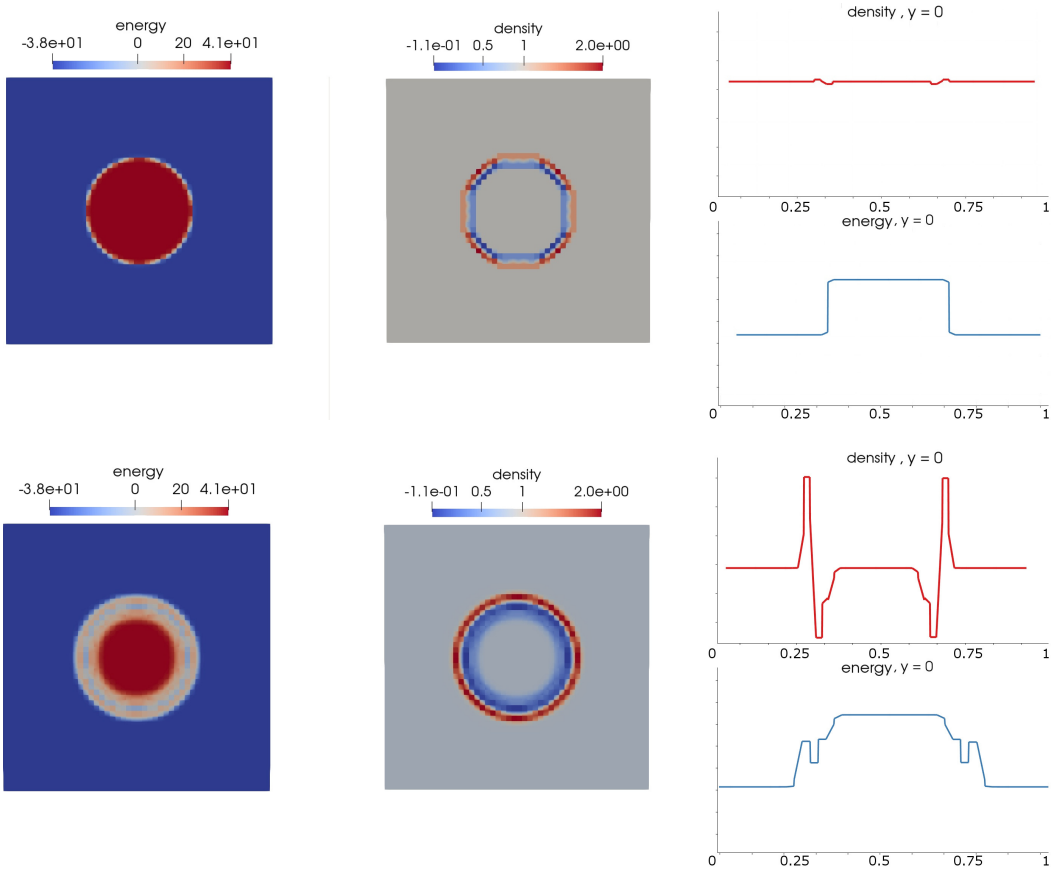


Figure 3.15: Unlimited solution - Energy, density, and their values over line y = 0, snapshots for first 4 time steps. The solution cannot progress beyond the last snapshot, as the oscillations are orders of magnitude larger than the physical solution



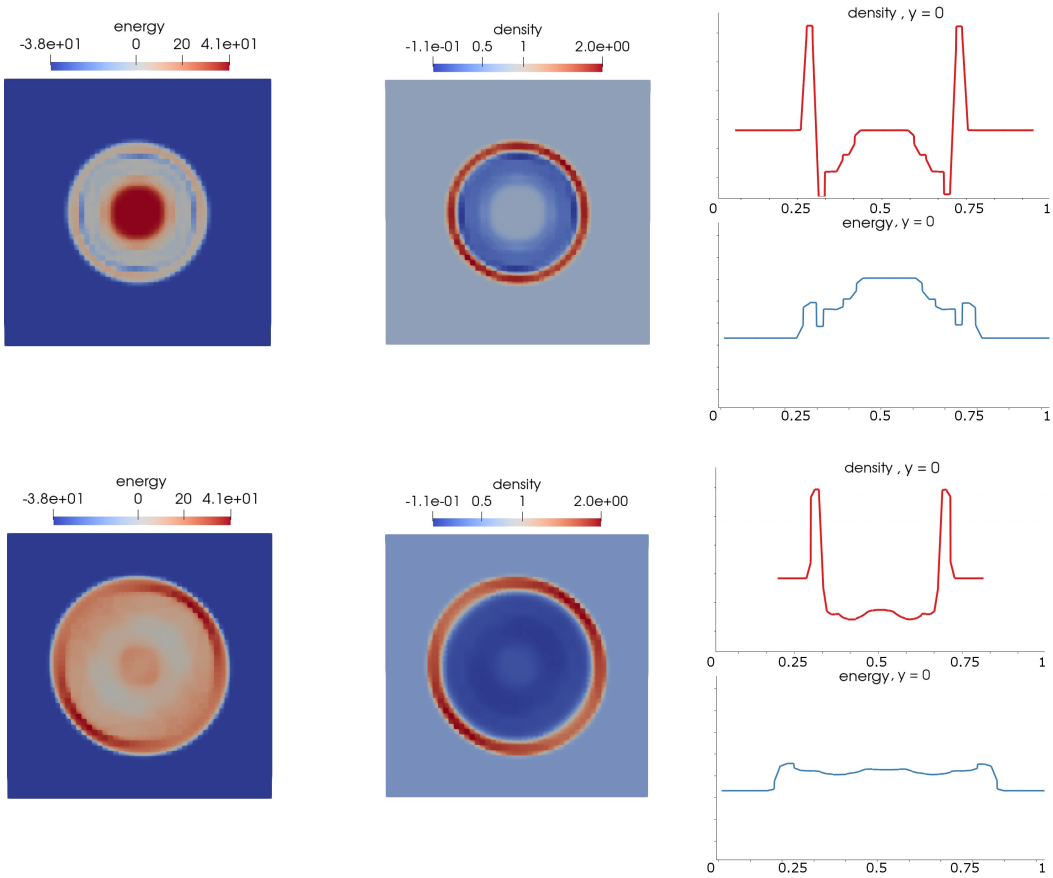


Figure 3.16: Solution limited with Vertex-based limiter - Energy, density, and their values over line y = 0, snapshots for $t = 0, t = 1.e-3, t = 2.e-3, t = 5.e-3$

3.9 Time-stepping and linearization

3.9.1 Time-stepping

The simple time discretization scheme that we use in section 3.4.1 allows us to simply implement the time-stepping in the following fashion:

Algorithm 3: Time-stepping procedure

Set: $y_0 =$ (initial solution)
Set: $ts = 1$ # initial time step
Set: $t = 0.00$ # initial time
# Loop over time steps
for ; $t < T$; $t = t + \tau$, $ts = ts + 1$ **do**
  Data: Solution from the previous time step y^{ts-1}
  1.Call procedure algorithm 1 to obtain $A, b\left(y^{ts-1}\right)$
  2.Solve the problem $Ay^{ts} = b\left(y^{ts-1}\right)$ # See note below
  3.(If necessary) postprocess y^{ts} using algorithm 2
  4.Calculate updated value of τ using equation (3.41)

Note The step 2 (solving the algebraic problem) is of course a key point in the overall process. Because of its importance, the aim of this work is not to describe, or even implement an algebraic solver for this purpose. Many scientific teams have spent many years on publicly available open-source solvers that are usable by the software we develop for the purpose of solving the MHD phenomena. We use the existing solvers.

3.10 Performance considerations

3.10.1 Parallelization

To be able to perform any large scale calculations, we need to be able to utilize the performance of hardware at maximum. Being able to use modern, multi-core computers is an absolute must to achieve good performance, as the execution time when using parallel execution can decrease by a factor of corresponding to the number of cores - and modern machines have tens of cores available.

The parallelization is possible at several places in the overall algorithm algorithm 1. But it makes most sense to parallelize the outer-most loop over elements, and over faces.

Another point for parallelization is the algebraic solver. As explained in section 3.9.1, we rely on existing software packages for finding the solution of the algebraic system equation (3.52) - all the used solvers support and heavily utilize parallelization.

3.10.2 Vectorization

Similarly as in section section 3.10.1, the goal here is to be able to solve the discretized problem in the most efficient (fastest) way possible. One of the features that (modern) hardware offers is to employ vectorization instructions - i.e. unary or binary instructions that operate on $N, N > 1$ values (in case of unary instructions),

or $N, N > 1$ pairs of values (in case of binary instructions) at the same time - the number N depends on the capabilities of the CPU, and on the precision (single or double). On the hardware that was available to perform calculations for this thesis (and based on always-used double precision), $N = \{4, 8\}$, using such instructions requires both using them in the code (one has to specify that these instructions shall be generated explicitly), and having the compiler aware of these, and able to utilize them in the generated machine code - both compilers used for work on this thesis (GNU gcc, and Microsoft Visual Studio) support vectorization instructions (SSE, SSE2, AVX, AVX2).

Vectorization helps heavily with respect to CPU time spent on calculation. In the algorithm algorithm 1, vectorized instructions are used in:

- evaluation of the expressions $a_{lm} += JxW_{\boldsymbol{j}}\ a_{lmK\boldsymbol{j}}$
- evaluation of expressions $b_l += JxW_{\boldsymbol{j}}\ b_{lK\boldsymbol{j}}|_{face|volumetric}$
- calculation of $JxW_{\boldsymbol{j}}$

3.10.3 Distribution

As discussed in section 3.1.1, the *domain decomposition* approach is taken to overcome the physical limitations (CPU physical size, RAM capacity, ...) of a single machine with shared memory.

This approach has several aspects, that are worth mentioning. The entire process of discretization of MHD equations (as well as any other system of PDEs) eventually leads to solving large (sparse) systems of linear equations, and then interpretation of the solution in terms of a global (defined on entire Ω) function - that is a linear combination of the global basis functions with the solution of the linear problem being the coefficients of this linear combination. From this it follows, that distributing only the triangulation would not by itself lead to radical increase of the size of problems we can tackle, there are other points where the algorithms employed need to take distribution of data among processors into account:

A) Distributed matrix and right hand side
   - It is necessary for each processor to be able to utilize the memory on the node it physically belongs to when writing values of calculated integrals into the algebraic structure (see steps $a_{lm} += ...$, $b_l += ...$ of algorithm 1).
   - Distribution of the matrix is actually very important from the memory capacity perspective. It is typically the largest data structure (together with preconditioner) used in the entire implementation - regardless whether the used method is FEM, DGFEM, or Finite Volumes for example.

B) Distributed algebraic solver When the algebraic structures (the matrix and the right hand side) are completed, the sought solution must again be sought in a distributed manner, otherwise the added cost would be transferring algebraic data from all processors to a common one, where the solution would be sought. This is, however, a theoretical possibility, as the data structures used in the solution of the algebraic problem (decompositions, preconditioners, ...) are typically too large to be stored on a single processor anyway.

C) Distributed solution

- Although it is quite obvious, it is noteworthy that as the algebraic structures are distributed, and so is the solution of the algebraic problem, the actual solution is again, distributed according to the domain decomposition.
- The distributed solution is the data structure that is most important from the perspective of the *ghost cells* - illustrated in figure 3.14, where for the distributed discontinuous Galerkin method, we need to be able to access the distributed solution values from all neighboring elements when performing numerical integration equation (3.61).

Message Passing Interface and deal.II

For all distributed computing purposes in this work, deal.II implementation of the Message Passing Interface (MPI) was used. MPI is a specification for a standard library for message passing for the purposes of distributed computing, and was originally introduced in Hariri et al. (1993). The library deal.II (Bangerth et al. (2015)) offers wrappers for low-level MPI functions, that are utilizable in the implementation of the methods of this work.

4 Adaptive Mesh Refinement

As the Adaptive Mesh Refinement (AMR) is a very important algorithm in the overall numerical solution, handling the multi-scale aspect of the studied problems, in this chapter, a description of what needs to be taken care of for the DG method, in the distributed settings, and what concrete decisions were made during this work, and justification of these in light of the requirements on the numerical solution.

Note that in what follows, the symbol T or $T_{i\in\{0,1,2,\dots\}}$ shall denote a general computational mesh and a particular mesh in the series of refined meshes respectively. We shall drop the subscript h from T_h referring to the dimension (maximum of diameters of all elements) of the mesh where possible.

4.1 Overview of the AMR

The general schema of any Adaptive Mesh Refinement algorithm is described in the algorithm algorithm 4:

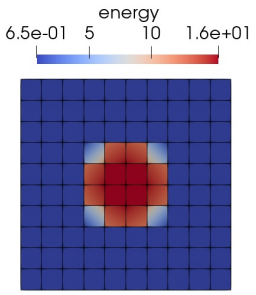
Algorithm 4: Generic AMR algorithm

Data: Mesh T_0
Result: A mesh T_n and a solution $\boldsymbol{y}_n$ on this mesh satisfying the solution acceptance criteria

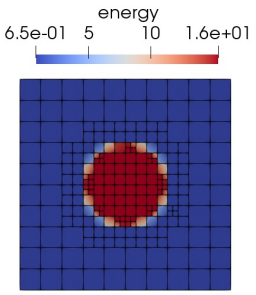
```
i = 0
while true do
    obtain solution y_i on T_i
    evaluate solution y_i acceptance criteria
    if solution acceptance criteria satisfied then
        return
    else
        identify subset T_i^r of all elements K ∈ T_i to be refined, T_i^r ⊆ T_i
        obtain T_{i+1} by refining (at least) all K ∈ T_i^r
        i = i + 1
```

In algorithm 4, *solution acceptance criteria* is usually either spatial error estimate threshold, or number of elements, etc. In the same description, the note that there might be other elements $K \notin T_i^r$ refined in order to maintain some desired attributes of the mesh, such as 1-irregularity (the refinement level difference of two elements sharing a common face is at most one), smoothness of mesh (there is e.g. no unrefined elements for which all, or a majority of neighboring elements would be refined), etc. An example of several steps of the algorithm (where step number

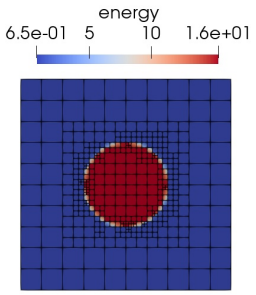
corresponds to the variable i in algorithm 4, is given in figure 4.1 below on a sample problem in 2 dimensions.



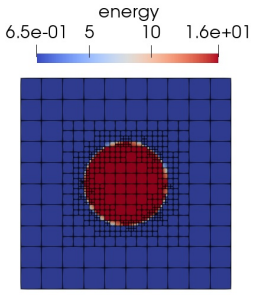
(a) AMR step 0 - 100 elements



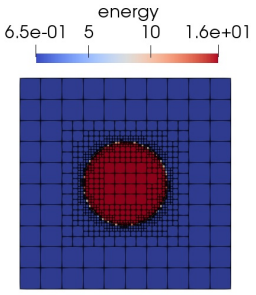
(b) AMR step 1 - 232 elements



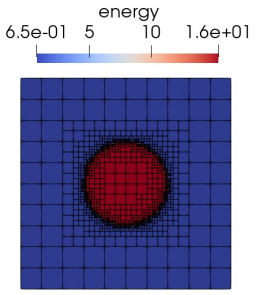
(c) AMR step 2 - 379 elements



(d) AMR step 3 - 610 elements



(e) AMR step 4 - 1016 elements



(f) AMR step 5 - 1364 elements

Figure 4.1: AMR steps

The benefit of AMR is clear. If we were to discretize the entire domain with elements small enough to capture the solution with the same quality as in the last AMR step, we would end up with > 40000 elements, where with AMR, the same is achieved with < 1400.

Since we are dealing with evolution equations, it is necessary to specify how the AMR algorithm relates with the non-AMR solution algorithm algorithm 3. There are several points we need to take into considerations:

- slope limiting as a postprocessing step after the solution must not be omitted in case of higher-order (e.g. linear) basis functions
- the solution needs to be transferred to the refined mesh in order to be able to assemble the matrix and the right-hand side on the refined mesh in the next adaptivity iteration

- since the solution evolves and the refinements that contribute to error reduction at time step n do not contribute to error reduction at time step $n+m$ (the solution was e.g. oscillating or potentially discontinuous at time step n, but is smooth at time step $n+m$), we also want to revert such refinements as the simulation time progresses, we call this process *coarsening* of elements. For this we shall define a set T_i^c of all elements to be coarsened.

The algorithm looks like this:

Algorithm 5: AMR for time-discretized problems

Set: $y_0 =$ (initial solution)
Set: $ts = 1$ # initial time step
Set: $t = 0.00$ # initial time
# Loop over time steps
for ; $t < T$; $t = t + \tau$, $ts = ts + 1$ **do**
  Data: Solution from the previous time step y^{ts-1} expressed on the mesh T_0^{ts}
  i = 0
  while *true* **do**
    call procedure algorithm 1 to obtain $A_i, b_i\left(y_i^{ts-1}\right)$ on the mesh T_i^{ts}
    solve the problem $A_i y_i^{ts} = b_i\left(y_i^{ts-1}\right)$
    post-process the solution y_i^{ts} using algorithm 2
    evaluate solution y_i^{ts} acceptance criteria
    if *solution acceptance criteria satisfied* **then**
      $y^{ts} = y_i^{ts}$
      break While loop
    else
      identify subset T_i^r of all elements $K \in T_i^{ts}$ to be refined, $T_i^r \subseteq T_i^{ts}$
      identify subset T_i^c of all elements $K \in T_i^{ts}$ to be coarsened, $T_i^c \subseteq T_i^{ts}$
      obtain T_{i+1}^{ts} by refining (at least) all $K \in T_i^r$ and coarsening a subset of T_i^c
      transfer the solution y_i^{ts} onto T_{i+1}^{ts}
      i = i + 1
  calculate updated value of τ using equation (3.41)

The output of the application of algorithm 5 is presented in figure 4.2.

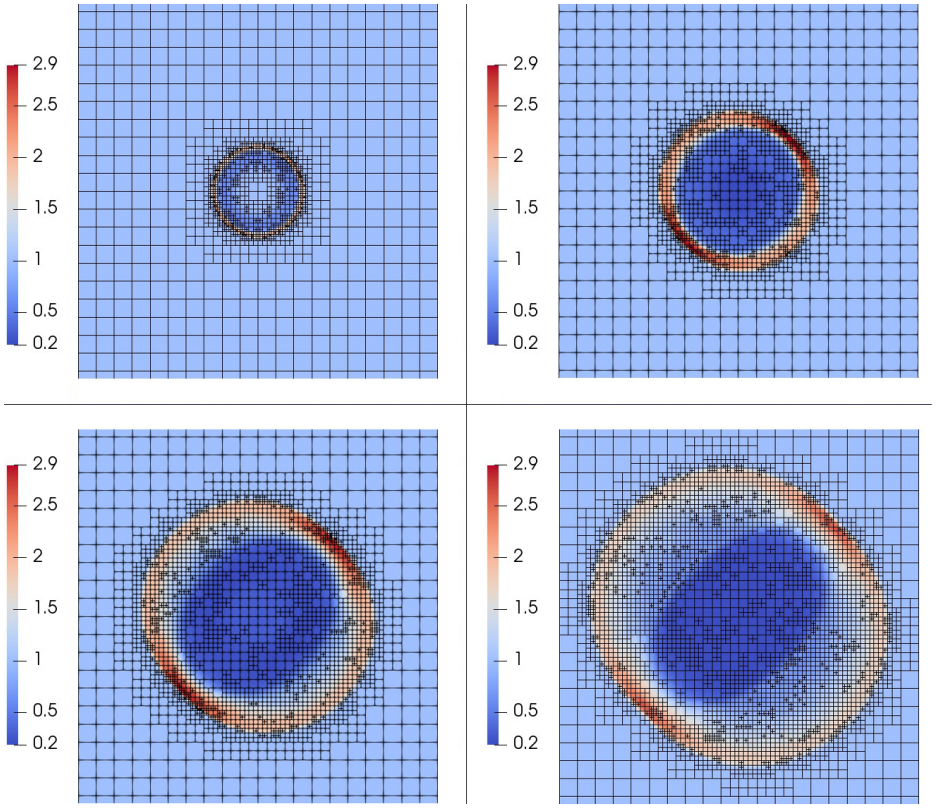


Figure 4.2: Solution obtained using AMR in 4 different time steps - ρ distribution on elements in $T(\Omega)$

The critical points of algorithm 5 are

- solution acceptance criteria evaluation,
- identification of subset T_i^r,
- identification of subset T_i^c.

For the last two points, we shall consider a function r

$$r: \; T_i^{ts} \to [0, +\infty), \tag{4.1}$$

which shall be called *refinement indicator*, and the set $T_i^r = \{K^r\}$ shall be then defined as a set of all such elements for which one of these criteria are satisfied:

$$r(K^r) > \alpha \cdot \max\left\{r(K) \;\middle|\; K \in T_i^{ts}\right\}, \text{ or} \tag{4.2}$$

$$r(K^r) > \beta \cdot \sum\left\{r(K) \;\middle|\; K \in T_i^{ts}\right\}. \tag{4.3}$$

The set $T_i^c = \{K^c\}$ shall be defined similarly as

$$r\left(K^c\right) < \gamma \cdot \max\left\{r\left(K\right) \mid K \in T_i^{ts}\right\}, \text{ or} \tag{4.4}$$

$$r\left(K^c\right) < \delta \cdot \sum\left\{r\left(K\right) \mid K \in T_i^{ts}\right\}. \tag{4.5}$$

Note that the parameters $0 < \gamma \leq \alpha < 1,\ 0 < \delta \leq \beta < 1$ are artificial, and do not affect the overall solution quality (as the solution acceptance criteria has to be met independently of choices of their values). Nevertheless, the choices may affect performance - e.g. for a high α, β, many steps are needed in order to satisfy the solution acceptance criteria, and on the other hand, if values are too low, there is an unnecessary high number of degrees of freedom that do not contribute substantially to the reduction of the solution error.

Please also note, that description of edge cases, and limitations are not given here for brevity. These cases include e.g. what happens if an element is both selected for coarsening and refinement, or how we maintain a 'minimal' refinement level so that we do not coarsen beyond a rational limit.

4.2 Adaptive-mesh refinement and DG

With the Discontinuous Galerkin method, using AMR brings additional complexity into the evaluation of face integrals described in equations (3.57) and (3.61). To describe the process in detail, definition of the numerical flux equation (3.13) needs to be taken into account. What is evaluated during the assembling procedure algorithm 1 is the term equation (3.61). In order to evaluate it, we need to consider values of the solution in quadrature points $\left\{\boldsymbol{x}_1^{ij}, ..., \boldsymbol{x}_{\boldsymbol{n}_f}^{ij}\right\}$ from both sides of $\Gamma_{ij} \in \Gamma_I$, that is from the two neighboring elements $K^i \in T, K^j \in T$. For $\Gamma_{ij} \in \Gamma_B$ the situation is obviously simpler as is not discussed further.

What needs to be evaluated is in fact the term

$$\mathbf{H}\left(\boldsymbol{\Psi}_h^k|_{ij}\left(\boldsymbol{x}\right), \boldsymbol{\Psi}_h^k|_{ji}\left(\boldsymbol{x}\right), \boldsymbol{n}_{ij}\left(\boldsymbol{x}\right)\right) \tag{4.6}$$

If the two neighboring elements are of the same size as in figure 4.3, calculation of this term from algorithmic perspective is performed as follows:

Algorithm 6: Assembling of numerical flux

Data: $K^i \in T_h$
Data: $\Gamma_{ij} \in \Gamma_I$ a face of K^i
Data: Quadrature points $\left\{\boldsymbol{x}_1^{ij}, ..., \boldsymbol{x}_{\boldsymbol{n}_f}^{ij}\right\}$
\# **1**. Triangulation and DOFs handling
Find the neighbor K^j (querying data structures representing the mesh T)
Find the set of degrees of freedom (basis functions) $v_h^i = v_h\left(K^i\right)$
Find the set of degrees of freedom (basis functions) $v_h^j = v_h\left(K^j\right)$
\# Loop over quadrature points
foreach $\boldsymbol{j}_f \in \{1, ..., \boldsymbol{n}_f\}$ **do**
  \# **2**. Previous solution values
  Find previous solution value $\boldsymbol{w}_{\boldsymbol{j}_f}^{ip} = \boldsymbol{\Psi}_h^k|_{ij}\left(\boldsymbol{x}_{\boldsymbol{j}_f}\right)$ on K^i
  Find previous solution value $\boldsymbol{w}_{\boldsymbol{j}_f}^{jp} = \boldsymbol{\Psi}_h^k|_{ij}\left(\boldsymbol{x}_{\boldsymbol{j}_f}\right)$ on K^i
  Set: $\boldsymbol{n}_{\boldsymbol{j}_f} = \boldsymbol{n}_{ij}\left(\boldsymbol{x}_{\boldsymbol{j}_f}\right)$
  \# **3**. Numerical flux evaluation
  $\mathbf{H}\left(\boldsymbol{\Psi}_h^k|_{ij}\left(\boldsymbol{x}\right), \boldsymbol{\Psi}_h^k|_{ji}\left(\boldsymbol{x}\right), \boldsymbol{n}_{ij}\left(\boldsymbol{x}\right)\right) = \mathbf{H}\left(\boldsymbol{w}_{\boldsymbol{j}_f}^{ip}, \boldsymbol{w}_{\boldsymbol{j}_f}^{jp}, \boldsymbol{n}_{\boldsymbol{j}_f}\right)$
  \# ... (Further processing as in algorithm 1)

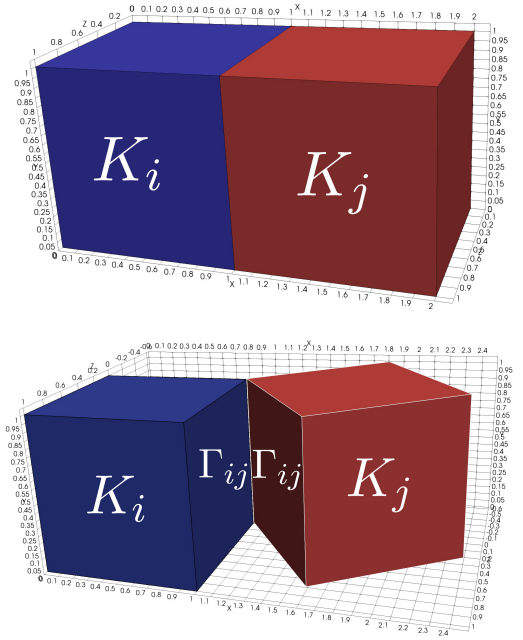


Figure 4.3: Neighbor elements K_i, K_j in the mesh (above), transformed in order to view Γ_{ij} (below).

If however, the two neighboring elements are not of the same size, situation gets more complicated. Note that for this work, the level of uniform refinement of the hexahedron K_i must differ from the level of uniform refinement of any neighboring element K_j with which it shares a common face Γ_{ij} at most by one. This means that the situation when neighboring elements are not of the same side looks always as in figure 4.4, but K_i can be on either side of Γ_{ij} as indicated in figure 4.4. Note that the edge on the whole side of element K_i is not physically stored anymore (at least not for assembling purposes).

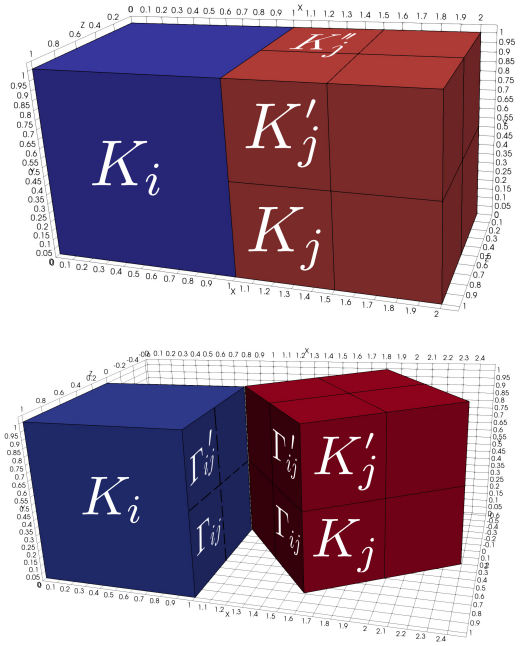


Figure 4.4: Neighbor elements $K_i, K_j, K_j', ...$ in the mesh (above), transformed in order to view $\Gamma_{ij}, \Gamma_{ij'}$ (below).

4.2.1 Periodic boundary conditions

Handling with periodic boundary conditions with AMR is primarily difficult because of implications for mesh partitioning - additional ghost cells need to be added to cells on boundaries with the periodic condition - in order to perform steps 1 and 2 in algorithm 6. See figure 4.5, and compare to figure 3.14.

The setup here is the same one used in section 3.1.1, showing ghost elements as in figure 3.14, but periodic boundary conditions were added to the left-right and top-bottom boundary part pairs. And the necessary elements were thusly added to ghost elements set for each affected processor (processor having any elements on the periodic boundaries).

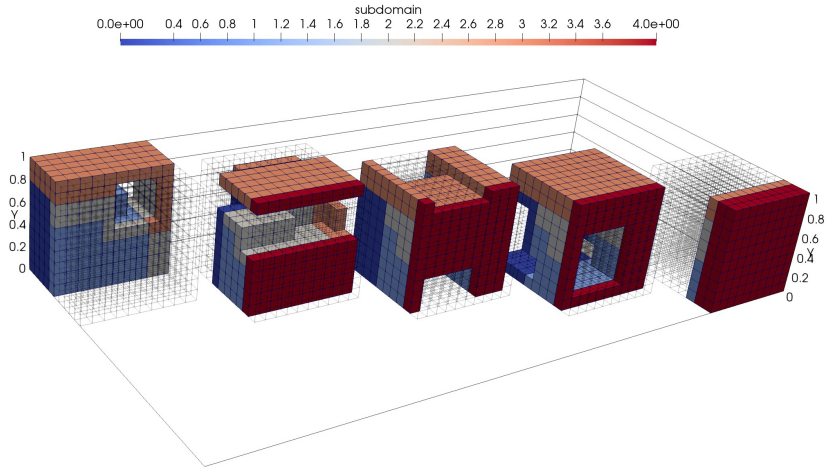


Figure 4.5: Processor-owned elements (0..4 left to right), with color-coded ghost elements from other processors. Ghost cells that have to be present here are dictated by fluxes - both across internal edges, and across periodic boundaries.

4.2.2 Relationship with slope limiters

As a key step in the slope limiting algorithm algorithm 2 is forming the set of all elements $K_{\boldsymbol{v}} = \left\{ K' \in T \mid \boldsymbol{v} \text{ is a vertex of } K \right\}$ for a particular vertex $\boldsymbol{v}$, and as this step tends to be rather expensive for distributed triangulations (which have different topology than vertex-based), caching of the already obtained mappings between the vertex $\boldsymbol{v}$ and the set $K_{\boldsymbol{v}}$ is employed. Of course with every mesh refinement, a lot of these cached values need to be forgotten.

In reality, after each mesh refinement step, there follows a step of redistributing the mesh (to have roughly equal number of elements owned by each processors - that is to avoid bottlenecks where one processors would do a significant portion of all processing and other processors would idly wait for it to finish assembling). This is illustrated in figures 5.8 to 5.13.

From this it follows, that carefully computing which entries in such a cache must be forgotten and which not would be very tedious and complex - that is why, for pragmatic reasons, after each refinement step, the entire cache is flushed. But in order not to suffer from slope limiting being a costly operation, the mesh refinement is performed only in every n-th time step. Another reason to do so is that with the unchanged mesh, the algebraic solver can keep the once computed matrix structures and data alike, which again saves computational time.

4.3 Reference solution approach

As the aim of this work is to prepare a universally usable solver for MHD problems, the refinement indicator equation (4.1) must ideally not be dependent on any attributes of the solved problem data (initial condition, boundary conditions, physical quantities such as γ, etc.). In order to achieve this, the so-calledreference solution approach is used.

The reference solution approach is not only problem, but also equation (physics) independent, which is truly invaluable for many types of physical problems, even so for multi-physics coupled problems, such as MHD.

4.3.1 Algorithm

Algorithm, given in line 7 is accompanied by an example in figures 4.6 to 4.9:

Algorithm 7: Reference solution AMR algorithm

Data: Pair of meshes - fine mesh T_0^f, and coarse mesh T_0^c - the fine mesh being one layer of refinement finer than the coarse one.

Data: Solution acceptance criteria in the form of threshold $0 < \alpha < 1$ representing the relative error estimate.

Note: This condition translates to: $\frac{\sum_{K^j \in T_i^c} ||\boldsymbol{y}_i - \boldsymbol{y}_i^c||_{L^2(K^j)}}{\sum_{K^j \in T_i^c} ||\boldsymbol{y}_i^c||_{L^2(K^j)}} < \alpha$, where $\boldsymbol{y}_i^c$ is a projection of $\boldsymbol{y}_i$ onto the coarse space.

Result: A fine mesh T_n^f and a solution $\boldsymbol{y}_n$ on this mesh satisfying the solution acceptance criteria

Note: In order to be sure that we will converge towards an acceptable solution, we need to define the refinement indicator (see equation (4.1)) in a way that the elements selected for refinement are those that contribute to the relative error (left hand side of the equation for the acceptance criteria) the most.

This means that the refinement indicator is defined as $r(K) = \frac{||\boldsymbol{y}_i - \boldsymbol{y}_i^c||_{L^2(K)}}{||\boldsymbol{y}_i^c||_{L^2(K)}}$.

i = 0

while *true* **do**

- obtain solution $\boldsymbol{y}_i$ on T_i^f
- project $\boldsymbol{y}_i$ onto T_i^c, obtaining coarse projection $\boldsymbol{y}_i^c$
- evaluate $||\boldsymbol{y}_i - \boldsymbol{y}_i^c||_{L^2(K)}$, $K \in T_i^c$
- **if** $\sum_{K^j \in T_i^c} ||\boldsymbol{y}_i - \boldsymbol{y}_i^c||_{L^2(K^j)} < \alpha$ *satisfied* **then**
  - return
- **else**
  - identify subset T_i^r of all elements $K \in T_i^c$ to be refined, $T_i^r \subseteq T_i^c$ in such a way that for some $0 < \alpha < 1$ (see equation (4.2)) we define the subset T_i^r as : $T_i^r = \left\{ K \in T_i^c : ||\boldsymbol{y}_i - \boldsymbol{y}_i^c||_{L^2(K)} > \alpha \max_{K' \in T_i^c} ||\boldsymbol{y}_i - \boldsymbol{y}_i^c||_{L^2(K')} \right\}$
  - obtain T_{i+1}^c by refining (at least) all $K \in T_i^r$
  - obtain T_{i+1}^f by uniformly refining T_{i+1}^c
  - i = i + 1

In figures figures 4.6 to 4.9, several AMR steps are presented, showing how the line 7 works in practice. The results are taken from Korous (2012). The error $||\boldsymbol{y}_i - \boldsymbol{y}_i^c||_{L^2(K)}$, $K \in T_i^c$
is shown in the top-right corner, and the two meshes - T_0^c and T_n^f and - at the bottom (left and right). The color of mesh elements corresponds to varying polynomial degree of basis functions on that element.

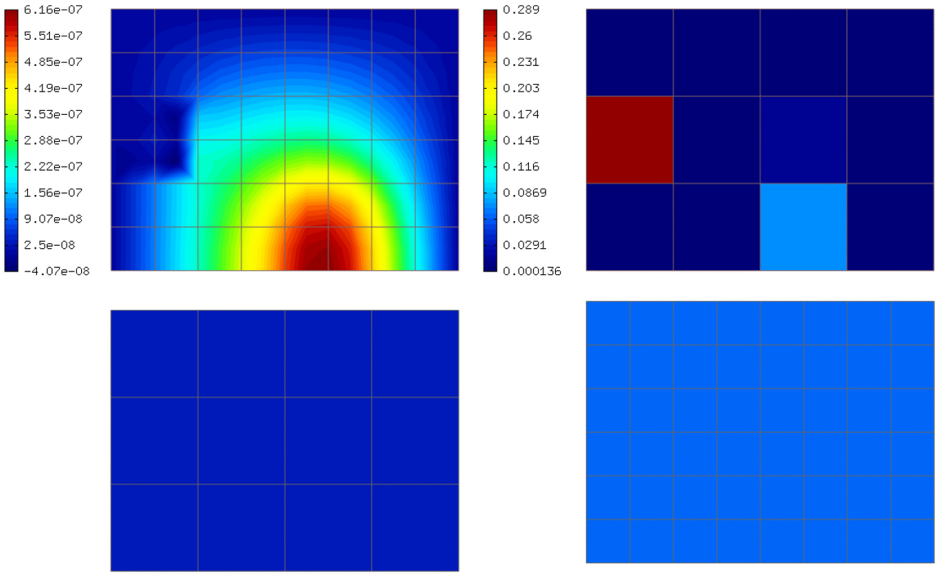


Figure 4.6: AMR Step 1, fine solution (top left), coarse mesh (bottom left), fine mesh (bottom right), element-wise error (top right)

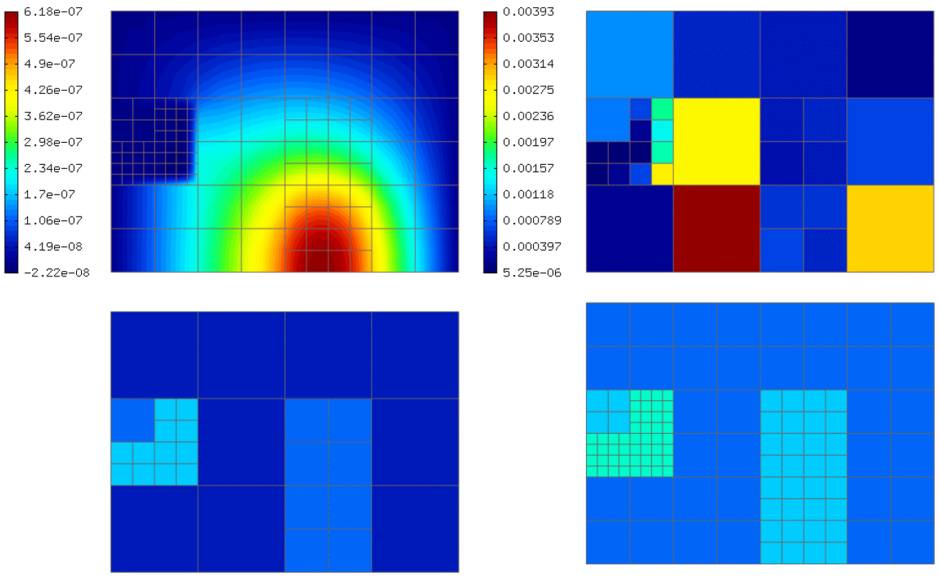


Figure 4.7: AMR Step 2, fine solution (top left), coarse mesh (bottom left), fine mesh (bottom right), element-wise error (top right)

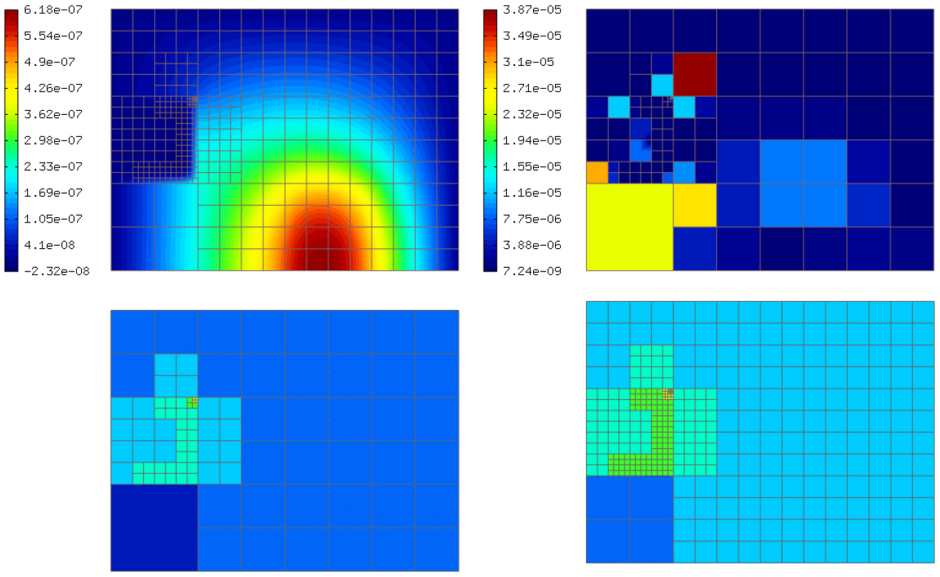


Figure 4.8: AMR Step 3, fine solution (top left), coarse mesh (bottom left), fine mesh (bottom right), element-wise error (top right)

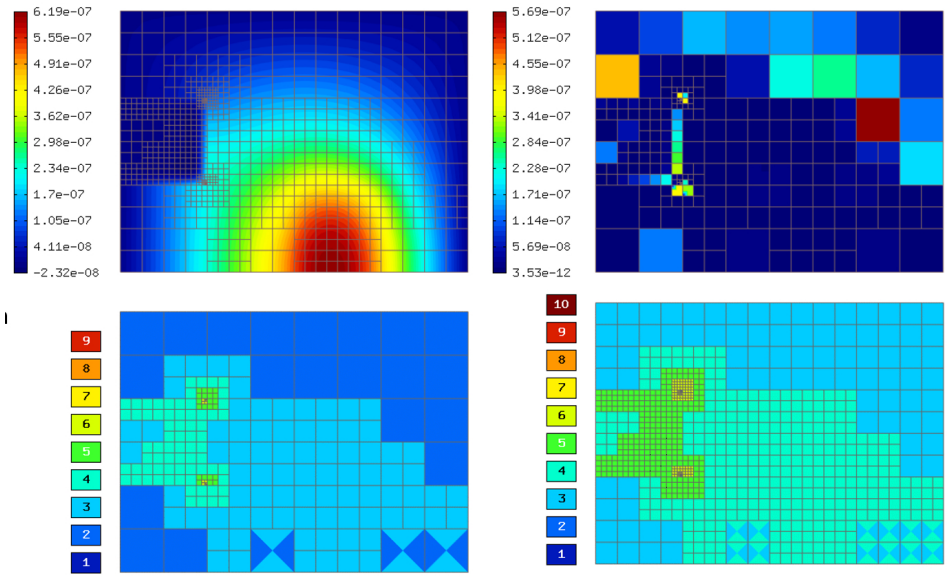


Figure 4.9: AMR Step 4, fine solution (top left), coarse mesh (bottom left), fine mesh (bottom right), element-wise error (top right)

4.3.2 Implementation notes

Regardless of whether the Reference solution-based AMR, or AMR based on another solution criteria and refinement indicator is used, a necessary implementation aspect is the iterative approach towards obtaining an acceptable solution - this applies both to line 7 as well as algorithm 4 - in both cases these iterations are handled via a *While* loop with the variable i.

In order to satisfy the solution acceptance criteria, and possibly some other technical criteria (minimum and maximum number of degrees of freedom for example), care must be taken in the sub-step (again, present in both presented algorithms) of identifying the elements for refinement (the set T_i^r). The way this set is constructed in equations (4.2) to (4.5) assumes the ability to calculate either $\max\left\{r\left(K\right) \mid K \in T_i^{ts}\right\}$, or $\sum\left\{r\left(K\right) \mid K \in T_i^{ts}\right\}$, prior to identifying the set T_i^r. This however, in a distributed computation, constitutes a map-reduce problem of the following nature:

A) On each processor separately, calculate the local contribution

B) On the master processor (rank #0), collect all local contribution

C) On the master processor calculate the result, distribute it to all processors

D) On each processor, use the global result to identify the local contribution to the set T_i^r

5 Results

In this section, results from the computation using the implemented software are presented. There are two classical benchmarks for 3-dimensional MHD equations, namely the MHD Blast Londrillo and Zanna (2000), Balsara and Spicer (1999), and the Orszag-Tang vortex Orszag and Tang (1979). And then the main section of this work contains results from the flux tube eruption model on and above the Sun's surface.

In most cases, we are interested in the distribution of plasma density ρ and the magnetic field B in the domain Ω.

5.1 Benchmarks

The benchmarks presented in this Section do not have an exact analytical solution, but the formation of waves and discontinuities is well studied, and benchmarking is usually performed on the basis of comparing the structure and presence of non-physical attributes.

5.1.1 Hardware specification

For all following benchmarks, the setup described below was used. The computational mesh T_h was formed in all cases by rectangular hexahedra. The value of the time step τ used was set according to the CFL condition (section 3.4.2). The Taylor basis functions (see Kuzmin (2010)) and Divergence-free basis functions (table 3.1) of order 0 (piecewise constant functions) and 1 (piecewise linear functions) were used. Where appropriate (for piecewise linear basis functions), the slope limiting technique from section 3.8.1 was used. Illustration of the obtained results follows below - these results were obtained using one node of the department's computational cluster with these parameters:

- CPU: Intel(R) Xeon(R) CPU E5-2680 v3 @ 2.50GHz,
- # of Cores: 48 per node (nodes 1, 2), 16 per node (nodes 3, 4),
- Vectorization support: AVX2,
- Parallelization implemented: Intel TBB,
- Distributed calculation implemented: OpenMPI,

- RAM: 512 GB,
- C/C++ Compiler: GNU gcc 5.4.0 .

5.1.2 MHD Blast

MHD Blast - original version

This benchmark has been used for decades - Zachary et al. (1994), Londrillo and Zanna (2000), Balsara and Spicer (1999) - in a variety of configurations and as a benchmark in software - e.g. (Stone et al., 2008a). The setup as described in Zachary et al. (1994), Londrillo and Zanna (2000) is defined (although in Londrillo and Zanna (2000) with interchanged $x-$ and $y-$ coordinates) by the initial conditions:

$$\begin{aligned} \gamma &= 5/3 \\ p_0(\boldsymbol{x}, t) &= 100 \quad \text{for} \quad |\boldsymbol{x}| < 0.1 \\ p_0(\boldsymbol{x}, t) &= 1 \quad \text{for} \quad |\boldsymbol{x}| \geq 0.1 \\ \rho(\boldsymbol{x}, t = 0) &= 1, \\ p(\boldsymbol{x}, t = 0) &= p_0(\boldsymbol{x}, t), \\ \boldsymbol{u}_1(\boldsymbol{x}, t = 0) &= 0, \\ \boldsymbol{u}_2(\boldsymbol{x}, t = 0) &= 0, \\ \boldsymbol{u}_3(\boldsymbol{x}, t = 0) &= 0, \\ \boldsymbol{B}_1(\boldsymbol{x}, t = 0) &= 0, \\ \boldsymbol{B}_2(\boldsymbol{x}, t = 0) &= 100, \\ \boldsymbol{B}_3(\boldsymbol{x}, t = 0) &= 0. \end{aligned} \tag{5.1}$$

Total energy is calculated using equations (1.1) to (1.3). The domain Ω is a square, with scaling quite arbitrarily used in the papers. In the case of Londrillo and Zanna (2000), $\Omega = [0, 1] \times [0, 1]$, in this work it is $\Omega = [-0.25, 0.25] \times [-0.25, 0.25]$.

It is obvious from the initial setup, that the example is true to its name, and it is in fact a blast of the over-pressured area $|\boldsymbol{x}| < 0.1$, where pressure p is $100\times$ larger than elsewhere in the domain. This setup is completed with simple outflow boundary condition equation (2.40). In the next figures, the solution as presented in Londrillo and Zanna (2000) is compared to the solution obtained with the approach described in this work. Note that the solution from Londrillo and Zanna (2000) needed to have the axes transformed ($x \rightleftharpoons y$) with respect to the original paper. The figure figure 5.1 is taken from the article Londrillo and Zanna (2000) from the page 33. Unfortunately the paper does not specify the precise time at which the snapshots are taken.

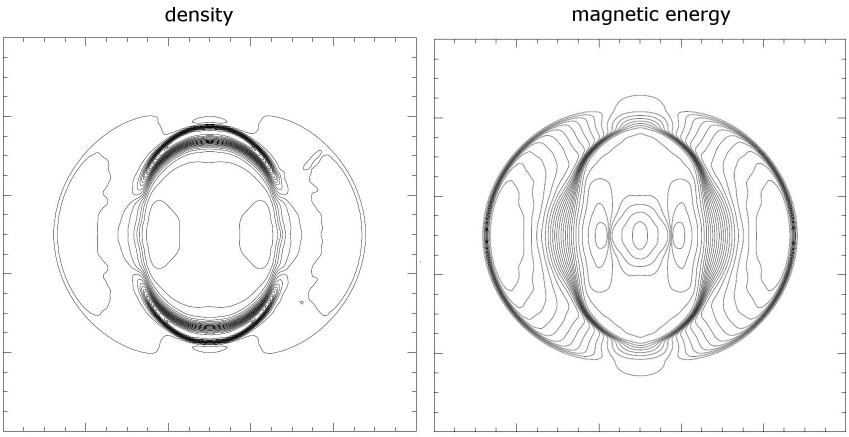


Figure 5.1: Results from Londrillo and Zanna (2000), density(left), magnetic energy(right)

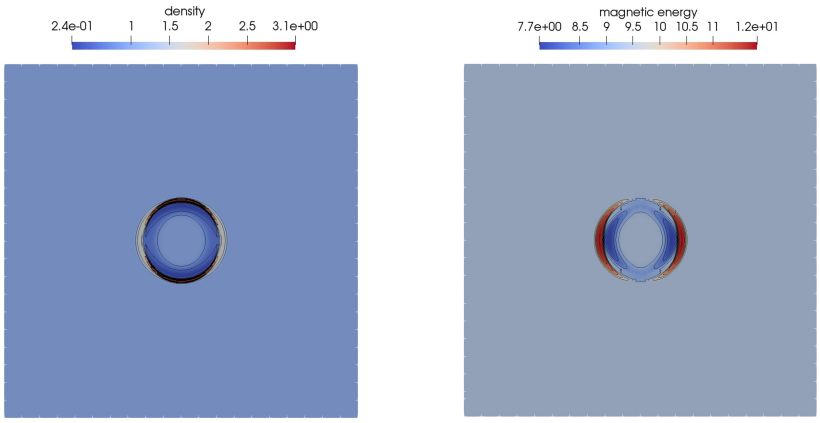


Figure 5.2: Obtained results, $t = 1 \times 10^{-3}$, density(left), magnetic energy(right)

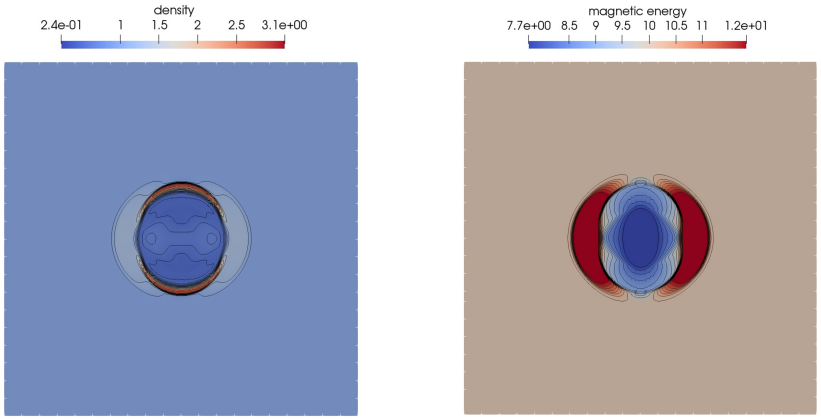


Figure 5.3: Obtained results, $t = 6 \times 10^{-3}$, density(left), magnetic energy(right)

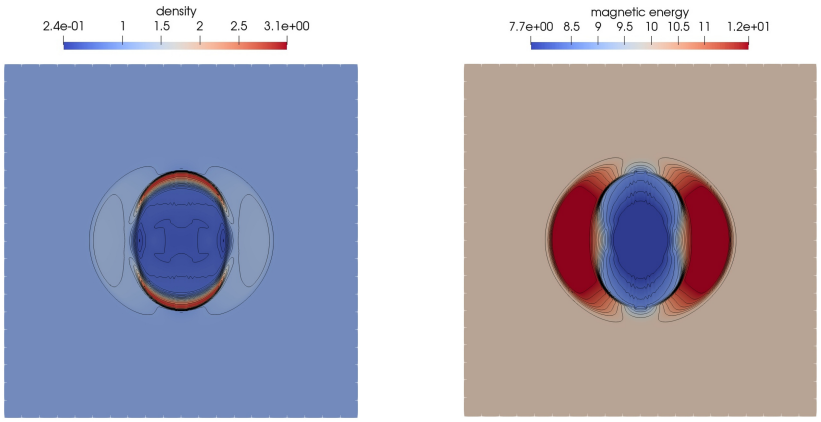


Figure 5.4: Obtained results, $t = 11 \times 10^{-3}$, density(left), magnetic energy(right)

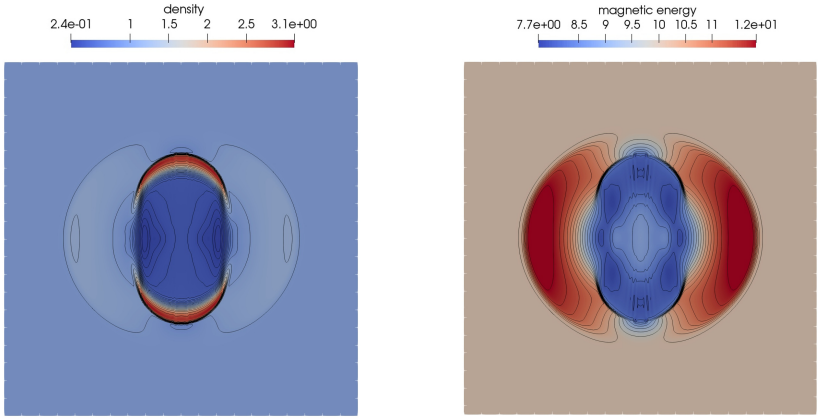


Figure 5.5: Obtained results, $t = 16 \times 10^{-3}$, density(left), magnetic energy(right)

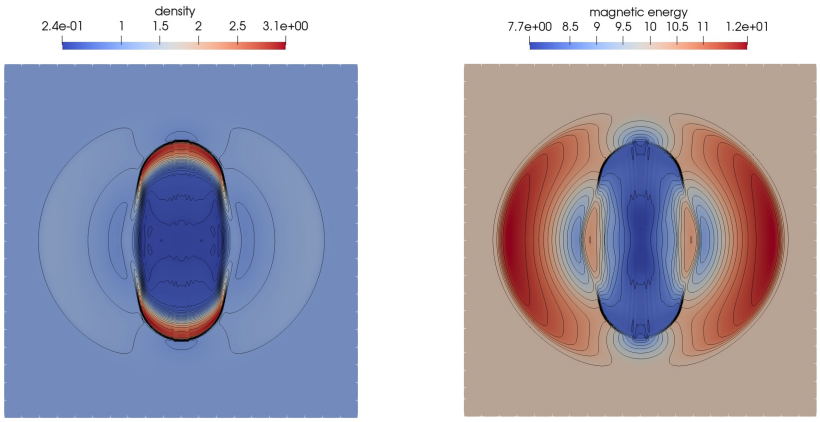


Figure 5.6: Obtained results, $t = 23 \times 10^{-3}$, density(left), magnetic energy(right)

The solution in figures 5.5 and 5.6 is apparently almost identical to that in figure 5.1. To demonstrate the distributed nature of the computation, in figure 5.7, color-mapping of elements $K \in T$ to processors owning the particular element is presented (see section 3.1.1 for details). Note that there were 48 processors used for the computation.

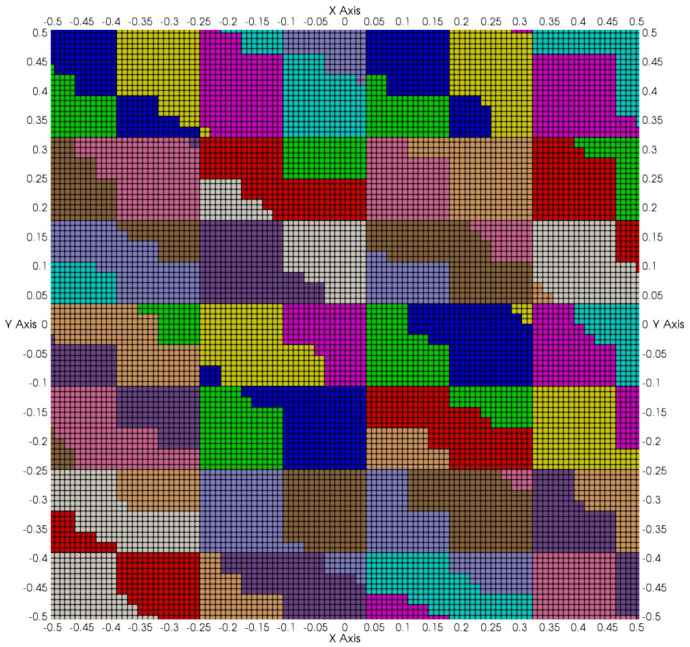


Figure 5.7: Color-mapping of elements to processors for the computation of figures 5.2 to 5.6

This however, was a static calculation with 200 mesh elements in the $x-$ and $y-$ dimensions. In order to see how the AMR (see chapter 4) performs, the same computation was performed in adaptive setup. Starting from a very coarse mesh of 10 elements in both dimensions, the evolving mesh and its distribution to processors are shown in figures 5.8 to 5.13 - the distribution to processors on the left, the mesh elements on the right.

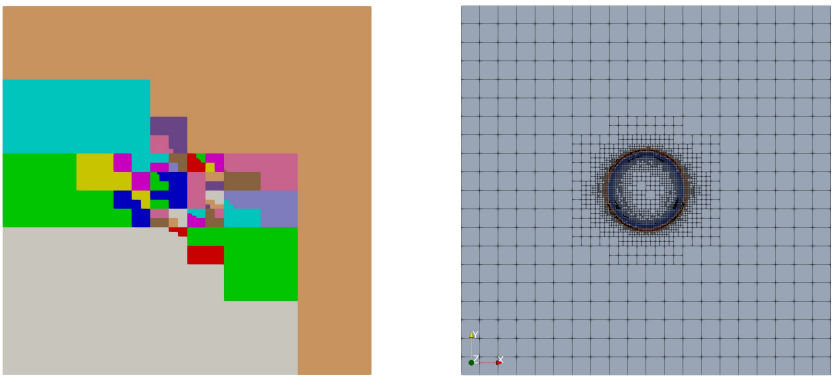


Figure 5.8: Obtained results, initial state, distribution of ρ on elements in $T(\Omega)$ (right) and distribution of these elements to processors (left)

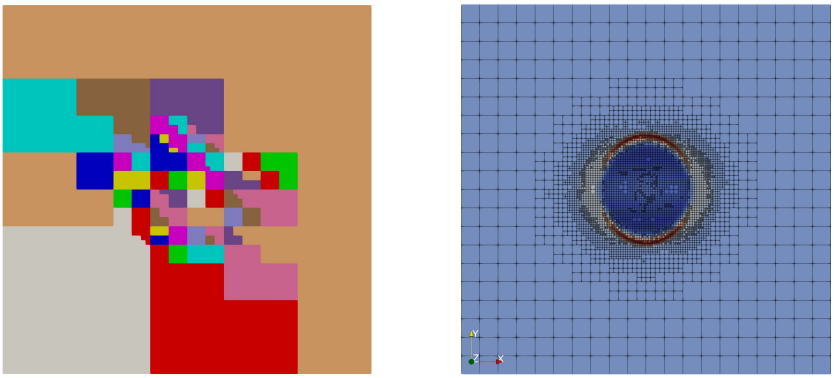


Figure 5.9: Obtained results, $t = 5 \times 10^{-3}$, distribution of ρ on elements in $T(\Omega)$ (right) and distribution of these elements to processors (left)

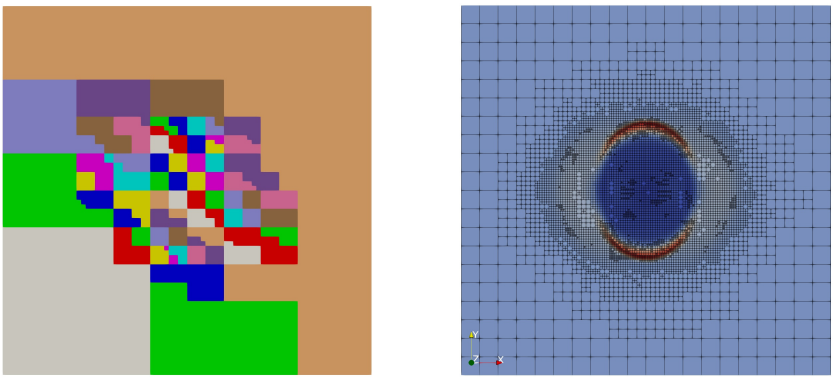


Figure 5.10: Obtained results, $t = 10 \times 10^{-3}$, distribution of ρ on elements in $T(\Omega)$ (right) and distribution of these elements to processors (left)

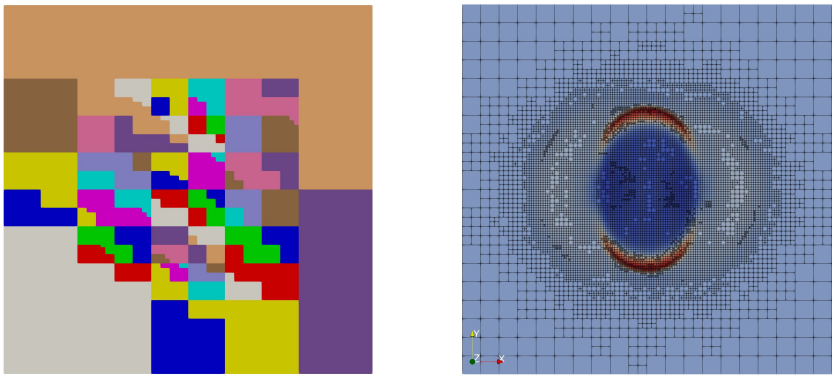


Figure 5.11: Obtained results, $t = 15 \times 10^{-3}$, distribution of ρ on elements in $T(\Omega)$ (right) and distribution of these elements to processors (left)

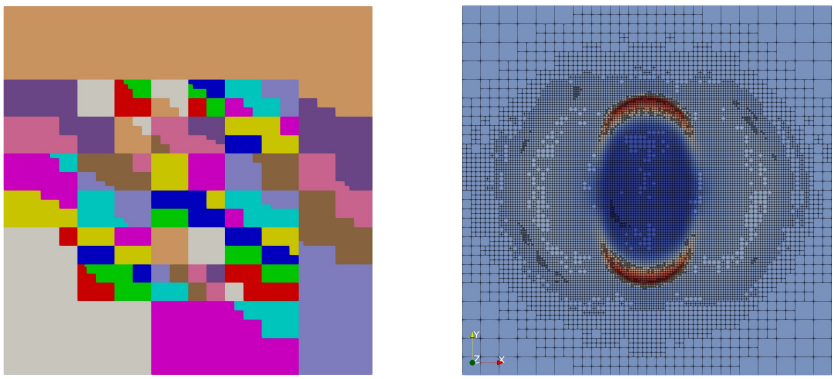


Figure 5.12: Obtained results, $t = 20 \times 10^{-3}$, distribution of ρ on elements in $T(\Omega)$ (right) and distribution of these elements to processors (left)

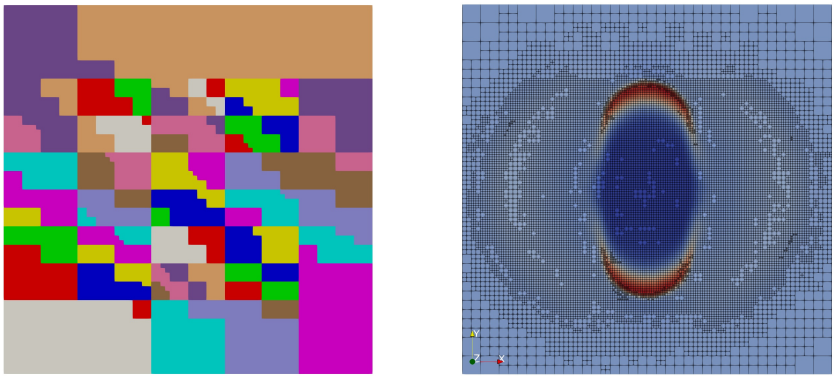


Figure 5.13: Obtained results, $t = 25 \times 10^{-3}$, distribution of ρ on elements in $T(\Omega)$ (right) and distribution of these elements to processors (left)

MHD Blast - extended version

An extended version of the benchmark has been used in Gaburov and Nitadori (2011), Stone et al. (2008b), and a similar problem was used also in Xisto et al. (2014) - the description of the benchmark is also available at:`http://www.astro.princeton.edu/~jstone/Athena/tests/blast/blast.html`. In this version, the domain dimensions are set as a rectangle: $\Omega = [-0.5, 0.5] \times [-0.75, 0.75]$. The

initial conditions are a little different than in the case of equation (5.1), and read:

$$\begin{aligned}
\gamma &= 5/3 \\
p_0(\boldsymbol{x}, t) &= 10 \quad \text{for} \quad |\boldsymbol{x}| < 0.1 \\
p_0(\boldsymbol{x}, t) &= 0.1 \quad \text{for} \quad |\boldsymbol{x}| \geq 0.1 \\
\rho(\boldsymbol{x}, t = 0) &= 1, \\
p(\boldsymbol{x}, t = 0) &= p_0(\boldsymbol{x}, t), \\
\boldsymbol{u}_1(\boldsymbol{x}, t = 0) &= 0, \\
\boldsymbol{u}_2(\boldsymbol{x}, t = 0) &= 0, \\
\boldsymbol{u}_3(\boldsymbol{x}, t = 0) &= 0, \\
\boldsymbol{B}_1(\boldsymbol{x}, t = 0) &= \frac{1}{\sqrt{2}}, \\
\boldsymbol{B}_2(\boldsymbol{x}, t = 0) &= \frac{1}{\sqrt{2}}, \\
\boldsymbol{B}_3(\boldsymbol{x}, t = 0) &= 0,
\end{aligned} \tag{5.2}$$

with periodic boundary conditions on the top-bottom, and left-right parts of the boundary. That is, with respect to equation (2.41), the two pairs $\Gamma_1, \Gamma_2, \Gamma_1^{'}, \Gamma_2^{'}$ are specified as follows:

$$\Gamma_1 = \{-0.5\} \times [-0.75, 0.75], \tag{5.3}$$

$$\Gamma_2 = \{0.5\} \times [-0.75, 0.75], \tag{5.4}$$

and

$$\Gamma_1^{'} = [-0.5, 0.5] \times \{-0.75\}, \tag{5.5}$$

$$\Gamma_2^{'} = [-0.5, 0.5] \times \{0.75\}. \tag{5.6}$$

The solution image taken from Gaburov and Nitadori (2011) is for comparison in figure 5.26.

Results

The first set of results are from calculations using piecewise-constant elements, on three successively uniformly refined meshes. The meshes used for computations figures 5.14 to 5.19 contained:

- 100×150 elements (left)
- 200×300 elements (middle)
- 400×600 elements (right)

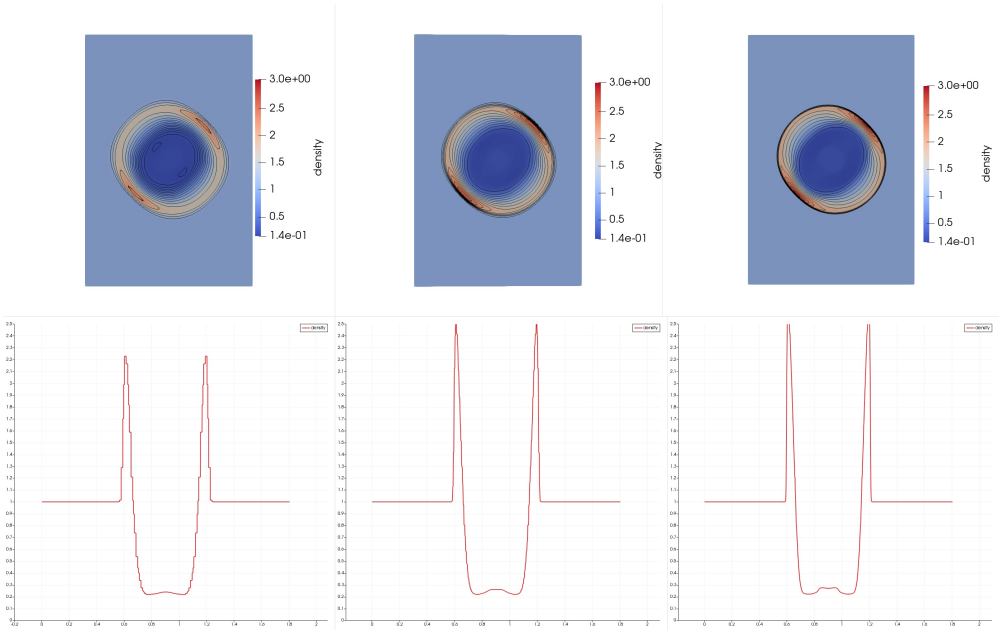


Figure 5.14: Obtained results, $t \approx 0.1$, distribution of ρ (top), with line distribution along bottom-left $\rightarrow$ top-right diagonal (bottom)

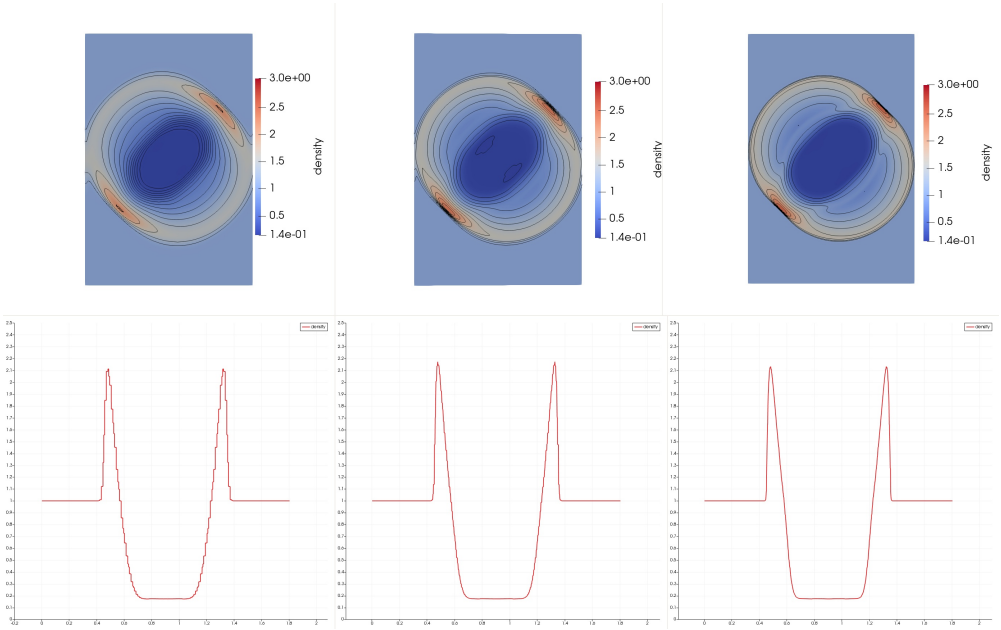


Figure 5.15: Obtained results, $t =\approx 0.2$, distribution of ρ (top), with line distribution along bottom-left $\rightarrow$ top-right diagonal (bottom)

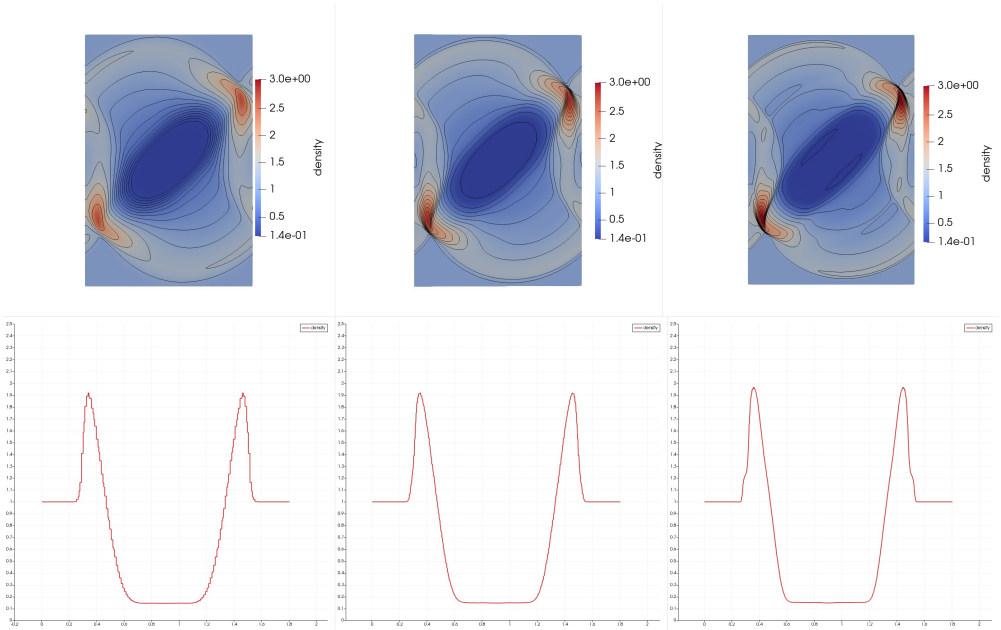


Figure 5.16: Obtained results, $t \approx 0.3$, distribution of ρ (top), with line distribution along bottom-left $\rightarrow$ top-right diagonal (bottom)

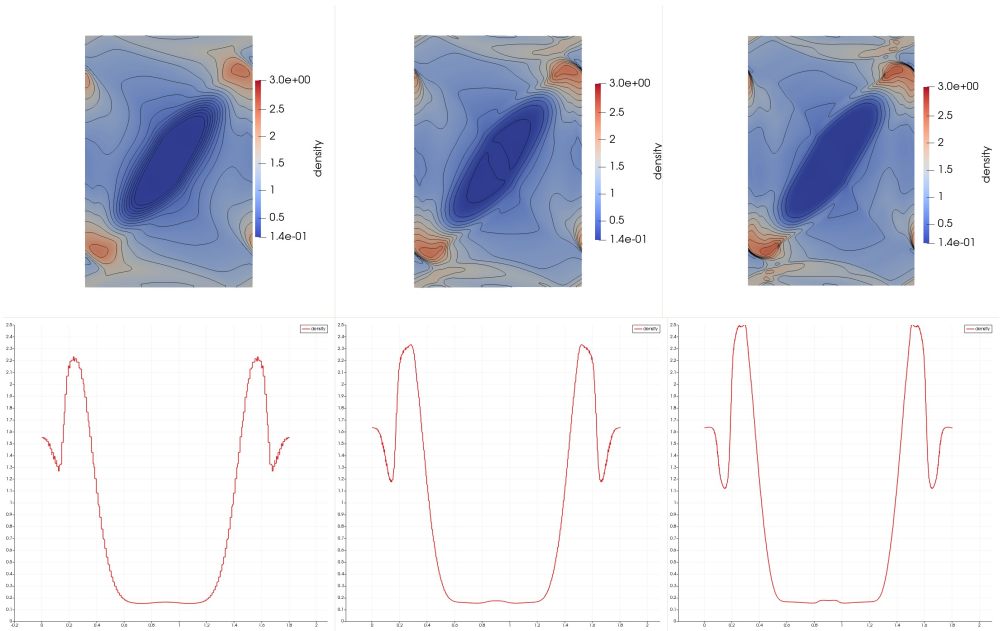


Figure 5.17: Obtained results, $t \approx 0.45$, distribution of ρ (top), with line distribution along bottom-left $\rightarrow$ top-right diagonal (bottom)

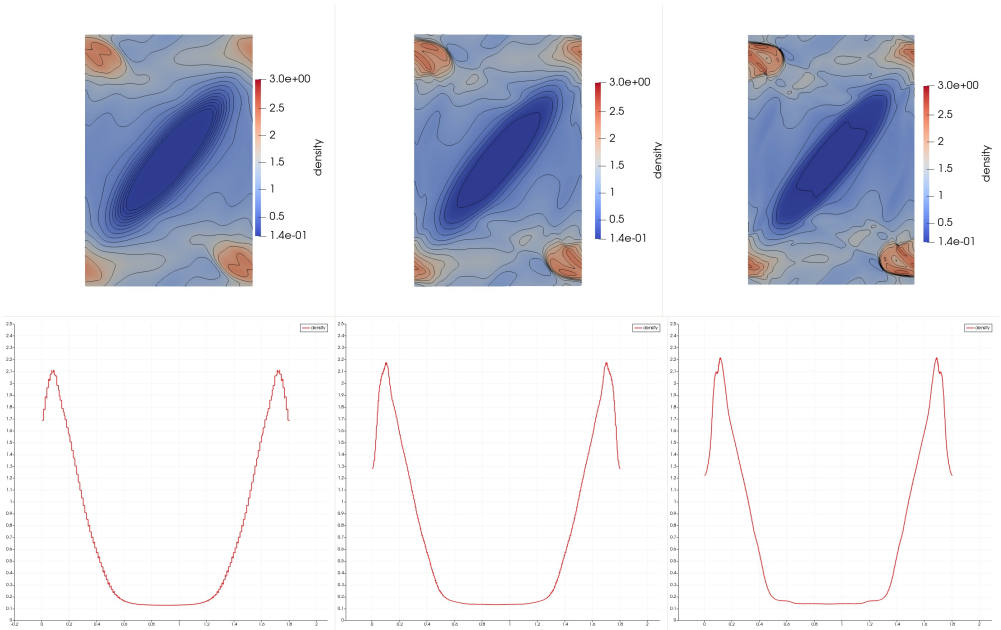


Figure 5.18: Obtained results, $t = \approx 0.75$, distribution of ρ (top), with line distribution along bottom-left $\rightarrow$ top-right diagonal (bottom)

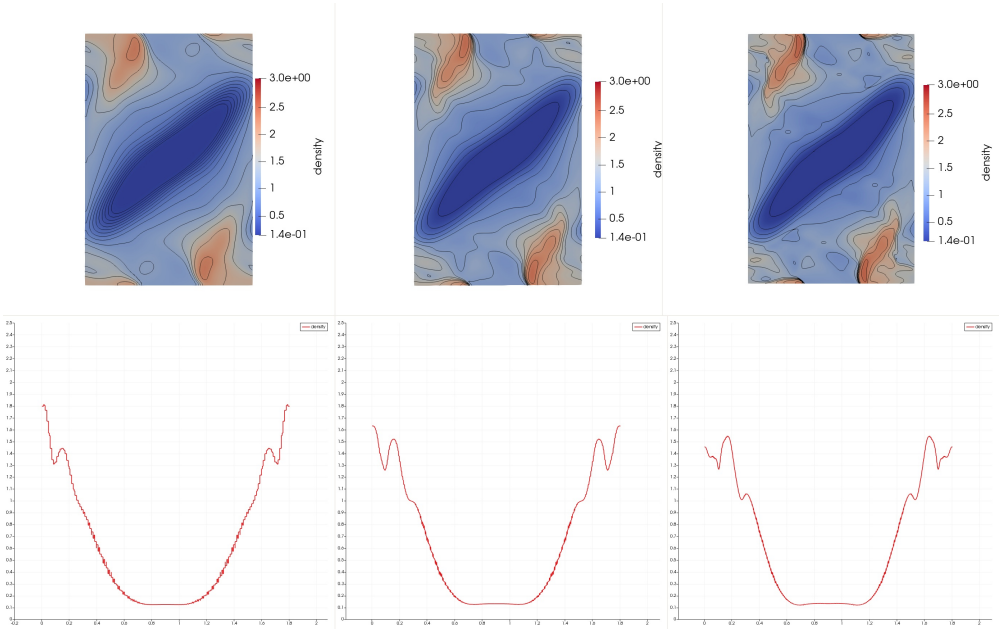


Figure 5.19: Obtained results, $t \approx 0.95$, distribution of ρ (top), with line distribution along bottom-left $\rightarrow$ top-right diagonal (bottom)

It is clearly visible, that the solution is somehow smeared, and the uniform mesh refinements improve the situation, but not greatly, and for a large cost of storage

size for storing much more mesh elements.

In order to amend the situation, and be able to obtain a higher-quality solution with a reasonable number of mesh elements, piecewise-linear basis functions need to be used. Results with piecewise-linear basis functions, on three successively uniformly refined meshes are given in figures 5.20 to 5.25. The meshes for these computations contained:

- 50×75 elements (left)
- 100×150 elements (middle)
- 200×300 elements (right)

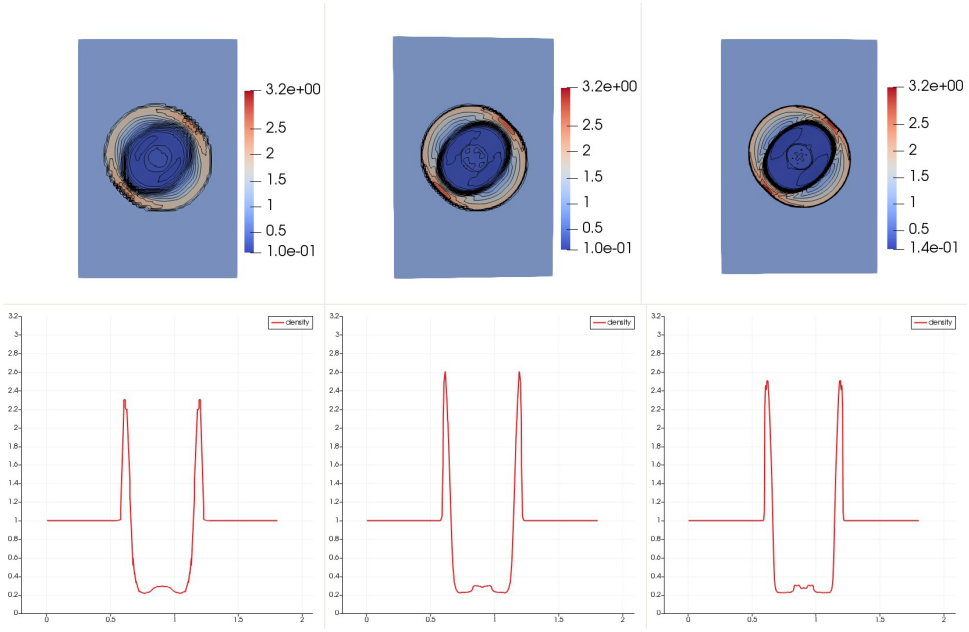


Figure 5.20: Obtained results, $t \approx 0.1$, distribution of ρ (top), with line distribution along bottom-left $\rightarrow$ top-right diagonal (bottom)

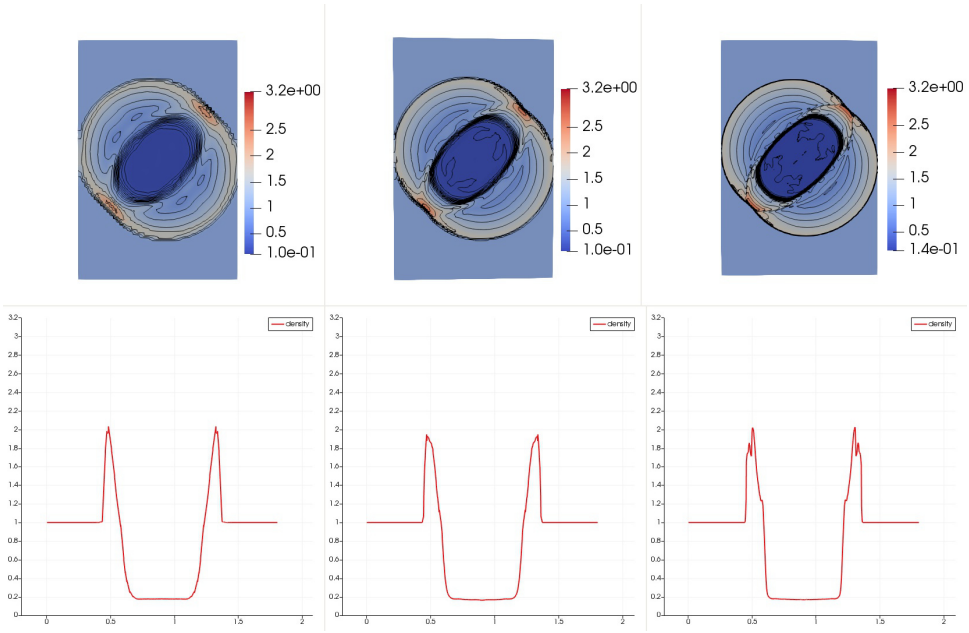


Figure 5.21: Obtained results, $t =\approx 0.2$, distribution of ρ (top), with line distribution along bottom-left $\rightarrow$ top-right diagonal (bottom)

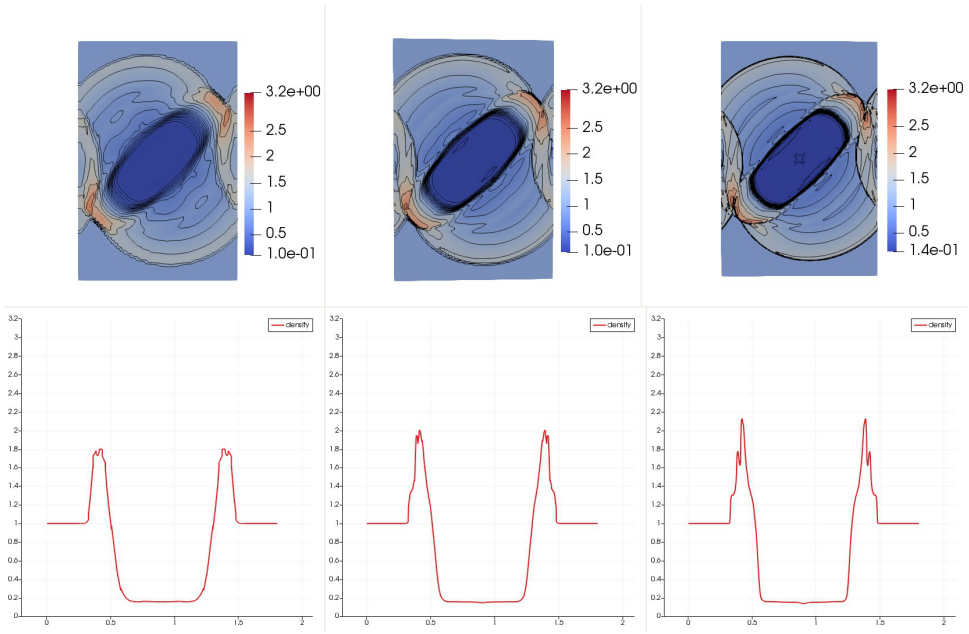


Figure 5.22: Obtained results, $t \approx 0.3$, distribution of ρ (top), with line distribution along bottom-left $\rightarrow$ top-right diagonal (bottom)

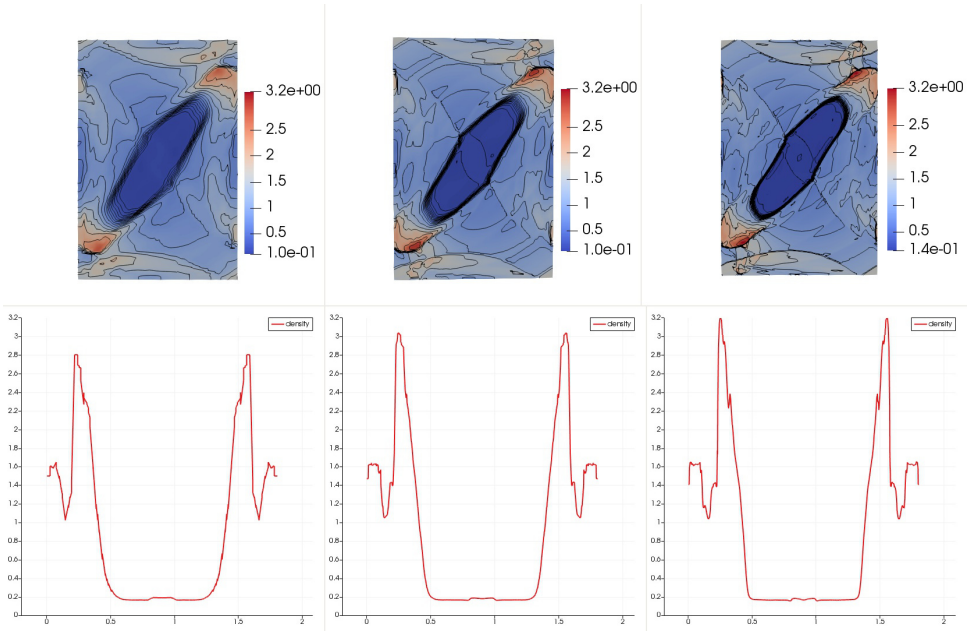


Figure 5.23: Obtained results, $t \approx 0.45$, distribution of ρ (top), with line distribution along bottom-left $\rightarrow$ top-right diagonal (bottom)

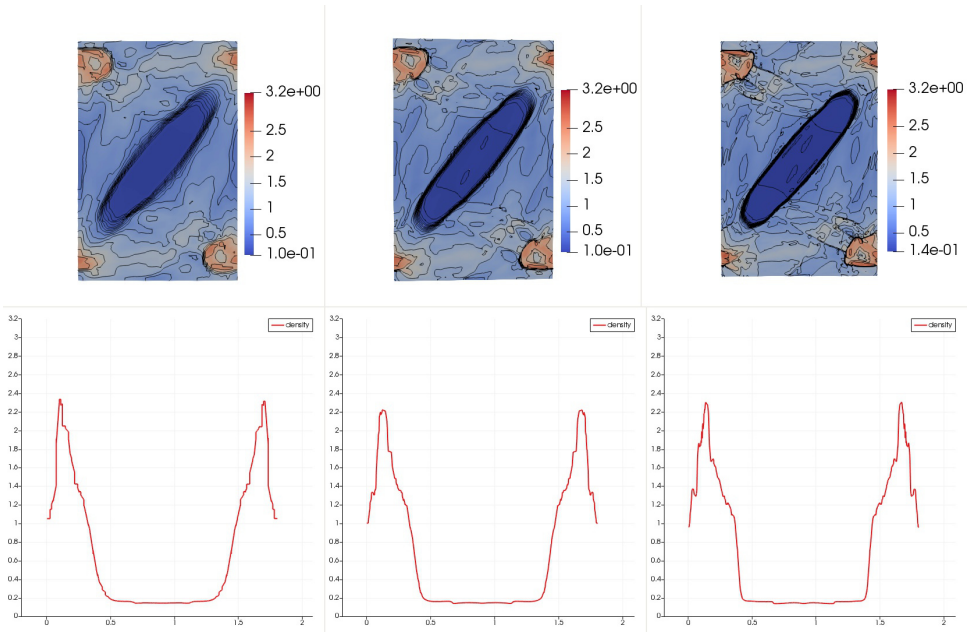


Figure 5.24: Obtained results, $t =\approx 0.75$, distribution of ρ (top), with line distribution along bottom-left $\rightarrow$ top-right diagonal (bottom)

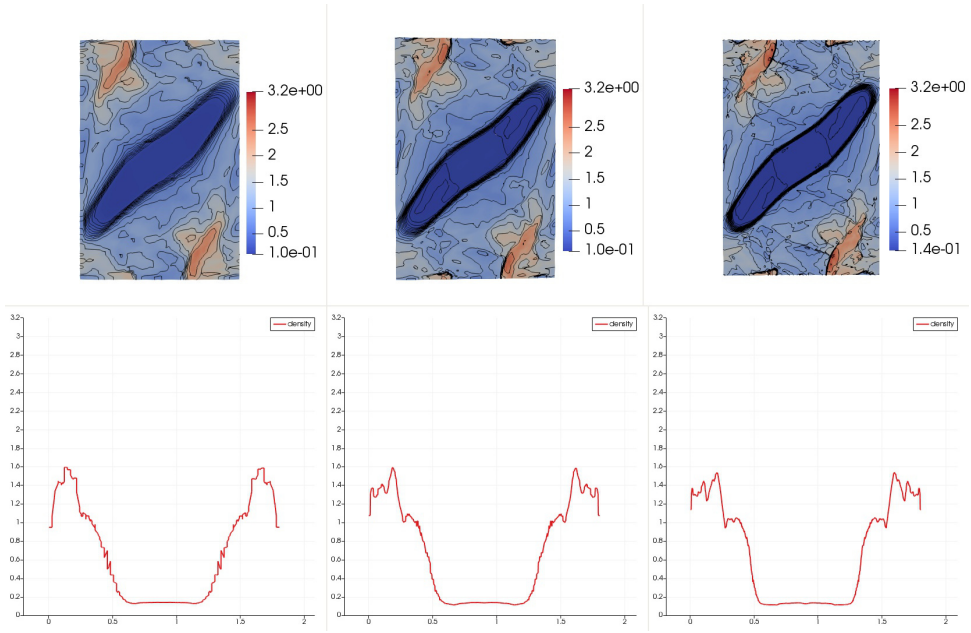


Figure 5.25: Obtained results, $t \approx 0.95$, distribution of ρ (top), with line distribution along bottom-left $\rightarrow$ top-right diagonal (bottom)

Now, the solution with piecewise-linear functions is of much higher quality and the solution snapshots displayed in figure 5.27 are practically identical (and even more detailed) as the solution from Gaburov and Nitadori (2011) in figure 5.26. Of course, this comes at a price of increased number of degrees of freedom, and associated increase in the storage size requirements, as well as the computation time, with respect to the solution with piecewise-constant basis functions.

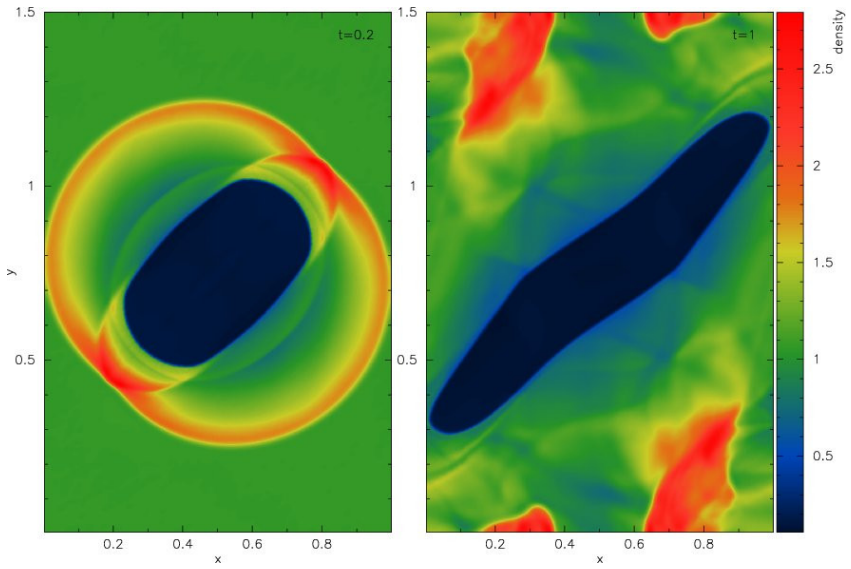


Figure 5.26: Reference solution, ρ distribution, $t \approx 0.2$ (left), $t \approx 1.0$ (right), from Gaburov and Nitadori (2011)

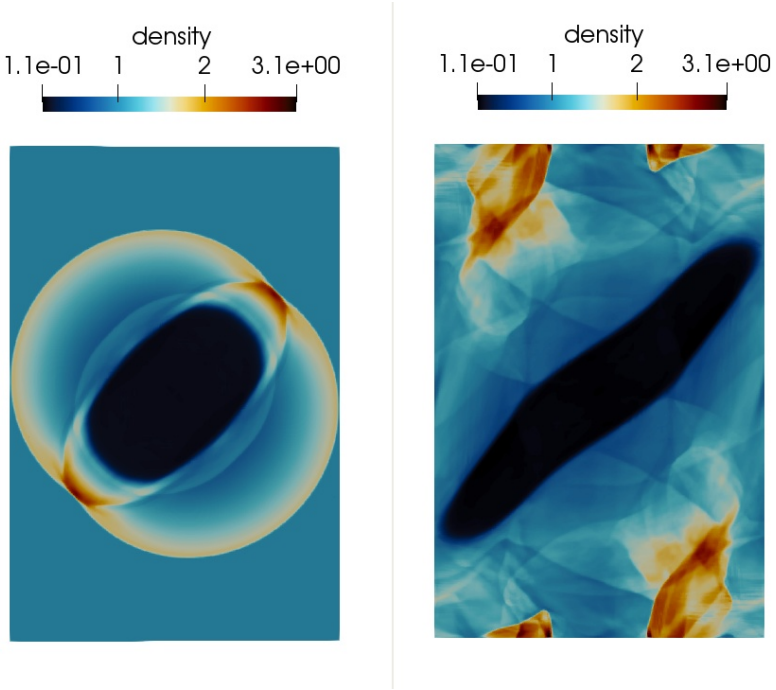


Figure 5.27: Obtained ρ distribution, $t \approx 0.2$ (left), $t \approx 1.0$ (right)

AMR for MHD Blast - extended version

One can see, that for the finest mesh of 200×300 elements, the solution obtained shows very detailed features, and is arguably of even higher quality than the reference one from Gaburov and Nitadori (2011). However, the calculation needs

$$200 \times 300 \ = 60000 \text{ elements}, \tag{5.7}$$

where each element contains 4 basis functions for each of $\rho, p, \pi_1, \pi_2, \pi_3$, and 11 basis functions for $\boldsymbol{B}$, that is 31 basis functions per element, in total $60000 \times 31 = 1860000$ degrees of freedom (DOFs). This number can be decreased, while keeping the same (or even higher) solution quality, with AMR - as illustrated in figures 5.28 to 5.35 below. On each of figures 5.28 to 5.35, the solution is displayed on the left, the adapted mesh in the middle, and the owning processor of a chunk of elements on the right. Note that there were 96 processors used for the computation.

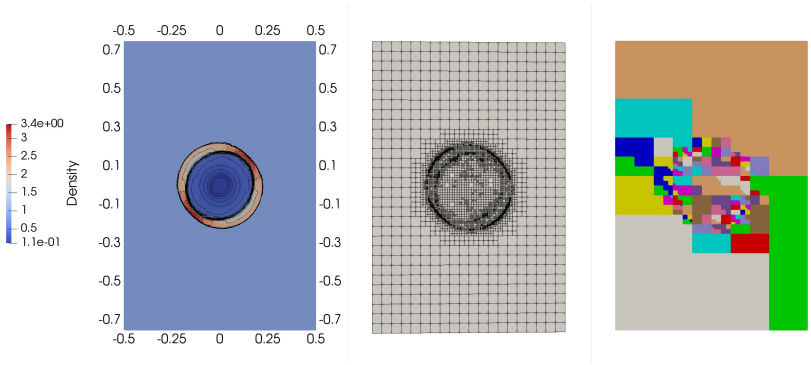


Figure 5.28: Obtained ρ distribution, mesh elements, and their owning processor. Time $t \approx 0.5e-1$.

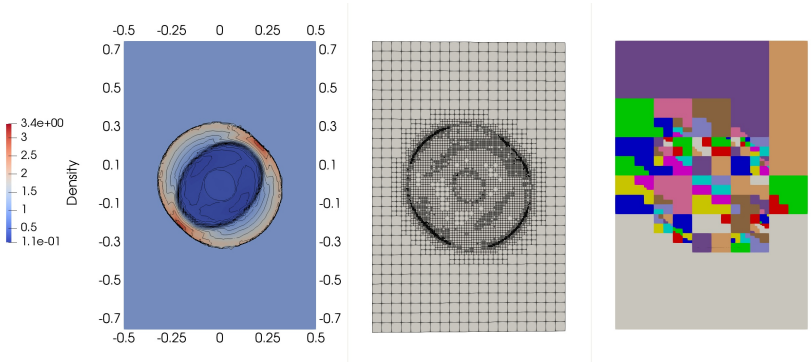


Figure 5.29: Obtained ρ distribution, mesh elements, and their owning processor. Time $t \approx 5e-1$.

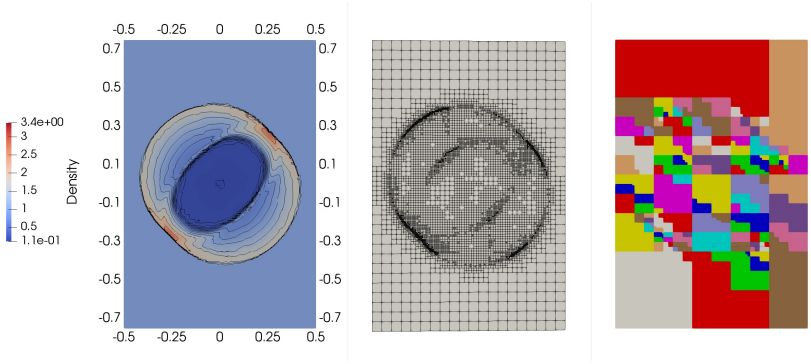


Figure 5.30: Obtained ρ distribution, mesh elements, and their owning processor. Time $t \approx 1.5e-1$.

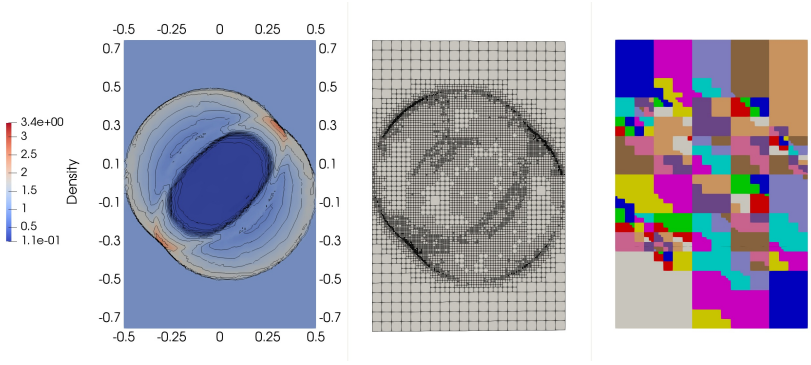


Figure 5.31: Obtained ρ distribution, mesh elements, and their owning processor. Time $t \approx 2e-1$.

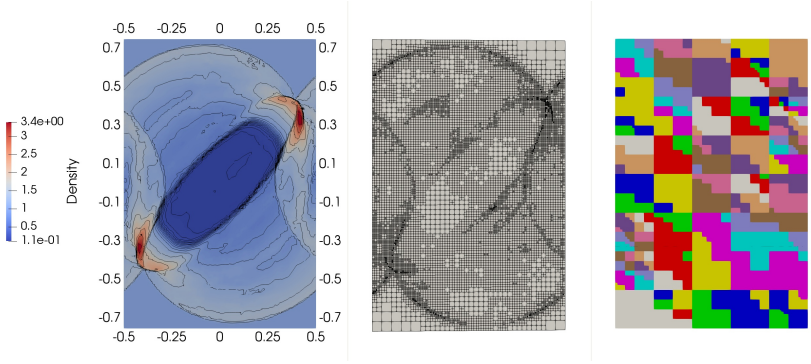


Figure 5.32: Obtained ρ distribution, mesh elements, and their owning processor. Time $t \approx 3.5e-1$.

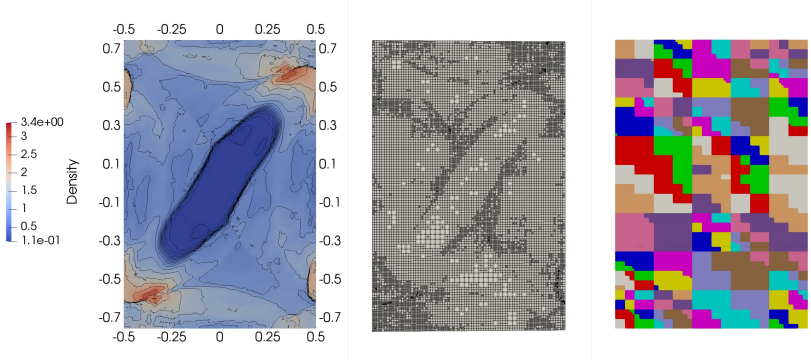


Figure 5.33: Obtained ρ distribution, mesh elements, and their owning processor. Time $t \approx 5.5e-1$.

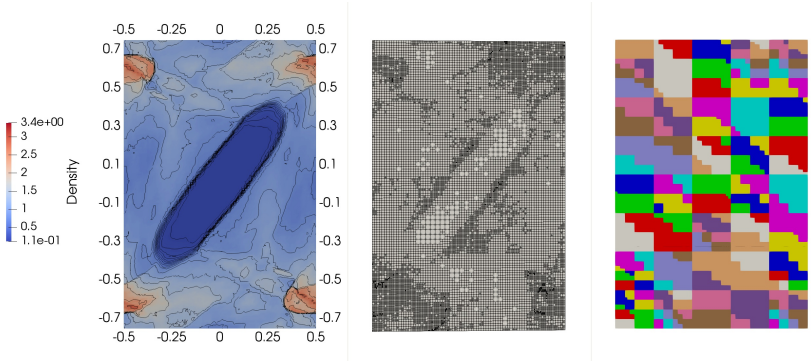


Figure 5.34: Obtained ρ distribution, mesh elements, and their owning processor. Time $t \approx 7e-1$.

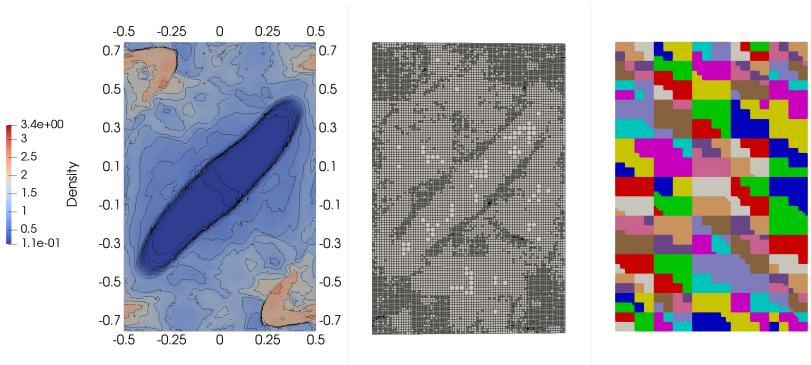


Figure 5.35: Obtained ρ distribution, mesh elements, and their owning processor. Time $t \approx 8.5e-1$.

In the table 5.1, mesh elements counts for all the displayed snapshots are given.

Time step	**Number of mesh elements**	**Number of DOFs**
$t \approx 0.5e-1$	5880	182280
$t \approx 1e-1$	7332	227292
$t \approx 1.5e-1$	8649	268119
$t \approx 2e-1$	10044	311364
$t \approx 3.5e-1$	14013	434403
$t \approx 5.5e-1$	16578	513918
$t \approx 7e-1$	17192	532952
$t \approx 8.5e-1$	18708	579948

Table 5.1: Mesh elements and DOFs counts for snapshots displayed in figures 5.28 to 5.35

Note that, as described in algorithm 5, the number of elements constantly evolves.

From table 5.1, it is cleary visible, that even for such a complex, and seemingly chaotic example, the AMR approach can cut down the number of mesh elements / DOFs radically, by 70-90 %, while achieving comparable results.

5.1.3 Orszag-Tang vortex

This problem was first described in Orszag and Tang (1979) and has been extensively used as a benchmark for 2- and 3- dimensional MHD code (Zachary et al. (1994), Londrillo and Zanna (2000), Skala and Barta (2012), Derigs et al. (2017), and many others). It is a simple model of the evolution of MHD turbulence including interactions between the several shock waves that appear. The Orszag-Tang system

is defined by the initial conditions:

$$
\begin{aligned}
\rho_0 &= \frac{25}{36\pi} \\
p_0 &= \frac{5}{12\pi} \\
B_0 &= \sqrt{\frac{1}{4\pi}} \\
\gamma &= 5/3 \\
\rho\left(\boldsymbol{x}, t=0\right) &= \rho_0, \\
p\left(\boldsymbol{x}, t=0\right) &= p_0, \\
\boldsymbol{u}_1\left(\boldsymbol{x}, t=0\right) &= -\sin(2\pi y), \\
\boldsymbol{u}_2\left(\boldsymbol{x}, t=0\right) &= \sin(2\pi x), \\
\boldsymbol{u}_3\left(\boldsymbol{x}, t=0\right) &= 0, \\
\boldsymbol{B}_1\left(\boldsymbol{x}, t=0\right) &= -B_0 \sin(2\pi y), \\
\boldsymbol{B}_2\left(\boldsymbol{x}, t=0\right) &= B_0 \sin(4\pi x), \\
\boldsymbol{B}_3\left(\boldsymbol{x}, t=0\right) &= 0, \\
U\left(\boldsymbol{x}, t=0\right) &= \frac{p_0}{\gamma-1} + U_m\left(\boldsymbol{x}, t\right) + U_k\left(\boldsymbol{x}, t\right),
\end{aligned}
\tag{5.8}
$$

where the last term is an application of equations (1.1) to (1.3). The domain Ω is set as $\Omega = [0,1] \times [0,1]$ and the problem is equipped with periodic boundary conditions on the top-bottom, and left-right parts of the boundary, similarly as in section 5.1.2. This configuration is strongly unstable, leading to a wide spectrum of propagating MHD modes and shock waves.

As before, in order to compare with reference papers, figures are presented from these papers - see figure 5.36. The images are taken from pages 30 (Londrillo and Zanna (2000)), and 20/282 (Zachary et al. (1994)) respectively. All are taken at $t = 0.5$.

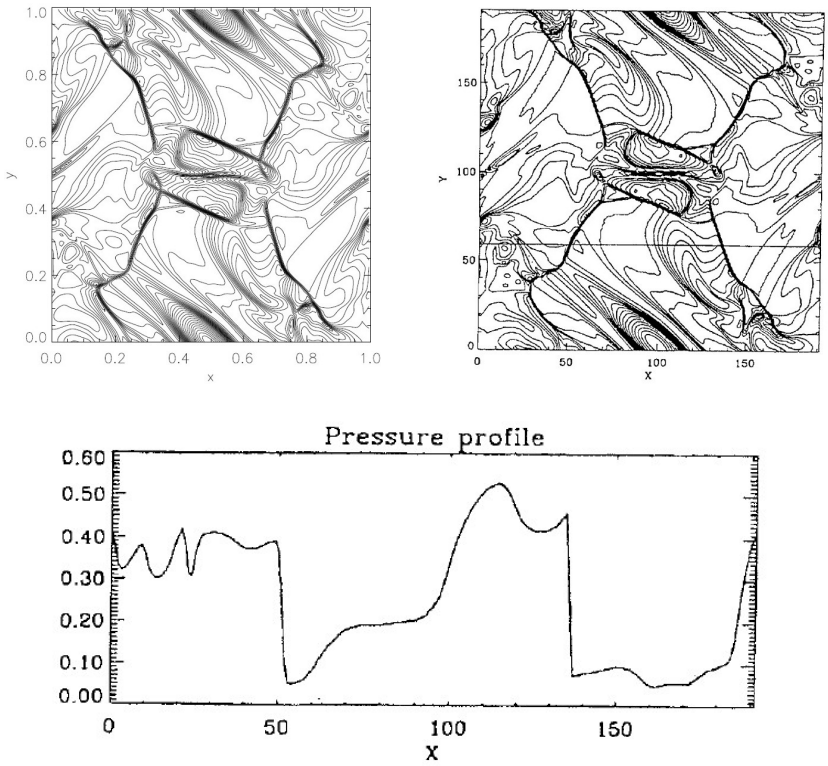


Figure 5.36: p isolines from Londrillo and Zanna (2000) (top left), p isolines from Zachary et al. (1994) (top right), p along $y = 0.3125$ from Zachary et al. (1994) (bottom).

Results obtained in the implemented software are presented in figures 5.37 to 5.43. The result on figure 5.43 is obviously in almost exact accordance with figure 5.36.

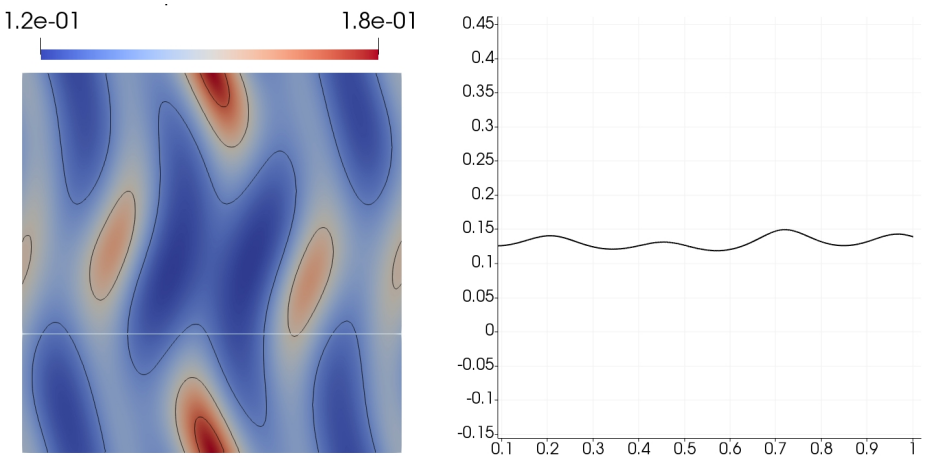


Figure 5.37: p distribution in Ω and p along $y = 0.3125$, $t = 0.03$

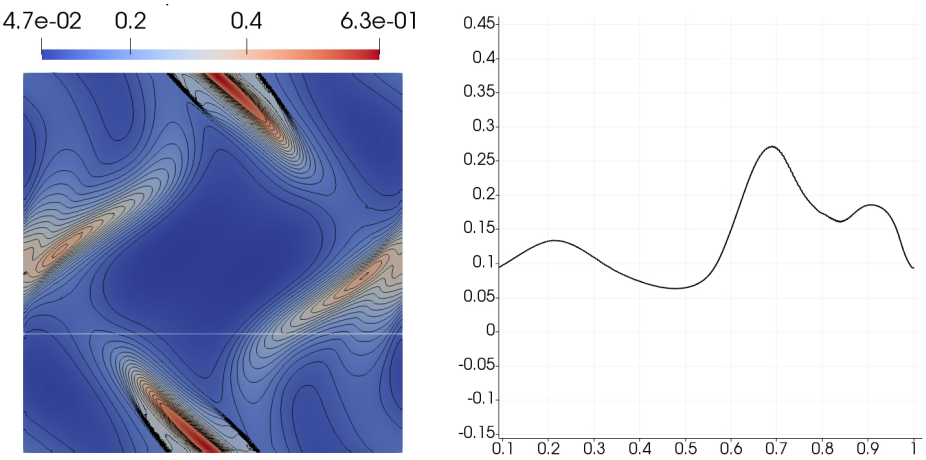


Figure 5.38: p distribution in Ω and p along $y = 0.3125$, $t = 0.18$

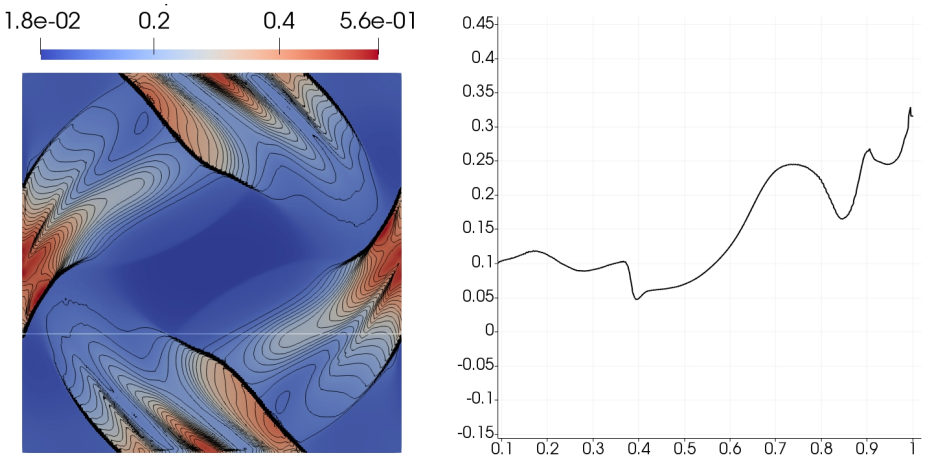


Figure 5.39: p distribution in Ω and p along $y = 0.3125$, $t = 0.28$

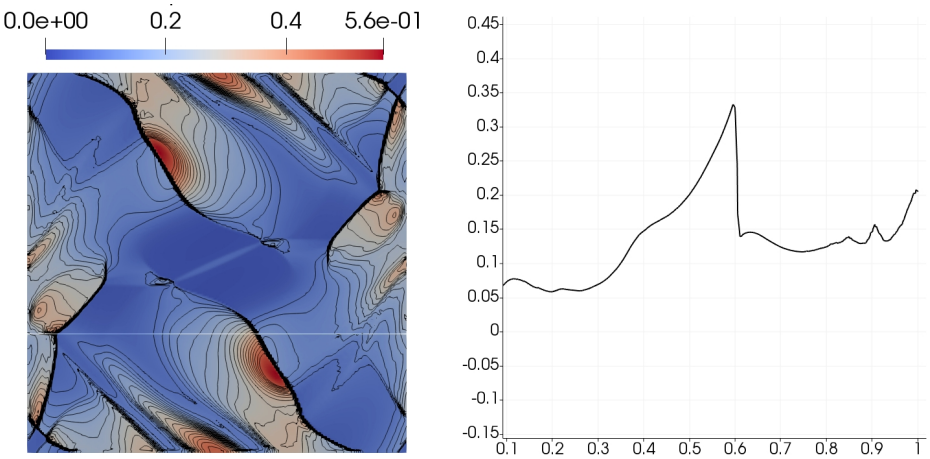


Figure 5.40: p distribution in Ω and p along $y = 0.3125$, $t = 0.38$

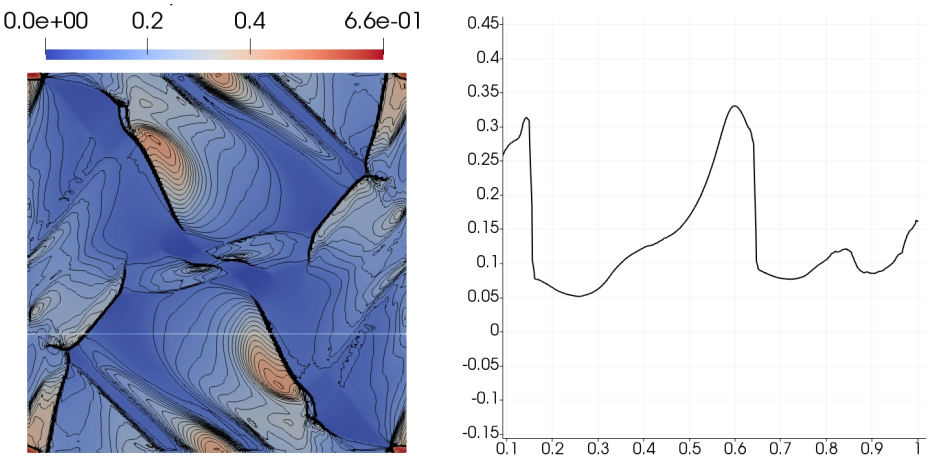


Figure 5.41: p distribution in Ω and p along $y = 0.3125$, $t = 0.46$

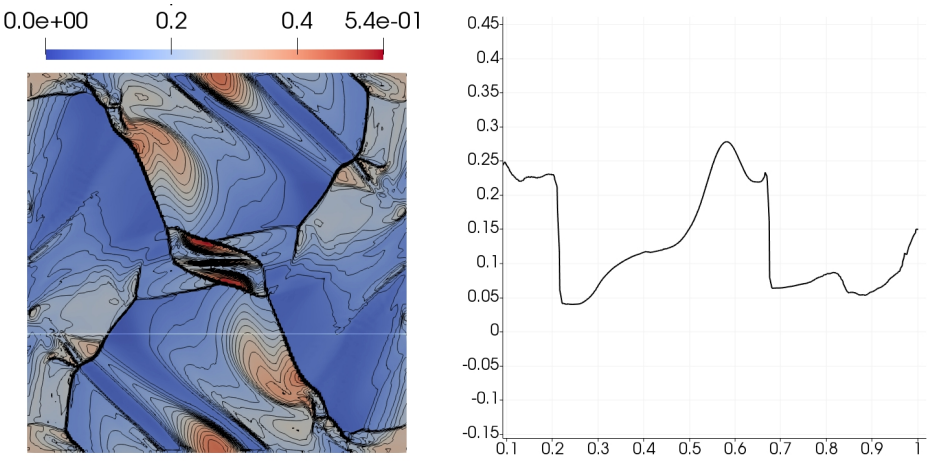


Figure 5.42: p distribution in Ω and p along $y = 0.3125$, $t = 0.48$

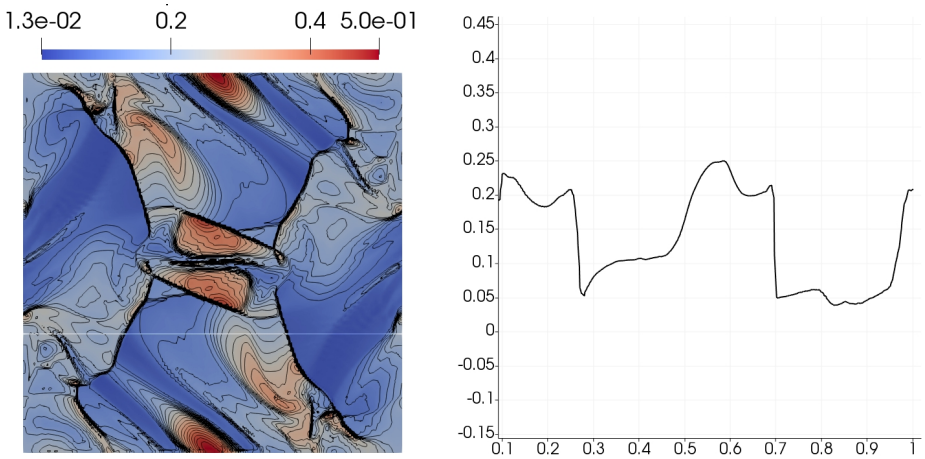


Figure 5.43: p distribution in Ω and p along $y = 0.3125$, $t = 0.5$

As noted above these figures, the resolution of all present shocks and discontinuities is very high, and is accordance with the reference results (figure 5.36) in Londrillo and Zanna (2000), and Zachary et al. (1994).

5.2 Flux tube eruption model

This model is based on the original Titov-Demoulin model from Titov and Demoulin (1999), as used in Kotrč et al. (2012), where the geometrical proportions, and equilibrium conditions are taken from Titov and Demoulin (1999).

5.2.1 Problem parameters

The model parameters are as follows. Note that k_B is the Boltzmann constant $k_B = 1.38064852 \times 10^{-23} \frac{\mathrm{J}}{\mathrm{K}}$, m_p is the plasma mass, and g gravitational acceleration.

Parameter values read

$$\begin{aligned}
\beta &= 0.05 \quad \text{... Plasma beta,} \\
L_G &= 2\ k_B \frac{T_{ext}}{(m_p g)} = 1.2 \times 10^8\,[\mathrm{m}]\,, \\
L_G &= 20 \quad \text{... Coronal height scale in dimension-less units,} \\
N_t &= 5 \quad \text{... Torus winding number,} \\
R &= 3 \quad \text{... Torus major radius,} \\
L &= 1.5 \quad \text{... Magnetic charge separation distance,} \\
d &= 1.5 \quad \text{... Geometrical factor,} \\
q &= \frac{\ln\left(8e^{-5/4}R\right)}{4} N_t \left(\frac{L}{R}\right)^2 \left[1 + \left(\frac{R}{L}\right)^2\right]^{3/2}, \\
q &\approx \text{Normalised magnetic charge corresponding to global equilibrium,} \\
H &= 2\ \frac{N_t^2}{R^2} \quad \text{... "Helicity" factor inside tho loop,} \\
\frac{T_{ext}}{T_{in}} &= 10 \quad \text{... Coronal/prominence temperature ratio.}
\end{aligned} \tag{5.9}$$

The domain Ω is taken as $[-2.5, 2.5] \times [-5, 5] \times [0, 5]$.

5.2.2 Initial condition

The model is equipped with an initial condition:

$$\rho\left(\boldsymbol{x}, t_0\right) = \exp\left(\frac{-z}{\frac{T_{ext}}{T_{in}} L_G}\right),\ \boldsymbol{x} \text{ inside the torus} \tag{5.10}$$

$$\rho\left(\boldsymbol{x}, t_0\right) = \frac{T_{in}}{T_{ext}} \exp\left(\frac{-z}{L_G}\right),\ \boldsymbol{x} \text{ outside the torus} \tag{5.11}$$

$$p\left(\boldsymbol{x}, t_0\right) = \beta \text{ everywhere} \tag{5.12}$$

$$\boldsymbol{v}\left(\boldsymbol{x}, t_0\right) = 0 \text{ everywhere} \tag{5.13}$$

$$\boldsymbol{B}\left(\boldsymbol{x}, t_0\right) \boldsymbol{B}_{in}\left(\boldsymbol{x}, t_0\right) + \boldsymbol{B}_{ext}\left(\boldsymbol{x}, t_0\right), \tag{5.14}$$

where $t_0 = 0$, and the equations for the magnetic field of the flux rope $\boldsymbol{B}_{in}\left(\boldsymbol{x}, t_0\right)$, magnetic field created by the (artificial) magnetic charges $\boldsymbol{B}_{ext}\left(\boldsymbol{x}, t_0\right)$ are the same as in equations (3) and (4) in Kotrč et al. (2012).

5.2.3 Boundary conditions

The model of equilibrium is equipped with these boundary conditions for density ρ, pressure p, and velocity $\boldsymbol{v}$:

$$\frac{\partial \rho}{\partial \boldsymbol{n}} = 0 \text{ everywhere,} \tag{5.15}$$

$$\frac{\partial p}{\partial \boldsymbol{n}} = 0 \text{ everywhere,} \tag{5.16}$$

$$\frac{\partial \boldsymbol{v}}{\partial \boldsymbol{n}} = 0 \text{ on the left, right, front, back, and top boundary, } z > 0, \tag{5.17}$$

$$\boldsymbol{v} = 0 \text{ on the bottom boundary, } z = 0, \tag{5.18}$$

$$\tag{5.19}$$

where the last equations represents the fixed ends of the flux rope on Sun's surface. The conditions for the magnetic field are more difficult to set, as we need to make sure that $\nabla \cdot \boldsymbol{B} = 0$ also for the magnetic field across the boundary. One way to achieve this is to set:

$$\frac{\partial B_{t_{1,2}}}{\partial \boldsymbol{n}} = 0, \tag{5.20}$$

$$\frac{\partial B_n}{\partial \boldsymbol{n}} = -\sum_{i=1,2} \frac{\partial B_{t_i}}{\partial \boldsymbol{t}_i}, \tag{5.21}$$

where $\boldsymbol{n}$ is the normal direction, $\boldsymbol{t}_i, i = 1, 2$ are the two tangential directions, and B_n, and $B_{t_{1,2}}$ stands for the normal, and two tangential components of the magnetic field $\boldsymbol{B}$. Also, U is calculated from the above so that the relation equation (1.3) holds.

6 Conclusion, outlook

As for the mathematical, numerical, and technical (software) parts for such tool to be delivered, all problems that were to be solved, such as

- Adaptive algorithm for the discretization of the time derivative (through CFL condition, see equation (3.41)),
- Shock-capturing for high-order DG scheme to prevent non-physical oscillations (through Vertex-based limiter, see algorithm 2),
- Adaptive algorithm for the discretization of the space derivatives (through AMR, entire chapter 4),
- A specific shapeset of basis and test functions based on Taylor expansions (through section 3.3),

have been solved, and moreover performance level meets the needs of the use cases. All this has been shown on benchmark problems (see sec:benchmarks), as well as real-world Titov-Demoulin-based simulation.

Of course, much can be improved upon, for example:

- Second-order scheme for the discretization of the time derivative,
- Adaptive algorithm for the discretization of the time derivative,
- Caching of values that are necessary in multiple spaces of the algorithm (utilizing the RAM),
- Further use of vectorization for evaluation of integral quantities,

but the original goal of preparing an easy-to-use, easy-to-extend, and well programmed, and tested software package, has been achieved:

- The code is able to utilize large-scale clusters through implementation being based on Message Passing Interface (MPI),
- The code is publicly available, and well documented - by following the schematics of the used numerical methods, as well as using clear naming conventions in the object model of the program,
- The code is quite ready for addition of new tests, new benchmarks, new examples, as well as easy parametrization of the existing ones.

6.1 Outloook

Further work will focus on real-world astrophysical problems, where the Titov-Demoulin-based simulations, although all mathematical and numerical apparatus is in place, still need work to satisfy the goal of being reliable and all-purpose tool for astrophysicists. Further work will focus on incorporating additional relevant physical phenomena - mainly study of the magnetic field reconnection - (Bárta et al., 2011), and other phenomena occurring both in solar physics and in industrial applications of plasma flow.

To sum up next steps with the already finished toolset, these are the logical next steps:

- Replicate fully the results of Kotrč et al. (2012), including all parameters, and boundary condition specifics,
- Run much more detailed simulation over a larger domain for a longer time, to be able to inspect the destructive behavior of the astrophysical event on small scales,
- Continuously improve the performance of the code, fix issues as they are discovered, and extend the implemented set of numerical schemes,
- Add additional relevant physics phenomena - resistivity, magnetic field reconnection, possibly relativistic effects,
- Get in touch with other possible users of the implemented software to enrich the set of possible use cases.

Bibliography

Balsara, D. S. and Spicer, D. S. (1999). A Staggered Mesh Algorithm Using High Order Godunov Fluxes to Ensure Solenoidal Magnetic Fields in Magnetohydrodynamic Simulations. *Journal of Computational Physics*, 149:270–292.

Bangerth, W., Davydov, D., Heister, T., Heltai, L., Kanschat, G., Kronbichler, M., Maier, M., Turcksin, B., and Wells, D. (2015). The `deal.II` library, version 8.4. *preprint*.

Barth, T. and Jespersen, D. (1989). The design and application of upwind schemes on unstructured meshes. In *27th Aerospace Sciences Meeting*, number AIAA-89-0366. American Institute of Aeronautics and Astronautics.

Batten, P., Clarke, N., Lambert, C., and Causon, D. (1997). On the choice of wavespeeds for the hllc riemann solver. *SIAM Journal on Scientific Computing*, 18(6):1553–1570.

Bárta, M., Büchner, J., Karlický, M., and Skála, J. (2011). Spontaneous current-layer fragmentation and cascading reconnection in solar flares. i. model and analysis. *The Astrophysical Journal*, 737(1):24.

Burstedde, C., Wilcox, L. C., and Ghattas, O. (2011). `p4est`: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees. *SIAM Journal on Scientific Computing*, 33(3):1103–1133.

Davis, T. (2006). *Direct Methods for Sparse Linear Systems*. Society for Industrial and Applied Mathematics.

Denner, F., Evrard, F., Serfaty, R., and van Wachem, B. G. (2017). Artificial viscosity model to mitigate numerical artefacts at fluid interfaces with surface tension. *Computers & Fluids*, 143:59 – 72.

Derigs, D., Winters, A., Gassner, G., Walch, S., and Bohm, M. (2017). Ideal glm-mhd: About the entropy consistent nine-wave magnetic field divergence diminishing ideal magnetohydrodynamics equations. 364.

Dolejší, V. and Feistauer, M. (2015). *Discontinuous Galerkin Method: Analysis and Applications to Compressible Flow*. Springer Series in Computational Mathematics. Springer International Publishing.

G., R. H. (1997). Thomas, j. w.: Numerical partial differential equations. finite difference methods. *ZAMM - Journal of Applied Mathematics and Mechanics / Zeitschrift für Angewandte Mathematik und Mechanik*, 77(5):386–386.

Gaburov, E. and Nitadori, K. (2011). Astrophysical weighted particle magnetohydrodynamics. *Monthly Notices of the Royal Astronomical Society*, 414(1):129–154.

Hariri, S., Park, J. B., Yu, F. K., Parashar, M., and Fox, G. C. (1993). A message passing interface for parallel and distributed computing. In *[1993] Proceedings The 2nd International Symposium on High Performance Distributed Computing*, pages 84–91.

Heroux, M. A., Bartlett, R. A., Howle, V. E., Hoekstra, R. J., Hu, J. J., Kolda, T. G., Lehoucq, R. B., Long, K. R., Pawlowski, R. P., Phipps, E. T., Salinger, A. G., Thornquist, H. K., Tuminaro, R. S., Willenbring, J. M., Williams, A., and Stanley, K. S. (2005). An overview of the trilinos project. *ACM Trans. Math. Softw.*, 31(3):397–423.

Intel Corporation (2017). Intel parallel studio.

J., R. W. and B., L. R. (2002). The use of classical lax–friedrichs riemann solvers with discontinuous galerkin methods. *International Journal for Numerical Methods in Fluids*, 40(334):479–486.

Kestener, P. et al. (1992). Ramses-gpu.

Kitware Inc. (2017). Paraview 5.5.

Korous, L. (2012). Adaptive hp discontinuous galerkin method for nonstationary compressible euler equations.

Korous, L. and Solin, P. (2012). An adaptive meshes for non-stationary compressible euler equations. *Computing*, 95(1):425–444.

Kotrč, P., Bárta, M., Schwartz, P., Kupryakov, Y., Kashapova, L., and Karlický, M. (2012). Modeling of ha eruptive events observed at the solar limb. 284.

Kuzmin, D. (2010). A vertex-based hierarchical slope limiter for p-adaptive discontinuous galerkin methods. *Journal of Computational and Applied Mathematics*, 233(12):3077 – 3085. Finite Element Methods in Engineering and Science (FEMTEC 2009).

Londrillo, P. and Zanna, L. D. (2000). High-order upwind schemes for multidimensional magnetohydrodynamics. *The Astrophysical Journal*, 530(1):508.

Ma, Z., Korous, L., and Santiago, E. (2012). Solving a suite of {NIST} benchmark problems for adaptive {FEM} with the hermes library. *Journal of Computational*

and Applied Mathematics, 236(18):4846 – 4861. {FEMTEC} 2011: 3rd International Conference on Computational Methods in Engineering and Science, May 9–13, 2011.

Miyoshi, T. and Kusano, K. (2005). A multi-state HLL approximate Riemann solver for ideal magnetohydrodynamics. *Journal of Computational Physics*, 208:315–344.

Mocz, P., Vogelsberger, M., Sijacki, D., Pakmor, R., and Hernquist, L. (2014). A discontinuous Galerkin method for solving the fluid and magnetohydrodynamic equations in astrophysical simulations. *mnras*, 437:397–414.

Norman, M., Wilson, J., et al. (1992). Zeus.

Orszag, S. A. and Tang, C.-M. (1979). Small-scale structure of two-dimensional magnetohydrodynamic turbulence. *Journal of Fluid Mechanics*, 90:129–143.

Rossmanith, J. A. (2013). High-Order Discontinuous Galerkin Finite Element Methods with Globally Divergence-Free Constrained Transport for Ideal MHD. *ArXiv e-prints*.

Skala, J. and Barta, M. (2012). LSFEM implementation of MHD numerical solver. *ArXiv e-prints*.

Skála, J., Baruffa, F., Büchner, J., and Rampp, M. (2015). The 3d mhd code goemhd3 for astrophysical plasmas with large reynolds numbers. *A&A*, 580:A48.

Solin, P., Korous, L., and Kus, P. (2014). Hermes2d, a c++ library for rapid development of adaptive -fem and -dg solvers. *Journal of Computational and Applied Mathematics*, 270:152 – 165. Fourth International Conference on Finite Element Methods in Engineering and Sciences (FEMTEC 2013).

Stone, J., Gardiner, T., Teuben, P., et al. (2008a). Athena.

Stone, J., Gardiner, T., Teuben, P., et al. (2008b). Athena.

Titov, V. and Demoulin, P. (1999). Basic topology of twisted magnetic configurations in solar flares. 351:707–720.

Xisto, C. M., Páscoa, J. C., and Oliveira, P. J. (2014). A pressure-based high resolution numerical method for resistive mhd. *Journal of Computational Physics*, 275:323 – 345.

Zachary, A., Malagoli, A., and Colella, P. (1994). A higher-order godunov method for multidimensional ideal magnetohydrodynamics. *SIAM Journal on Scientific Computing*, 15(2):263–284.