Universität zu Köln

Entropy Stable Discontinuous Galerkin Methods for Multi-Component Euler and Ideal Magnetohydrodynamics Equations with Chemical Networks in Julia

INAUGURAL-DISSERTATION

zur

Erlangung des Doktorgrades

der Mathematisch-Naturwissenschaftlichen Fakultät

der Universität zu Köln

vorgelegt von

Christof Martin Czernik

aus Dormagen

September 18, 2022

Berichterstatter:	Prof. Dr. Gregor Gassner Prof. Dr. Philipp Birken
Tag der mündlichen Prüfung:	28.11.2022

Danksagung

Diese Arbeit ist während meiner Tätigkeit als wissenschaftlicher Mitarbeiter am Mathematischen Institut der Universität zu Köln entstanden. Mein wissenschaftliches Projekt wurde unter anderem finanziert von dem European Research Council (EXTREME, project no. 714487).

Ganz herzlich bedanken möchte ich mich bei meinem Betreuer Prof. Dr. Gregor Gassner, der mir in seiner sympathischen Arbeitsgruppe die Möglichkeit zur Promotion gegeben hat. Hiermit möchte ich mich nicht nur für die hervorragende Betreuung bedanken, sondern ebenfalls für den ausgezeichneten Austausch und die akademische Freiheit bei der Umsetzung dieser Arbeit. Auch abseits des fachlichen Dialogs gab es immer reichlich spannende Gesprächsthemen.

Des Weiteren möchte ich mich hiermit auch bei der gesamten Arbeitsgruppe bedanken, mit der ich viele tolle Momente verbringen durfte. Auch in Zeiten von Corona gab es immer gute Laune und anregende Gesprächsthemen. Danken möchte ich zudem Michael, der mir immer mit hilfreichen Tipps zur Seite stand und mit dem ich immer wieder gerne ein Seminar organisieren würde. Ein besonderer Dank gilt Andrés, der mir bei Fragen immer zur Seite stand und mit dem man immer viele lustige Gespräche führen konnte.

Ich bedanke mich zudem auch bei Prof. Dr. Birken für die Zweitbegutachtung dieser Arbeit.

Ganz persönlich bedanken möchte ich mich bei meiner Familie, die auch in schwierigen Zeiten immer ein offenes Ohr für mich hatte. Hierbei geht mein Dank auch an das neueste Mitglied meiner Familie, Annika.

Mein ganz besonderer Dank geht an meine Frau Sissy, die mich nun seit über 10 Jahren auf meinem Lebensweg begleitet. Ohne ihre bedingungslose Liebe und unabdingbare Unterstützung wäre diese Arbeit nicht möglich gewesen.

Abstract

We present a high-order entropy-stable discontinuous Galerkin spectral element method (DGSEM) for multi-component Euler and multi-component ideal magnetohydrodynamics (MHD) equations with chemical reaction terms written in Julia. Instead of using a completely self-made code, we extend the already existing and proven simulation framework Trixi.jl, so that we can make our new introduced features available to the public.

For this purpose, we extend the simulation framework Trixi.jl with multi-component Euler and multi-component ideal MHD equations. Since we place value on entropy-stable processes, we add an entropy-conservative flux function for the multi-component Euler equations from the literature and propose an entropy-conservative flux function for the multi-component ideal MHD equations.

Trixi.jl contains a very effective shock-capturing method where the high-order DGSEM can be blended with a first-order Finite Volume (FV) scheme for cartesian meshes. To be able to simulate applications with more complex geometries we are going to extend this feature to unstructured and curvilinear meshes and make it work for multi-component equations.

Another feature in Trixi.jl is the positivity-preserving limiter, which is able to rescue a solution in difficult situations. In the literature, however, another method has emerged which, based on the shock-capturing method used in Trixi.jl, is able to preserve the positivity of density and pressure for single-component simulations. In this work we add this new positivity-preserving scheme to Trixi.jl and propose slight modifications for the multi-component case.

The new multi-component equations give us the opportunity to introduce chemical reactions into Trixi.jl. Since we advocate the use of packages in this work,

we will add an external package specialized on the solution of chemical reactions and give an overview and introduction to all important packages in Trixi.jl.

Finally, we provide numerical test cases that verify the theoretical properties of the new introduced features and demonstrate the strengths and weaknesses of our method. Additionally, we demonstrate the capabilities of our method with complex numerical examples.

Contents

Symbols

t	time variable
$\frac{\partial}{\partial t}$	partial time derivative
x	space variable
$\frac{\partial}{\partial x}$	partial space derivative
u	vector of state variables
f	flux function
m	number of state variables
Ω	computational domain
$\mathcal{J}$	jacobian
ϕ	smooth test function
T	end time
$[\![(\cdot)]\!]$	jump oprator
λ	eigenvalue of the flux jacobian
U	entropy function
F	entropy flux
s	physical specific entropy
w, w_k	vector of entropy variables, quadrature weights
ρ	density
ρ_k	partial density of component k
v	velocity
$\vec{v}$	vector of velocities in three dimensions
p	pressure
$R, \tilde{R}, R_{\text{inf}}$	gas constant, specific gas constant, universal gas constant
T_{gas}	temperature
γ	total heat capacity ratio
c_p	specific heats at constant pressure
c_v	specific heats at constant volume
Γ	non-conservative terms
B	magnetic field
$\vec{B}$	vector of magnetic field
c_h	hyperbolic divergence cleaning
ψ	divergence-correcting field
N	number of components
r_k	specific gas constant of species k
m_k	molar mass of specied k
c_{vk}	specific heats at constant volume of species k
c_{pk}	specific heats at constant pressure of species k
s_k	specific entropy of species k
Y_k	mass fraction of species k
S	source term

ω_k	chemical production rate of species k
W_k	molecular weight of species k
K_l^f, K_l^b	forward and backward reaction rates of each chemical reaction l
nr	total number of chemical reactions
$\tau_{k,l}^f, \tau_{k,l}^b$	forward and backward stoichiometric coefficients of species k and reaction l
q_k	chemical heat released by species k
Da	Damkohler number
N_q	number of elements
Q_i	element i of the domain
$\Delta t, \Delta x$	timestep, spatial distance measure of mesh
$\langle . \rangle$	scalar product
$\overline{f}$	average surface flux
f^*	numerical two-point flux
E	reference space
N_p	polynomial degree
ξ, η	reference coordinates
δ	Kronecker delta
M	mass matrix
D	differentiation matrix
B	boundary matrix
$f^\#$	entropy-conservative two-point flux
α	blending factor
$\mathcal{T}$	shock threshold
β	factor of deviation from safe solution
$P^{a \to b}$	projection operator from a to b
$\tilde{F}_i, \tilde{D}_i$	set of formation, set of destruction of species i
$\tilde{\mathcal{R}}_j$	set of reactants of reaction j
$\mathcal{R}$	right eigenvectors of the flux
Θ	diagonal matrix with the eigenvalues of the flux

Acronyms

1D, 2D, 3D	1-Dimension, 2-Dimensions, 3-Dimensions
AMR	Adaptive Mesh Refinement
CFD	Computational Fluid Dynamics
CFL	Courant-Friedrich-Levy
DG	Discontinuous Galerkin
DGSEM	Discontinuous Galerkin Spectral Element Method
EC	Entropy Conservative
ENO	Essentially Non-Oscillatory
EOC	Experimental Order of Convergence
ES	Entropy Stable
FD	Finite Difference
FE	Finite Element
FV	Finite Volume
GLM	Generalized Lagrangian Multiplier
HLL	Harten-Lax-Leer
LGL	Legendre-Gauss-Lobatto
LLF	Local Lax-Friedrichs
MHD	Magnetohydrodynamic
NE	Neighboring Elements
ODE	Ordinary Differential Equations
PDE	Partial Differential Equation
RK	Runge-Kutta
SBP	Summation By Parts
SSP	Strong-Stability Preserving
SSPRK	Strong-Stability Preserving Runge-Kutta
TVD	Total Variation Diminishing
WENO	Weighted Essentially Non-Oscillatory

1 Introduction

This thesis deals with fluid mechanical processes that are to be approximated or simulated with the help of numerical methods, also called *computational fluid dynamics* (CFD). In the following, we will describe the physical and chemical processes as well as the associated problems that we are trying to deal with. We will show the influence of numerical simulations on our world today and what possibilities they still hold. In addition, we look at the state-of-the-art methods currently being used by science and industry and explain why high-order methods are becoming increasingly important for the future. We will then go into the research questions and briefly outline the structure of this thesis.

1.1 Physical and Chemical Processes

Fluid mechanics is the science of studying the physical behavior of fluids and gases. It has a very wide field of application, like mechanical engineering, astrophysics, medicine, and chemical engineering to name a few. An important subfield are gas dynamics, which deal with density-variable (i.e. compressible) flows. This is usually the case for transonic and supersonic flows e.g. flows around an aircraft leading to abrupt changes in temperature and density. These abrupt changes usually lead to shock waves and compression shocks which play a crucial role in this thesis. There are applications (e.g. in astrophysics) in which a flow may be influenced by magnetic fields or gravity. Further applications may work with several different gases or fluids that interact with each other and can even lead to chemical reactions, as in combustion processes or detonations. As we will see, our focus is precisely on these last applications.

Most of these applications are quite complex, so that an analytical solution borders on impossibility. However, with the help of analytical and experimental methods, certain macroscopic behaviour can be derived, which can then be described in the form of equations. In this thesis we are mainly concerned with

the Euler equations of fluid dynamics, which are a special case of the Navier-Stokes equations neglecting viscosity and heat conductivity. They are stated as a system of *partial differential equations* (PDEs), which belong to the class of non-linear hyperbolic conservation laws consisting of the conservation of mass, momentum and energy. These equations are used for a variety of applications where friction and heat conduction are not important. However, additional effects may need to be described for certain applications, such as the influence of magnetic fields. Therefore, the Euler equations can be extended to the so called *magnetohydrodynamic* (MHD) equations, with which it is possible to describe the flow of magnetofluids like plasmas. Another extension that can be made is the multi-component extension, so that the flow of several different gases can be described at the same time. This can be taken even further by allowing chemical reactions between these different components and thus leading to more complex applications including combustion processes as well as detonation processes, which can be modeled by additional *ordinary differential equations* (ODEs).

1.2 Solution through Numerical Discretization

Although we can set up (simplified) equations for the observed physical and chemical processes, it is usually not possible to solve them analytically. As we know, there is still no general existence and uniqueness proof of a solution for the Navier-Stokes equations, which are basically the Euler equations extended by viscosity and heat conductivity. However, the solution of these equations is of such great interest to society that it even counts as one of the seven millennium problems of mathematics. The solution to these problems carries a prize money of one million US dollars.

Now we are faced with the problem that many technical and physical applications are modeled with the help of these equations. So how can we be sure that the plane we have designed will not crash if we do not know the exact solution to this problem? One option is to recreate physical behaviour experimentally, which often involves a great deal of effort and expense. For an aircraft, you have to build huge wind tunnels and construct each component individually. Of course, this also leads to the fact that many processes cannot be reproduced at all or only to a limited extent. If, for example, we want to build an aircraft

engine or a wind turbine as efficiently as possible, this means that we have to try out many different configurations of wing positions and angles to arrive at the optimum configuration.

A possible solution to this problem is the numerical simulation of this kind of problems. Especially in the last decades, the application area of numerical simulations has greatly expanded with the increasing computing power of computers. To keep pace with this development, however, it is also necessary to construct numerical methods that can exploit the full potential of these new possibilities. With the rise of the computer, supercomputers are also becoming more common, advertising even higher performance. Here, for example, it is useful to apply methods that can be parallelized to a high degree and act with high accuracy. This way, it is possible to find even faster and better results for questions that deserve our attention. An everyday problem here is, for example, modern weather forecasting, which would not be possible without adapted numerical methods and computational clusters.

1.3 Motivation for High-Order Schemes

As we have seen in the previous section, it is important to keep pace with technical developments and to derive better numerical methods. Nowadays, science and industry still like to use state of the art lower-order methods, as these have proven to be robust over time. These lower-order methods differ from the high-order methods in the way that low-order methods can achieve higher accuracy only by grid refinement. In contrast, some high-order methods, like the DG method, can achieve higher accuracies also by increasing the polynomial degree. This can have the advantage that high-order methods with the same number of degrees of freedom (i.e. including grid resolution and polynomial degree) as low-order methods can achieve more accurate results than low-order methods [1].

In our research area this would be e.g. the finite difference (FD) method as well as the finite volume (FV) method. Although, these methods have higher-order variants by increasing the stencil, their mainly used lower-order variants struggle to resolve finer features. These inaccuracies may lead to unacceptable solutions in the long run [2]. Analytical knowledge of these methods is,

however, at a high level, as many scientists have worked with them over the years. But it is precisely because of this kind of fundamental research, that we are now able to construct improved higher-order methods that can show off with greatly increased accuracy and better efficiency in terms of less work for a given accuracy [1]. Of course, high-order methods also have downsides, depending on the area of application, which we will discuss in the following.

One of the biggest problems, for example, is robustness. For instance, in our applications, the positivity of certain quantities such as density or pressure is indispensable, otherwise we would violate physical laws leading to wrong solutions. In this thesis we are studying the discontinuous Galerkin (DG) method, which is a conservative, high-order accurate, easy parallelizable method and able to handle complex geometries. It is closely related to the finite element (FE) method and differs roughly in the discontinuous interfaces between the elements. The high-order DG method has a habit of oscillating at shocks [3], which can lead to negative densities and pressure. It is easy to imagine that this problem means more work for us, as we have to try to circumvent these kind of difficulties with special procedures and ideas. However, this also increases the complexity of the methods. As is so often the case, the choice of the preferable method depends on weighing the advantages and disadvantages of each individual method. The trade-off is usually between the three cornerstones of robustness, speed and accuracy. The long-term goal here, however, is to someday construct a robust, fast and accurate scheme that science and industry can work with to solve the great problems of our time. We hope that this thesis will help us to get a little bit closer to this goal.

1.4 Research Questions and Objectives

Now that we have a rough idea of the area in which this thesis is set, let us look at the aim of our work. In doing so, we will raise some questions that we will try to answer in the course of this thesis. The success of this endeavor will be discussed in the conclusion at the end.

(1) How well does the high-order DGFV hybrid method perform for multi-component equations applications?

As already mentioned, lower-order methods are still commonly used for many applications. Even though these are robust, they usually provide less accurate solutions. In this thesis we want to investigate whether our high-order method is suitable for multi-component equations by comparing its robustness to the first-order FV method.

(2) Is it possible to construct an entropy-stable high-order DG method for the multi-component ideal MHD equations?

High-order methods are not usually known for their robustness and stability. One idea to address this difficulty is to construct entropy-stable high-order methods [4–6]. In the case of our DG method, we achieve this with the help of entropy-conservative (EC) fluxes which have been known for some time for the single-component Euler and ideal MHD equations. For the multi-component equations these entropy-conservative fluxes are relatively new, only recently Gouasmi et al. [7] had derived an entropy-conservative flux for the multi-component Euler equations. However, since we also want to work with ideal MHD equations, we derive an entropy-conservative flux for the multi-component ideal MHD equations.

(3) How well does the high-order DG method perform when we allow chemical networks in addition to the multi-component equations?

The addition of chemical networks for multi-component equations leads to a variety of new applications. However, these usually imply further complexity for the numerical methods. This is usually not a problem for robust low-order methods, but might be problematic for high-order methods like the DG method. Here, we will investigate whether the shock-capturing method by Hennemann et al. [8], already integrated in Trixi.jl, still works well when we allows chemical networks or whether something needs to be changed in the shock-capturing method.

(4) What do we need to add to the simulation framework Trixi.jl so that we can use it for our thesis?

In order to answer the questions posed in this thesis, it is necessary to build a suitable simulation code with which we are able to properly test our new introduced features in this thesis. For such a task usually a completely self-made code is build, which is no longer used afterwards. A better idea is to extend an already existing and proven simulation framework so that we can make our new introduced features available to the general public. However, this is usually not that easy, because, for example, data structures do not fit or certain features are not integrated. In this thesis we will rely on the numerical simulation framework Trixi.jl, see for instance Ranocha et al. [9], which we have to extend with 1D and 2D multi-component equations for Euler and ideal MHD together with suitable entropy-conservative fluxes. Here we will introduce and derive an entropy-conservative flux for the multi-component ideal MHD equations. For our future applications we will also need to find a way to integrate chemical networks and extend the existing shock-capturing method for curvilinear grids. In addition, we will integrate a positivity-preserving limiter which is based on the shock-capturing method, which was developed by Rueda-Ramírez and Gassner [10]. Following on from this we propose an extension of the positivity-preserving limiter for multi-component equations.

1.5 Outline of the Thesis

The rest of this thesis is structured as follows.

Chapter 2

First, we start with the physical model which leads us to the governing equations. Here we look at the special case of conservation laws that this thesis focuses on and explain what weak solutions are all about and why we emphasise an entropy condition. We give a brief insight into the formulation of the Euler and ideal MHD equations for the single-component and multi-component case as well as for chemical networks.

Chapter 3

Based on the previous chapter, we then go into mathematical modelling by presenting the building blocks of our numerical method with which we want to solve the previously mentioned equations numerically. Here, we first introduce the well-known state-of-the-art first-order FV method, which we want to use as a safety net for our final scheme. We then go into the derivation of the high-order discontinuous Galerkin spectral element method (DGSEM). In order to increase the robustness of the DGSEM, we then go into entropy-stability and derive entropy-conservative fluxes for the multi-component Euler and ideal MHD equations. Since the resulting entropy-stable high-order DG method is not really suitable for shock-heavy simulations, we introduce a shock-capturing scheme in which the first-order FV method is blended with the high-order DGSEM. Here we also go into more detail on how to choose the shock indicator and how the convex blending works. In very rare cases, an application cannot be simulated robustly even with the very well working shock-capturing method. For this case, a positivity-preserving limiter is introduced which is based on the shock-capturing method and extended for the multi-component case. At the end, we will discuss adaptive mesh refinement (AMR), which we will also use in our applications for efficiency reasons, and briefly explain how time integration works.

Chapter 4

Subsequently, we will discuss the simulation framework and external packages used in this thesis. We will give an introduction to the numerical simulation framework Trixi.jl, covering its main features, code structure, and how it can be used by users and developers. Since Trixi.jl has outsourced the time integration procedure using an external package DifferentialEquations.jl, we will briefly discuss the features and benefits of this package as well. Chemical networks, which are a separate system of ODEs, are also solved using a specialized external package called *KROME*. For this purpose, a Julia wrapper KROME.jl was specially built with which KROME can also be used in Julia. We will discuss the application as well as the advantages and disadvantages of the package. Towards the end, we will briefly elaborate on the mesh generation available in Trixi.jl and discuss the external package called *HOHQMesh* in more detail. Finally, we will revisit all these parts and show how to build an

elixir in Trixi.jl.

Chapter 5

Now we also have to show that our developed method works properly by running certain verification tests. We will first show the high-order accuracy of the method by running experimental order of convergence (EOC) tests for the multi-component Euler and ideal MHD equations and verify the conservation properties. We will then put the shock-capturing properties of the method to the test by running the well-known Sod's shock tube test, for example. The other building blocks of our method, such as the positivity-preserving limiter, adaptive mesh refinement and chemical networks, will also be examined here.

Chapter 6

The next chapter is probably one of the most exciting for readers, as it presents the results of the developed method for more complex cases. Here, we will specifically address applications with the multi-component equations. We will simulate applications with complicated and stiff chemical networks, sometimes even using AMR and curvilinear grids and show the benefits and drawbacks of our numerical scheme.

Chapter 7

In the final chapter, we will recap our successes and discuss the advantages and disadvantages of our developed method. Last but not least, we will take a look at future research questions that have not yet been answered by us.

2 Physical Model - Governing Equations

As mentioned in the previous chapter, we are interested in solving physical and chemical processes numerically by providing a suitable numerical scheme. For this, however, it is necessary to investigate the problem analytically. Therefore, we are interested in the mathematical groundwork of the underlying equations we are trying to solve in this thesis. The continuous analysis of the equations allows us to identify certain properties and constraints (e.g. entropy inequalities) that we will try to mimic at the discrete level.

In the following sections, we will give an overview of conservation laws, a specific subfield of PDEs on which we focus on this thesis. Thereupon, we will derive important representatives of this class of PDEs, namely the Euler equations as well as the ideal MHD equations. Next, we will extend those equations, which have been stated in a single-component formulation, into a multi-component formulation which allows to describe fluid dynamics of multiple different components. On this basis, it is then possible for us to describe chemical networks which are able to converse individual components into one and other.

2.1 Conservation Laws

In nature, there are many processes that work with the principle of conservation of some quantity. One of the most famous is the law of conservation of energy, which states that energy cannot increase or decrease in an isolated system over time, it can only transform into another form of energy. Phenomena like this can be represented quite easily as systems of time-dependent PDEs, which can be written mathematically as follows:

$$\frac{\partial}{\partial t}u(x,t) + \frac{\partial}{\partial x}f(u(x,t)) = 0. \tag{2.1}$$

Here, we use $u : \mathbb{R} \times \mathbb{R} \to \mathbb{R}^m$ as an m-dimensional vector of quantities, like mass, momentum or energy and $f : \mathbb{R}^m \to \mathbb{R}^m$ as the flux functions for the system of conservation laws on the domain $\Omega \subset \mathbb{R}$. To see how these conservation laws are derived from physical principles, we refer to Leveque [11]. Equation (2.1) can also be rewritten in the so called *quasi-linear* form

$$\frac{\partial}{\partial t}u(x,t) + \mathcal{J}(u)\frac{\partial}{\partial x}u(x,t) = 0 \tag{2.2}$$

with $\mathcal{J}(u) = \frac{\partial}{\partial u}f$ being the Jacobian. If $\mathcal{J}$ has real eigenvalues and is diagonalizable, then these equations belong to a special subset of PDEs called *hyperbolic* PDEs, which have their own special properties we can take advantage of while deriving a fitting numerical scheme.

A very important property is for example the wave-like behaviour of the solutions, which means that information can only travel with a finite speed over the domain. On the basis of this property, we are allowed to derive numerical schemes which do not have to take the information over the entire domain to calculate the update at a particular point, which results in better parallelizable schemes and therefore in faster schemes. Another related property is that information travels along characteristics of the equations which allows us to track the propagation of the information resulting in a better understanding of how to derive a suitable Riemann solver, as we will see in the following subsections. The problem of deriving suitable Riemann solvers has a great depth and is out of the scope of this thesis, so we refer for further details to the following sources [1, 12, 13].

2.1.1 Weak Formulation

Due to the potential non-linearity of the equations, discontinuities may arise in the solution even if the initial condition has been choosen to be smooth [14]. Should this be the case, it follows that the differentiability of (2.1) is no longer valid. Therefore, we can manipulate the equations to redirect the derivative onto a smooth function, leading to the so called *weak form*. According to the Lax-Wendroff theorem [15], if a conservative method, which is able to conserve quantities in a discrete sense, converges, it is then guaranteed to converge to a weak solution.

To achieve this, we first need a smooth test function $\phi \in \mathcal{C}^\infty$ with compact support $\Omega \times [0, T)$ which we introduce by multiplying it with each conservation equation in (2.1)

$$\phi \frac{\partial}{\partial t} u(x,t) + \phi \frac{\partial}{\partial x} f(u(x,t)) = 0. \tag{2.3}$$

Now we integrate the equations in space and time

$$\int_0^T \int_\Omega \left(\phi \frac{\partial}{\partial t} u(x,t) + \phi \frac{\partial}{\partial x} f(u(x,t)) \right) dx dt = 0. \tag{2.4}$$

By using integration by parts, we can move the derivative onto the test function as follows

$$\int_0^T \int_\Omega u(x,t) \frac{\partial}{\partial t} \phi + f(u(x,t)) \frac{\partial}{\partial x} \phi = - \int_\Omega \phi(x,0) u(x,0) dx. \tag{2.5}$$

This weak formulation (2.5) is fundamental for designing and analyzing numerical methods, like our high-order DGSEM. The solution of the weak formulation (2.5) is called *weak solution* and is usually not unique. In fact, there is an infinite amount of different weak solutions and it is not trivial to find a correct physical solution [16]. This circumstance follows from the fact that we are trying to model the real world with a simplified approach which neglects some fundamental physical processes. To find the physical correct solution, it is necessary to identify the missing piece which was neglected initially in the model. Since we are working with gas dynamics, it only seems natural that we have to satisfy the second law of thermodynamics, which states that the physical entropy has to increase for a spontaneous process like a shock wave. It turns out that this condition is partly sufficient to converge to a physical correct solution [1, 17]. Numerical schemes for gas dynamics satisfying this condition in a discrete sense seem to confirm this assumption by resulting in more robust solutions [5]. But let us go back to the beginning and take a closer look at the development and behaviour of shocks.

2.1.2 Rankine-Hugoniot Jump Condition

To get a better understanding of shocks, it is necessary to analyze the associated characteristics of the equations. Characteristics are curves $x(t)$ in the

$x-t$ plane, along which the solution of the PDE travels at a constant characteristic speed leading to an ODE. We speak of a shock when two characteristics intersect with each other [13]. Similar to the characteristics a shock can move with a certain velocity called *shock speed* s [18], which is related to jumps in conservative variables u and flux functions f satisfying the Rankine-Hugoniot condition

$$s[\![u]\!] = [\![f]\!], \tag{2.6}$$

with the jump operator

$$[\![a]\!] := a_R - a_L, \tag{2.7}$$

where the underscript L, R denote the left and right states of the shock. It follows that the solution u containing discontinuities represents a weak solution precisely when it satisfies the Rankine-Hugoniot condition (2.6) across shocks and satisfies the differential form of the conservation law in smooth regions [19].

2.1.3 Entropy Condition

Now that we have introduced the set of weak solutions, we are faced with the problem of identifying a unique and physically correct solution [14, 17]. For a non-linear field it follows the Lax entropy condition

$$\frac{\partial}{\partial x}\lambda_p(u_L) > s > \frac{\partial}{\partial x}\lambda_p(u_R), \quad p = 1, \dots, m \tag{2.8}$$

from the Rankine-Hugoniot condition with $\lambda_1 = \frac{\partial}{\partial x} f(u)$ for the scalar case ($m = 1$), condition (2.8) means that characteristics should go into the shock instead of evolving out of the shock as time advances [11]. Solutions should satisfy this condition when a shock is present to not be an unphysical solution.

In this thesis, we want to derive physical solutions with the help of entropy conditions by extending this idea to a formal framework we can use later on. Therefore, we introduce the convex function called *entropy function* $U(u)$ and the corresponding *entropy flux* $F(u)$ which we call *entropy flux pair* (U, F). This entropy flux pair must satisfy the following condition

$$\left\langle \frac{\partial}{\partial u} U, \frac{\partial}{\partial u} f \right\rangle = \frac{\partial}{\partial u} F, \tag{2.9}$$

with the new set of entropy variables $w := \frac{\partial}{\partial u} U$. We require the entropy variables to symmetrize the system of conservation laws. Therefore, it is possible to treat the entropy variables as independent variables which we can write as

$$\frac{\partial}{\partial t} u + \frac{\partial}{\partial x} f = \frac{\partial}{\partial w} u \frac{\partial}{\partial t} w + \frac{\partial}{\partial w} f \frac{\partial}{\partial x} w = 0, \tag{2.10}$$

with $\frac{\partial}{\partial w} u$ and $\frac{\partial}{\partial w} f$ being symmetric matrices. Taking a look at thermodynamics, we know that entropy is produced by dissipative processes, hence we introduce viscous terms to our conservation laws with the aim to identify a correct physical solution by regularizing the conservation law

$$\frac{\partial}{\partial t} u^\epsilon + \frac{\partial}{\partial x} f(u^\epsilon) = \epsilon \frac{\partial^2}{\partial x^2} u^\epsilon, \tag{2.11}$$

so that the shock discontinuities are smoothed out entirely guaranteering the existence of a strong solution which satisfies the conservation laws in differential form [1, 20]. On this basis, we are able to define the so called *entropy solution* as the limit of this viscous solution

$$u = \lim_{\epsilon \to 0} u^\epsilon, \quad \epsilon > 0. \tag{2.12}$$

With the help of the introduced entropy variables w, we are able to derive a regularized entropy equation

$$\frac{\partial}{\partial t} U^\epsilon + \frac{\partial}{\partial x} F^\epsilon = \epsilon \left\langle w^\epsilon, \frac{\partial^2}{\partial x^2} u^\epsilon \right\rangle, \tag{2.13}$$

which can be rewritten as an inequality

$$\frac{\partial}{\partial t} U^\epsilon + \frac{\partial}{\partial x} F^\epsilon \leq \epsilon \frac{\partial}{\partial x} \left\langle w^\epsilon, \frac{\partial}{\partial x} u^\epsilon \right\rangle, \tag{2.14}$$

using the product rule for the viscous terms as well as the symmetric positive definitely property of the Hessian of the entropy function [21]. If we now take the limit $\epsilon \to 0$ and integrate over the domain Ω, we obtain an entropy inequality

$$\int_\Omega \frac{\partial}{\partial t} U + \frac{\partial}{\partial x} F dx \leq 0 \tag{2.15}$$

which states that the (mathematical) entropy can only remain the same or decrease in time [20]. This property is directly linked to the concept of thermodynamic entropy which states the same reversed property. In other words, the mathematical entropy is related to the physical entropy, except that it has the opposite sign.

2.1.4 Riemann Problem

To incorporate the entropy stability property into our numerical schemes, we want to derive entropy stable Riemann solvers. A so called *Riemann problem* [13, 22] considers a single interface with two different constant states u_L and u_R meeting at the same spatial position. As we will see in the next chapter, our numerical methods for hyperbolic conservation laws are developed based on the Riemann problem 2.1. For the sake of simplicity, we consider the following

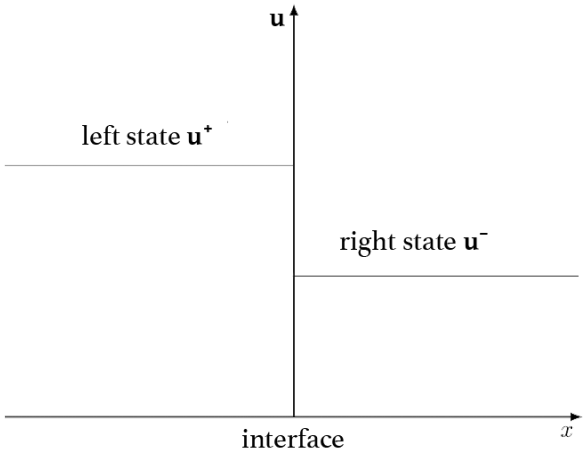


Figure 2.1: An example Riemann problem in one space dimension with the jump $[\![u]\!] = u^+ - u^-$, where u^- is the right interface state of u and u^+ is the left interface state of u.

Cauchy problem for a one-dimensional system of hyperbolic conservation laws

$$\frac{\partial}{\partial t}u + \frac{\partial}{\partial x}f(u) = 0, \tag{2.16}$$

and the piecewise constant initial condition

$$u_0(x) = \begin{cases} u^L, & x \leq 0 \\ u^R, & x > 0. \end{cases} \tag{2.17}$$

A system with m eigenvalues, each corresponding to a characteristic, results in m waves spreading out from the starting point. Depending on the characteristics these waves can be classified as a shock wave, a contact wave and rarefraction wave [17]. For more details we refer to the literature, specifically [17] and [11] and to Sod's shock tube problem, see Figure 2.2. The goal is to

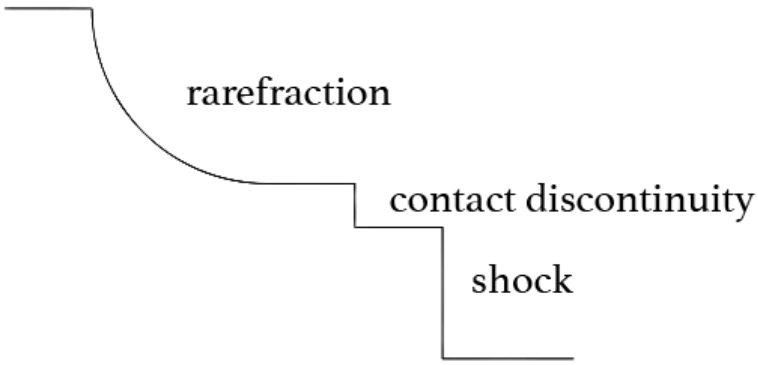


Figure 2.2: Different wave formations appearing in Sod's shock tube test case.

find a solution of this Riemann problem by finding an intermediate state u_M which is connected to u_L and u_R by a discontinuity satisfying the Rankine-Hugoniot condition. This solution can be found numerically with a so called *Riemann solver*. Exact Riemann solvers are available in the literature for many hyperbolic systems [23], but they are computationally expensive. An alternative is to use approximate Riemann solvers. Although approximative Riemann solvers can be very dissipative they usually yield a robust scheme with good numerical results and are therefore a common choice.

An easy and well-known example for an approximative Riemann solver is the so called *Rusanov* [24] or *local Lax-Friedrichs* [25] flux. The idea is to take

a rudimentary central flux and add a viscous term dependent on the maximum wave speed $\lambda_{\max} = \max\{\max_p |\lambda_p(u_L)|, \max_p |\lambda_p(u_R)|\}$ of the system of equations

$$f^* = \frac{1}{2}(f_L + f_R) - \frac{\lambda_{\max}}{2}(u_R - u_L). \tag{2.18}$$

This choice of Riemann solver neglects most of the knowledge about all the special types of waves in the Riemann problem by simply taking the fastest wave speed. As a result, the LLF is a very robust numerical scheme regardless of the dissipative character that this choice brings with it. By studying the waves of the Riemann problem more accurately, it is possible to derive less dissipative and therefore more accurate Riemann solvers, which, however, have an increased complexity and computational costs. These solvers are sometimes fitted for a specific use case and have to be choosen more carefully. Further well-known solvers are for example the Roe [26] or Harten-Lax-van-Leer (HLL) [27] Riemann solver.

2.2 Euler Equations

To get a better idea of the Equations used in this thesis, we start with the underlying Euler equations which are a set of non-linear hyperbolic conservation laws and form the basis for our considered equations. They represent a special case and simplification of the well-known Navier-Stokes equations by neglecting viscosity and thermal conductivity. Although considering zero viscosity is helpful in a mathematical sense and leads to hyperbolic characteristics, it confronts us with problems at shocks due to the lack of dissipative properties. The Euler equations are used for fluid mechanics of gaseous flows and are based on the fundamental principles of mass, momentum and energy conservation.
Therefore, the Euler equations in 2D can be stated as

$$\frac{\partial}{\partial t}\underbrace{\begin{pmatrix}\rho\\ \rho v_1\\ \rho v_2\\ \rho e^t\end{pmatrix}}_{u} + \frac{\partial}{\partial x_1}\underbrace{\begin{pmatrix}\rho v_1\\ \rho v_1^2 + p\\ \rho v_1 v_2\\ (\rho e^t + p)v_1\end{pmatrix}}_{f(u)} + \frac{\partial}{\partial x_2}\underbrace{\begin{pmatrix}\rho v_2\\ \rho v_1 v_2\\ \rho v_2^2 + p\\ (\rho e^t + p)v_2\end{pmatrix}}_{g(u)} = \begin{pmatrix}0\\0\\0\\0\end{pmatrix} \tag{2.19}$$

with the vector of conserved quantities u, the flux vector $f(u)$ in x_1-direction and the flux vector $g(u)$ in x_2-direction. Here, ρ is the density of the gas, v_1, v_2

are the velocities in x- and y-direction, $\rho v_1, \rho v_2$ the momentum in x_1- and x_2-direction and e^t denotes the specific total energy. By choosing the ideal gas law as equation of state, the pressure p is given by

$$p = \rho RT = (\gamma - 1)(\rho e^t - \frac{1}{2}\rho(v_1^2 + v_2^2)) \tag{2.20}$$

where R is the gas constant, T is the temperature, and $\gamma = \frac{c_p}{c_v}$ is the total heat capacity ratio with c_p denoting the specific heats at constant pressure and c_v denoting the specific heats at constant volume.

Due to its crucial role in this thesis, we also want to emphasize the mathematical entropy

$$U = -\frac{\rho s}{\gamma - 1}, \tag{2.21}$$

with the physical specific entropy

$$s = log(p) - \gamma log(\rho), \tag{2.22}$$

and the entropy flux

$$F = -\frac{\rho s}{\gamma - 1} v, \tag{2.23}$$

with $v = (v_1, v_2)$. Smooth solutions fullfil the entropy-conservation property,

$$\frac{\partial}{\partial t} U + \frac{\partial}{\partial x_1} F_1 + \frac{\partial}{\partial x_2} F_2 = 0, \tag{2.24}$$

which is usually not guaranteed for solutions featuring shocks. In this thesis we try to improve our numerical solutions by using the entropy inequality

$$\frac{\partial}{\partial t} U + \frac{\partial}{\partial x_1} F_1 + \frac{\partial}{\partial x_2} F_2 \leq 0, \tag{2.25}$$

as an additional criterion for our weak solutions which we then call entropy stable solution. The entropy variables are denoted by

$$w = U'(u) = \left(\frac{\gamma - s}{\gamma - 1} - \frac{\rho v^2}{2p}, \frac{\rho v_1}{p}, \frac{\rho v_2}{p}, -\frac{\rho}{p} \right)^T. \tag{2.26}$$

2.3 Ideal Magnetohydrodynamics Equations

Now that we got to know the Euler equations, we are able to extend these to the ideal MHD equations by coupling the effect of magnetic fields to the model. By introducing an additional *generalized Lagrangian multiplier* (GLM) [28, 29] part into the equations, we obtain an improved version called *ideal GLM-MHD* equations, which is able to deal with the so called *divergence-free* condition by transporting divergence of the magnetic field away. In other words, this means that the magnetic field which is initially assumed to be divergence-free does not remain divergence-free during the simulation since numerical errors occur which are then transported away [30]. To obtain physical correct solutions when the divergence-free condition is not fulfilled by the numerical scheme, it is needed to add non-conservative terms to the equations since they are an essential part of the system [31] and needed to fulfill condition (2.8).

Thus, the equations are simply an extension of the Euler equations with the flux vector $f(u)^{\text{Euler}}$, $g(u)^{\text{Euler}}$ by an MHD flux vector part $f(u)^{\text{MHD}}$, $g(u)^{\text{MHD}}$ and an GLM flux vector part $f(u)^{\text{GLM}}$, $g(u)^{\text{GLM}}$ as well as the non-conservative terms $\Gamma_{x_1}^{\text{Powell}}$, $\Gamma_{x_2}^{\text{Powell}}$, $\Gamma_{x_1}^{\text{GLM}}$, $\Gamma_{x_2}^{\text{GLM}}$ with the vector of quantities u. Here, we added v_3 as an additional velocity in x_3-direction with $\vec{v} = (v_1, v_2, v_3)$, $\vec{B} = (B_1, B_2, B_3)^T$ denotes the magnetic field vector, c_h denotes the hyperbolic divergence cleaning speed and ψ the so called *divergence-correcting* field. Since we work with $\mu_0 = 1$ in this thesis, we dropped the permeability factor of the medium in the equations stated below. We point out here, however, that these equations are now no longer conservation equations.

$$
\begin{aligned}
&\frac{\partial}{\partial t}\underbrace{\begin{pmatrix}\rho\\ \rho v_1\\ \rho v_2\\ \rho v_3\\ \rho e\\ B_1\\ B_2\\ B_3\\ \Psi\end{pmatrix}}_{u}
+\frac{\partial}{\partial x_1}\underbrace{\begin{pmatrix}\rho v_1\\ \rho v_1^2+p\\ \rho v_1 v_2\\ \rho v_1 v_3\\ (\frac{1}{2}\rho||\vec{v}||^2+\frac{\gamma p}{\gamma-1})v_1\\ 0\\ 0\\ 0\\ 0\end{pmatrix}}_{f(u)^{\text{Euler}}}
+\underbrace{\begin{pmatrix}0\\ \frac{1}{2}||\vec{B}||^2-B_1^2\\ -B_1B_2\\ -B_1B_3\\ v_1||\vec{B}||^2-B_1(\vec{v}\cdot\vec{B})\\ 0\\ v_1B_2-v_2B_1\\ v_1B_3-v_3B_1\\ 0\end{pmatrix}}_{f(u)^{\text{MHD}}}
+\underbrace{\begin{pmatrix}0\\ 0\\ 0\\ 0\\ c_h\psi B_1\\ c_h\psi\\ 0\\ 0\\ c_hB_1\end{pmatrix}}_{f(u)^{\text{GLM}}}\\
&+\frac{\partial}{\partial x_2}\underbrace{\begin{pmatrix}\rho v_2\\ \rho v_1 v_2\\ \rho v_2^2+p\\ \rho v_2 v_3\\ (\frac{1}{2}\rho||\vec{v}||^2+\frac{\gamma p}{\gamma-1})v_2\\ 0\\ 0\\ 0\\ 0\end{pmatrix}}_{g(u)^{\text{Euler}}}
+\underbrace{\begin{pmatrix}0\\ -B_1B_2\\ \frac{1}{2}||\vec{B}||^2-B_2^2\\ -B_2B_3\\ v_2||\vec{B}||^2-B_2(\vec{v}\cdot\vec{B})\\ v_2B_1-v_1B_2\\ 0\\ v_2B_3-v_3B_2\\ 0\end{pmatrix}}_{g(u)^{\text{MHD}}}
+\underbrace{\begin{pmatrix}0\\ 0\\ 0\\ 0\\ c_h\psi B_2\\ c_h\psi\\ 0\\ 0\\ c_hB_2\end{pmatrix}}_{g(u)^{\text{GLM}}}\\
&+\underbrace{\frac{\partial B_1}{\partial x_1}\begin{pmatrix}0\\ B_1\\ B_2\\ B_3\\ \vec{v}\cdot\vec{B}\\ v_1\\ v_2\\ v_3\\ 0\end{pmatrix}}_{\Gamma_{x_1}^{\text{Powell}}}
+\underbrace{\frac{\partial\psi}{\partial x_1}\begin{pmatrix}0\\ 0\\ 0\\ 0\\ v_1\psi\\ 0\\ 0\\ 0\\ v_1\end{pmatrix}}_{\Gamma_{x_1}^{\text{GLM}}}
+\underbrace{\frac{\partial B_2}{\partial x_2}\begin{pmatrix}0\\ B_1\\ B_2\\ B_3\\ \vec{v}\cdot\vec{B}\\ v_1\\ v_2\\ v_3\\ 0\end{pmatrix}}_{\Gamma_{x_2}^{\text{Powell}}}
+\underbrace{\frac{\partial\psi}{\partial x_2}\begin{pmatrix}0\\ 0\\ 0\\ 0\\ v_2\psi\\ 0\\ 0\\ 0\\ v_2\end{pmatrix}}_{\Gamma_{y}^{\text{GLM}}}
=\begin{pmatrix}0\\0\\0\\0\\0\\0\\0\\0\\0\end{pmatrix}.
\end{aligned}
\tag{2.27}
$$

By choosing the calorically perfect gas assumption, we can close the system with pressure given by

$$
p=(\gamma-1)\left(\rho e^t-\frac{1}{2}(\rho||\vec{v}||^2+||\vec{B}||^2+\psi^2)\right). \tag{2.28}
$$

Again we are also interested into the entropy pair $(U, F(U))$ of the underlying equations which leads to the same mathematical and physical entropy as well as the same entropy flux as in the Euler equations case. The entropy variables read

$$w = U'(u) = \left(\frac{\gamma - s}{\gamma - 1} - \frac{\rho ||\vec{v}||^2}{2p}, \frac{\rho v_1}{p}, \frac{\rho v_2}{p}, \frac{\rho v_3}{p}, -\frac{\rho}{p}, \frac{\rho B_1}{p}, \frac{\rho B_2}{p}, \frac{\rho B_3}{p}, \frac{\rho \psi}{p} \right)^T. \tag{2.29}$$

2.4 Multi-Component Euler Equations

In contrast to the single-component Euler equations which observe a single chemical species, like hydrogen, the multi-component Euler equations are able to describe multiple different chemical species, like hydrogen mixed with oxygen. As we will see in a further section, chemical species are also able to interact through chemical reaction networks allowing to simulate various complex natural phenomena and technical applications like combustion and detonation processes. In the following, we want to show how the single-component Euler equations can be extended into the multi-component case and which special properties have to be taken into account for our applications.

The multi-component Euler equations used in this thesis can be stated as

$$\frac{\partial}{\partial t} \underbrace{\begin{pmatrix} \rho_1 \\ \vdots \\ \rho_N \\ \rho v_1 \\ \rho v_2 \\ \rho e^t \end{pmatrix}}_{u} + \frac{\partial}{\partial x_1} \underbrace{\begin{pmatrix} \rho_1 v_1 \\ \vdots \\ \rho_N v_1 \\ \rho v_1^2 + p \\ \rho v_1 v_2 \\ (\rho e^t + p) v_1 \end{pmatrix}}_{f(u)} + \frac{\partial}{\partial x_2} \underbrace{\begin{pmatrix} \rho_1 v_2 \\ \vdots \\ \rho_N v_2 \\ \rho v_1 v_2 \\ \rho v_2^2 + p \\ (\rho e^t + p) v_2 \end{pmatrix}}_{g(u)} = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \tag{2.30}$$

whereas the total mass continuity equation is replaced by N continuity equations for each of the species with ρ_k being the partial density of species k and $\rho = \sum_{k=1}^{ns} \rho_k$ being the total density of the mixture of all species. Using the ideal gas law we can close the system stating the pressure p as

$$p := \sum_{k=1}^{N} \rho_k \tilde{R}_k T, \quad \tilde{R}_k = \frac{R_{\text{inf}}}{m_k}, \tag{2.31}$$

with $\tilde{R}_k$ being the specific gas constant and m_k being the molar mass of species k. The temperature T can be determined using a calorically perfect gas assumption from the following equation

$$\rho e = \sum_{k}^{N} \rho_k e_k, \quad e_k := c_{vk} T, \tag{2.32}$$

which holds for the compressible Euler case with $\rho e := \rho e^t - (\rho v)^2/2\rho$ being the internal energy. Following the Mayer's relation we get

$$R = c_p - c_v, \quad \tilde{R}_k = c_{pk} - c_{vk}, \tag{2.33}$$

which leads to

$$\gamma = \frac{c_p}{c_v}, \quad \gamma_k = \frac{c_{pk}}{c_{vk}}. \tag{2.34}$$

Therefore, we can compute

$$\gamma = \frac{\sum_{k=1}^{N} \rho_k c_{vk} \gamma_k}{\sum_{k=1}^{N} \rho_k c_{vk}} \tag{2.35}$$

so that we can calculate the pressure as

$$p = (\gamma - 1)(\rho e^t - \frac{1}{2}\rho ||v||^2). \tag{2.36}$$

In the case of multi-component Euler, we can state the mathematical entropy as

$$U = -\rho s := -\sum_{k=1}^{N} \rho_k s_k \tag{2.37}$$

with the specific entropy of species k as

$$s_k := c_{vk} \log(T) - \tilde{R}_k \log(\rho_k) \tag{2.38}$$

and the entropy flux

$$F = -\rho s v \tag{2.39}$$

with s being the specific entropy of the mixture. We obtain the following entropy variables [7]

$$w = U'(u) = \frac{1}{T}\begin{pmatrix} g_1 - \frac{1}{2}v_1^2 & \dots & g_N - \frac{1}{2}v_1^2 & v_1 & -1 \end{pmatrix}^T, \tag{2.40}$$

with

$$g_k := h_k - Ts_k, \quad h_k := e_k + \tilde{R}_k T, \quad 1 \le k \le N, \tag{2.41}$$

being the Gibbs function and specific enthalpy of species k.

To reduce confusion, let us briefly mention here that there are several equivalent forms with the same weak solution of the multi-component equations model. The one we use in this thesis is usually referred to as the *symmetric* formulation, where the total density continuity equation is replaced by N species continuity equations. Another often used form is the so called *unsymmetric* formulation where we use the single-component Euler equations and add $N-1$ species continuity equations as follows

$$\frac{\partial}{\partial t}\underbrace{\begin{pmatrix} \rho \\ \rho v_1 \\ \rho v_2 \\ \rho e^t \\ \rho Y_1 \\ \vdots \\ \rho Y_{N-1} \end{pmatrix}}_{u} + \frac{\partial}{\partial x_1}\underbrace{\begin{pmatrix} \rho v \\ \rho v_1^2 + p \\ \rho v_1 v_2 \\ (\rho e^t + p)v_1 \\ \rho Y_1 v_1 \\ \vdots \\ \rho Y_{N-1} v_1 \end{pmatrix}}_{f(u)} + \frac{\partial}{\partial x_2}\underbrace{\begin{pmatrix} \rho v_2 \\ \rho v_1 v_2 \\ \rho v_2^2 + p \\ (\rho e^t + p)v_2 \\ \rho Y_1 v_2 \\ \vdots \\ \rho Y_{N-1} v_2 \end{pmatrix}}_{g(u)} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}, \tag{2.42}$$

where Y_k denotes the mass fraction of species k and the remaining mass fraction Y_N can be calculated by $Y_N = 1 - \sum_{k=1}^{N-1} Y_k$. Other less frequently encountered formulations are the *Gamma* formulation and the $\frac{1}{\gamma-1}$ formulation which can be viewed in [32].

Although the multi-component equations look similar to the single-component case, they encounter problems that do not occur in the single-component case and are characteristic to the multi-component case.

Positivity

A well known problem of multi-component equations is the mass fraction positivity issue. Physically, all mass fractions have to be nonnegative $Y_k \geq 0, k = 1, \ldots, N$ as otherwise we would get negative partial densities $\rho Y_k < 0$. Numerically this condition can not be guaranteed trivially.

A possibility to ensure positive mass fractions is to use a so called *uncoupled* approach for the numerical solution of multi-component equations. In short, with this approach we treat the single-component Euler equations and the additional species equations seperately which means that we use a classical single-component Euler scheme for the Euler part and simply advect the additional species equations seperately [33]. This approach is simple to implement and provides positive mass fractions, but leads to poor solutions since it is lacking in a theoretical point of view [34]. This is, in fact, a reason we are deriving an entropy-conservative flux for the so called *coupled* approach of the ideal multi-component GLM-MHD equations seen in the next chapter. Briefly, it means that we are working with the system of equations in (2.42) as a whole instead of treating the species equations seperately. The interested reader is referred to Larrouturou [34], who proposed a modification of the numerical flux, so that the positivity of mass fractions is guaranteed. The system used in this thesis satisfies the mass fraction positivity by construction.

Pressure Oscillations

An even more serious problem are the pressure oscillations developing at material fronts seperating species which are not evoked due to the high-order accuracy of the scheme, but are an intrinsic problem of the multi-component nature of the equations which can arise even with a first-order scheme [33, 35]. These pressure oscillations are in fact a huge problem compared to the mass fraction positivity since they can not be circumvented so easily. So far, it is known to be solvable by introducing a non-conservative scheme which is contrary to the most important property of our desired scheme [32]. Furthermore, even though it is possible to control the conservation errors to a decent degree, it is not able to handle strong shocks which is another main task of our required scheme in this thesis. We therefore refrain from presenting this solution idea in more detail at this point and refer to Abgrall [32] for the so called

quasi conservative approach. However, this means for us that we are not able to tackle the pressure oscillation problem so easily and have to hope that our other mechanisms introduced in this thesis are sufficient to prevent/minimize this kind of problem in our simulations.

2.5 Multi-Component Ideal Magnetohydrodynamics Equations

Now that we have seen how to extend the single-component Euler equations to the multi-component case, we are able to extend the ideal GLM-MHD equations analogue to the multi-component case. Therefore, the ideal GLM-MHD equations written in a more compact form compared to (2.27) can be stated as

$$\frac{\partial}{\partial t}\underbrace{\begin{pmatrix} \rho_1 \\ \vdots \\ \rho_N \\ \rho v_1 \\ \rho v_2 \\ \rho v_3 \\ \rho e^t \\ B_1 \\ B_2 \\ B_3 \\ \psi \end{pmatrix}}_{u} + \frac{\partial}{\partial x_1}\underbrace{\begin{pmatrix} \rho_1 v_1 \\ \vdots \\ \rho_N v_1 \\ \rho v_1^2 + p + \frac{1}{2}||\vec{B}||^2 - B_1^2 \\ \rho v_1 v_2 - B_1 B_2 \\ \rho v_1 v_3 - B_1 B_3 \\ (\frac{1}{2}\rho||\vec{v}||^2 + \frac{\gamma p}{\gamma - 1})v_1 + v_1||\vec{B}||^2 - B_1(\vec{v}\cdot\vec{B}) + c_h\psi B_1 \\ c_h\psi \\ v_1 B_2 - v_2 B_1 \\ v_1 B_3 - v_3 B_1 \\ c_h B_1 \end{pmatrix}}_{f(u)}$$

$$
+\frac{\partial}{\partial x_2}\underbrace{\begin{pmatrix}
\rho_1 v_2 \\
\vdots \\
\rho_N v_2 \\
\rho v_1 v_2 - B_1 B_2 \\
\rho v_2^2 + p + \frac{1}{2}||\vec{B}||^2 - B_2^2 \\
\rho v_2 v_3 - B_2 B_3 \\
(\frac{1}{2}\rho||\vec{v}||^2 + \frac{\gamma p}{\gamma - 1})v_2 + v_2||\vec{B}||^2 - B_2(\vec{v}\cdot\vec{B}) + c_h\psi B_2 \\
v_2 B_1 - v_1 B_2 + c_h\psi \\
0 \\
v_2 B_3 - v_3 B_2 \\
c_h B_2
\end{pmatrix}}_{g(u)}
$$

$$
+\underbrace{\frac{\partial B_1}{\partial x_1}\begin{pmatrix}
0 \\ \vdots \\ 0 \\ B_1 \\ B_2 \\ B_3 \\ \vec{v}\cdot\vec{B} \\ v_1 \\ v_2 \\ v_3 \\ 0
\end{pmatrix}}_{\Gamma_{x_1}^{\text{Powell}}}
+\underbrace{\frac{\partial \psi}{\partial x_1}\begin{pmatrix}
0 \\ \vdots \\ 0 \\ 0 \\ 0 \\ 0 \\ v_1\psi \\ 0 \\ 0 \\ 0 \\ v_1
\end{pmatrix}}_{\Gamma_{x_1}^{\text{GLM}}}
+\underbrace{\frac{\partial B_2}{\partial x_2}\begin{pmatrix}
0 \\ \vdots \\ 0 \\ B_1 \\ B_2 \\ B_3 \\ \vec{v}\cdot\vec{B} \\ v_1 \\ v_2 \\ v_3 \\ 0
\end{pmatrix}}_{\Gamma_{x_2}^{\text{Powell}}}
+\underbrace{\frac{\partial \psi}{\partial x_2}\begin{pmatrix}
0 \\ \vdots \\ 0 \\ 0 \\ 0 \\ 0 \\ v_2\psi \\ 0 \\ 0 \\ 0 \\ v_2
\end{pmatrix}}_{\Gamma_{x_2}^{\text{GLM}}}
=\begin{pmatrix}
0 \\ \vdots \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0
\end{pmatrix}
\tag{2.43}
$$

with $f(u)$ and $g(u)$ being the flux vectors in x_1- and x_2-direction incorporated with the GLM flux vector part. Similar to the multi-component Euler case, we can calculcate the pressure as

$$
p = (\gamma - 1)\left(\rho e^t - \frac{1}{2}(\rho||\vec{v}||^2 + ||\vec{B}||^2 + \psi^2)\right) \tag{2.44}
$$

with

$$
\gamma = \frac{\sum_{k=1}^{N}\rho_k c_{vk}\gamma_k}{\sum_{k=1}^{N}\rho_k c_{vk}}. \tag{2.45}
$$

The entropy pair is the same as the one from the multi-component Euler case,

$$(U, F(U)) = (-\rho s, -\rho s v), \tag{2.46}$$

which leads to the entropy variables

$$w = U'(u) = \frac{1}{T}\left(g_1 - \frac{1}{2}v_1^2, \ldots, g_N - \frac{1}{2}v_1^2, v_1, v_2, v_3, -1, B_1, B_2, B_3, \psi\right)^T. \tag{2.47}$$

The entropy variables, together with the entropy flux potential,

$$\mathcal{F} = (\sum_{k=1}^{N} r_k \rho_k v_1) + \frac{1}{2T} u ||\vec{B}^2|| + \frac{1}{T} c_h B_1 \psi, \tag{2.48}$$

will be fundamental for our entropy-conservative flux derivation in the next chapter.

2.6 Chemical Networks

The extension of the Euler and ideal GLM-MHD equations to the multi-component case opens up many new possibilities for simulation applications. One great new opportunity is to simulate chemical reactions, which have many areas of application like combustion processes, hypersonic reacting flows, or pre-mixed detonations. Chemical reactions cause the generation and/or destruction of chemical species under the restraint of mass conservation. Depending on the partial densities and temperature, a conversion of species takes place. In the following, we will show how we are able to extend the multi-component equations stated in the previous sections. We will give an example based on the multi-component Euler equations leading to the multi-component reactive Euler equations. The extension of the ideal GLM-MHD equations is done analogously.

Chemical reactions represent a source term which has to be added to our system of equations as follows

$$\frac{\partial}{\partial t} u + \frac{\partial}{\partial x_1} f(u) + \frac{\partial}{\partial x_2} g(u) = S(u), \tag{2.49}$$

with a source term S containing the production rate of each species k. The multi-component reactive Euler equations can then be written as

$$\frac{\partial}{\partial t}\underbrace{\begin{pmatrix}\rho_1\\ \vdots\\ \rho_N\\ \rho v_1\\ \rho v_2\\ \rho e\end{pmatrix}}_{u}+\frac{\partial}{\partial x_1}\underbrace{\begin{pmatrix}\rho_1 v_1\\ \vdots\\ \rho_N v_1\\ \rho v_1^2+p\\ \rho v_1 v_2\\ (\rho e^t+p)v_1\end{pmatrix}}_{f(u)}+\frac{\partial}{\partial x_2}\underbrace{\begin{pmatrix}\rho_1 v_2\\ \vdots\\ \rho_{ns} v_2\\ \rho v_1 v_2\\ \rho v_2^2+p\\ (\rho e^t+p)v_2\end{pmatrix}}_{g(u)}=\underbrace{\begin{pmatrix}\dot{\omega}_1\\ \vdots\\ \dot{\omega}_N\\ 0\\ 0\\ 0\end{pmatrix}}_{S(u)} \tag{2.50}$$

with

$$\dot{\omega}_k = W_k \sum_{l=1}^{nr}(\tau_{k,l}^b - \tau_{k,l}^f)\left(K_l^f \prod_{k=1}^{N}\left(\frac{\rho_k}{W_k}\right)^{\tau_{k,l}^f} - K_l^b \prod_{k=1}^{N}\left(\frac{\rho_k}{W_k}\right)^{\tau_{k,l}^b}\right) \tag{2.51}$$

where ω_k is the production rate, W_k the molecular weight of each species k. K_l^f and K_l^b are the forward and backward reaction rates of each chemical reaction $l = 1, \ldots, nr$ where nr is the total number of reactions. Furthermore, $\tau_{k,l}^f$ and $\tau_{k,l}^b$ are the stoichiometric coefficients of species k as integer numbers of the elementary reaction mechanism consisting of N species and nr reactions

$$\sum_{k=1}^{N}\tau_{k,l}^f \chi_k \Leftrightarrow \sum_{k=1}^{N}\tau_{k,l}^b \chi_k, \quad l = 1, \ldots, nr \tag{2.52}$$

with χ_k being the chemical formulation of species k. The pressure p is given by

$$p = (\gamma - 1)\left(\rho e^t - \frac{1}{2}(\rho||v||^2) - \sum_{k=1}^{N} q_k \rho_k\right) \tag{2.53}$$

where q_k is the chemical heat released during a chemical reaction. Chemical reaction equations are usually pretty stiff depending on the reaction rates K. One of the two important formulations of reaction rates used in this thesis is the *Arrhenius* form

$$K = \mathcal{A}T^{\mathcal{B}} \exp\left(\frac{-T_{ign}}{T}\right) \tag{2.54}$$

with $\mathcal{A}$ being some empirical coefficient, $\mathcal{B}$ being some empirical exponent and T_{ign} being the empirical activation temperature of the chemical reaction. A very complete collection of reaction rates and coefficients can be found in the CHEMKIN databases [36]. A tougher formulation leading to more stiffness is the Heaviside form which is used in very temperature sensitive settings with usually high activation energies

$$K = \begin{cases} Da & T \geq T_{ign} \\ 0 & T < T_{ign} \end{cases} \tag{2.55}$$

with Da being the Damkohler number. The latter formulation works like a step function and activates and deactivates the chemical reactions with full power depending on the actual temperature in contrast to the Arrhenius form which distributes the chemical reaction rate across a temperature range.

3 Mathematical Model - Numerical Scheme

3.1 Introduction

In the previous chapter, we have familiarized ourselves with the governing equations we want to solve. Now, we want to develop a numerical scheme which allows us to solve these equations for our applications in the best way possible. Since the perfect method does not exist, we have to make compromises and try to combine the best of all worlds in the hope to create a robust and well working numerical scheme for our kind of applications. Hence, we will introduce two different numerical schemes, list their advantages and disadvantages, and explain how we combine those properly.

The following chapter is organized as follows. First, we will introduce a very robust and easy to implement numerical method called *Finite Volume* (FV) method, which is well-known in the world of fluid dynamics and will act as a safety net for our numerical scheme. Here, we go into the mathematical modeling of the FV method and discuss the major advantages (robustness) and disadvantages in relation to our applications. Second, we will introduce our numerical goal scheme called *discontinuous Galerkin spectral element method* (DGSEM) and again provide the mathematical formulation as well as the advantages and disadvantages of the scheme. Since robustness is an important endeavour for us, we will also address the importance of entropy stability for our schemes and derive suitable entropy-conservative fluxes for our governing equations. Following this, we will provide a convex blending scheme which connects our high-order DG solver with our safety net first-order FV solver to the so called *DGFV hybrid* method. In this regard, we will explain why the DGFV hybrid method is needed in the presence of shocks and how it is able to produce robust solutions. Although the shock-capturing blending scheme does a good job in producing robust solutions, sometimes it might be necessary to

use an additional positivity-preserving scheme. In this work we will introduce a positivity-preserving scheme specially developed for our shock-capturing blending scheme which is able to rescue our solution a posteriori. Finally, we will discuss the mathematical model of chemical networks and time integration. We want to mention briefly, that we use the method of lines with which we can separate the temporal discretization from the spatial discretization. Therefore, we will first deal with the spatial discretization and only at the end with the temporal discretization.

3.2 Baseline: Finite Volume Method

In this section, we will show how to discretize the spatial derivative of our equations with the FV method. Although we are working with hyperbolic systems of conservation laws, we will derive the FV method for scalar conservation laws in 1D which works analogously.

For simplicity, let us say we want to construct the FV method for the non-linear conservation laws in one space dimension

$$\frac{\partial u}{\partial t} + \frac{\partial f(u)}{\partial x} = 0. \tag{3.1}$$

First, we need to discretize the domain $\Omega = [x_L, x_R]$ into $(N_q + 1)$ equidistant non-overlapping so called *control volumes* (or *finite volumes*)

$$Q_i = [x_{i-\frac{1}{2}}, x_{i+\frac{1}{2}}], \quad \cup_{i=0}^{N_q} Q_i = \Omega \tag{3.2}$$

bounded by the cell boundaries $x_{i-\frac{1}{2}}, x_{i+\frac{1}{2}}$ associated with the point of coordinate x_i, defined as

$$x_i = x_L + (i + \frac{1}{2})\Delta x, \quad i = 0, \ldots, N_q. \tag{3.3}$$

Hereby, we define the control volume size $\Delta x = \frac{x_R - x_L}{N_q + 1}$ with the cell borders

$$x_{i-\frac{1}{2}} = x_i - \frac{\Delta x}{2} = x_L + i\Delta x, \quad i = 0, \ldots, N_q + 1 \tag{3.4}$$

$$x_{i+\frac{1}{2}} = x_i + \frac{\Delta x}{2} = x_L + (i+1)\Delta x, \quad i = 0, \ldots, N_q + 1. \tag{3.5}$$

In contrast to finite difference (FD) methods (which we will not discuss further in this thesis and refer to [37]), we use the integral form of the conservation law as a starting point to derive the FV method. This means, that instead of approximating point values like the FD methods do, we will trace the cell mean value as an integral of our quantity u over each one of these finite volumes

$$u_i^n := \frac{1}{\Delta x}\int_{Q_i} u(x,t^n)dx \equiv \frac{1}{\Delta x}\int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} u(x,t^n)dx, \tag{3.6}$$

for at some discrete time $t^n := n\Delta t, n = 0,1,\ldots,$ with time step Δt. Now we can integrate our conservation law (3.1) by integrating over the spatial sub-domain Q_i

$$\frac{\partial}{\partial t}\int_{Q_i} u(x,t)dx + f\left(u\left(x_{i+\frac{1}{2}},t\right)\right) - f(u(x_{i-\frac{1}{2}},t)) = 0, \tag{3.7}$$

which gives us the integral form of the conservation law. Since we want to calculate an approximation for the solution of the cell averages u_i^{n+1} at some time t^{n+1} in the future, we can also integrate over a discrete time sub-domain $[t^n, t^{n+1}]$

$$\int_{Q_i} u(x,t^{n+1})dx - \int_{Q_i} u(x,t^n)dx = \int_{t^n}^{t^{n+1}} f(u(x_{i-\frac{1}{2}},t))dt - \int_{t^n}^{t^{n+1}} f(u(x_{i+\frac{1}{2}},t))dt, \tag{3.8}$$

with a time step $\Delta t = t^{n+1} - t^n$ in the size of the discrete time sub-domain. Since we are working with cell mean values, we rearrange and divide by the spatial domain size Δx

$$\frac{1}{\Delta x}\int_{Q_i} u(x,t^{n+1})dx = \tag{3.9}$$

$$\frac{1}{\Delta x}\int_{Q_i} u(x,t^n)dx - \frac{1}{\Delta x}\left[\int_{t^n}^{t^{n+1}} f(u(x_{i-\frac{1}{2}}))dt - \int_{t^n}^{t^{n+1}} f(u(x_{i+\frac{1}{2}}))\right]. \tag{3.10}$$

Then, we define our average surface fluxes as follows

$$\overline{f}^n_{i+\frac{1}{2}} := \frac{1}{\Delta t}\int_{t^n}^{t^{n+1}} f(u(x_{i+\frac{1}{2}}))dt,$$
$$\overline{f}^n_{i-\frac{1}{2}} := \frac{1}{\Delta t}\int_{t^n}^{t^{n+1}} f(u(x_{i-\frac{1}{2}}))dt, \tag{3.11}$$

and can therefore insert (3.11) as well as (3.6) into (3.9) leading to

$$u_j^{n+1} = u_j^n - \frac{\Delta t}{\Delta x}(\overline{f}^n_{i+\frac{1}{2}} - \overline{f}^n_{i-\frac{1}{2}}). \tag{3.12}$$

To get a fully discrete method, we now only have to find an approximation of (3.11) based on the values u^n at time t^n. As we are working with hyperbolic conservation laws, we know that information propagates as a wave with finite speed. Therefore, we only need to evaluate the values u_i^n, u_{i+1}^n on both sides of the surface $x_{i+\frac{1}{2}}$, respectively u_{i-1}^n, u_i^n for $x_{i-\frac{1}{2}}$, as long as we adjust the time step size Δt appropriately to the fastest wave speeds in the hyperbolic system. This means that we use can use some numerical flux function

$$\overline{f}^n_{i+\frac{1}{2}} \approx f^{*n}_{i+\frac{1}{2}} = f^*(u_i^n, u_{i+1}^n), \tag{3.13}$$
$$\overline{f}^n_{i-\frac{1}{2}} \approx f^* n_{i-\frac{1}{2}} = f^*(u_{i-1}^n, u_i^n), \tag{3.14}$$

as an approximation for the average surface flux which leads to the fully discrete FV method

$$u_j^{n+1} = u_j^n - \frac{\Delta t}{\Delta x}\left(f^*(u_i^n, u_{i+1}^n) - f^*(u_{i-1}^n, u_i^n)\right). \tag{3.15}$$

As we can see, the FV method has many obvious advantages. First of all, it is a very simple method to understand and to implement. Due to its derivation on the basis of the integral form which is closer to the physics, it is able to handle strong shocks when using appropriate numerical flux functions, which is the main property we rely on in our shock capturing blending scheme. Since we derived a conservative discretization using control volumes with conservative numerical fluxes between these control volumes, we obtain a fully conservative method if one disregards the boundary conditions.

One of the main disadvantages of the FV method is the representation of the values as a piecewise constant polynomial (cell mean value) resulting in a low first-order accuracy. There are possibilities to construct higher order FV methods by reconstructions like for example piecewise linear polynomials leading to second order accuracy. Another idea to construct more robust high-order schemes is to examine multiple candidate stencils like the essentially non-oscillatory (ENO) as well as weighted essentially non-oscillatory (WENO) methods [38, 39]. Such stencil based high-order FV methods do have the downside to become computationally expensive [40]. This is where our next method can show its full strength since it can adjust its accuracy arbitrarily high with ease.

3.3 High-Order: Discontinuous Galerkin Spectral Element Method

Now, that we have seen how to build a FV method which will be acting as a safety net due to its shock robustness capabilities, we will present a high-order discontinuous Galerkin spectral element method (DGSEM) which will function as the method of choice in the absence of shocks due to its accuracy advantage.

3.3.1 DGSEM in 1D

Just like in the derivation of the FV method, we will construct the DGSEM method for the conservation laws in one space dimension. Similar to the FV method, we divide the domain $\Omega = [x_L, x_R]$ into non-overlapping elements

$$Q_i = \left[x_i - \frac{\Delta x_i}{2}, x_i + \frac{\Delta x_i}{2} \right], \tag{3.16}$$

in the physical domain with center x_i. To save computing operations, we transform every element Q_i onto the same reference space $E = [-1, 1]$ of size 2

$$Q_i \underset{\xi(x)}{\overset{x(\xi)}{\rightleftharpoons}} E, \tag{3.17}$$

by the linear coordinate mapping

$$x(\xi) = x_i + \frac{\Delta x_i}{2}\xi, \quad \xi \in [-1, 1]. \tag{3.18}$$

Since the Jacobian of this transformation is given by

$$\mathcal{J} = \frac{\partial x}{\partial \xi} = \frac{\Delta x_i}{2}, \tag{3.19}$$

we can transform the whole conservation law into reference space

$$\mathcal{J}\frac{\partial u}{\partial t} + \frac{\partial f(u)}{\partial \xi} = 0, \tag{3.20}$$

with the flux function

$$\frac{\partial f(u)}{\partial x} = \frac{\partial f(u)}{\partial \xi}\frac{\partial \xi}{\partial x} = \mathcal{J}^{-1}\frac{\partial f(u)}{\partial \xi}, \tag{3.21}$$

and the inverse Jacobian $\mathcal{J}^{-1} = \frac{\partial \xi}{\partial x}$.

Let us now come to the heart of the DG method which is the variation formulation. Since our conservation law contains a derivative onto the flux function, it means that our flux function has to be differentiable. Due to the fact that this is usually not the case, especially for non-linear conservation laws, we have to use a trick to overcome this issue. A clever idea is to move the derivative from the flux function onto some other smooth function. Therefore, we multiply our conservation law with an infinitely differentiable smooth testfunction $\phi(\xi)$ and integrate the equation over the reference element E

$$\int_E \mathcal{J}\frac{\partial u}{\partial t}\phi d\xi + \int_E \frac{\partial f(u)}{\partial \xi}\phi d\xi = 0. \tag{3.22}$$

Now we can use integration by parts to move the derivative from the flux function onto the smooth testfunction

$$\int_E \mathcal{J}\frac{\partial u}{\partial t}\phi d\xi + [f\phi]_{-1}^{1} - \int_E f\frac{\partial \phi}{\partial \xi}d\xi = 0, \tag{3.23}$$

leading to the so called *weak formulation* which works as the basis for the DG method.

Similar to the FV method, we allow the solution to be discontinuous across element interfaces. Therefore, we have a Riemann problem on the interface which will be solved approximately due to computational efficiency. Typical

choices in this thesis are the *local Lax-Friedrichs* (LLF) [25] or *Harten-Lax-Leer* (HLL) [27] numerical flux. This means we can replace the surface flux by our approximative Riemann solver f^*

$$\int_E \mathcal{J}\frac{\partial u}{\partial t}\phi d\xi + [f^*\phi]_{-1}^1 - \int_E f\frac{\partial \phi}{\partial \xi}d\xi = 0. \tag{3.24}$$

By using integration by parts once again on the so called *volume integral*, we are able to create a *strong formulation*

$$\int_E \mathcal{J}\frac{\partial u}{\partial t}\phi d\xi + [(f^* - f)\phi]_{-1}^1 + \int_E \frac{\partial f}{\partial \xi}\phi d\xi = 0. \tag{3.25}$$

as a basis for the DG method which is equivalent to the weak formulation due to the so called *summation-by-parts* (SBP) property for the *Legendre-Gauss-Lobatto* (LGL) operator with collocation which is able to mimic integration by parts in discrete. For more information about SBP operators we refer to [41].

In contrast to the FV method, which used first-order accurate cell mean values inside the finite volumes, we now use a polynomial of degree N_p inside each element, allowing us to switch between different orders of accuracy by adjusting the polynomial degree N_p. Fortunately, we are working in the reference space and are able to do this so called *polynomial Ansatz* with the reference coordinate ξ, allowing us to build just one Ansatz for the reference space instead of building one Ansatz for each physical element. We use a nodal interpolation of the solution,

$$u(x(\xi), t)|_{Q_i} \approx u^{Q_i}(\xi, t) = \sum_{j=0}^{N_p} u_j^{Q_i}(t) l_j(\xi), \tag{3.26}$$

where we use the Lagrange basis functions

$$l_j(\xi) = \prod_{i=0, i\neq j}^{N_p} \frac{\xi - \xi_i}{\xi_j - \xi_i}, \quad j = 0, \ldots, N_p, \tag{3.27}$$

which satisfy the Kronecker's property $l_j(\xi_i) = \delta_{ij}, i, j = 0, \ldots, N_p$, meaning that

$$\delta_{ij} = \begin{cases} 1, i = j \\ 0, i \neq j. \end{cases} \tag{3.28}$$

This way we are able to use collocated quadrature rules since we use a nodal polynomial representation based on LGL points $\{\xi_i\}_{i=0}^{N_p}$, which includes the boundaries $x = -1$ and $x = 1$, in the reference element. Now our polynomial coefficients are also directly our values at the grid points, leading to

$$u^{Q_i}(\xi_i, t) = \sum_{j=0}^{N_p} u_j^{Q_i}(t) l_j(\xi_i) = u_i^{Q_i}(t). \tag{3.29}$$

We use the same Ansatz for the flux function $f(u)$ which leads to

$$f(u) \approx \sum_{j=0}^{N_p} f_j(t) l_j(\xi). \tag{3.30}$$

A problem arises due to the fact that we try to approximate our flux function $f(u)$ with a polynomial of degree N_p although the flux function is not necessarily linear in u which means that the flux function might have a much higher polynomial degree. A good example is the Burgers equation with $f(u) = \frac{u^2}{2}$ which leads to a flux polynomial of degree $2N_p$. To be efficient we use collocation of the interpolation to be able to calculate the coefficients directly,

$$f_j(t) = f(u(\xi_j, t)) \tag{3.31}$$

Unfortunately, this approach has aliasing errors [42].

Now we can derive the DGSEM by discretizing the spatial part of the transformed conservation law

$$\underbrace{\frac{\Delta x_i}{2} \int_{-1}^{1} \frac{\partial u}{\partial t} l_j(\xi) d\xi}_{I} + \underbrace{\int_{-1}^{1} \frac{\partial f(u)}{\partial \xi} l_j(\xi) d\xi}_{II} = 0. \tag{3.32}$$

We begin to discretize the first summand I with our polynomial approach for the solution u

$$\frac{\Delta x_i}{2} \int_{-1}^{1} \frac{\partial u}{\partial t} l_j(\xi) d\xi = \frac{\Delta x_i}{2} \int_{-1}^{1} \left(\sum_{l=0}^{N_p} \frac{\partial u}{\partial t}(t) l_j(\xi) \right) l_i(\xi) d\xi, \tag{3.33}$$

and evaluate the integrals numerically with an LGL quadrature

$$\frac{\Delta x_i}{2}\int_{-1}^{1}\left(\sum_{l=0}^{N_p}\frac{\partial u}{\partial t}(t)l_l(\xi)\right)l_j(\xi)d\xi \approx \frac{\Delta x_i}{2}\sum_{k=0}^{N_p}\left(\sum_{l=0}^{N_p}\frac{\partial u}{\partial t}(t)l_l(\xi_k)\right)l_j(\xi_k)w_k. \tag{3.34}$$

Property (3.28) leads to

$$\frac{\Delta x_i}{2}\frac{\partial u_j(t)}{\partial t}w_j, \quad j=0,...,N_p, \tag{3.35}$$

due to the Kronecker property. This result can also be written in a compact matrix-vector notation

$$\frac{\Delta x_i}{2}\int_{-1}^{1}\frac{\partial u}{\partial t}l_j(\xi)d\xi \approx \frac{\Delta x_i}{2}\underline{\underline{M}}\,\underline{\dot{u}}(t), \tag{3.36}$$

with the LGL mass matrix

$$M_{lk} = \langle l_l, l_k\rangle_{N_p} = \delta_{lk}w_k, \quad l,k=0,\ldots,N_p. \tag{3.37}$$

Here we use the double underlined notation to outline a matrix and a single underlined notation to outline a vector. As we already showed above, we apply integration by parts on the second summand II

$$\int_{-1}^{1}\frac{\partial f(u)}{\partial \xi}l_j(\xi)d\xi = [f(u)l_j(\xi)]_{-1}^{1} - \int_{-1}^{1}f(u)\frac{\partial l_j(\xi)}{\partial \xi}d\xi, \tag{3.38}$$

and use an approximative Riemann solver on the surface flux

$$[f(u)l_j(\xi)]_{-1}^{1} - \int_{-1}^{1}f(u)\frac{\partial l_j(\xi)}{\partial \xi}d\xi \approx \left[f^*(u^+,u^-)l_j(\xi)\right]_{-1}^{1} - \int_{-1}^{1}f(u)\frac{\partial l_j(\xi)}{\partial \xi}d\xi. \tag{3.39}$$

Since the integral is of order $2N_p-1$ and our LGL-SBP quadrature is exact for polynomials of degree $2N_p-1$, we can integrate the volume part exactly

$$\begin{aligned}\left[f^*(u^+,u^-)l_j(\xi)\right]_{-1}^{1} - \int_{-1}^{1}f(u)\frac{\partial l_j(\xi)}{\partial \xi}d\xi \\ = \left[f^*(u^+,u^-)l_j(\xi)\right]_{-1}^{1} - \sum_{k=0}^{N_p}\left(\sum_{l=0}^{N_p}f_l(t)l_l(\xi_k)\right)\frac{l_j(\xi_k)}{\partial \xi_k}w_k,\end{aligned} \tag{3.40}$$

which can be written as

$$\left[f^*(u^+,u^-)l_j(\xi)\right]_{-1}^{1} - \sum_{k=0}^{N_p}\left(\sum_{l=0}^{N_p} f_l(t)l_l(\xi_k)\right)\frac{l_j(\xi_k)}{\partial\xi_k}w_k$$
$$= \left[f^*(u^+,u^-)l_j(\xi)\right]_{-1}^{1} - \sum_{k=0}^{N_p} f_k(t)D_{kj}w_k, \tag{3.41}$$

with the differentiation matrix

$$D_{kj} = \frac{\partial l_j(\xi_k)}{\partial\xi_k}, \quad j,k = 0,\ldots,N_p. \tag{3.42}$$

Again, we are able to write our result in a compact matrix-vector notation

$$\left[f^*(u^+,u^-)l_j(\xi)\right]_{-1}^{1} - \sum_{k=0}^{N_p} f_k(t)D_{kj}w_k = \underline{\underline{B}}\,\underline{f}^* - \underline{\underline{D}}^T\,\underline{\underline{M}}\,\underline{f}, \tag{3.43}$$

with

$$\underline{\underline{B}} = \mathrm{diag}(-1,0,0,\ldots,0,0,1). \tag{3.44}$$

All in one we obtain the weak formulation in matrix-vector notation

$$\frac{\Delta x_l}{2}\underline{\underline{M}}\,\dot{\underline{u}}(t) + \underline{\underline{B}}\,\underline{f}^* - \underline{\underline{D}}^T\,\underline{\underline{M}}\,\underline{f} = 0, \tag{3.45}$$

and since we use the LGL-Operator with the SBP property

$$(\underline{\underline{M}}\,\underline{\underline{D}}) + (\underline{\underline{M}}\,\underline{\underline{D}})^T = \underline{\underline{B}}, \tag{3.46}$$

we are able to reformulate the weak formulation into an equivalent strong formulation

$$\frac{\Delta x_l}{2}\underline{\underline{M}}\,\dot{\underline{u}}(t) + \underline{\underline{M}}\,\underline{\underline{D}}\,\underline{f} = -\underline{\underline{B}}\left[\underline{f}^* - \underline{f}\right]. \tag{3.47}$$

The strong form can be rearranged so that we can discretize the conservation equation directly in space with the DGSEM.

$$\frac{\Delta x_l}{2}\dot{\underline{U}} + \underline{\underline{D}}\,\underline{f} = -\underline{\underline{M}}^{-1}\underline{\underline{B}}\left[\underline{f}^* - \underline{f}\right], \tag{3.48}$$

which shows that the DGSEM for non-linear conservation laws can be constructed independently for arbirary flux functions. Again, to get a full space discretization, we have to choose a proper numerical flux for the surface term like the local Lax-Friedrichs flux.

3.3.2 DGSEM in 2D and Curvilinear Elements

For the sake of completeness, we will additionally derive the DGSEM in 2D with curvilinear elements since we will also be working with two dimensional applications using different kind of geometries in this thesis. The interested reader is referred to the book of Kopriva [43], which we used as orientation for this subsection.

A way to do this is to create a mapping between the physical space and the reference space similar to what we already did for the 1D case, but this time we want to do a mapping of square elements into a curvilinear space. This way it is possible to use the tensor-product ansatz to extend the DGSEM easily to two dimensions by applying the 1D method in each direction.

We start with our conservation law in 2D

$$\frac{\partial}{\partial t}u + \frac{\partial}{\partial x_1}f + \frac{\partial}{\partial x_2}g = 0, \tag{3.49}$$

which we want to transform with a curvilinear mapping. Herefore, it is useful to use the chain rule for the spatial derivatives resulting in

$$\frac{\partial}{\partial x_1}f = \frac{\partial}{\partial \xi}u\frac{\partial}{\partial x_1}\xi + \frac{\partial}{\partial \eta}u\frac{\partial}{\partial x_1}\eta, \tag{3.50}$$

and

$$\frac{\partial}{\partial x_2}g = \frac{\partial}{\partial \xi}u\frac{\partial}{\partial x_2}\xi + \frac{\partial}{\partial \eta}u\frac{\partial}{\partial x_2}\eta, \tag{3.51}$$

what can also be presented as

$$\nabla_x f = \begin{pmatrix} f_{x_1} \\ f_{x_2} \end{pmatrix} = \begin{pmatrix} \xi_{x_1} & \eta_{x_1} \\ \xi_{x_2} & \eta_{x_2} \end{pmatrix} \begin{pmatrix} f_\xi \\ f_\eta \end{pmatrix}, \tag{3.52}$$

or vice versa

$$\nabla_\xi f = \begin{pmatrix} f_\xi \\ f_\eta \end{pmatrix} = \begin{pmatrix} x_{1\xi} & x_{2\xi} \\ x_{1\eta} & x_{2\eta} \end{pmatrix} \begin{pmatrix} f_{x_1} \\ f_{x_2} \end{pmatrix}. \tag{3.53}$$

Usually, we only know the one direction of the mapping, namely from the reference square to the physical space $\vec{x}(\xi, \eta)$ which means that we need to derive the

so called *metric terms* $\xi_{x_1}, \xi_{x_2}, \eta_{x_1}, \eta_{x_2}$ from the inverse of the transformation, it follows

$$\begin{pmatrix} \xi_{x_1} & \eta_{x_1} \\ \xi_{x_2} & \eta_{x_2} \end{pmatrix} = \begin{pmatrix} x_{1\xi} & x_{2\xi} \\ x_{1\eta} & x_{2\eta} \end{pmatrix}^{-1} = \frac{1}{\mathcal{J}} \begin{pmatrix} x_{2\eta} & -x_{2\xi} \\ -x_{1\eta} & x_{1\xi} \end{pmatrix}, \tag{3.54}$$

with the Jacobian $\mathcal{J} := x_{1\xi}x_{2\eta} - x_{1\eta}x_{2\xi}$. Now we are able to transform the underlying equations (3.49)

$$\mathcal{J}\frac{\partial}{\partial t}u + \frac{\partial}{\partial \xi}\tilde{f} + \frac{\partial}{\partial \eta}\tilde{g} = 0, \tag{3.55}$$

with

$$\tilde{f} := f(x_{2\eta}) + g(-x_{1\eta}) \qquad \tilde{g} := f(-x_{2\xi}) + (x_{1\xi}), \tag{3.56}$$

and respectively

$$\begin{aligned} \tilde{f}_\xi &= f_\xi(x_{2\eta}) + f(x_{2\eta\xi}) + g_\xi(-x_{1\eta}) + g(-x_{1\eta\xi}) \\ \tilde{g}_\eta &= f_\eta(-x_{2\xi}) + f(-x_{2\xi\eta}) + g_\eta(x_{1\xi}) + g(x_{1\xi\eta}). \end{aligned}$$

As can be seen, the transformed equation looks similar to the transformed 1D equation in the subsection above which means that the derivation of our DGSEM in 2D is straight forward.

Again, we divide the domain $\Omega = [x_{1L}, x_{1R}] \times [x_{2L}, x_{2R}]$ into non-overlapping elements $Q_{i,j}$ for which we have to determine the mapping $\vec{x}^{Q_{i,j}}(\xi, \eta)$. The metric terms determined by this mapping can then be used for the transformation of each element into the reference element $E = [-1, 1]^2$. In this reference space we are again able to use our polynomial Ansatz

$$u(\vec{x}, t)|_{Q_{i,j}} \approx u^{Q_{i,j}}(\xi, \eta, t) = \sum_{k,l=0}^{N_p} u_{k,l}^{Q_{i,j}}(t) l_k(\xi) l_l(\eta). \tag{3.57}$$

For presentation reasons we drop the superscript $Q_{i,j}$ in the following.

Now we are again ready to apply the variation formulation of the conservation law

$$\iint_E (Ju_t + \tilde{f}_\xi + \tilde{g}_\eta) l_i(\xi) l_j(\eta) d\xi d\eta = 0, \quad i, j = 0, ..., N_p. \tag{3.58}$$

Below we will take a closer look at the three parts of the equation (3.58). We start with the first term containing the time derivative where we insert our polynomial Ansatz function as follows

$$\iint_E \mathcal{J}(\xi,\eta) u_t l_i(\xi) l_j(\eta) d\xi d\eta \approx \iint_E \mathcal{J}(\xi,\eta) \left[\sum_{k,l}^{N_p} \dot{u}_{kl}(t) l_k(\xi) l_l(\eta) \right] l_i(\xi) l_j(\eta) d\xi d\eta. \tag{3.59}$$

Again, we are using the inexact but efficient LGL-quadrature for the integrals leading to

$$\begin{aligned} \iint_E \mathcal{J} u_t l_i(\xi) l_j(\eta) d\xi d\eta \approx & \sum_{n,m=0}^{N_p} w_n w_m \left[\sum_{k,l=0}^{N_p} \dot{u}_{kl} l_k(\xi_n) l_l(\eta_m) \right] \\ & l_i(\xi_n) l_j(\eta_m) \mathcal{J}(\xi_n,\eta_m) \\ = & \mathcal{J}(\xi_i,\eta_j) w_i w_j \dot{u}_{ij}. \end{aligned} \tag{3.60}$$

Now we take the second term containing the spatial derivative in η direction

$$\iint_E \tilde{f}_\xi l_i(\xi) l_j(\eta) d\xi d\eta = \int_{-1}^{1} l_j(\eta) \left[\int_{-1}^{1} \tilde{f}_\xi l_i(\xi) d\xi \right] d\eta \tag{3.61}$$

$$\approx \sum_{m=0}^{N_p} w_m \underbrace{l_j(\eta_m)}_{\delta_{mj}} \left[\int_{-1}^{1} \tilde{f}_\xi(\xi,\eta_m) l_i(\xi) d\xi \right] d\eta \tag{3.62}$$

$$= w_j \left[\int_{-1}^{1} \tilde{f}_\xi(\xi,\eta_j) l_i(\xi) d\xi \right], \tag{3.63}$$

where the last equation is similar to the 1D case apart from the extra integration weight. Analogous to the 1D case, we receive

$$\iint_E \tilde{f}_\xi l_i(\xi) l_j(\eta) d\xi d\eta \approx w_j \left[\tilde{f}^*(1,\eta_j) l_i(1) - \tilde{f}^*(-1,\eta_j) l_j(-1) - \sum_{n=0}^{N_p} w_n \tilde{f}_{nj} D_{ni} \right]. \tag{3.64}$$

It also follows the tensor product Ansatz for the flux function using a polyno-

mial of degree N_p on the LGL points

$$\tilde{f}(u(\xi,\eta,t)) \approx \tilde{f}(\xi,\eta,t) = \sum_{i,j=0}^{N_p} \underbrace{\tilde{f}(u_{ij}(t))}_{\tilde{f}_{ij}} l_i(\xi) l_j(\eta). \tag{3.65}$$

We do the same for the second partial derivative

$$\iint_E \tilde{g}_\eta l_i(\xi) l_j(\eta) d\xi d\eta \approx w_i \left[\tilde{g}^*(\xi_i, 1) l_j(1) - \tilde{g}^*(\xi_i, -1) l_j(-1) - \sum_{m=0}^{N_p} w_m \tilde{g}_{im} D_{mj} \right], \tag{3.66}$$

leading to the 2D DGSEM

$$\dot{u}_{ij} = -\frac{1}{J_{ij}} \left[\tilde{f}^*(1,\eta_j) \frac{l_i(1)}{w_i} - \tilde{f}^*(-1,\eta_j) \frac{l_i(-1)}{w_i} - \sum_{n=0}^{N_p} \frac{w_n}{w_i} D_{ni} \tilde{f}_{nj} \right] \tag{3.67}$$

$$- \frac{1}{J_{ij}} \left[\tilde{g}^*(\xi_i, 1) \frac{l_j(1)}{w_j} - \tilde{g}^*(\xi_i, -1) \frac{l_j(-1)}{w_j} - \sum_{n=0}^{N_p} \frac{w_n}{w_j} D_{nj} \tilde{g}_{ni} \right]. \tag{3.68}$$

The strong form matrix-vector notation

$$\underline{\underline{\mathcal{J}}}\,\underline{\underline{\dot{u}}} + \underline{\underline{D}}\,\underline{\underline{\tilde{f}}} + \underline{\underline{\tilde{g}}}\,\underline{\underline{D}}^T = \underline{\underline{S}} \left[\underline{\underline{\tilde{f}}}^* - \underline{\underline{\tilde{f}}} \right] + \left[\underline{\underline{\tilde{g}}}^* - \underline{\underline{\tilde{g}}} \right] \underline{\underline{S}}. \tag{3.69}$$

Let us now turn to the advantages and disadvantages of the DGSEM. As we have seen, the DGSEM derivation is somewhat more complex than the FV derivation, whereas the generic discretization of non-linear conservation equations is not much more difficult in the end. Furthermore, we are able to calculate high-order accurate numerical simulations with low dispersion and dissipation errors which allows us to resolve very fine structures of a solution. The high-order capability also allows for more efficient calculations, since it is able to produce evenly accurate results as, for example, the FV method but with less degrees of freedom [1, 44]. Another advantage of the DGSEM is the easy parallelization as well as the possible use of curvilinear meshes. Although the standard DGSEM as derived above has great advantages, like for example

higher-order accuracy compared to the FV method before, it has also disadvantages like stability issues for non-linear systems of conservation laws as well as oscillation problems in areas where shocks are present which may lead to negative densities or pressure. These and other issues will be adressed in the following sections.

3.4 Entropy Stability

As we already know, our goal is to build a numerical scheme suitable for conservation laws, which means we are trying to reproduce physical characteristics of conservation laws in a discrete sense. The more physical characteristics can be reproduced in a discrete sense by a numerical scheme, the better. Since we are mostly working with conservation laws, an exception here are the GLM-MHD equations, it is clear that our numerical scheme has to be able to conserve certain quantities in a discrete sense, like for example mass conservation, which is also conserved in a physical sense. This property is also necessary to produce weak solutions, which are an infinite class of solutions to which our numerical scheme converges. However, as we saw, mass conservation is not enough to identify a unique and physical solution. A way to identify these solutions is to reproduce the second law of thermodynamics, which states that our numerical scheme has to satisfy the entropy-stability property in a discrete sense. In this thesis, we are able to clearly show the benefits of the entropy stability property in practice (see chapter 5 and 6) which leads to increased robustnes for non-linear and/or underresolved simulations.

Since the DGSEM is a relatively new scheme, it has to deal with problems other schemes already have findings for. A good example is the FV community, which is already advanced in a theoretical point of view. Therefore, it would be beneficial to be able to rewrite parts of the DGSEM to be compatible with the FV method. Due to the SBP property of our derived DGSEM which can therefore be written in strong form, we are able to rewrite the volume integrals of the DGSEM to be compatible with the FV method

$$\underline{\underline{D}}\,\underline{f} = \sum_{m=0}^{N} D_{im} f_m^l = \frac{\overline{f}_{i+1}^l - \overline{f}_i^l}{w_i}, \tag{3.70}$$

using the finite volume type flux differencing form [45, 46]. Now it is possible

to construct an entropy-conserving high-order DGSEM, when we compute the flux difference as

$$\frac{\overline{f}^l_{i+1} - \overline{f}^l_i}{w_i} \approx 2 \sum_{m=0}^{N_p} D_{im} f^{\#,l}_{EC}(u_i, u_m) \tag{3.71}$$

with an entropy-conserving two-point numerical flux function leading to a entropy-conserving FV scheme and D being part of diagonal norm SBP operator [47–49].

We would like to briefly mention here that this approximation is due to the findings of Fisher and Carpenter [47] and it is possible to derive the standard DG method by simple averages. Thereupon, it was extended to construct entropy-stable schemes by Fisher and Carpenter [45] as well as kinetic energy preserving schemes by Gassner et al. [50] and pressure equilibrium preserving schemes by Shima et al. [51]. Another scheme with such properties was also proposed by Ranocha [52].

In the following subsections we will derive an entropy conserving flux for the multi-component Euler and the multi-component Ideal GLM-MHD equations.

3.4.1 Entropy Conservative Flux for Multi-Component Euler Equations

To derive an entropy-conserving flux for the multi-component Euler equations, we follow Gouasmi et al. [7] where we need to satisfy the entropy-conservation condition by Tadmor [53]

$$[\![w]\!] \cdot f^{\#} = [\![\mathcal{F}]\!], \tag{3.72}$$

where w denotes the entropy variables derived in chapter 2,

$$f^{\#} = [f^{\#}_{1,1}, f^{\#}_{1,2}, \ldots, f^{\#}_{1,N}, f^{\#}_2, f^{\#}_3]^T, \tag{3.73}$$

denotes the entropy-conserving interface flux for the N-component Euler equations and $\mathcal{F} = \sum_{k=1}^{N} \tilde{R}_k \rho_k u$ denotes the entropy potential function. We define the notation for the jump operator, arithmetic, and logarithmic means between

two states at an interface, a_L and a_R, as

$$[\![a]\!]_{(L,R)} := a_R - a_L, \tag{3.74}$$

$$\overline{a}_{(L,R)} := \frac{1}{2}(a_L + a_R), \tag{3.75}$$

$$a^{\ln}_{(L,R)} := [\![a]\!]_{(L,R)} / [\![\ln(a)]\!]_{(L,R)}. \tag{3.76}$$

The logarithmic mean can be evaluated by the procedure given in [54]. Following Roe's technique [26], we will rewrite the jump terms as a linear combination of jumps in algebraic variables

$$z = [z_{1,1}, z_{1,2}, \ldots, z_{1,N}, z_2, z_3] = \left[\rho_1, \rho_2, \ldots, \rho_N, v_1, \frac{1}{T}\right]. \tag{3.77}$$

It follows for the potential jump function

$$[\![\mathcal{F}]\!] = \sum_{k=1}^{N} \tilde{R}_k [\![\rho_k v_1]\!] = \left(\sum_{k=1}^{N} \tilde{R}_k \overline{z_{1,k}}\right) [\![z_2]\!] + \sum_{k=1}^{N} \tilde{R}_k \overline{z_2} [\![z_{1,k}]\!]. \tag{3.78}$$

For the entropy variables, we start with the first N components

$$\frac{g_k}{T} - \frac{u^2}{2T} = \frac{e_{0k}}{T} + c_{vk} + \tilde{R}_k - c_{vk}\ln(T) + \tilde{R}_k \ln(\rho_k) - \frac{u^2}{2T} \tag{3.79}$$

$$= e_{0k} z_3 + c_{vk} + \tilde{R}_k + c_{vk}\ln(z_3) + \tilde{R}_k \ln(z_{1,k}) - \frac{1}{2} z_2^2 z_3, \tag{3.80}$$

which leads to

$$\left[\!\!\left[\frac{g_k}{T} - \frac{u^2}{2T}\right]\!\!\right] = [\![z_{1,k}]\!] \frac{\tilde{R}_k}{z^{\ln}_{1,k}} - [\![z_2]\!] \overline{z_2 z_3} + [\![z_3]\!] \left(e_{0k} + \frac{c_{vk}}{z^{\ln}_3} - \frac{1}{2}\overline{z_2^2}\right). \tag{3.81}$$

For the remaining entropy variables, we obtain

$$\left[\!\!\left[\frac{u}{T}\right]\!\!\right] = \overline{z_3}[\![z_2]\!] + \overline{z_2}[\![z_3]\!], \tag{3.82}$$

and

$$-\left[\!\!\left[\frac{1}{T}\right]\!\!\right] = -[\![z_3]\!]. \tag{3.83}$$

Now we are able to insert our results into the entropy-conservation condition (3.72)

$$\sum_{k=1}^{N} z_{1,k}\left(\frac{\tilde{R}_k}{z_{1,k}^{\ln}} f_{1,k}^{\#}\right) + [\![z_3]\!]\left((-\overline{z_3 z_2})\sum_{k=1}^{N} f_{1,k}^{\#} + \overline{z_3} f_2^{\#}\right) \tag{3.84}$$

$$+ [\![z_3]\!]\left(\sum_{k=1}^{N}(e_{0k} + c_{vk}\frac{1}{z_3^{\ln}} - \frac{1}{2}\overline{z_2^2}) f_{1,k}^{\#} + \overline{z_2} f_2^{\#} - f_3^{\#}\right) \tag{3.85}$$

$$= \left(\sum_{k=1}^{N} \tilde{R}_k \overline{z_{1,k}}\right)[\![z_2]\!] + \sum_{k=1}^{N} \tilde{R}_k \overline{z_2}[\![z_{1,k}]\!]. \tag{3.86}$$

Now we can formulate this scalar condition into a system of $N+3$ equations

$$\frac{\tilde{R}_k}{z_{1,k}^{\ln}} f_{1,k}^{\#} = \tilde{R}_k \overline{z_2}, \quad 1 \le k \le N, \tag{3.87}$$

$$(-\overline{z_3 z_2})\sum_{k=1}^{N} f_{1,k}^{\#} + \overline{z_3} f_2^{\#} = \left(\tilde{R}_k \overline{z_{1,k}}\right), \tag{3.88}$$

$$\sum_{k=1}^{N}\left(e_{0k} + c_{vk}\frac{1}{z_3^{\ln}} - \frac{1}{2}\overline{z_2^2}\right) f_{1,k}^{\#} + \overline{z_2} f_2^{\#} - f_3^{\#} = 0. \tag{3.89}$$

Therefore, the entropy conservative flux for the multi-component Euler equations is

$$f_{1,k}^{\#} = \rho_k^{\ln}\overline{v_1} \tag{3.90}$$

$$f_2^{\#} = \frac{1}{\overline{1/T}}\left(\sum_{k=1}^{N} \tilde{R}_k \overline{\rho_k}\right) + \overline{v_1}\sum_{k=1}^{N} f_{1,k}^{\#} \tag{3.91}$$

$$f_3^{\#} = \sum_{k=1}^{N}\left(e_{0k} + c_{vk}\frac{1}{(1/T)^{\ln}} - \frac{1}{2}\overline{v_1^2}\right) f_{1,k}^{\#} + \overline{v_1} f_2^{\#}. \tag{3.92}$$

If we use this numerical flux in the surface term as well as the volume term of the DGSEM, we get an entropy-conservative DGSEM which is only feasable for

smooth simulations. For more complex simulations with shocks a dissipative numerical flux is needed in the surface leading to an entropy-stable DGSEM which is known to be more robust and provides more physical solutions than the standard DGSEM derived the section before. We also show these advantageous properties in practice in chapter 5 and 6.

3.4.2 Entropy-Conservative Flux for Multi-Component Ideal GLM-MHD Equations

Similar to the entropy-conserving flux for the multi-component Euler equations, we are now able to introduce an entropy-conserving flux for the multi-component ideal GLM-MHD equations. Again, we have to satisfy condition (3.72) where w denotes the entropy variables (2.47) for the multi-component ideal GLM-MHD equations. Furthermore, we need the entropy flux potential (2.48) to calculate the N-component ideal GLM-MHD entropy-conservative interface flux

$$f^{\#} = \left[f^{\#}_{1,1}, f^{\#}_{1,2}, \dots, f^{\#}_{1,N}, f^{\#}_{2}, f^{\#}_{3}, f^{\#}_{4}, f^{\#}_{5}, f^{\#}_{6}, f^{\#}_{7}, f^{\#}_{8}, f^{\#}_{9}\right]^T. \tag{3.93}$$

We will use the algebraic variables

$$z = [z_{1,1}, z_{1,2}, \dots, z_{1,N}, z_2, z_3, z_4, z_5, z_6, z_7, z_8, z_9]^T \tag{3.94}$$

$$= \left[\rho_1, \rho_2, \dots, \rho_N, v_1, v_2, v_3, \frac{1}{T}, B_1, B_2, B_3, \psi\right]^T. \tag{3.95}$$

First, we rewrite our entropy variables into an algebraic variable formulation leading to

$$\frac{1}{T}\left(g_k - \frac{1}{2}||\vec{v}||^2\right) = e_{0k}z_5 + c_{vk} + \tilde{R}_k + c_{vk}\ln z_5 + \tilde{R}_k \ln z_{1,k} - \frac{1}{2}z_5(z_2^2 + z_3^2 + z_4^2), \tag{3.96}$$

for $1 \leq k \leq N$ as well as

$$\frac{v_1}{T} = z_2 z_5, \quad \frac{v_2}{T} = z_3 z_5, \quad \frac{v_3}{T} = z_4 z_5 \tag{3.97}$$

$$\frac{-1}{T} = -z_5, \quad \frac{B_1}{T} = z_5 z_6, \quad \frac{B_2}{T} = z_5 z_7 \tag{3.98}$$

$$\frac{B_3}{T} = z_5 z_8, \quad \frac{\psi}{T} = z_5 z_9. \tag{3.99}$$

Now, we have to examine the jump of the, in algebraic variables formulated, entropy variables, leading to

$$\left[\!\left[\frac{1}{T}(g_k - \frac{1}{2}||\vec{u}||^2)\right]\!\right] = [\![z_{1,k}]\!]\frac{\tilde{R}_k}{z_{1,k}^{\ln}} - [\![z_2]\!]\overline{z_2}\,\overline{z_5} - [\![z_3]\!]\overline{z_3}\,\overline{z_5} - [\![z_4]\!]\overline{z_4}\,\overline{z_5} + [\![z_5]\!]\left(e_{0k} + \frac{c_{vk}}{z_5^{\ln}} - \frac{1}{2}(\overline{z_2^2} + \overline{z_3^2} + \overline{z_4^2})\right), \tag{3.100}$$

for $1 \leq k \leq N$ and

$$\left[\!\left[\frac{v_1}{T}\right]\!\right] = [\![z_5]\!]\overline{z_2} + [\![z_2]\!]\overline{z_5}, \quad \left[\!\left[\frac{v_2}{T}\right]\!\right] = [\![z_5]\!]\overline{z_3} + [\![z_3]\!]\overline{z_5} \tag{3.101}$$

$$\left[\!\left[\frac{v_3}{T}\right]\!\right] = [\![z_5]\!]\overline{z_3} + [\![z_3]\!]\overline{z_5}, \quad \left[\!\left[\frac{-1}{T}\right]\!\right] = -[\![z_5]\!] \tag{3.102}$$

$$\left[\!\left[\frac{B_1}{T}\right]\!\right] = [\![z_5]\!]\overline{z_6} + [\![z_6]\!]\overline{z_5}, \quad \left[\!\left[\frac{B_2}{T}\right]\!\right] = [\![z_5]\!]\overline{z_7} + [\![z_7]\!]\overline{z_5} \tag{3.103}$$

$$\left[\!\left[\frac{B_3}{T}\right]\!\right] = [\![z_5]\!]\overline{z_8} + [\![z_8]\!]\overline{z_5}, \quad \left[\!\left[\frac{\psi}{T}\right]\!\right] = [\![z_5]\!]\overline{z_9} + [\![z_9]\!]\overline{z_5}. \tag{3.104}$$

Additionally, we need to examine the jumps of the entropy flux potential

$$\begin{aligned}[\![\mathcal{F}]\!] = {} & [\![z_2]\!]\left(\sum_{k=1}^{N}\tilde{R}_k\overline{z_{1,k}}\right) + \sum_{k=1}^{N}\tilde{R}_k\overline{z_2}[\![z_{1,k}]\!] + [\![z_9]\!]c_h\overline{z_5}\,\overline{z_6} + [\![z_6]\!]c_h\overline{z_5}\,\overline{z_9} \\ & + [\![z_5]\!]c_h\overline{z_6 z_9} + [\![z_5]\!]\frac{1}{2}(\overline{z_2 z_6^2} + \overline{z_2 z_7^2} + \overline{z_2 z_8^2}) + [\![z_2]\!]\frac{1}{2}(\overline{z_5}\,\overline{z_6^2} + \overline{z_5}\,\overline{z_7^2} + \overline{z_5}\,\overline{z_8^2}) \\ & + [\![z_6]\!](\overline{z_2}\,\overline{z_5}\,\overline{z_6}) + [\![z_7]\!](\overline{z_2}\,\overline{z_5}\,\overline{z_7}) + [\![z_8]\!](\overline{z_2}\,\overline{z_5}\,\overline{z_8}) - \overline{z_6}([\![z_5]\!](\overline{z_2 z_6} + \overline{z_3 z_7} + \overline{z_4 z_8}) \\ & + [\![z_2]\!]\overline{z_5}\,\overline{z_6} + [\![z_3]\!]\overline{z_5}\,\overline{z_7} + [\![z_4]\!]\overline{z_5}\,\overline{z_8} + [\![z_6]\!]\overline{z_5}\,\overline{z_2} + [\![z_7]\!]\overline{z_5}\,\overline{z_3} + [\![z_8]\!]\overline{z_5}\,\overline{z_4}).\end{aligned} \tag{3.105}$$

Following the entropy conservation conditions (3.72) we obtain the following system of N+8 equations

$$f_{1,k}^{\#}\tilde{R}_k\frac{1}{z_{1,k}^{\ln}}=\tilde{R}_k\overline{z_2},\quad 1\le k\le N, \tag{3.106}$$

$$f_2^{\#}\overline{z_5}-(\overline{z_2}\,\overline{z_5})\sum_{k=1}^{N}f_{1,k}^{\#}=\sum_{k=1}^{N}\tilde{R}_k\overline{z_{1,k}^{\ln}}+\frac{1}{2}\overline{z_5}(\overline{z_6^2}+\overline{z_7^2}+\overline{z_8^2})-\overline{z_6}\,\overline{z_6}\,\overline{z_5}, \tag{3.107}$$

$$f_3^{\#}\overline{z_5}-\overline{z_3}\,\overline{z_5}\sum_{k=1}^{N}f_{1,k}^{\#}=-\overline{z_6}\,\overline{z_7}\,\overline{z_5}, \tag{3.108}$$

$$f_4^{\#}\overline{z_5}-\overline{z_4}\,\overline{z_5}\sum_{k=1}^{N}f_{1,k}^{\#}=-\overline{z_6}\,\overline{z_8}\,\overline{z_5}, \tag{3.109}$$

$$\sum_{k=1}^{N}\left(e_{0k}+\frac{c_{vk}}{z_5^{\ln}}-\frac{1}{2}(\overline{z_2^2}+\overline{z_3^2}+\overline{z_4^2})\right)f_{1,k}^{\#}-f_5^{\#}+\overline{z_2}f_2^{\#}+\overline{z_3}f_3^{\#}+\overline{z_4}f_4^{\#} \tag{3.110}$$
$$+\overline{z_6}f_6^{\#}+\overline{z_7}f_7^{\#}+\overline{z_8}f_8^{\#}+\overline{z_9}f_9^{\#}=c_h\overline{z_6z_9}+\frac{1}{2}(\overline{z_2z_6^2}+\overline{z_2z_7^2}+\overline{z_2z_8^2})$$
$$-\overline{z_6}(\overline{z_2z_6}+\overline{z_3z_7}+\overline{z_4z_8}),$$

$$f_6^{\#}\overline{z_5}=c_h\overline{z_5}\,\overline{z_9}+\overline{z_9}\,\overline{z_2}\,\overline{z_6}-\overline{z_6}\,\overline{z_5}\,\overline{z_2}, \tag{3.111}$$

$$f_7^{\#}\overline{z_5}=\overline{z_5}\,\overline{z_2}\,\overline{z_7}-\overline{z_5}\,\overline{z_6}\,\overline{z_3}, \tag{3.112}$$

$$f_8^{\#}\overline{z_5}=\overline{z_5}\,\overline{z_2}\,\overline{z_8}-\overline{z_6}\,\overline{z_5}\,\overline{z_4}, \tag{3.113}$$

$$f_9^{\#}\overline{z_5}=c_h\overline{z_5}\,\overline{z_6}. \tag{3.114}$$

By a simple transformation we get the entropy-conservative flux function in algebraic formulation

$$f_{1,k}^{\#} = \overline{z_2} z_{1,k}^{\ln}, \quad 1 \le k \le N, \tag{3.115}$$

$$f_2^{\#} = \overline{z_2} \sum_{k=1}^{N} f_{1,k}^{\#} + \sum_{k=1}^{N} \tilde{R}_k \frac{\overline{z_{1,k}}}{\overline{z_5}} + \frac{1}{2}(\overline{z_6^2} + \overline{z_7^2} + \overline{z_8^2}) - \overline{z_6}^2, \tag{3.116}$$

$$f_3^{\#} = \overline{z_3} \sum_{k=1}^{N} f_{1,k}^{\#} - \overline{z_6}\,\overline{z_7}, \tag{3.117}$$

$$f_4^{\#} = \overline{z_4} \sum_{k=1}^{N} f_{1,k}^{\#} - \overline{z_6}\,\overline{z_8}, \tag{3.118}$$

$$\begin{aligned} f_5^{\#} = {} & \sum_{k=1}^{N} (e_{0k} + \frac{c_{vk}}{z_5^{\ln}} - \frac{1}{2}(\overline{z_2^2} + \overline{z_3^2} + \overline{z_4^2})) f_{1,k}^{\#} + \overline{z_2} f_2^{\#} + \overline{z_3} f_3^{\#} + \overline{z_4} f_4^{\#} \\ & + \overline{z_6} f_6^{\#} + \overline{z_7} f_7^{\#} + \overline{z_8} f_8^{\#} + \overline{z_9} f_9^{\#} - c_h \overline{z_6 z_9} - \frac{1}{2}(\overline{z_2 z_6^2} + \overline{z_2 z_7^2} + \overline{z_2 z_8^2}) \\ & + \overline{z_6}(\overline{z_2 z_6} + \overline{z_3 z_7} + \overline{z_4 z_8}), \end{aligned} \tag{3.119}$$

$$f_6^{\#} = c_h \overline{z_9}, \tag{3.120}$$

$$f_7^{\#} = \overline{z_2}\,\overline{z_7} - \overline{z_6}\,\overline{z_3}, \tag{3.121}$$

$$f_8^{\#} = \overline{z_2}\,\overline{z_8} - \overline{z_6}\,\overline{z_4}, \tag{3.122}$$

$$f_9^{\#} = c_h \overline{z_6}. \tag{3.123}$$

Therefore, the entropy-conservative interface flux $f^{\#}$ for the multi-component ideal GLM-MHD equations is defined as

$$
\begin{aligned}
f^{\#}_{1,k} &= \overline{v_2}\rho_k^{\ln}, \quad 1 \le k \le N \\
f^{\#}_2 &= \overline{v_1}\sum_{k=1}^{N} f^{\#}_{1,k} + \frac{1}{\overline{1/T}}\left(\sum_{k=1}^{N}\tilde{R}_k\overline{\rho_k}\right) + \frac{1}{2}(\overline{B_1^2}+\overline{B_2^2}+\overline{B_3^2}) - \overline{B_1}^2, \\
f^{\#}_3 &= \overline{v_2}\sum_{k=1}^{N} f^{\#}_{1,k} - \overline{B_1}\,\overline{B_2}, \\
f^{\#}_4 &= \overline{v_3}\sum_{k=1}^{N} f^{\#}_{1,k} - \overline{B_1}\,\overline{B_3}, \\
f^{\#}_5 &= \sum_{k=1}^{N}\left(e_{0k} + \frac{c_{vk}}{(1/T)^{\ln}} - \frac{1}{2}(\overline{v_1^2}+\overline{v_2^2}+\overline{v_3^2})\right) f^{\#}_{1,k} + \overline{v_1}f^{\#}_2 + \overline{v_2}f^{\#}_3 + \overline{v_3}f^{\#}_4 \\
&\quad + \overline{B_1}f^{\#}_6 + \overline{B_2}f^{\#}_7 + \overline{B_3}f^{\#}_8 + \overline{\psi}f^{\#}_9 - c_h\overline{B_1\psi} - \frac{1}{2}(\overline{v_1B_1^2}+\overline{v_1B_2^2}+\overline{v_1B_3^2}) \\
&\quad + \overline{B_1}(\overline{v_1B_1}+\overline{v_2B_2}+\overline{v_3B_3}),
\end{aligned}
\tag{3.124}
$$

$$
\begin{aligned}
f^{\#}_6 &= c_h\overline{\psi}, \\
f^{\#}_7 &= \overline{v_1}\,\overline{B_2} - \overline{B_1}\,\overline{v_2}, \\
f^{\#}_8 &= \overline{v_1}\,\overline{B_3} - \overline{B_1}\,\overline{v_3}, \\
f^{\#}_9 &= c_h\overline{B_1}.
\end{aligned}
\tag{3.125}
$$

Using entropy-conservative interface fluxes leads to an entropy-conservative scheme with virtually no dissipation. To obtain a more stable scheme we need to add dissipation, for example by using the LF type dissipation,

$$
f^{\#}(u_L,u_R) = f^{EC}(u_L,u_R) - \frac{1}{2}\mathcal{D}\left[\!\left[u\right]\!\right]_{(L,R)}, \tag{3.126}
$$

with the *Roe-type* dissipation matrices

$$
\mathcal{D} = \mathcal{R}|\Theta|\mathcal{R}^{-1}, \tag{3.127}
$$

and $\mathcal{R}$ being the matrix of right eigenvectors and Θ being a diagonal matrix with the eigenvalues of the flux [55]. The derivation of the dissipation matrices

for the Euler and ideal GLM-MHD case is not trivial, thus we refer the reader to [7] and [31]. We want to point out that the LLF scheme can also be written as a Roe-type operator, hereby all the diagonal entries of Θ are filled with the maximum eigenvalue [55].

3.5 Shock-Capturing Blending Scheme

Even if an entropy-stable high-order DGSEM shows an improved robustness over the standard high-order DGSEM, it still continues to struggle with shocks. To overcome this issue, we combine the best of both worlds, the robustness of the first-order FV method as well as the accuracy of the high-order DGSEM. Therefore, our recipe deals with the combination of our high-order nodal DGSEM with the first-order FV method. A plausible idea is to construct a FV method on a subcell grid which directly uses the nodal LGL values of the high-order DGSEM as subcell *average* values for the FV method. This gives us the opportunity to construct the FV method in a similar fashion to the DGSEM with an identical surface term and a volume term which we are able to blend. Further advantages are that we do not have to store additional degrees of freedom for the FV method and that we meet our important conservation property.

The SBP property of our DGSEM which makes the weak formulation and the strong formulation equivalent and finally leads to a split-form DG scheme, is not only needed to be able to use symmetric two-point flux functions with additional desirable properties such as entropy-conservation derived for FV methods. Due to the fact that we are now able to rewrite the DGSEM volume term in a FV kind of surface flux, we are able to interchange the DGSEM volume term, which is the main issue for oscillations while shocks are present, with the robust and shock approved first-order FV method when needed.

Roughly speaking, we want to use our high-order entropy-stable DGSEM as much as possible to obtain more accurate results while we want to switch to the robust first-order FV method when stability problems occur. Since the surface terms of the DGSEM are algebraically equal to the FV surface terms, we do not have to worry about this part of the scheme. The difficulty lies in the volume part of the DGSEM which is the high-order polynomial inside the DG element. By dividing each DG element into $N_p + 1$ so called *subcells* of size w_i,

we are able to use our first-order FV method inside the DG element, see Figure 3.1. Now, we get an accurate high-order and a robust first-order solution of the

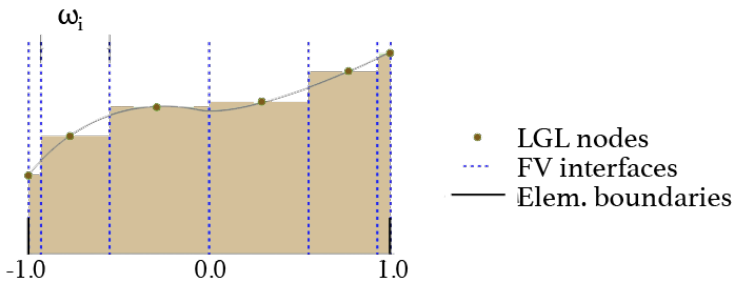


Figure 3.1: Schematic of a DG element in reference space divided into FV subcells of size w_i adapted from [55].

volume term which can then be chosen appropriately or be blended together. This way, we do not have to use pure FV inside the DG element, but just as much as needed, which prevents the FV method to dominate and provide a first-order result. Usually, it is already sufficient to use a maximum of 50 percent FV method to stabilize the solution. The following procedure is based on Hennemann et al. [8]. Similar procedures have also been used in [55–58].

3.5.1 Shock Indicator

An important question now is how to decide if a DG element needs blending with the FV method and how much blending is needed to stabilize the solution. Since the DGSEM is working with polynomials inside the element, an idea by Persson and Peraire [59] is to pick a quantity of interest, usually a quantity which has to be positive like pressure, and compare its modal energy of the highest polynomial modes to its overall modal energy. Since we derived our DGSEM as a nodal based scheme with Lagrange polynomials, we first have to transform our observed quantity into a modal based representation with Legendre polynomials defined as

$$\langle \zeta, \zeta \rangle_{L^2} = \langle \sum_{j=0}^{N_p} m_j \tilde{L}_j, \sum_{j=0}^{N_p} m_j \tilde{L}_j \rangle_{L^2} = \sum_{i,j=0}^{N_p} m_i m_j \langle \tilde{L}_i, \tilde{L}_j \rangle_{L^2} = \sum_{j=0}^{N_p} m_j^2, \quad (3.128)$$

with the modal coefficients $\{m_j\}_{j=0}^{N_p}$, m_{N_p} being the highest mode and ζ being the indicator variable. Now if we want to calculate the energy share of the highest mode compared to the total energy of the polynomial, we calculate

$$\mathcal{E} = \left(\frac{m_{N_p}^2}{\sum_{j=0}^{N_p} m_j^2} \right). \tag{3.129}$$

To overcome possibly arising odd/even effects, Hennemann et al. [8] proposed to evaluate additionally the second highest mode m_{N_p-1} compared to the total energy of the polynomial without the influence of the highest mode

$$\mathcal{E} = \max \left(\frac{m_{N_p}^2}{\sum_{j=0}^{N_p} m_j^2}, \frac{m_{N_p-1}^2}{\sum_{j=0}^{N_p-1} m_j^2} \right). \tag{3.130}$$

The question that now arises is how we know if the energy share of the highest/second highest mode is indeed critical and that if the element needs FV stabilisation. Again, [59] brings a threshold into play which decides if the energy share presents a shock in the element or not

$$\mathcal{T}(N_p) = a \cdot 10^{-c(N_p+1)^{\frac{1}{4}}}. \tag{3.131}$$

Based on this, [8] has proposed the parameters $a = 0.5, c = 1.8$ resulting from insights of various numerical experiments leading to

$$\mathcal{T}(N_p) = 0.5 \cdot 10^{-1.8(N_p+1)^{\frac{1}{4}}}. \tag{3.132}$$

Whether this choice is perfect is up for debate, but it actually works well for our kind of applications (see Chapter 6).

3.5.2 Blending Factor α

Now that we know the modal energy and also have a suitable threshold, we are able to translate these findings into a blending factor $\alpha \in [0, 1]$ which states the share of FV needed for stabilisation. An easy mapping would for example be a step function

$$\alpha = \begin{cases} 1, & \mathcal{E} \geq \mathcal{T} \\ 0, & \mathcal{E} < \mathcal{T}, \end{cases} \tag{3.133}$$

where we use full FV if the highest mode energy compared to the total mode energy is greater or equal to the designed threshold and full DG if not. Another possibility is to do a smooth mapping, that enables values between 0 and 1 like the one from [8]

$$\alpha = \frac{1}{1 + \exp(\frac{-9.21024}{\mathcal{T}}(\mathcal{E} - \mathcal{T}))}, \tag{3.134}$$

where we get $\alpha \approx 0$ for $\mathcal{E} = 0$ and $\alpha \approx 1$ for $\mathcal{E} = 1$.

For performance reasons, it is also possible to clip extreme edge values and use just one scheme instead of both (which would be necessary for $\alpha \in (0, 1)$)

$$\tilde{\alpha} := \begin{cases} 0, & \alpha < \alpha_{\min} \\ \alpha, & \alpha_{\min} \leq \alpha \leq 1 - \alpha_{\min} \\ 1, & 1 - \alpha_{\min} < \alpha, \end{cases} \tag{3.135}$$

where we usually choose $\alpha_{\min} = 0.001$.

Apart from this, we can also adjust the range of our blending coefficient by setting a maximum $\alpha \in [0, \alpha_{\max} \leq 1]$ which helps to get more accurate results in simulations with rather weak shocks. As reported by Henneman et al. [8], in some cases sudden changes in the discretization operator might generate artifacts in the solution. Therefore, it is advantageous to propagate the blending coefficient also to neighboring elements (NE)

$$\alpha^{\text{final}} = \max_{NE}\{\alpha, 0.5\alpha_{NE}\}, \tag{3.136}$$

where we take the calculated blending coefficient for the corresponding element or at least fifty percent of the highest blending coefficient of the elements sharing a face with this element.

3.5.3 Convex Blending

The basis for our scheme is the split form DGSEM in strong form which can be written as

$$\mathcal{J}\underline{\dot{u}} + \underline{\mathcal{R}}^{DG} = J\underline{\dot{u}} + \underline{\underline{M}}^{-1}\left[\underline{\underline{\Delta}}\,\underline{\tilde{f}} + \underline{\underline{B}}(\underline{f}^* - \underline{f})\right] = 0, \tag{3.137}$$

with the standard FV differencing matrix

$$\underline{\underline{\Delta}} := \begin{pmatrix} -1 & 1 & 0 & \dots & \dots & 0 \\ 0 & -1 & 1 & 0 & \dots & 0 \\ \dots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & -1 & 1 & 0 \\ 0 & \dots & \dots & 0 & -1 & 1 \end{pmatrix}, \tag{3.138}$$

for the volume term. Our first-order FV discretization we derived in section 3.2 has the form

$$\dot{\underline{u}}_i + \frac{1}{\Delta x_i}(f^*_{i+\frac{1}{2}} - f^*_{i-\frac{1}{2}}) = 0, \tag{3.139}$$

with the control volume size Δx_i. Since we are working on a subcell grid based on Gauss-Lobatto nodes, our control volumes have different sizes which are given by the Gauss-Lobatto quadrature weights leading to

$$\mathcal{J}\dot{\underline{u}}_i + \frac{1}{w_i}(f^*_{(i,i+1)} - f^*_{(i-1,i)}) = 0, \tag{3.140}$$

whereby $\mathcal{J}$ denotes the Jacobian which transforms the DG element into the reference element on the interval $E = [-1, 1]$ and $i \in \{0, \dots, N_p\}$ denotes the subcell elements inside the DG element. This FV discretization on the subcell grid can also be written in matrix vector notation as

$$\mathcal{J}\dot{\underline{u}} + \underline{\mathcal{R}}^{FV} := J\dot{\underline{u}} + \underline{\underline{M}}^{-1}\underline{\underline{\Delta}}\,\underline{f}^{FV} = 0, \tag{3.141}$$

with $\underline{f}^{FV} = (f^*_{(L,0)}, f^*_{(0,1)}, \dots, f^*_{(N_p-1,N_p)}, f^*(N_p, R))^T \in \mathbb{R}^{N_p+2}$. By introducing the following fluxes

$$\tilde{f}^{FV}_0 = f_0, \tag{3.142}$$

$$\tilde{f}^{FV}_i = f^{FV}_i = f^*_{(i-1,i)}, \quad i \in \{1, \dots, N_p\} \tag{3.143}$$

$$\tilde{f}^{FV}_{N_p+1} = f_{N_p}, \tag{3.144}$$

we are able to rewrite (3.141) in a similar fashion as the strong form DGSEM

$$\mathcal{J}\dot{\underline{u}} + \underline{\mathcal{R}}^{FV} = \mathcal{J}\dot{\underline{u}} + \underline{\underline{M}}^{-1}\left[\underline{\underline{\Delta}}\,\tilde{\underline{f}}^{FV} + \underline{\underline{B}}(\underline{f}^* - \underline{f})\right] = 0. \tag{3.145}$$

Now we can see that it is possible to directly blend the first-order FV operator with the high-order DG operator as follows

$$\underline{R} := \alpha \underline{\mathcal{R}}^{FV} + (1-\alpha)\underline{\mathcal{R}}^{DG}, \tag{3.146}$$

with the blending function α (3.134). Since we got both schemes on the same subcell grid, we can simply put everything together

$$\mathcal{J}\underline{\dot{u}} + \underline{\underline{M}}^{-1}\left[\underline{\underline{\Delta}}\left(\alpha\underline{\tilde{f}}^{FV} + (1-\alpha)\underline{\tilde{f}}\right) + \underline{\underline{B}}(\underline{f}^* - \underline{f})\right] = 0. \tag{3.147}$$

Therefore, we only have to calculate the volume contribution of both schemes and blend them together to obtain our hybrid DGSEM solution. A more compact notation can be taken from [8], where we can apply our blending directly to the volume flux

$$\tilde{f}_0^\alpha = f_0, \tag{3.148}$$

$$\tilde{f}_i^\alpha := \alpha f^*_{(i-1,i)} + (1-\alpha)\tilde{f}_i \quad i \in \{1,\dots,N_p\} \tag{3.149}$$

$$\tilde{f}^\alpha_{N_p+1} = f_{N_p} \tag{3.150}$$

and obtain the original strong formulation for our hybrid scheme

$$\mathcal{J}\underline{\dot{u}} + \underline{\mathcal{R}}^{FV} = \mathcal{J}\underline{\dot{u}} + \underline{\underline{M}}^{-1}\left[\underline{\underline{\Delta}}\,\underline{\tilde{f}}^\alpha + \underline{\underline{B}}(\underline{f}^* - \underline{f})\right] = 0. \tag{3.151}$$

3.6 Positivity-Preserving Scheme

Although our hybrid scheme, consisting of a high-order DGSEM and a first-order FV, works pretty well in capturing shocks and thereby stabilizing the solution, there are difficult simulations in which it might not be sufficient. A common method to overcome this issue is to monitor the simulation and to intervene if necessary with a posteriori scheme. A well-known scheme is the positivity-preserving limiter by Zhang and Shu [60].

Another possibility, developed by Rueda-Ramírez and Gassner [10], is to use the hybrid scheme framework for an a posteriori positivity limiter. Since we are calculating the volume update with our high-order DGSEM as well as our first-order FV method, we are able to use the FV method solution as a safe solution. Based on this, it is possible to recalculate the volume update with

a stronger blending coefficient α leading to a larger FV share in the solution or completely switch to the FV solution which is known to be very robust and positivity-preserving under the right CFL condition and Riemann solvers.

In the following subsections we will discuss the main idea of the positivity-preserving scheme from Rueda-Ramírez and Gassner [10] and how to apply it for the critical quantities like density or pressure. In addition, we will introduce an extension to the scheme to make it work for multi-component simulations.

3.6.1 Density Correction

As a basis for this scheme, we assume that the calculated FV solution is a safe solution (taking into account a suitable Riemann flux and time integration), which preserves the positivity of critical quantities like density or pressure. Since our main high-order scheme also needs a suitable high-order time integration scheme to achieve high-order accuracy, usually a high-order Runge-Kutta (RK) time integration scheme like the strong-stability preserving (SSP) RK methods of Spiteri and Ruuth [61] is used which, applied with pure FV method look as follows

$$u_{\text{safe}}^{s+1} = a_{ss}u^s + \Delta t^s \dot{u}^{\text{s, FV}} + \sum_{i=1}^{s-1} \left(a_{si}u^i + \Delta t b_{si}\dot{u}^i \right), \tag{3.152}$$

with the RK coefficients a_{si}, b_{si} and the time-step size $\Delta t^s = b_{ss}\Delta t$ of the s RK-stage.

Now, we apply a threshold to the solution, which prohibits too strong deviations from the safe solution

$$\rho \geq \beta \rho_{\text{safe}}, \tag{3.153}$$

by the factor $\beta \in (0, 1]$. A good value, according to [10], is $\beta = 0.1$ which means that the hybrid solution has to be larger than ten percent of the safe FV solution

$$\rho \geq 0.1 \rho_{\text{safe}}. \tag{3.154}$$

On closer inspection, this is even a stronger condition than positivity

$$\rho \geq \beta \rho_{\text{safe}} > 0, \tag{3.155}$$

as long as the safe solution really is positive.

Since we are using the stage update of the RK method for the pure FV method as well as for the hybrid method, we obtain the following relation for the density quantity

$$\rho_{\text{safe}} - \rho = \Delta t^s \left(\dot{\rho}^{\text{s, FV}} - \dot{\rho}^{\text{s}} \right). \tag{3.156}$$

Then, we rewrite this relation by adding zero to the equation

$$\rho_{\text{safe}} - \rho_{\text{new}} + a_\rho = \Delta t^s \left(\dot{\rho}^{\text{s,FV}} - \dot{\rho}^{\text{s}} \right), \tag{3.157}$$

which means that we added and subtracted $\beta\rho_{\text{safe}}$ to the equation and defined $\rho_{\text{new}} = \beta\rho_{\text{safe}}$ as well as $a_p = \beta\rho_{\text{safe}} - \rho$. This way, we can control our positivity condition (3.153) by checking if it violates the condition $a_p > 0$. If this should be the case, we have to correct the blending factor by correcting the solution to be $\rho = \rho_{\text{new}}$. As a consequence, we want to change the hybrid solution of the density ρ to our new solution ρ_{new} by rewriting (3.157) as follows

$$\rho_{\text{safe}} - \rho_{\text{new}} = \Delta t^s \left(\dot{\rho}^{\text{s,FV}} - \dot{\rho}^{\text{s,new}} \right), \tag{3.158}$$

with $\dot{\rho}^{\text{s,new}} = \dot{\rho}^s + \frac{a_\rho}{\Delta t^s}$. Therefore, we can recalculate our blending coefficient a posteriori as

$$\alpha_{\text{new}} = \alpha + \frac{a_\rho}{\Delta t^s \left(\dot{\rho}^{\text{s,FV}} - \dot{\rho}^{\text{s,DG}} \right)}, \tag{3.159}$$

so that we are able to compute the blending difference $\Delta\alpha = \alpha_{\text{new}} - \alpha$ which we can use to calculate the corrected solution and its time derivative as

$$u_{\text{new}}^{s+1} = u^{s+1} + \Delta\alpha \Delta t^s \left(\dot{u}^{\text{FV}} - \dot{u}^{\text{DG}} \right), \tag{3.160}$$

$$\dot{u}_{\text{new}}^{s+1} = \dot{u}^{s+1} + \Delta\alpha \left(\dot{u}^{\text{FV}} - \dot{u}^{\text{DG}} \right). \tag{3.161}$$

Due to the element-wise blending of the hybrid method, it is also necessary to apply the positivity-preserving element-wise by calculating the new blending coefficient as the maximum over all degrees of freedom N_p of an element

$$\alpha_{\text{new}} = \max_i \{\alpha_{\text{new}}^i\}, \quad i \in \{1, \ldots, N_p\}. \tag{3.162}$$

3.6.2 Pressure Correction

Now that we have corrected the density quantity, it is necessary to check if we also have to correct the resulting pressure quantity based on the similar condition $p \geq \beta p_{\text{safe}}$. Since the pressure does not depend linearly on the blending coefficient, we have to solve a non-linear equation for α_{new}

$$g(\alpha_{\text{new}}) = p(u_{\text{new}}(\alpha_{\text{new}})) - \beta p_{\text{safe}} = 0, \tag{3.163}$$

which can be solved with any suitable iterative method like Newton's method

$$\alpha_{\text{new}}^{n+1} = \alpha^n - \frac{g(\alpha^n)}{\partial p(\alpha^n)/\partial \alpha}. \tag{3.164}$$

Due to the chain rule we have to calculcate

$$\frac{\partial p}{\partial \alpha} = \frac{\partial u}{\partial \alpha}\frac{\partial p}{\partial u}, \tag{3.165}$$

where we get

$$\frac{\partial u}{\partial \alpha} = \Delta t^s \left(\dot{u}^{\text{FV}} - \dot{u}^{\text{DG}}\right), \tag{3.166}$$

derived from the RK update of the hybrid scheme. The second part, $\frac{\partial p}{\partial u}$, is dependent on the equation of state of the underlying equations. Again, we compute the corrected solution by using our new blending coefficient as we did for the density correction in the step before.

3.6.3 Partial Density Correction

Although this positivity-preserving scheme works fine by satisfying the positivity constraint for the overall density and pressure for the multi-component versions of equations, we cannot guarantee that the density components satisfy the positivity constraints. Therefore, we have to adapt the density correction described above to obtain positivity in each partial density.

Instead of applying the density correction to the total density ρ, we have to apply it to each density component $\rho_k, k \in \{1, \ldots, N\}$, which results in the positivity of the total density simultaneously. Therefore, we calculate

$$a_{\rho_k} = \beta \rho_{k,\text{safe}} - \rho_k, \quad k \in \{1, \ldots, N\}, \tag{3.167}$$

and check if $\exists k : a_{\rho_k} > 0$. Is that the case, we calculate our new blending coefficient

$$\alpha_{\text{new}} = \alpha + \max_k \left\{ \frac{a_{\rho_k}}{\Delta t^s \left(\dot{\rho}_k^{\text{s,FV}} - \dot{\rho}_k^{\text{s,DG}} \right)} \right\}, \tag{3.168}$$

so that even the most critical component is saved. Like before, these steps have to be done for all degrees of freedom of the underlying DG element. As we will see in chapter 5, the positivity-preserving scheme is able to rescue simulations even under very difficult circumstences.

3.7 Adaptive Mesh Refinement

Sometimes it might be that simulations have different regions with flow features that need high resolution, be it a turbulent flow or a shock wave, or regions where the solution is very regular. If this is the case, it is possible to save some computational effort with the adaptive mesh refinement (AMR) technique by refining the mesh locally and dynamically when necessary and otherwise working with very coarse elements, see [62, 63]. We focus on a 2:1 mesh refinement by halving an element in one dimension. In order to integrate the AMR into our numerical scheme with conforming grids, we must now also allow for non-conforming grids. How this works with the DG method is explained in the following subchapter.

3.7.1 Non-Conforming h-Refinement

Until now, we have worked with a conforming mesh. For the AMR we want to split elements with a 2:1 refinement which can be seen in Figure 3.2 which is called *h-refinement* since we split the interface at a hanging node, see for example [64]. The halving of the elements creates some problems that have to be solved. First, it forces us to introduce a new data structure for storing the different sized elements. A simple and straightforward method is a tree-data structure that works with parent elements and child elements. Second, how do we determine when such a refinement happens? Here, we can now take a shortcut and simply use the shock indicator of the shock-capturing process. The last and most difficult question is how we move our solution between these different sized refinement levels since the interfaces do not match anymore. A

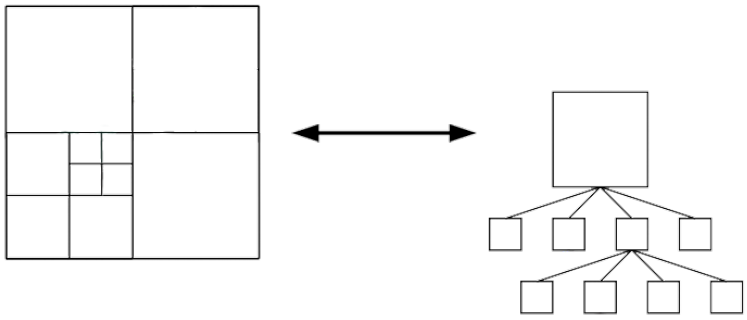


Figure 3.2: A schematic tree data structure of a non-conforming mesh.

way to solve this issue is to construct conservative prolongation (coarse to fine mesh) and restriction (fine to coarse mesh) operators with the mortar element method [65, 66] discussed in the following.

3.7.2 Mortar Element Method

For the DG method, the only thing we need to do between element interfaces is to calculate the numerical fluxes between those [67, 68]. In the mortar element method we do not calculate the numerical fluxes directly, but first project the solution of each element interface onto the mortar. The numerical fluxes are then computed on the mortar and are then projected back to the respective element interface. Therefore, we first have to derive eight different operators. That means, we need four operators which project the solution of the refined element onto their own respective mortar $P^{e_{R_1}\to m_1}, P^{e_{R_2}\to m_2}$ as well as the backprojection $P^{m_1\to e_{R_1}}, P^{m_2\to e_{R_2}}$. This works as follows; first we need to select the polynomial order of our mortar space to be the maximum of polynomial degree N_p and M_1 and respectively N_p and M_2

$$J_1 = \max(N_p, M_1), \quad J_2 = \max(N_p, M_2). \tag{3.169}$$

Now we want to derive the projections named above using $\mathbb{L}_2$-projection. Let us assume we got the values u_{R_1}, u_{R_2} so that

$$\int_{-1}^{1} u_{m_1 R_1} l_i^{m_1}(\xi) d\xi = \int_{-1}^{1} u_{R_1} l_i^{m_1}(\xi) d\xi, \quad i = 0, ..., J_1 \tag{3.170}$$

$$\int_{-1}^{1} u_{m_2 R_2} l_i^{m_2}(\xi) d\xi = \int_{-1}^{1} u_{R_2} l_i^{m_2}(\xi) d\xi, \quad i = 0, ..., J_2, \tag{3.171}$$

with the Lagrange polynomials $l_i^{m_1}, l_i^{m_2}$ on the mortar grids m_1, m_2. Now we can do an $\mathbb{L}_2$-projection to obtain

$$\int_{-1}^{1} \left(\sum_{j=0}^{J_1} u_{m_1 R_1 j} l_j^{m_1}(\xi) \right) l_i^{m_1}(\xi) d\xi = \int_{-1}^{1} \left(\sum_{k=0}^{M_1} u_{R_1 ik} l_k(\xi) \right) l_i^{m_1}(\xi) d\xi \tag{3.172}$$

$$\Leftrightarrow \sum_{j=0}^{J_1} u_{m_1 R_1 j} \underbrace{\int_{-1}^{1} l_j^{m_1}(\xi) l_i^{m_1}(\xi) d\xi}_{=(I_1)} = \sum_{k=0}^{M_1} u_{R_1 ik} \underbrace{\int_{-1}^{1} l_k(\xi) l_i^{m_1}(\xi) d\xi}_{=(II_1)}, \tag{3.173}$$

and

$$\int_{-1}^{1} \left(\sum_{j=0}^{J_2} u_{m_2 R_2 j} l_j^{m_2}(\xi) \right) l_i^{m_2}(\xi) d\xi = \int_{-1}^{1} \left(\sum_{k=0}^{M_2} u_{R_2 ik} l_k(\xi) \right) l_i^{m_2}(\xi) d\xi \tag{3.174}$$

$$\Leftrightarrow \sum_{j=0}^{J_2} u_{m_2 R_2 j} \underbrace{\int_{-1}^{1} l_j^{m_2}(\xi) l_i^{m_2}(\xi) d\xi}_{=(I_2)} = \sum_{k=0}^{M_2} u_{R_2 ik} \underbrace{\int_{-1}^{1} l_k(\xi) l_i^{m_2}(\xi) d\xi}_{=(II_2)}. \tag{3.175}$$

Using exact quadrature rules for the integrals leads to the mass matrices of the mortars

$$(I_1) = \int_{-1}^{1} l_j^{m_1}(\xi) l_i^{m_1}(\xi) d\xi =: M_{ij}^{m_1}, \quad i, j = 0, ..., J_1 \tag{3.176}$$

$$(I_2) = \int_{-1}^{1} l_j^{m_2}(\xi) l_i^{m_2}(\xi) d\xi =: M_{ij}^{m_2}, \quad i, j = 0, ..., J_2, \tag{3.177}$$

and

$$(II_1) = \int_{-1}^{1} l_k(\xi) l_i^{m_1}(\xi) d\xi = \sum_{j=0}^{J_1} w_j l_i^{m_1}(\xi_j) \underbrace{l_k(\xi_j)}_{=\delta_{kj}} \tag{3.178}$$

$$= w_k l_i^{m_1}(\xi_k) =: \mathcal{V}_{ik}^{e_{R_1} \to m_1}, \quad k = 0, ..., N_{R_1};\ i = 0, ..., J_1 \tag{3.179}$$

$$(II_2) = \int_{-1}^{1} l_k(\xi) l_i^{m_2}(\xi) d\xi = \sum_{j=0}^{J_2} w_j l_i^{m_2}(\xi_j) \underbrace{l_k(\xi_j)}_{=\delta_{kj}} \tag{3.180}$$

$$= w_k l_i^{m_2}(\xi_k) =: \mathcal{V}_{ik}^{e_{R_2} \to m_2}, \quad k = 0, ..., N_{R_2};\ i = 0, ..., J_2. \tag{3.181}$$

It follows in matrix-vector notation

$$\underline{\underline{M}}^{m_1} \underline{u}_{m_1 R_1} = \underline{\underline{\mathcal{V}}}^{e_{R_1} \to m_1} \underline{u}_{R_1} \tag{3.182}$$

$$\underline{\underline{M}}^{m_2} \underline{u}_{m_2 R_2} = \underline{\underline{\mathcal{V}}}^{e_{R_2} \to m_2} \underline{u}_{R_2}, \tag{3.183}$$

where $\mathcal{V}$ denotes a rectangular matrix which leads to the solution of the refined elements onto the mortar as

$$\underline{u}_{m_1 R_1} = (\underline{\underline{M}}^{m_1})^{-1} \underline{\underline{\mathcal{V}}}^{e_{R_1} \to m_1} \underline{u}_{R_1} \tag{3.184}$$

$$\underline{u}_{m_2 R_2} = (\underline{\underline{M}}^{m_2})^{-1} \underline{\underline{\mathcal{V}}}^{e_{R_2} \to m_2} \underline{u}_{R_2}. \tag{3.185}$$

Now we know how to do the outward projection but not how to project the calculated numerical flow back. It follows

$$\sum_{k=0}^{M_1} f_{R_1 k}^* \int_{-1}^{1} l_k(\xi) l_i(\xi) d\xi = \sum_{j=0}^{J_1} f_{m_1 R_1 j}^* \int_{-1}^{1} l_j^{m_1}(\xi) l_i(\xi) d\xi \tag{3.186}$$

$$\sum_{k=0}^{M_2} f_{R_2 k}^* \int_{-1}^{1} l_k(\xi) l_i(\xi) d\xi = \sum_{j=0}^{J_2} f_{m_2 R_2 j}^* \int_{-1}^{1} l_j^{m_1}(\xi) l_i(\xi) d\xi, \tag{3.187}$$

which can be written in matrix-vector notation

$$\underline{\underline{M}}^{e_{R_1}} \underline{f}_{R_1}^* = \underline{\underline{\mathcal{V}}}^{m_1 \to e_{R_1}} \underline{f}_{m_1 R_1}^* \tag{3.188}$$

$$\underline{\underline{M}}^{e_{R_2}} \underline{f}_{R_2}^* = \underline{\underline{\mathcal{V}}}^{m_2 \to e_{R_2}} \underline{f}_{m_2 R_2}^*, \tag{3.189}$$

leading to

$$\Rightarrow \underline{f}^*_{R_1} = \underbrace{(\underline{\underline{M}}^{e_{R_1}})^{-1}\underline{\underline{\mathcal{V}}}^{m_1\to e_{R_1}}}_{=:\underline{\underline{P}}^{m_1\to e_{R_1}}}\underline{f}^*_{m_1R_1} \tag{3.190}$$

$$\Rightarrow \underline{f}^*_{R_2} = \underbrace{(\underline{\underline{M}}^{e_{R_2}})^{-1}\underline{\underline{\mathcal{V}}}^{m_2\to e_{R_2}}}_{=:\underline{\underline{P}}^{m_2\to e_{R_2}}}\underline{f}^*_{m_2R_2}. \tag{3.191}$$

The remaining operators $P^{e_L\to m_1}$, $P^{e_L\to m_2}$, $P^{m_1\to e_L}$ and $P^{m_2\to e_L}$ are then simply calculated by projection.

It should be mentioned here that we described the standard mortar element method for simplicity. There are also entropy-stable mortar element methods, as for example in Friedrich et al. [64]. These entropy-stable mortar element methods are more expensive than the one described above, but are relevant to prove entropy stability.

3.8 Time Integration

As we have seen, we have built a numerical scheme to solve the spatial derivatives of the governing equations leading to a system of ODE's which can be solved with classical numerical methods. Due to the high-order property of our numerical scheme, it is necessary to use a high-order ODE solver so that the temporal error does not dominate the spatial error too much and therefore the order of the overall procedure remains intact. Furthermore, we also need an ODE solver whose stability area is large enough for our purposes, see Gassner and Kopriva [69].

Although there are many different procedures to solve a system of ODE's like explicit, implicit, or implicit-explicit time integrators, we choose the explicit high-order SSP Runge-Kutta procedures for our scheme. This has the advantage that such an explicit method is less computationally intensive than an implicit method, since no system of equations has to be solved, and it is also easier to implement. Since we are working with governing equations which have hyperbolic character, we are able to calculate the maximum wave speeds and are thus able to state a suitable time step restriction, which is needed to guarantee the stability of the scheme. Consequently, an implicit procedure is

not necessary in this case, as we generally do not consider a stiff problem in which it would be useful, the interested reader is nevertheless referred to [70]. It should be mentioned briefly that in this work we will indeed have to deal with stiff problems, but these stem from the chemical reaction networks and are therefore not a problem for us as they are solved with an external package KROME [71] that is able to apply subcycling. For the sake of completeness, it should be mentioned here that time integration is also a current research topic, like local time stepping methods for AMR simulations [72, 73] or space-time DG methods [74].

Since we work with an external package for time integration in this work, it is easy to switch between many different time integration methods (see chapter 4). Nevertheless, for the sake of completeness, we want to mention one of the most common time integration methods for our kind of problems. As we know solutions to hyperbolic PDE's are not always smooth, especially when talking about non-linear systems of hyperbolic PDE's used in this thesis where shock waves can develop even from smooth initial conditions leading to spurious oscillations or overshoots. Therefore, a desirable non-linear stability requirement for numerical methods would be the *strong stability preserving* (SSP) [61, 75]. A class of time integration methods designed for hyperbolic PDE's are the explicit *strong-stability preserving Runge-Kutta methods* (SSPRK) [61].

A stable time integration scheme alone is not sufficient to guarantee the stability of the overall scheme. What is required is an additional time step condition called *Courant-Friedrich-Levy* (CFL) time step condition [76] which would not be necessary using an implicit time integration scheme. However, we are quite familiar with our hyperbolic system and know the maximum wave velocities, so that we can estimate a stable timestep. In our case the CFL condition can be stated as

$$\Delta t = \frac{CFL}{|\lambda_{\max}|}\Delta x_{\text{eff}}(N_p), \quad CFL \in (0, 1] \tag{3.192}$$

with the approximation of the maximal wave speed $\lambda_{\max}$, the characteristic length of the mesh discretization $\Delta x_{\text{eff}}(N_p)$ which in our case is dependent on the element size of the DG element Δx and the polynomial degree N_p in 1D. Simply explained, with the help of this condition we can ensure that wave propagation or information propagation does not take place beyond one element,

as our procedure could not act this.

A typical problem of chemical reactions, especially for problems with high stiffness, is the time step restriction which is usually way lower than the one of the advection step of the conservation law. That is also the reason why it is common to treat the advection step of the underlying conservation law and the reaction step of the ODE system separately. This way it is possible to solve the advection step with an already tailored numerical scheme and the ODE system with an efficient ODE solver. Due to the large time step difference it is then possible to subcycle the reaction part to catch up with the advection step which leads to a reduced computational effort than treating the system as a whole. This means we have an advection step of the flow

$$S_a: \quad u_t + f(u)_{x_1} + g(u)_{x_2} = 0 \tag{3.193}$$

without any chemical reactions and a reaction step

$$S_r: \quad \frac{\partial Y_k}{\partial t} = \frac{\dot{\omega}_k}{\rho}, \quad k = 1, \ldots, N \tag{3.194}$$

with a fixed total density and constant specific internal energy. Now we can use either a first-order accurate Lie-Trotter splitting scheme [77]

$$\vec{u}^{n+1} = S_r^{(\Delta t)} \circ S_a^{(\Delta t)} u^n, \tag{3.195}$$

or a second order Strang splitting scheme [78]

$$\vec{u}^{n+1} = S_a^{(\frac{\Delta t}{2})} \circ S_r^{\Delta t} \circ S_a^{(\frac{\Delta t}{2})} u^n, \tag{3.196}$$

to approximate a solution for the next time step. In the following thesis we use our derived numerical scheme for the advection part and an external chemistry solver for the reaction part. Due to the high stiffness applications used in this thesis, we have refrained from resorting to special splitting solvers implemented in OrdinaryDiffEq.jl. For the interested reader, the Julia packages Catalyst.jl [79] and OrdinaryDiffEq.jl [80] are recommended.

4 Simulation Framework and Packages

To build a simulation framework with the capabilities needed for this thesis is a massive undertaking. Therefore, it is only plausible to use an already existing simulation framework and benefit from what is already there. In general, it is easier to add missing features into an existing code, as long as the data structures fit the project, than reinventing the wheel and programming a complete framework yourself. This approach is not only time saving, it is also an opportunity to add your knowledge and features to a broader audience which might result in further scientific continuation from other scientists. Especially in today's fast-paced world with more and more different programming languages, algorithms, and methods, it is important that new ideas can be tried out quickly and easily without disproportionate effort. In order to remain true to these thoughts, we have decided to use a relatively new but already well established simulation framework called Trixi.jl [9] which was designed with a similar thought in mind.

In this chapter, we will briefly introduce the main framework we are using in this thesis in Section 1, which is providing useful features and compatible data structures for our research. Here, we will look at the main functionalities the code already offers as well as the needed functionalities we introduced into the framework. Furthermore, we will describe the rough structure of the simulation framework and show how convenient it typically is to install and start a simulation. Thereby, we will briefly touch on how everyone is able to extend the existing code with additional features. Afterwards, we show interesting functionalities of a powerful package already existing in Julia, which is used as a key ingredient in the simulation framework, namely the DifferentialEquations.jl package [80] in Section 2. Next, we will introduce another package into the existing simulation framework, called *KROME* [71], which opens the door to the world of applied chemistry in Section 3. Last, we will take a look in the mesh generation possibilities of our chosen framework and show how easy it is to create a mesh needed for our purposes.

4.1 Introduction to Trixi.jl

The simulation framework used for this thesis is the so called Trixi.jl open-source code written in Julia. Although still very young, it is already a very powerful computational fluid dynamics (CFD) code for mainly hyperbolic partial differential equations (PDEs). It is meant to be fast, easy to understand, and extensible [9]. Among others, this is the case because of its mainly underlying highly parallelizable discontinuous Galerkin (DG) method as well as the povided parallelization techniques, the modular implementation and the comprehensive documentation. These are just a few of the properties that have been convincing points to use this simulation framework as a foundation stone for our research.

In the following subsections, we will enumerate the most important features of Trixi.jl as well as highlight features we brought into this framework for our research. We will outline the basic structure the code is implemented. Further, we will give a short introduction into the installation process of Trixi.jl and show how we can start and modify an already available simulation file, called *elixir*. In the end, we will roughly discuss the possibilities and steps to participate in the expansion of the simulation framework.

4.1.1 Main Features

Although Trixi.jl is an adaptive high-order numerical simulation framework of hyperbolic PDEs in Julia and its features are therefore strongly tailored to this task area, it is not only limited to this area. Anyway, the focus of Trixi.jl lies in the simulation of high-order methods for hyperbolic PDEs in up to three space dimensions. Similar to our research focus, Trixi.jl is specialized in hyperbolic PDEs as described in chapter 2. It supports many different equations as for example compressible Euler equations and ideal magnetohydrodynamics (MHD) equations, which have been extended by us to multi-component equations which are even allowing chemical network simulations. To give an impression of the many features this framework provides, we list the main features Trixi.jl offers [81] and mark our involvement with *added in thesis* in corner brackets where functionalities not yet integrated into the main code are marked with a (*):

- 1D, 2D, and 3D simulations on line/quad/hex/simplex meshes

  - Cartesian and curvilinear meshes
  - Conforming and non-conforming meshes
  - Structured and unstructured meshes
  - Hierarchical quadtree/octree grid with adaptive mesh refinement
  - Forests of quadtrees/octrees with p4est via P4est.jl
- High-order accuracy in space in time
- Discontinuous Galerkin methods
  - Kinetic energy-preserving and entropy-stable methods based on flux differencing for single-component equations
  - Entropy-stable methods based on flux differencing for multi-component Euler equations by Gouasmi **[added in thesis]**
  - Entropy-stable methods based on flux differencing for multi-component ideal MHD equations adapted from Gouasmi**[added in thesis]**
  - Entropy-stable shock capturing for cartesian meshes
  - Entropy-stable shock capturing for curvilinear meshes **[added in thesis]**
  - Positivity-preserving limiting by Zhang & Shu
  - Positivity-preserving limiting for single-component Equations by Rueda-Ramírez & Gassner **[added in thesis]***
  - Positivity-preserving limiting for multi-component Equations adapted from Rueda-Ramírez & Gassner **[added in thesis]***
  - Finite difference summation by parts (SBP) methods
- Compatible with the SciML ecosystem for ordinary differential equations
  - Explicit low-storage Runge-Kutta time integration
  - Strong stability preserving methods
  - CFL-based and error-based time step control
- Native support for differentiable programming
  - Forward mode automatic differentiation via ForwardDiff.jl

- Periodic and weakly-enforced boundary conditions
- Multiple governing equations:
  - Compressible Euler equations
  - Magnetohydrodynamics (MHD) equations
  - Multi-component compressible Euler equations [**added in thesis**]
  - Multi-component MHD equations [**added in thesis**]
  - Acoustic perturbation equations
  - Hyperbolic diffusion equations for elliptic problems
  - Lattice-Boltzmann equations (D2Q9 and D3Q27 schemes)
  - Shallow water equations
  - Several scalar conservation laws (e.g., linear advection, Burgers' equation)
- Multi-physics simulations
  - Self-gravitating gas dynamics
  - Chemical networks with KROME.jl [**added in thesis**]*****
- Shared-memory parallelization via multithreading
- Visualization and postprocessing of the results
  - In-situ and a posteriori visualization with Plots.jl
  - Interactive visualization with Makie.jl
  - Postprocessing with ParaView/VisIt via Trixi2Vtk

As can be seen, this thesis has brought many useful extensions into the simulation framework like multi-component Euler and ideal MHD equations, entropy-stable DGSEM for multi-component Euler and ideal MHD equations, single-component positivity-preserving limiting by Rueda-Ramírez and Gassner adapted to multi-component flows, chemical networks with KROME as well as the extension of the entropy-stable shock capturing scheme to curvilinear grids.

4.1.2 Code Structure

Now that we have seen the main features of Trixi.jl, we can take a quick look at the overall code structure of this framework which allows us to easily extend the existing code base with additional features. A discretization of our conservation laws can be divided in roughly two parts. Since we are using partial differential equations, it follows that we got partial derivatives for at least two variables. In our case, these are usually one time derivative and between one and three spatial derivatives (1D-3D). Then it is a common approach to do a semidiscretization using the so called method of lines. The idea is to discretize all spatial derivatives so that we obtain an ordinary differential equation (ODE) problem which is only dependent on the time derivative. A great advantage of this method is that ODEs are very well researched and therefore have a wide range of methods for solving such kind of equations.

This is also one of the reasons why the method of lines approach is used in Trixi.jl [9]. By discretizing the spatial part of the equations with a high-order DGSEM, we obtain an ODE problem which can be solved by another well designed subpackage called *OrdinaryDiffEq.jl*, which is part of the package DifferentialEquations.jl [80] discussed later in this chapter.

This means that our discretization in two parts looks as follows. In the first part, we semidiscretize by only discretizing the spatial parts of the equations with a solver integrated in Trixi.jl. The choice of the mesh, the equations, the initial condition, and boundary condition as well as the source term also belong to the semidiscretization part. In the second part, we now got an ODE problem which can be solved by a well suited time integration scheme out of the OrdinaryDiffEq.jl package [80].

Since the time integration scheme does not live in the Trixi.jl environment, it is a not trivial task to intervene in the time integration steps and stages. For some features, however, it is necessary to be able to perform a task between time integration steps and stages. The solution for this is called a callback. A callback is a function that is called at each time step or stage and briefly interrupts the time integration. Such a callback function can be used for input and output files, the step size control of the time integration scheme, a screen output, AMR, positivity limiting as well as using a chemical network solver.

4.1.3 User Experience

The purpose of an open source simulation framework is to make it accessible to the general public. This applies in particular for potential users as well as developers. In order to get a little closer to this goal, it is important that the code is not only easy to understand but also easy to use. That is why we will now briefly outline the installation process of Trixi.jl for new users and explain how to start and configure an existing test case called elixir.

Let us begin with the installation process. Since Trixi.jl is a Julia package, it is very easy to install and run. First of all, we need to install Julia on our machine. This step differs depending on the operating system and can be done following the official Julia installation instructions [82]. Once we started Julia, we have to import Julia's builtin package manager by executing the following command:

```
julia> import Pkg
```

Now we can install all necessary packages to run Trixi.jl, including Trixi.jl itself as well as OrdinaryDiffEq.jl which is needed by Trixi.jl to run a simulation:

```
julia> Pkg.add(["Trixi", "OrdinaryDiffEq"])
```

Other useful packages for users are *Trixi2Vtk* and *Plots* which can be used to visualize solutions generated by Trixi:

```
julia> Pkg.add(["Trixi2Vtk, "Plots"])
```

Now that we have installed all necessary and useful packages, we can load the Trixi package:

```
julia> using Trixi
```

Since Trixi already has a huge base of test cases, they had to be subdivided into subfolders in the *examples/* subdirectory. To see a list of all examples already existing in Trixi, we simply have to run the following command:

```
julia> get_examples()
```

Now we can start any out of hundreds already existing test cases in Trixi with the command *trixi_include*. An exemplary test case can be executed by:

```
julia> trixi_include("examples/p4est_2d_dgsem/
                     elixir_eulermulti_diffraction_90_amr.jl")
```

An easy way to visualize the solution is by loading the Plots package:

```
julia> using Plots
```

and use the plot function to gerenate a plot:

```
julia> plot(sol)
```

Resulting in a heatmap of all primitive variables in Figure (4.1).

To modify an existing example, we can change the parameters of desire in the elixir file or simply overwrite these at the start of the simulation. The latter case can be done very easily once we know the typical function names in the elixir file. Let us illustrate this process by modifying a few parameters in the default example. As we could see at the start of the default example, it runs to the final time of $T_{end} = 0.6$:

```
Time integration
-----------------------
Start time:         0.0
Final time:         0.6        ← elixir
time integrator:  SSPRK54
```

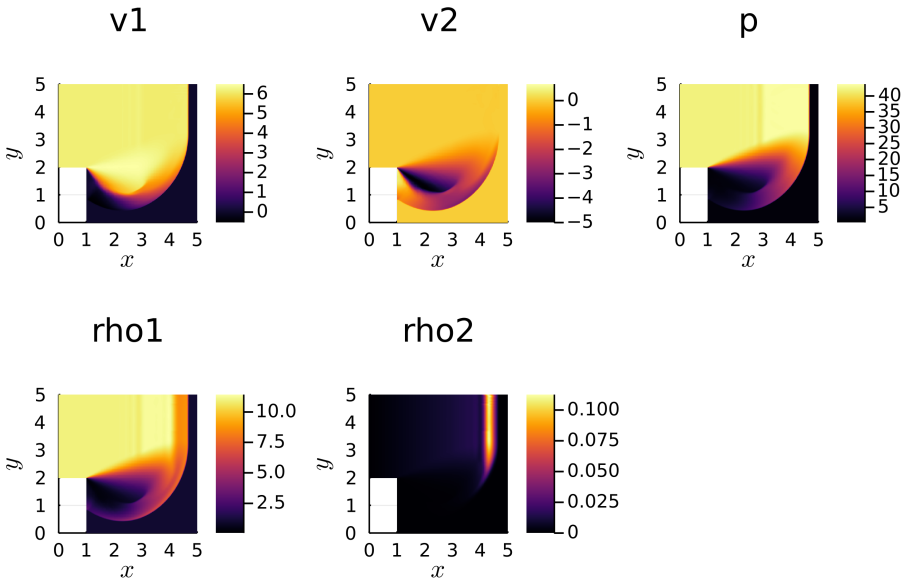


Figure 4.1: A reactive multi-component Euler diffraction simulation with AMR and 20700 total degrees of freedom in the beginning using a HOHQMesh generated P4est mesh. The simulation runs until the endtime T=0.6 with CFL=0.9 using the SSPRK54 time integration scheme as well as the third-order DGFV hybrid scheme.

Should we decide to run the simulation until time $T_{new} = 0.2$, we can simply overwrite the timespan tuple *tspan* with our new endtime as follows:

```
julia> trixi_include("examples/p4est_2d_dgsem/
                     elixir_eulermulti_diffraction_90_amr.jl",
                     tspan=(0.0, 0.2))
```

Resulting in a simulation running to time $T_{end} = 0.2$:

```
Time integration
-----------------------
Start time:         0.0
Final time:         0.2        ← new
time integrator:  SSPRK54
```

Let us now take a look at the solution by generating a plot:

```
julia> plot(sol)
```

Resulting in a less developed shock front heatmap of all primitive variables in Figure (4.2).

In a similar manner, it is also possible to write and use a completely new initial condition. So far, the elixir uses an initial condition already predefined in Trixi.jl:

```
DGFloat64 SemidiscretizationHyperbolic
------------------------------
spatial dimensions:  ....  2
mesh:  ..................  P4estMesh{2, Float64}
equations:  .............  CompressibleEulerMulticomponentEquations2D
initial condition:  .....  initial_condition_diffraction_90
boundary conditions:  ...  6
  B3:  ..................  typeof(boundary_condition_slip_wall)
  B4:  ..................  typeof(boundary_condition_slip_wall)
  B6:  BoundaryConditionDirichlet{typeo...tial_condition_diffraction_90)}
  B1:  ..................  typeof(boundary_condition_slip_wall)
  B5:  ..................  typeof(boundary_condition_slip_wall)
  B2:  ..................  typeof(boundary_condition_slip_wall)
source terms:  ..........  nothing
solver:  ................  DG
total DOFs:  ............  20700
```

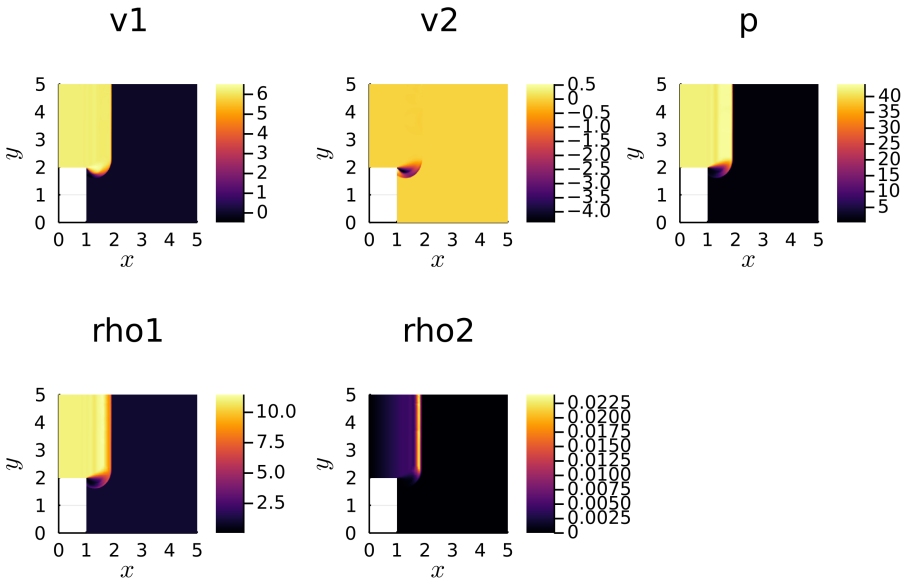


Figure 4.2: A reactive multi-component Euler diffraction simulation with AMR and 20700 total degrees of freedom in the beginning using a HOHQMesh generated P4est mesh. The simulation runs until the endtime T=0.2 with CFL=0.9 using the SSPRK54 time integration scheme as well as a third-order hybrid scheme.

We now want to use an own initial condition, which is not already included in Trixi.jl, without modifying the existing elixir. In this example we want to increase the effect of the chemistry source term by increasing the temperatures. We can simply define a new initial condition function in the Julia REPL as follows:

```
julia> function new_IC(x, t,
                       equations::CompressibleEulerMulticomponentEquations2D)
   if x[1] < 0.5
      rho1 = 11.0
      rho2 = eps()
      rho = rho1+rho2
      rho_v1 = rho * 6.18
      rho_v2 = rho * 0.0
      rho_e = 4970.0
   else
      rho1 = 1.0
      rho2 = eps()
      rho = rho1+rho2
      rho_v1 = rho * 0.0
      rho_v2 = rho * 0.0
      rho_e = 550.0
   end
   return SVector(rho_v1, rho_v2, rho_e, rho1, rho2)
end
```

Now, we are ready to start the elixir again and override the previously used initial condition with our new created initial condition:

```
julia> trixi_include("examples/p4est_2d_dgsem/
                     elixir_eulermulti_diffraction_90_amr.jl",
                     tspan=(0.0, 0.2), initial_condition=new_IC)
```

The analysis callback shows that our initial condition has really been overwritten with our new created initial condition:

```
DGFloat64 SemidiscretizationHyperbolic
------------------------------
spatial dimensions:  ....  2
mesh:  ..................  P4estMesh{2, Float64}
equations:  .............  CompressibleEulerMulticomponentEquations2D
initial condition:  .....  new_IC        ← new
boundary conditions:  ...  6
  B3:  ..................  typeof(boundary_condition_slip_wall)
  B4:  ..................  typeof(boundary_condition_slip_wall)
  B6:  BoundaryConditionDirichlet{typeo...tial_condition_diffraction_90)}
  B1:  ..................  typeof(boundary_condition_slip_wall)
  B5:  ..................  typeof(boundary_condition_slip_wall)
  B2:  ..................  typeof(boundary_condition_slip_wall)
source terms:  ..........  nothing
solver:  ................  DG
total DOFs:  ............  20700
```

To take a look at the solution of this new test case, again, we just create a heatmap plot:

```
julia> plot(sol)
```

Resulting in a different heatmap where the influence of the chemistry source terms dominate in Figure (4.3). Due to the much higher temperatures on the overall domain, the reactant component ρ_1 is quickly transformed into the product component ρ_2 and is therefore only present at the inflow boundary.

4.1.4 Developer Experience

Developers as well as more advanced users often have further demands on a code than just easy usability. A code that is as clearly structured as possible with many modular approaches is often the way to go. This way new developers and users have the opportunity to integrate new features without having to change other parts of the code. On the one hand, this allows faster familiarization with the source code, as only specific code fragments need to be understood, and on the other hand, it reduces the general programming effort. Not all

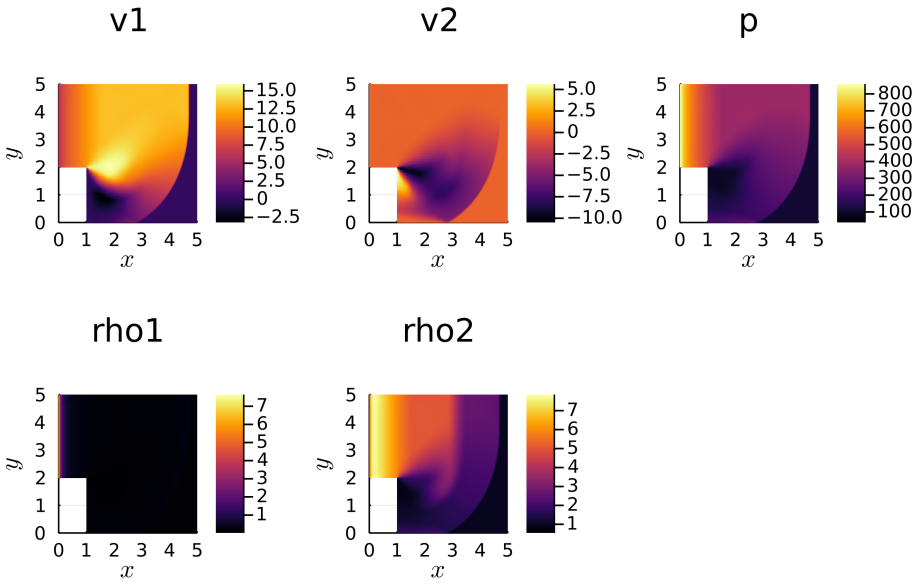


Figure 4.3: A reactive multi-component Euler diffraction simulation with adapted initial conditions and AMR as well as 20700 total degrees of freedom in the beginning using a HOHQMesh generated P4est mesh. The simulation runs until the endtime T=0.2 with CFL=0.9 using the SSPRK54 time integration scheme as well as a third-order hybrid scheme.

code modules have to be adapted to the new features. In the following, we will briefly describe the installation process of Trixi for developers and then give an example of how easy it can be to integrate new features into this framework.

The installation process is quite straightforward. This time, however, we assume that we have already installed Julia on our machine. Since we want to edit files in Trixi, we need a local copy on our machine. This means we download Trixi from github using the following command:

```
git clone git@github.com:trixi-framework/Trixi.jl.git
```

Then we go to the folder where Trixi lives:

```
cd Trixi.jl
```

Now we first have to install all dependencies for Trixi:

```
julia --project=@.  -e 'import Pkg; Pkg.instantiate()'
```

Since developers also have to visualize solutions, we also need to install post-processing tools:

```
julia -e 'import Pkg; Pkg.add(["Trixi2Vtk", "Plots"])'
```

Last but not least, we need an ODE solver package for time integration:

```
julia -e 'import Pkg; Pkg.add("OrdinaryDiffEq")'
```

To now use our local Trixi code instead of the Trixi package, we have to start Julia as follows:

```
julia --project=@.
```

Let us assume that we need an additional feature for our research not included in Trixi, this can be e.g. new governing equations, a new positivity limiter, or even a different solver. Since Trixi is build in a modular manner, we can just create a new file and implement our new features. Let us take a new positivity limiter as an example. Currently, Trixi has a folder called *callbacks_stage* where all callback functions are stated which can be called between stages of the time integration.

```
callbacks_stage
* callbacks_stage.jl
* positivity_zhang_shu.jl
* positivity_zhang_shu_dg1d.jl
* positivity_zhang_shu_dg2d.jl
* positivity_zhang_shu_dg3d.jl
* positivity_rueda-ramirez_gassner.jl
* positivity_rueda-ramirez_gassner_dg1d.jl
* positivity_rueda-ramirez_gassner_dg2d.jl
```

More precisely, the positivity limiters are mostly used here, since they are called not only at each step but also at each stage. If we want to add a new limiter, we can simply use the existing limiters as a guide. Assuming we want to build a new positivity limiter called *please_survive* for the 2D case, we create two new files called *positivity_please_survive.jl* and *positivity_please_survive_dg2d.jl*.

```
callbacks_stage
* callbacks_stage.jl
* positivity_zhang_shu.jl
* positivity_zhang_shu_dg1d.jl
* positivity_zhang_shu_dg2d.jl
* positivity_zhang_shu_dg3d.jl
* positivity_rueda-ramirez_gassner.jl
* positivity_rueda-ramirez_gassner_dg1d.jl
* positivity_rueda-ramirez_gassner_dg2d.jl
* positivity_please_survive.jl
* positivity_please_survive_dg2d.jl
```

Let us suppose that we are able to reuse the existing infrastructure of *positivity_zhang_shu.jl* so we just copy everything into *positivity_please_survive.jl* and rename it properly. Now we can build our actual method for limiting into *positivity_please_survive_dg2d.jl*. Afterwards, we just need to adjust two more files. First, we have to include our new method file *positivity_please_survive.jl* into our *callbacks_stage.jl* file and second, we have to export our new function

PositivityPreservingLimiterPleaseSurvive into the main Trixi module *Trixi.jl*.

Assuming we have tested our new method extensively and found it to be good, we can now make it available to other users of Trixi. A good way to do this is to create a new branch on github containing all necessary changes to the source code. Since Trixi has control structures to check the operability and coverage of the code, we also need to create a test example which uses our new developed positivity limiter. Now, if we think everything is just fine, we can simply create a pull request on github to merge our branch into the main code of Trixi. Main developers will now join in and scrutinize the code and, if necessary, make suggestions for improvements. If everything is OK, the code will be merged into the main code. It can be as simple as this.

4.2 Time Integration with DifferentialEquations.jl

As we have already indicated in section 1, the simulation framework we work with can roughly be divided into two parts. One part is the semidiscretization of the spatial dimension with methods implemented in Trixi.jl and the other part is the time integration of the resulting ODE from Trixi which is done with an external package. In the following, we will give a short overview about the main features of this powerful package and give an introduction into the ODE solver afterwards.

4.2.1 Main Features

DifferentialEquations.jl is a julia written package which allows to numerically solve different kinds of differential equations. The documentation of this package lists the following solvable equations:

- Discrete equations (function maps, discrete stochastic (Gillespie/Markov) simulations)
- Ordinary differential equations (ODEs) **[used in thesis]**
- Split and Partitioned ODEs (Symplectic integrators, IMEX Methods)
- Stochastic ordinary differential equations (SODEs or SDEs)
- Stochastic differential-algebraic equations (SDAEs)

- Random differential equations (RODEs or RDEs)
- Differential algebraic equations (DAEs)
- Delay differential equations (DDEs)
- Neutral, retarded, and algebraic delay differential equations (NDDEs, RDDEs, and DDAEs)
- Stochastic delay differential equations (SDDEs)
- Experimental support for stochastic neutral, retarded, and algebraic delay differential equations (SNDDEs, SRDDEs, and SDDAEs)
- Mixed discrete and continuous equations (Hybrid Equations, Jump Diffusions)
- (Stochastic) partial differential equations ((S)PDEs) (with both finite difference and finite element methods)

As we can see, the DifferentialEquations.jl package provides a variety of solvable equations, including ODEs, which have to be solved in Trixi.jl. Since the included ODE solver OrdinaryDiffEq.jl is so extensive, it is also completely independent and usable on its own, although it is only a component package in the DifferentialEquations.jl ecosystem [80]. Since even this component package is far too extensive, we will just give an abbreviated overview over the different available ODE solvers.

- Non-Stiff Equations
  - Explicit Runge-Kutta Methods
    - Euler - Canonical forward Euler method
    - Midpoint - Second order midpoint method
    - Heun - Second order Heun's method
    - RK4 - Canonical Runge-Kutta Order 4 method
    - ... and at least 17 more
  - Explicit Strong-Stability Preserving Runge-Kutta Methods for Hyperbolic PDEs (Conservation Laws)

    - SSPRK22 - Two-stage, second order strong stability preserving method of Shu and Osher
    - SSPRK33 - Three-stage, third order strong stability preserving method of Shu and Osher
    - SSPRK54 - Five-stage, fourth order strong stability preserving method of Spiteri and Ruuth [**used in thesis**]
    - ... and at least 10 more
  - Low-Storage Methods
    - ORK256 - 5-stage, second order low-storage method for wave propogation equations.
    - CarpenterKennedy2N54 - The five-stage, fourth order low-storage method of Carpenter and Kennedy [**used in thesis**]
    - ... and at least 41 more
  - Parallelized Explicit Extrapolation Methods
  - Adams-Bashforth Explicit Methods
- Stiff Equations
  - SDIRK Methods
  - Fully-Implicit Runge-Kutta Methods
  - Rosenbrock Methods
  - Stabilized Explicit Methods
  - ... and many more

Since Trixi.jl is predominantly specialised in conservation laws, we mostly use explicit strong-stability preserving Runge-Kutta methods for hyperbolic PDEs as well as low-storage methods.

4.2.2 ODE Solver

Let us now go into more detail how to use the ODE solver of OrdinaryDiffEq.jl. Let us assume we got to solve an ordinary differential equation

$$\frac{du}{dt} = f(u, p, t),$$

on some time interval $t \in [0, T]$ with some right hand side $f(u, p, t)$ dependent on the current state variable u, some parameter p and the current time t. Considering we want to solve some ODE with this package we simply do the following. First we need to install OrdinaryDiffEq.jl:

```
julia> using OrdinaryDiffEq
```

and define our right hand side with our desired function, here $5u - 3$:

```
julia> f(u,p,t) = 5u-3
```

with the initial condition of $u(2) = 1$:

```
julia> u2 = 1
```

We now want to numerically solve this ordinary equation for time $t = 3.0$:

```
julia> tspan = (2.0, 3.0)
```

Now we have all the ingredients needed to define our ODE problem:

```
julia> prob = ODEProblem(f, u2, tspan)
```

and solve it with OrdinaryDiffEq using the Tsitouras 5/4 Runge-Kutta method:

```
julia> sol = solve(prob, Tsit5(), reltol=1e-8, abstol=1e-8)
```

By loading the Plots package:

```
julia> using Plots
```

we can now plot our numerical solution:

```
julia> plot(sol,linewidth=5,title="Solution to the ODE", xaxis="Time
            (t)",yaxis="u(t)",label="Numerical Solution")
```

and overlay it with the exact solution $u(t) = \frac{2}{5}\exp(5(t-2)) + \frac{3}{5}$:

```
julia> plot!(sol.t, t->0.4*exp(5*(t-2))+0.6,lw=3,ls=:dash,label="True
            Solution")
```

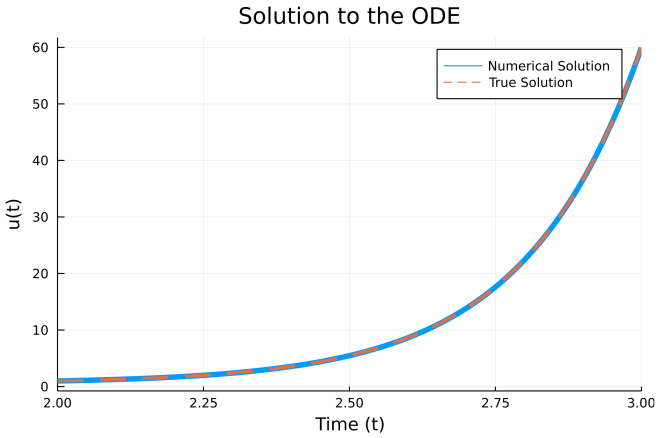


Figure 4.4: Numerical and exact ODE solution for $tspan = (2.0, 3.0)$.

4.2.3 Integration in Trixi.jl

With Trixi.jl it basically works the same way. We have to define our ODE problem *prob*, which we name *ode* in Trixi, where we have to state our right hand side (which first has to be semidiscretized by Trixi.jl) together with an initial condition as well as the time interval *tspan* to be calculated.

We have to load the necessary packages, here OrdinaryDiffEq.jl, first:

```
using OrdinaryDiffEq
```

Let us assume we want to solve some hyperbolic PDE with Trixi. Therefore, we first have to carry out the semidiscretization. We collect all data structures and functions needed for the spatial discretization of our hyperbolic PDE:

```
semi = SemidiscretizationHyperbolic(mesh, equations, initial_condition,
solver, boundary_conditions=boundary_conditions, source_terms=nothing,
chemistry_terms=chemistry_term)
```

Now we can state our ODE problem with the just created semidiscretized right hand side together with some time interval tspan $= (0.0, 0.6)$:

```
ode = semidiscretize(semi, tspan)
```

and solve it using some time integration scheme from OrdinaryDiffEq.jl:

```
sol = solve(ode, SSPRK54(), dt=0.1, save_everystep=false,
            callback=callbacks);
```

4.3 Chemical Networks with KROME.jl

Since we introduced multi-component equations into Trixi.jl as one part of this thesis, we now also have the possibility to introduce chemical reactions by implementing a chemical networks solver into Trixi.jl. So far, our multi-component equations in 2D looked as follows:

$$\frac{\partial}{\partial t}u + \frac{\partial}{\partial x_1}f(u) + \frac{\partial}{\partial x_2}g(u) = S(u), \tag{4.1}$$

whereas, for the multi-component Euler equations, we got

$$u = \begin{pmatrix} \rho v_1 \\ \rho v_2 \\ \rho e_t \\ \rho y_1 \\ \rho y_2 \\ \vdots \\ \rho y_N \end{pmatrix}, f(u) = \begin{pmatrix} \rho v_1^2 + p \\ \rho v_1 v_2 \\ (\rho e_t + p) v_1 \\ \rho v_1 y_1 \\ \rho v_1 y_2 \\ \vdots \\ \rho v_1 y_N \end{pmatrix}, g(u) = \begin{pmatrix} \rho v_1 v_2 \\ \rho v_2^2 + p \\ (\rho e_t + p) v_2 \\ \rho v_2 y_1 \\ \rho v_2 y_2 \\ \vdots \\ \rho v_2 y_N \end{pmatrix}, S(u) = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

for the vectors of the conserved variables, the flux in the x_1- and x_2-direction and source terms. To add the chemical reaction feature to this equations, we now have to add source terms with rate of changes $\dot{\omega}_i$ to the individual component concentrations [83, 84]

$$S(u) = \left(0, 0, 0, \dot{\omega}_1, \dot{\omega}_2, \ldots, \dot{\omega}_N\right).$$

In addition to the already existing multi-component advection solver, for (4.1) with $S(u) = 0$, we now need an additional solver for systems of ODEs of chemical kinetics.

A very powerful package to solve this kind of ODEs is KROME. This Fortran package, consisting of a Python pre-processor, was invented to simplify the usage of chemical networks in large numerical simulations in any numerical code [71]. In addition to solving chemical networks, KROME also provides a large set of features corresponding to physical phenomenon connected to chemistry. Especially astrophysical phenomena like photochemistry, heating, cooling, dust treatment, and reverse kinetics [71] can be simulated with KROME. In the following subsection, we will focus on reaction rates and show how easy it is to introduce new chemical networks into Trixi.jl with KROME.

4.3.1 KROME in Trixi.jl

To understand what is happening under the hood of KROME, we have to explain the rate equations first. These ODEs describe the formation and destruction of the density components via a given set of chemical reactions. The total production rate of each number density n_i is the difference between the

production (first sum) and the destruction (second sum) of every single reaction

$$\frac{dn_i}{dt} = \sum_{j \in \tilde{F}_i} \left(K_j^f \prod_{r \in \tilde{\mathcal{R}}_j} n_{r(j)} \right) - \sum_{j \in \tilde{D}_i} \left(K_j^b \prod_{r \in \tilde{\mathcal{R}}_j} n_{r(j)} \right). \tag{4.2}$$

Hereby we denote K_j^f, K_j^b to be the forward or backward reaction rate coefficient of the jth reaction belonging to the set of formation $\tilde{F}_i$ or the set of destruction $\tilde{D}_i$ of species i. In addition, $\tilde{\mathcal{R}}_j$ is the set of reactants of reaction j and $n_{r(j)}$ the number density of each reactants of reaction j.

Instead of solving these ODEs in Trixi.jl itself, again, we want to use an already existing package in Trixi. For this purpose, a Julia wrapper had been created and added to the Trixi framework named KROME.jl. We have integrated the necessary infrastructure for this package into Trixi.jl in such a way that new chemical networks can be implemented with minimal effort. In the following, we will show how easy it can be done.

Suppose we have a multi-component elixir with two components, a and b, as a starting point. We first have to state the reaction equations, which, in this example, looks as follows:

$$a \to b,$$

in a separate file. Therefore, we create a new file *diffraction_reaction* and store it in the elixir folder. Let us assume that we use the Arrhenius form, which means that the reaction rate is dependent of the temperature and looks as follows

$$\omega = -\tilde{K} \rho Y e^{\frac{-\tilde{T}}{T_{gas}}}.$$

where $T_{gas} = \frac{p}{\rho}$ is the temperature, $\tilde{K} = 2566.4$ some constant and $\tilde{T} = 50$ the constant activation temperature. Then our *diffraction_reaction* file has to look as follows:

```
@format:idx,R,P,rate
        1,FK1,FK2,2566.4d0*n(1)*exp(-50d0/Tgas)
```

where we have to state the index number of the reaction, the reactant, the product, and the reaction rate in Arrhenius form. Once we created this file, we go into our Trixi.jl folder and build our KROME configuration by executing:

```
 JULIA_KROME_CUSTOM_ARGS="-n;/path_to_Trixi.jl/examples
/path_to_elixir_folder/diffraction_reaction;" julia -e 'using Pkg;
Pkg.build("KROME")'
```

followed by starting julia. The only thing we need to do now is to activate the chemistry in the elixir file. For this purpose, we choose the in this thesis introduced chemistry term called *chemical_reaction_network*:

```
chemistry_term = chemical_reaction_network
```

and pass this to the semidiscretization:

```
semi = SemidiscretizationHyperbolic(mesh, equations, initial_condition,
solver, boundary_conditions=boundary_conditions, source_terms=nothing,
chemistry_terms=chemistry_term)
```

Despite that, we need to activate KROME as a chemistry callback:

```
chemistry_callback = KROMEChemistryCallback()
```

and pass this callback to the callback set:

```
callbacks = CallbackSet(summary_callback,
                        analysis_callback,
                        alive_callback,
                        save_solution,
                        chemistry_callback,
                        stepsize_callback)
```

Then, we start our elixir file as we always do while the *magic* happens behind the scenes. As we can see, including new chemical reactions into Trixi.jl is very simple since we just need a reaction file which has to be compiled with KROME as well as activated chemistry terms and callbacks in the elixir file.

4.4 Mesh Generation

An important feature for numerical simulations is the mesh generation. Trixi.jl is able to create and handle different kinds of meshes. In the following subsections we will give a brief overview of the provided mesh types and their features in Trixi.jl and focus on the mesh generation of complex geometries with HOHQMesh.jl for Trixi.

4.4.1 Main Features

The most simple meshes are created with the Trixi.jl internal mesh type called *TreeMesh* which is only able to use cartesian coordinates in 1D, 2D and 3D for hypercubical domains. To create and use curvilinear meshes in 1D, 2D and 3D, we can use *StructuredMesh* by setting domain boundary curves or by setting a complete transformation mapping. For more complex geometries, we can use the unstructured mesh type *UnstructuredMesh2D* which enables arbitrary domains generated by HOHQMesh.jl. An example how to use HOHQMesh.jl with Trixi.jl will be given in the next subsection. Further mesh types are *P4estMesh* which uses the sophisticated P4est library and allows for curvilinear meshes in 2D and 3D as well as the *DGMultiMesh* which allows for affine meshes but is only usable with the *DGMulti* solver. It follows an overview of these mentioned mesh types and their features in Trixi, which can also be read upon on [85].

Features	TreeMesh	StructuredMesh	UnstructuredMesh2D	P4estMesh	DGMultiMesh
Spatial Dimension	1D, 2D, 3D	1D, 2D, 3D	2D	2D, 3D	1D, 2D, 3D
Coordinates	cartesian	curvilinear	curvilinear	curvilinear	affine
Connectivity	h-nonconforming	conforming	conforming	h-nonconforming	conforming
Element Type	line, square, cube	line, quad, hex	quad	quad, hex	simplex, quad, hex
Adaptive Mesh Refinement	Yes	No	No	Yes	No
Solver Type	DGSEM	DGSEM	DGSEM	DGSEM	DGMulti
Domain	hypercube	mapped hypercube	arbitrary	arbitrary	arbitrary
Weak Form	Yes	Yes	Yes	Yes	Yes
Flux Differencing	Yes	Yes	Yes	Yes	Yes
Shock Capturing	Yes	Yes **[added in thesis]**	Yes **[added in thesis]**	Yes **[added in thesis]**	No
Nonconservative Equations	Yes	Yes	Yes	Yes	Yes

4.4.2 Trixi.jl with HOHQMesh

Since some of our applications have to be calculated on more complex geometries, we have to use a mesh type which is able to work with these geometries. For this reason, we introduced shock-capturing capabilities for the curvilinear mesh types *StructuredMesh*, *UnstructuredMesh2D* and *P4estmesh* in Trixi. In the following we will focus on *UnstructuredMesh2D* and *P4estMesh* which are able to read in 2D geometries created with HOHQMesh.jl.

HOHQMesh.jl is a high-order hex-quad mesh generator created and developed in Fortran by David Kopriva and wrapped into a julia package. Since it provides high-order boundary curve information, it is very well suited for high-order spectral element methods like the DGSEM. More information can be found here [86]. In the following, we will show how to set up a mesh with HOHQMesh and integrate it into Trixi.

Since HOHQMesh is integrated into the Trixi framework as a registered Julia package, we can simply install it using the Julia package manager:

```
julia> import Pkg; Pkg.add("HOHQMesh")
```

Let us say we want to create a mesh that looks like Figure (4.5), therefore, we need to create a control file which can be read by HOHQMesh.jl. Among other things, this control file states the domain which we want to mesh and autogenerates a fitting mesh for it. Usually, the control file can be divided into two blocks, the control input and the model.

In the control input we have to set the run parameters with the names of the mesh file, the plot file, and stats file. Furthermore, we have to set the mesh file format to *ABAQUS* required by *P4estMesh* in Trixi, some polynomial order like $N_p = 1$ of the interpolant which represents the curved boundaries on the elements and the plot file format which can be set to *skeleton* for the element boundaries without nodes or *sem* for the element boundaries with nodes:

```
\begin{CONTROL_INPUT}
   \begin{RUN_PARAMETERS}
      mesh file name = diffraction.mesh
      plot file name = diffraction.tec
      stats file name = diffraction.txt
      mesh file format = ABAQUS
      polynomial order = 1
      plot file format = skeleton
   \end{RUN_PARAMETERS}
```

Additionally, we have to set the background grid size inside the control input block, which is controlled by a spatial step size $[\Delta x_1 = 0.1, \Delta x_2 = 0.1, \Delta x_3 = 0.0]$:

```
   \begin{BACKGROUND_GRID}
      background grid size [0.1, 0.1, 0.0]
   \end{BACKGROUND_GRID}
```

Last but not least, we can activate a smoothing routine to create nicer quadrilateral elements with the following block:

```
   \begin{SPRING_SMOOTHER}
      smoothing = ON
      smoothing type = LinearAndCrossBarSpring
      number of iterations = 25
   \end{SPRING_SMOOTHER}
\end{CONTROL_INPUT}
```

Now, we can create our own outer boundaries in the model block. This can be done by connecting single lines stating a start point and end point as follows:

```
\begin{MODEL}
   \begin{OUTER_BOUNDARY}
      \begin{END_POINTS_LINE}
         name = B1
         xStart = [0.0,2.0,0.0]
         xEnd = [1.0,2.0,0.0]
      \end{END_POINTS_LINE}
      \begin{END_POINTS_LINE}
         name = B2
         xStart = [1.0,2.0,0.0]
         xEnd = [1.0,0.0,0.0]
      \end{END_POINTS_LINE}
      \begin{END_POINTS_LINE}
         name = B3
         xStart = [1.0,0.0,0.0]
         xEnd = [5.0,0.0,0.0]
      \end{END_POINTS_LINE}
      \begin{END_POINTS_LINE}
         name = B4
         xStart = [5.0,0.0,0.0]
         xEnd = [5.0,5.0,0.0]
      \end{END_POINTS_LINE}
      \begin{END_POINTS_LINE}
         name = B5
         xStart = [5.0,5.0,0.0]
         xEnd = [0.0,5.0,0.0]
      \end{END_POINTS_LINE}
      \begin{END_POINTS_LINE}
         name = B6
         xStart = [0.0,5.0,0.0]
         xEnd = [0.0,2.0,0.0]
      \end{END_POINTS_LINE}
   \end{OUTER_BOUNDARY}
\end{MODEL}
```

Since we have finished our control file, it is time to create the mesh. Therefore, we have to load HOHQMesh.jl first:

```
julia> using HOHQMesh
```

Because we need to tell *HOHQMesh* where to find our control file, we have to state the path

```
control_file = joinpath("out", "diffraction.control")
```

and can then generate our mesh with *HOHQMesh* using the *generate_mesh* function:

```
output = generate_mesh(control_file)
```

To visualize our generated mesh, we can open the *diffraction.tec* file with Paraview leading to 4.5.

4.5 Creating an Elixir

Now we have the most important building blocks for our work and can begin to build the first application examples. Based on the examples considered in this chapter, we will now create a test case that combines all the building blocks in one Elixir. More specifically, we create a test case with Trixi.jl using a time integration procedure from OrdinaryDiffEq.jl, a chemical network created with KROME.jl, and a mesh we create with HOHQMesh.jl using P4estMesh. We use P4estMesh here so that we can also calculate our test case with AMR.

In the following, we want to test the detonation diffraction problem for the reactive multi-component Euler equations on the prior with HOHQMesh developed unstructured and partially curved grid using the *P4estMesh* type with AMR. Hereby, we test a detonation diffraction on an angle of ninety degrees which is a quite challenging task for high-order schemes like ours since the pressure or density might drop below zero. In particular, we got an instream

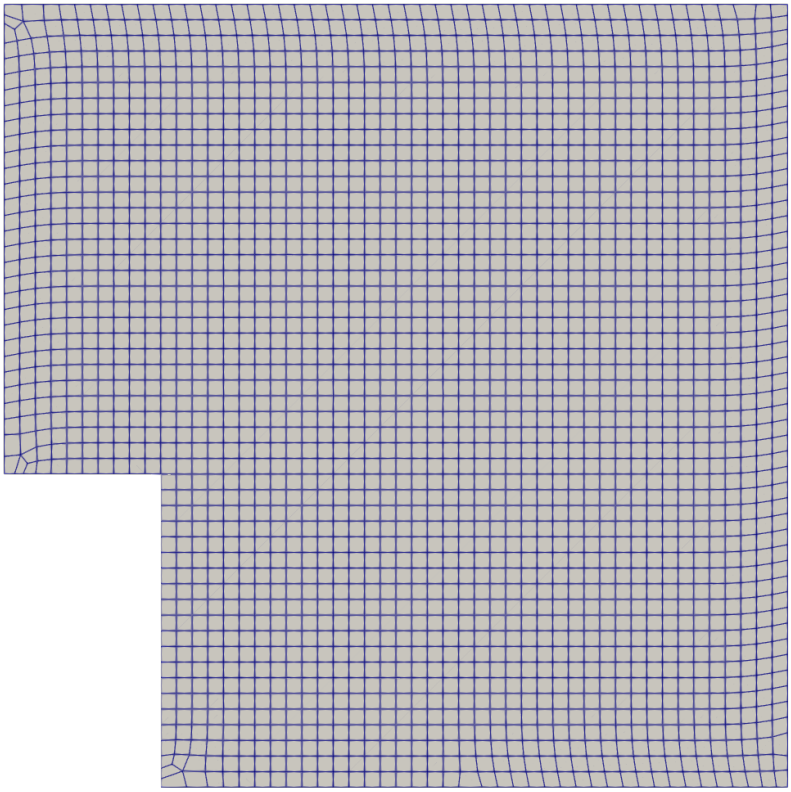


Figure 4.5: A with HOHQMesh.jl created mesh for the ninety degrees diffraction corner test case.

boundary on the left developing a shock front moving to the right. This shock wave is then diffracted at the ninety degrees corner. Furthermore, due to the high temperature at the shock front, a detonation takes place, which transforms one component into the other.

The initial condition looks as follows

$$(\rho, v_1, v_2, \rho e, Y_a, Y_b) = \begin{cases} (11, 6.18, 0, 970, 1, 0), & x < 0.5 \\ (1, 0, 0, 55, 1, 0), & x \geq 0.5 \end{cases}, \tag{4.3}$$

with the following parameters regarding the components

$$(\gamma_a, \gamma_b) = (1.2, 1.2) \tag{4.4}$$
$$(\tilde{R}_a, \tilde{R}_b) = (0.287, 0.287) \tag{4.5}$$
$$(q_a, q_b) = (50, 0). \tag{4.6}$$

On the upper left instream boundary we set the initial solution using Dirichlet boundary conditions whereas all other boundaries are set to be reflective. We use the already in subsection 4.3 discussed reaction

$$a \to b$$

as well as the temperature dependent Arrhenius form for reaction rates

$$\omega = -\tilde{K}\rho Y e^{\frac{-\tilde{T}}{T_{gas}}}.$$

where $T_{gas} = \frac{p}{\rho}$ is the temperature, $\tilde{K} = 2566.4$ is some constant and $\tilde{T} = 50$ the constant activation temperature.

The simulation will run until $T_{end} = 0.6$ with $CFL = 0.9$ using the SSPRK54 time integration scheme. We set the grid to have elements of size $\Delta x_1 = \Delta x_2 = 0.1$ resulting in 2300 elements for the underlying geometry. We solve this example with our fourth-order DGSEM shock capturing scheme which leads to overall 36800 degrees of freedom. Since we use AMR with three more levels ($\Delta x_1 = \Delta x_2 = 0.05$, $\Delta x_1 = \Delta x_2 = 0.025$, $\Delta x_1 = \Delta x_2 = 0.0125$), we will have more degrees of freedom at the shock front during the simulation.

Now, that we have all necessary information about our test case, we can start to create our elixir called *elixir_eulermulti_diffraction_90_amr.jl*. In the following, we will use some parts from the previous subsections. We begin by loading all necessary packages for our example:

```
01 using Downloads:  download
02 using OrdinaryDiffEq
03 using KROME
04 using Trixi
```

In this case, we have to be able to download the not in Trixi.jl created mesh (line 1), we need our time integration scheme from OrdinaryDiffEq.jl (line 2), the chemical network solver KROME (line 3), and our Trixi framework (line 4) to solve the spatial discretization as well as to put everything together. Now we have to choose the right equations and hand over the necessary parameters stated in (4.4):

```
05 equations = CompressibleEulerMulticomponentEquations2D(
06                           gammas = (1.2, 1.2),
07                           gas_constants = (0.287, 0.287),
08                           heat_of_formations = (50.0, 0.0))
```

Next, we have to create an initial condition function using the values stated in (4.3):

```
09 function initial_condition_diffraction_90(x, t,
10               equations::CompressibleEulerMulticomponentEquations2D)
11     if x[1] < 0.5
12         rho1 = 11.0
13         rho2 = 0.0
14         rho = rho1 + rho2
15         rho_v1 = rho * 6.18
16         rho_v2 = rho * 0.0
17         rho_e = 970.0
18     else
19         rho1 = 1.0
20         rho2 = 0.0
21         rho = rho1 + rho2
22         rho_v1 = rho * 0.0
23         rho_v2 = rho * 0.0
24         rho_e = 55.0
25     end
26     return SVector(rho_v1, rho_v2, rho_e, rho1, rho2)
27 end
```

and set this function as initial condition:

```
28 initial_condition = initial_condition_diffraction_90
```

Since this particular test case is not periodic, we need to define specific boundary conditions as described above:

```
29 boundary_condition_dirichlet = BoundaryConditionDirichlet(
30                                 initial_condition)
31 boundary_conditions = Dict( :B1 => boundary_condition_slip_wall,
32                               :B2 => boundary_condition_slip_wall,
33                               :B3 => boundary_condition_slip_wall,
34                               :B4 => boundary_condition_slip_wall,
35                               :B5 => boundary_condition_slip_wall,
36                               :B6 => boundary_condition_dirichlet)
```

In addition, we also have to define our chemistry terms, which are already predefined in Trixi.jl called *chemical_reaction_network*:

```
37 chemistry_term = chemical_reaction_network
```

Now, we have to choose our solver, including the polynomial order $N_p = 2$, the surface (LLF), and volume fluxes (central) as well as the shock capturing scheme with the associated shock capturing variable pressure:

```
surface_flux = flux_lax_friedrichs
volume_flux = flux_central
polydeg = 2
basis = LobattoLegendreBasis(polydeg)
indicator_sc = IndicatorHennemannGassner(equations, basis,
                                         alpha_max = 1.0,
                                         alpha_min = 0.0,
                                         alpha_smooth = true,
                                         variable = pressure)
volume_integral = VolumeIntegralShockCapturingHG(indicator_sc;
                    volume_flux_dg = volume_flux,
                    volume_flux_fv = surface_flux)
solver = DGSEM(polydeg = polydeg, surface_flux = surface_flux,
               volume_integral = volume_integral)
```

Furthermore, we have to choose an appropriate mesh, in this case we use the *P4estMesh* type for the *HOHQMesh* created mesh called *diffraction.inp*:

```
mesh_file = joinpath("examples/p4est_2d_dgsem","diffraction.inp")
mesh = P4estMesh{2}(mesh_file)
```

Now we got everything we need for the semidiscretization with Trixi.jl and can hand it over to create a semidiscretization:

```
semi = SemidiscretizationHyperbolic(mesh, equations,
       initial_condition, solver,
       boundary_conditions = boundary_conditions,
       source_terms = nothing,
       chemistry_terms = chemistry_term)
```

The spatial semidiscretization is now completed, so we can start to set the necessary parameter for the ODE solver as well as requiered callbacks. Since our ODE problem needs a time interval, we define the tuple *tspan* and hande it over to our ODE problem:

```
59 tspan = (0.0, 0.6)
60 ode = semidiscretize(semi, tspan)
```

Moreover, we want to have a summary, an analysis of our simulation as well as a saved solution every 100 timesteps:

```
61 summary_callback = SummaryCallback()
62 analysis_interval = 100
63 analysis_callback = AnalysisCallback(semi,
64                                       interval = analysis_interval)
65 alive_callback = AliveCallback(analysis_interval = analysis_interval)
66 save_solution = SaveSolutionCallback(interval = 100,
67                                       save_initial_solution = true,
68                                       save_final_solution = true)
```

Due to the fact that we want to use AMR in this simulation, we have to set the necessary parameters of the AMR callback. For the AMR, we use the same indicator of Hennemann and Gassner with the exact same parameters as for the shock capturing above:

```
69 amr_indicator = IndicatorHennemannGassner(semi,
70                                           alpha_max = 1.0,
71                                           alpha_min = 0.0,
72                                           alpha_smooth = true,
73                                           variable = pressure)
```

In addition, we have to set the base, med and max level with the corresponding thresholds for our AMR:

```
74 amr_controller = ControllerThreeLevel(semi, amr_indicator,
75                   base_level = 0,
76                   med_level = 1, med_threshold = 0.05,
77                   max_level = 3, max_threshold = 0.1)
```

Here we choose to use the underlying mesh as a base level, which will be refined to the *med_level = 1* if the AMR indicator is above the threshold *med_threshold=0.05* and to the *max_level = 3* if the AMR indicator strikes out above the threshold of *max_threshold = 0.1*. Now we put the AMR indicator and the AMR controller into the corresponding AMR callback:

```
78 amr_callback = AMRCallback(semi, amr_controller,
79                 interval = 5,
80                 adapt_initial_condition = true,
81                 adapt_initial_condition_only_refine = true)
```

and want the mesh to change every five timesteps. Speaking of timesteps, instead of taking uniform timesteps which usually do not fit for the whole simulation, we want to use adaptive timesteps using the CFL condition which we override with the following callback:

```
82 stepsize_callback = StepsizeCallback(cfl = 0.9))
```

The last callback we need, is the chemistry callback which activates the reaction network for every timestep:

```
83 chemistry_callback = KROMEChemistryCallback())
```

Now we have to collect all defined callbacks:

```
84 callbacks = CallbackSet(summary_callback,
85                          analysis_callback,
86                          alive_callback,
87                          save_solution,
88                          amr_callback,
89                          stepsize_callback,
90                          chemistry_callback)
```

We can run the simulation by using OrdinaryDiffEq.jl with a corresponding time integration scheme SSPRK54, our ODE problem, a dummy time step size and the just now defined callbacks:

```
91 sol = solve(ode, SSPRK54(), dt=1.0,
92             save_everystep = false, callback = callbacks);
93 summary_callback()
```

To start the simulation we now have to compile the chemical network (see subsection 4.3), create the necessary mesh (see subsection 4.4), start Julia and Trixi.jl (see subsection 4.1) and run the just created elixir as follows:

```
julia> trixi_include("examples/p4est_2d_dgsem/
                      elixir_eulermulti_diffraction_90_amr.jl")
```

To visualize the simulation with *Trixi2Vtk* we simply have to load it:

```
julia> using Trixi2Vtk
```

and then create vtu files out of the Trixi.jl produced h5 files which can be visualized by Paraview:

```
julia>  trixi2vtk(joinpath("out", "solution_0*.h5"), output_directory =
"out")
```

The result can be seen in Figure 4.6.

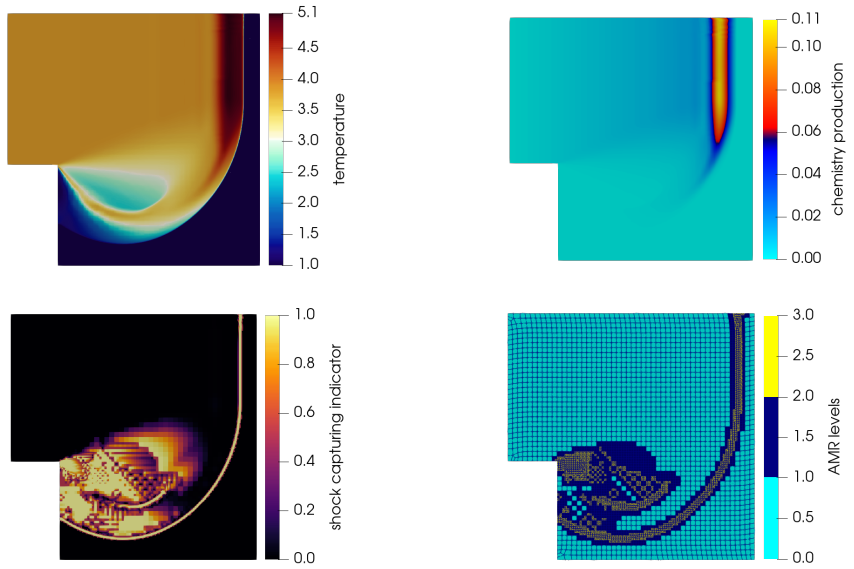


Figure 4.6: Solution of the created elixir in section 4.5 for the detonation diffraction test case.

5 Validation of the Numerical Schemes

In this chapter, we will show that the basic underlying schemes work properly and fulfill important properties for robustness. Therefore, we first provide the experimental order of convergence (EOC) to demonstrate the high-order capability of our scheme as well as the correct implementation of the underlying methods. Thereupon, entropy-stability as well as entropy-conservation will be examined which is expected to be an important cornerstone of receiving the correct solution out of the weak solution set. Since high-order schemes are quite fragile when shocks appear in the simulation, we also have to verify that our scheme is capable of providing (correct) solutions when shocks are present. Sometimes, there are situations where even our very robust shock-capturing scheme is not capable of providing a solution since some physical values, usually density or pressure, become negative using high-order simulations. As our future applications, especially when chemical networks come into play, will suffer from such problems, it is important to overcome these. Hence, we will show that our positivity-preserving scheme, which is based on our shock-capturing scheme, is capable of keeping the solution alive by switching back to a safe first-order scheme. Since some of these applications have only small areas in the domain where higher refinement is needed, they can also profit from adaptive mesh refinement (AMR), which manages to refine and coarsen the mesh dynamically during the simulation. We will show that this feature is able to produce faster and better results with less computational effort. Last but not least, we will verify the correct implementation of the chemical network solver by solving a one dimensional detonation wave example.

5.1 Multi-Component Euler Equations

5.1.1 Convergence Studies

Since our main goal is to present the advantages of high-order methods over low-order methods like the first-order FV method, we have to show that our DG

method really is a high-order scheme. For this purpose, we will verify our standard DGSEM as well as the entropy-conserving DGSEM and entropy-stable DGSEM to be high-order by showing the experimental order of convergence (EOC) using a manufactured solution. We verify the experimental order of convergence in 1D. The following test cases are adapted from [9].

Setup

To conduct our convergence tests, we create an analytical solution by using a smooth initial condition in combination with a source term, also referred to as *manufactured solution*. As initial condition we choose

$$(\rho, p, v, Y_1, Y_2, Y_3, Y_4) = \left(\phi, \phi^2, 1, \tfrac{1}{15}, \tfrac{2}{15}, \tfrac{4}{15}, \tfrac{8}{15}\right),$$

with $\phi = (2 + 0.1 * sin(\pi * (x - t)))$ and the following parameters regarding the different components

$$\begin{aligned}\left(\gamma_1, \gamma_2, \gamma_3, \gamma_4\right) &= (1.4, 1.6, 1.8, 1.2)\\ (\tilde{R}_1, \tilde{R}_2, \tilde{R}_3, \tilde{R}_4) &= (0.4, 0.6, 0.8, 0.2).\end{aligned}$$

The simulation takes place in the spatial domain $\Omega = [-1, 1]$ using periodic boundary conditions. The additional source terms for the manufactured solution are given by

$$(\rho, p, v, Y_1, Y_2, Y_3, Y_4) = \left(0, \phi, \phi, \tfrac{1}{15}, \tfrac{2}{15}, \tfrac{4}{15}, \tfrac{8}{15}\right),$$

with $\phi = \frac{-((0.4 * sin(\pi(t-x)) - 8) + 1) * 0.1\pi * (\pi - 1) * cos(\pi(t-x))}{2}$.

Solver

The simulation is observed up to the final time of $T_{end} = 0.4$ with a time step size based on the CFL condition of $CFL = 0.5$. In time we use the five-stage, fourth-order low-storage method of Carpenter and Kennedy, whereas we use the DGFV hybrid method with polynomial degrees $N_p = 3$ and $N_p = 4$ in space.

DOF	$\|\|\rho v\|\|_2$	$\|\|\rho v\|\|_\infty$	EOC_2	EOC_∞
128	8.36e-08	4.94e-07	—	—
256	5.16e-09	3.11e-08	4.02	3.99
512	3.26e-10	1.96e-09	3.99	3.99
1024	2.02e-11	1.22e-10	4.01	4.00
2048	1.27e-12	7.65e-12	4.00	4.00
4096	8.02e-14	5.10e-13	3.98	3.91
mean			4.00	3.98

DOF	$\|\|\rho e\|\|_2$	$\|\|\rho e\|\|_\infty$	EOC_2	EOC_∞
128	2.49e-07	1.35e-06	—	—
256	1.53e-08	9.07e-08	4.03	3.89
512	1.14e-09	6.71e-09	3.75	3.76
1024	6.27e-11	4.26e-10	4.18	3.98
2048	3.97e-12	2.69e-11	3.98	3.99
4096	2.32e-13	1.73e-12	4.10	3.95
mean			4.01	3.91

DOF	$\|\|\rho_1\|\|_2$	$\|\|\rho_1\|\|_\infty$	EOC_2	EOC_∞
128	5.19e-09	2.80e-08	—	—
256	3.17e-10	1.59e-09	4.03	4.14
512	2.43e-11	1.07e-10	3.71	3.89
1024	1.24e-12	6.78e-12	4.29	3.98
2048	7.58e-14	3.95e-13	4.03	4.10
4096	4.19e-15	2.49e-14	4.18	3.99
mean			4.05	4.02

DOF	$\|\|\rho_2\|\|_2$	$\|\|\rho_2\|\|_\infty$	EOC_2	EOC_∞
128	1.04e-08	5.59e-08	—	—
256	6.35e-10	3.18e-09	4.03	4.14
512	4.85e-11	2.14e-10	3.71	3.89
1024	2.48e-12	1.36e-11	4.29	3.98
2048	1.52e-13	7.89e-13	4.03	4.10
4096	8.38e-15	4.98e-14	4.18	3.99
mean			4.05	4.02

DOF	$\|\|\rho_3\|\|_2$	$\|\|\rho_3\|\|_\infty$	EOC_2	EOC_∞
128	2.07e-08	1.12e-07	—	—
256	1.27e-09	6.36e-09	4.03	4.14
512	9.70e-11	4.28e-10	3.71	3.89
1024	4.96e-12	2.71e-11	4.29	3.98
2048	3.03e-13	1.58e-12	4.03	4.10
4096	1.68e-14	9.96e-14	4.18	3.99
mean			4.05	4.02

DOF	$\|\|\rho_4\|\|_2$	$\|\|\rho_4\|\|_\infty$	EOC_2	EOC_∞
128	4.15e-08	2.24e-07	—	—
256	2.54e-09	1.27e-08	4.03	4.14
512	1.94e-10	8.56e-10	3.71	3.89
1024	9.93e-12	5.42e-11	4.29	3.98
2048	6.06e-13	3.16e-12	4.03	4.10
4096	3.35e-14	1.99e-13	4.18	3.99
mean			4.05	4.02

Table 5.1: Experimental Order of Convergence for the conservative variables using the standard DGSEM with polynomial degree $N_p = 3$ and different degrees of freedom (DOF).

Result

First, we take a look at our standard DGSEM, thus we use a central Riemann solver for the volume flux and a dissipative Riemann solver, namely the local Lax-Friedrichs Riemann solver, for the surface flux. Since most of our experiments will be performed with a fourth-order scheme, we focus on the polynomial degree $N_p = 3$ which should result in a fourth-order convergence rate, see table 5.1. Second, we examine an entropy-stable DGSEM which means that we use an entropy-conserving Riemann solver, namely the multi-component

DOF	$\|\|\rho v\|\|_2$	$\|\|\rho v\|\|_\infty$	EOC_2	EOC_∞
128	8.52e-08	5.12e-07	—	—
256	5.47e-09	3.23e-08	3.96	3.99
512	3.25e-10	2.02e-09	4.07	4.00
1024	2.08e-11	1.22e-10	3.97	4.05
2048	1.28e-12	7.53e-12	4.02	4.02
4096	8.09e-14	4.93e-13	3.99	3.93
mean			4.00	4.00

DOF	$\|\|\rho e\|\|_2$	$\|\|\rho e\|\|_\infty$	EOC_2	EOC_∞
128	5.60e-07	2.50e-06	—	—
256	5.69e-08	2.27e-07	3.30	3.47
512	4.01e-09	1.56e-08	3.83	3.86
1024	3.60e-10	1.74e-09	3.48	3.16
2048	1.53e-11	9.19e-11	4.55	4.24
4096	8.49e-13	5.08e-12	4.17	4.18
mean			3.87	3.78

DOF	$\|\|\rho_1\|\|_2$	$\|\|\rho_1\|\|_\infty$	EOC_2	EOC_∞
128	2.05e-08	7.52e-08	—	—
256	2.17e-09	7.15e-09	3.24	3.39
512	1.54e-10	5.97e-10	3.82	3.58
1024	1.38e-11	6.47e-11	3.49	3.21
2048	5.85e-13	3.49e-12	4.56	4.21
4096	3.22e-14	1.93e-13	4.18	4.18
mean			3.86	3.72

DOF	$\|\|\rho_2\|\|_2$	$\|\|\rho_2\|\|_\infty$	EOC_2	EOC_∞
128	4.11e-08	1.50e-07	—	—
256	4.33e-09	1.43e-08	3.24	3.39
512	3.08e-10	1.19e-09	3.82	3.58
1024	2.75e-11	1.29e-10	3.49	3.21
2048	1.17e-12	6.97e-12	4.56	4.21
4096	6.44e-14	3.85e-13	4.18	4.18
mean			3.86	3.72

DOF	$\|\|\rho_3\|\|_2$	$\|\|\rho_3\|\|_\infty$	EOC_2	EOC_∞
128	8.21e-08	3.01e-07	—	—
256	8.67e-09	2.86e-08	3.24	3.39
512	6.17e-10	2.39e-09	3.82	3.58
1024	5.51e-11	2.59e-10	3.49	3.21
2048	2.34e-12	1.39e-11	4.56	4.21
4096	1.29e-13	7.70e-13	4.18	4.18
mean			3.86	3.72

DOF	$\|\|\rho_4\|\|_2$	$\|\|\rho_4\|\|_\infty$	EOC_2	EOC_∞
128	1.64e-07	6.02e-07	—	—
256	1.73e-08	5.72e-08	3.24	3.39
512	1.23e-09	4.78e-09	3.82	3.58
1024	1.10e-10	5.18e-10	3.49	3.21
2048	4.68e-12	2.79e-11	4.56	4.21
4096	2.58e-13	1.54e-12	4.18	4.18
mean			3.86	3.72

Table 5.2: Experimental Order of Convergence for the conservative variables using the entropy-stable DGSEM with polynomial degree $N_p = 3$ and different degrees of freedom (DOF).

entropy-conservative flux for the volume flux and the local Lax-Friedrichs type Riemann solver for the surface flux, see table 5.2.

For verifying reasons only, we also examine an entropy-conservative scheme, which means that we use an entropy-conserving Riemann solver, namely the multi-component entropy-conservative flux, both, for the volume flux as well as for the surface flux, see table 5.3. This scheme does only work, when we are using a smooth test case which does not develop any shocks during the simulation. Therefore, this scheme is not used for any kind of application dur-

ing this thesis, since it would not fulfill necessary laws of thermodynamics for non-smooth kind of applications. To illustrate the odd-even effect for entropy-

DOF	$\|\|\rho v\|\|_2$	$\|\|\rho v\|\|_\infty$	EOC_2	EOC_∞
128	1.39e-06	5.19e-06	—	—
256	1.73e-07	6.49e-07	3.00	3.00
512	2.17e-08	8.13e-08	3.00	3.00
1024	2.71e-09	1.02e-08	3.00	3.00
2048	3.38e-10	1.27e-09	3.00	3.00
4096	4.23e-11	1.60e-10	3.00	2.99
mean			3.00	3.00

DOF	$\|\|\rho e\|\|_2$	$\|\|\rho e\|\|_\infty$	EOC_2	EOC_∞
128	2.68e-06	1.12e-05	—	—
256	3.35e-07	1.39e-06	3.00	3.01
512	4.19e-08	1.73e-07	3.00	3.00
1024	5.24e-09	2.16e-08	3.00	3.00
2048	6.55e-10	2.71e-09	3.00	3.00
4096	8.19e-11	3.40e-10	3.00	2.99
mean			3.00	3.00

DOF	$\|\|\rho_1\|\|_2$	$\|\|\rho_1\|\|_\infty$	EOC_2	EOC_∞
128	5.52e-08	2.62e-07	—	—
256	6.89e-09	3.23e-08	3.00	3.02
512	8.61e-10	4.02e-09	3.00	3.01
1024	1.08e-10	5.02e-10	3.00	3.00
2048	1.35e-11	6.28e-11	3.00	3.00
4096	1.68e-12	7.89e-12	3.00	2.99
mean			3.00	3.00

DOF	$\|\|\rho_2\|\|_2$	$\|\|\rho_2\|\|_\infty$	EOC_2	EOC_∞
128	1.10e-07	5.24e-07	—	—
256	1.38e-08	6.46e-08	3.00	3.02
512	1.72e-09	8.04e-09	3.00	3.01
1024	2.15e-10	1.00e-09	3.00	3.00
2048	2.69e-11	1.26e-10	3.00	3.00
4096	3.36e-12	1.58e-11	3.00	2.99
mean			3.00	3.00

DOF	$\|\|\rho_3\|\|_2$	$\|\|\rho_3\|\|_\infty$	EOC_2	EOC_∞
128	2.21e-07	1.05e-06	—	—
256	2.76e-08	1.29e-07	3.00	3.02
512	3.44e-09	1.61e-08	3.00	3.01
1024	4.31e-10	2.01e-09	3.00	3.00
2048	5.38e-11	2.51e-10	3.00	3.00
4096	6.73e-12	3.16e-11	3.00	2.99
mean			3.00	3.00

DOF	$\|\|\rho_4\|\|_2$	$\|\|\rho_4\|\|_\infty$	EOC_2	EOC_∞
128	4.42e-07	2.10e-06	—	—
256	5.51e-08	2.58e-07	3.00	3.02
512	6.89e-09	3.22e-08	3.00	3.01
1024	8.61e-10	4.02e-09	3.00	3.00
2048	1.08e-10	5.02e-10	3.00	3.00
4096	1.35e-11	6.31e-11	3.00	2.99
mean			3.00	3.00

Table 5.3: Experimental Order of Convergence for the conservative variables using the entropy-conservative DGSEM with polynomial degree $N_p = 3$ and different degrees of freedom (DOF).

conservative schemes, a phenomenon already observed in the literature [87–89], we show that the scheme indeed has a lower experimental order of convergence as expected and opposed to the standard and entropy-stable scheme, see table 5.4. This phenomenon can only be observed for odd polynomial orders and therefore, the last convergence tables have been obtained with an odd polynomial degree of $N_p = 3$ and an even polynomial degree of $N_p = 2$. It can be observed, that although the correct third-order convergence rate has been

achieved for the even polynomial degree $N_p = 2$, the higher odd polynomial degree of $N_p = 3$ does not deliver a fourth-order convergence rate but instead just a third-order convergence rate. Since an entropy-conservative scheme is not favorable in real world applications anyway, this phenomenon does not present a problem.

DOF	$\|\|\rho v\|\|_2$	$\|\|\rho v\|\|_\infty$	EOC_2	EOC_∞
128	9.90e-06	3.80e-05	—	—
256	1.16e-06	3.96e-06	3.09	3.26
512	1.31e-07	4.42e-07	3.15	3.16
1024	1.67e-08	7.16e-08	2.96	2.63
2048	1.63e-09	7.54e-09	3.36	3.25
4096	1.90e-10	9.52e-10	3.11	2.98
mean			3.13	3.06

DOF	$\|\|\rho e\|\|_2$	$\|\|\rho e\|\|_\infty$	EOC_2	EOC_∞
128	2.78e-05	1.19e-04	—	—
256	4.80e-06	2.09e-05	2.54	2.52
512	6.98e-07	2.68e-06	2.78	2.96
1024	1.06e-07	4.76e-07	2.73	2.49
2048	1.13e-08	6.44e-08	3.23	2.89
4096	1.06e-09	6.01e-09	3.41	3.42
mean			2.94	2.86

DOF	$\|\|\rho_1\|\|_2$	$\|\|\rho_1\|\|_\infty$	EOC_2	EOC_∞
128	9.61e-07	3.94e-06	—	—
256	1.69e-07	6.34e-07	2.51	2.64
512	2.61e-08	1.05e-07	2.70	2.59
1024	3.80e-09	1.85e-08	2.78	2.50
2048	4.11e-10	2.55e-09	3.21	2.86
4096	3.90e-11	1.93e-10	3.40	3.73
mean			2.92	2.86

DOF	$\|\|\rho_2\|\|_2$	$\|\|\rho_2\|\|_\infty$	EOC_2	EOC_∞
128	1.92e-06	7.89e-06	—	—
256	3.38e-07	1.27e-06	2.51	2.64
512	5.21e-08	2.10e-07	2.70	2.59
1024	7.60e-09	3.71e-08	2.78	2.50
2048	8.23e-10	5.10e-09	3.21	2.86
4096	7.79e-11	3.85e-10	3.40	3.73
mean			2.92	2.86

DOF	$\|\|\rho_3\|\|_2$	$\|\|\rho_3\|\|_\infty$	EOC_2	EOC_∞
128	3.84e-06	1.58e-05	—	—
256	6.76e-07	2.54e-06	2.51	2.64
512	1.04e-07	4.20e-07	2.70	2.59
1024	1.52e-08	7.41e-08	2.78	2.50
2048	1.65e-09	1.02e-08	3.21	2.86
4096	1.56e-10	7.71e-10	3.40	3.73
mean			2.92	2.86

DOF	$\|\|\rho_4\|\|_2$	$\|\|\rho_4\|\|_\infty$	EOC_2	EOC_∞
128	7.69e-06	3.16e-05	—	—
256	1.35e-06	5.07e-06	2.51	2.64
512	2.09e-07	8.41e-07	2.70	2.59
1024	3.04e-08	1.48e-07	2.78	2.50
2048	3.29e-09	2.04e-08	3.21	2.86
4096	3.12e-10	1.54e-09	3.40	3.73
mean			2.92	2.86

Table 5.4: Experimental Order of Convergence for the conservative variables using the entropy-conservative DGSEM with polynomial degree $N_p = 2$ and different degrees of freedom (DOF).

5.1.2 Mass, Momentum, Energy and Entropy Conservation

Since we are studying Euler equations, it is important to verify that our scheme is able to comply with the conservation property of mass, momentum, and energy which are intrinsic for the Euler equations. Moreover, being able to control the entropy is a key ingredient for a robust numerical scheme while being able to converge to a physical correct and unique solution. For this purpose an additional conservation equation, namely the entropy-conservation, has to be satisfied. To meet this additional property, special conservative fluxes have been derived. To show that these fluxes are really entropy-conservative, we use a simple test case with a rather weak shock since entropy-conservation can only be achieved by disabling dissipation terms which would then stabilize the solution.

Setup

To check the entropy-conservation, we initialize the initial condition with a weak shock wave

$$(\rho, p, v_1, v_2, Y_1, Y_2, Y_3, Y_4)$$
$$= \begin{cases} (1.1691, 1.245, 0.1882, 0.1882, \frac{1}{15}, \frac{2}{15}, \frac{4}{15}, \frac{8}{15}), & x \leq 0.5 \quad \text{(left side)} \\ (1.0, 1.0, 0.0, 0.0, \frac{1}{15}, \frac{2}{15}, \frac{4}{15}, \frac{8}{15}), & x > 0.5 \quad \text{(right side)} \end{cases}$$

in the spatial domain [-2, 2] using periodic boundary conditions.

Solver

We run the simulation to the final time $T_{end} = 2.0$ with a relatively small time step using $CFL = 0.3$ to have smaller time integration errors. Again, we use the five-stage, fourth-order low-storage method of Carpenter and Kennedy for time integration. For this test we choose polynomial degree $N_p = 3$ resulting in a fourth-order DG scheme and $DOF = 128$ degrees of freedom in each spatial dimension.

Result

First, we want to check the mass, momentum and energy conservation. For this purpose, we calculate the differences in total mass and total momentum accumulated over the entire domain at $T_{end} = 2$ compared to the total mass

and momentum accumulated over the whole domain at $T_{start} = 0$ for the EC DGSEM in 1D and 2D, see table 5.5 and table 5.6. As can be seen, we get mass, momentum and energy conservation within machine precision. Second, we

	ρv_1	ρ_1	ρ_2	ρ_3	ρ_4	ρe
$\lvert\Delta U\rvert$	6.8E-17	8.6E-16	1.7E-15	3.4E-15	6.9E-15	2.4E-16

Table 5.5: Difference in accumulated total mass, momentum, and energy over the whole domain at time $T_{end} = 2$ to the accumulated total mass, momentum amd energy over the entire domain at time $T_{start} = 0$ in 1D calculated with the fourth-order EC DGSEM and $DOF = 128$ degrees of freedom.

	ρv_1	ρv_2	ρ_1	ρ_2	ρ_3	ρ_4	ρe
$\lvert\Delta U\rvert$	1.8E-16	1.9E-16	2.7E-14	5.5E-14	1.1E-13	2.2E-13	4.1E-14

Table 5.6: Difference in accumulated total mass, momentum, and energy over the whole domain at time $T_{end} = 2$ to the accumulated total mass, momentum and energy over the entire domain at time $T_{start} = 0$ in 2D calculated with the fourth-order EC DGSEM and $DOF = 128^2$ degrees of freedom.

want to verify that our scheme really is entropy-conservative or entropy-stable. Therefore, we analyse the total entropy change throughout the simulation over the entire domain. For an entropy-conservative scheme we do not expect the entropy to change during the simulation, whereas for an entropy stable scheme we do expect an entropy change. Our results show that we get an entropy change of zero within machine accuracy for the entropy-conservative scheme, while for the entropy-stable scheme we can see a significant entropy drop as expected, see table 5.7.

5.2 Multi-Component Ideal MHD Equations

Similar to the multi-component Euler case, we now want to show that our entropy-stable and entropy-conservative method is also high-order for the multi-component ideal MHD equations. Therefore, we will show the experimental order of convergence using a periodic solution adapted from the single-component

	EC	ES
$\lvert\Delta S\rvert$ in 1D	8.0E-17	4.1E-04
$\lvert\Delta S\rvert$ in 2D	2.3E-17	2.8E-05

Table 5.7: Total entropy change over the entire domain at time $T_{end} = 2$ in 1D and 2D calculated with the fourth-order entropy-conservative DG method and the entropy-stable DG method using $DOF = 128$ degrees of freedom in each spatial dimension.

ideal MHD equation convergence test from [90]. Furthermore, we will also show that the basic conservation properties are fulfilled by our method. The following test cases are adapted from [9].

5.2.1 Convergence Studies

In the following, we will show the experimental order of convergence with the single-component ideal MHD convergence setup taken from [90] adapted to the multi-component case. More precisely, we use a smooth Alfvén wave test case in 2D. Here, an Alfvén wave with forty-five degrees inclination to the x-axis moves over a periodic domain and returns to the start position every integer time step $t \in \mathbb{N}$, so that the final solution can be compared with the starting solution.

Setup

The initial condition can be stated as

$$\begin{pmatrix} \rho \\ p \\ v_1 \\ v_2 \\ v_3 \\ B_1 \\ B_2 \\ B_3 \\ \psi \\ Y_1 \\ Y_2 \end{pmatrix} = \begin{pmatrix} 1.0 \\ 0.1 \\ -\phi_B * \sin(\alpha) \\ \phi_B * \cos(\alpha) \\ 0.1 * \cos(2\pi\phi_x) \\ \cos(\alpha) + v_1 \\ \sin(\alpha) + v_2 \\ v_3 \\ 0 \\ \frac{1}{3} \\ \frac{2}{3} \end{pmatrix} \tag{5.1}$$

with $\alpha = \frac{\phi}{4}$, $\phi_x = x * \cos(\alpha) + y * \sin(\alpha)$ and $\phi_B = 0.1 * \sin(2\pi\phi_x)$ and the following parameters regarding the components

$$\begin{aligned} (\gamma_1, \gamma_2) &= (2.0, 4.0) \\ \left(\tilde{R}_1, \tilde{R}_2\right) &= (1.0, 2.0)\,. \end{aligned} \tag{5.2}$$

The simulation takes place in the spatial domain $\Omega = [0, \sqrt{2}] \times [0, \sqrt{2}]$ using periodic boundary conditions.

Solver

The simulation takes place in the temporal domain $T = [0.0, 2.0]$ with a time step size determined by the CFL condition $CFL = 0.5$ as well as the GLM speed $GLM = 0.5$. We use the five-stage, fourth-order low-storage method of Carpenter and Kennedy for the time integration as well as the fourth-order entropy-stable DGSEM with polynomial degree $N_p = 3$ for the spatial integration using the LLF surface flux as well as the in this thesis derived EC volume flux.

Result

For a polynomial degree of $N_p = 3$, we should expect a convergence order of four, which we indeed achieve as we see in table 5.8. Hereby, we calculated the

L_2 and L_∞ error as well as the corresponding EOCs for $16 - 512$ degrees of freedom per spatial dimension.

DOF	$\|\|\rho v_1\|\|_2$	$\|\|\rho v_1\|\|_\infty$	EOC_2	EOC_∞
16	3.04e-04	1.43e-03	—	—
32	1.58e-05	1.29e-04	4.27	3.47
64	9.08e-07	7.42e-06	4.12	4.12
128	5.63e-08	4.82e-07	4.01	3.94
256	3.51e-09	3.03e-08	4.00	3.99
512	2.19e-10	1.89e-09	4.00	4.00
mean			4.08	3.91

DOF	$\|\|\rho v_2\|\|_2$	$\|\|\rho v_2\|\|_\infty$	EOC_2	EOC_∞
16	3.04e-04	1.43e-03	—	—
32	1.58e-05	1.29e-04	4.27	3.47
64	9.08e-07	7.42e-06	4.12	4.12
128	5.63e-08	4.82e-07	4.01	3.94
256	3.51e-09	3.03e-08	4.00	3.99
512	2.19e-10	1.89e-09	4.00	4.00
mean			4.08	3.91

DOF	$\|\|\rho v_3\|\|_2$	$\|\|\rho v_3\|\|_\infty$	EOC_2	EOC_∞
16	4.83e-04	2.71e-03	—	—
32	3.42e-05	1.83e-04	3.82	3.89
64	1.34e-06	1.10e-05	4.68	4.05
128	7.99e-08	6.78e-07	4.06	4.02
256	4.96e-09	4.25e-08	4.01	3.99
512	3.09e-10	2.66e-09	4.00	4.00
mean			4.12	3.99

DOF	$\|\|\rho e\|\|_2$	$\|\|\rho e\|\|_\infty$	EOC_2	EOC_∞
16	3.43e-04	1.05e-03	—	—
32	1.67e-05	8.53e-05	4.36	3.62
64	1.04e-06	6.69e-06	4.01	3.67
128	6.53e-08	4.41e-07	3.99	3.92
256	4.08e-09	2.78e-08	4.00	3.99
512	2.55e-10	1.74e-09	4.00	4.00
mean			4.07	3.84

DOF	$\|\|B_1\|\|_2$	$\|\|B_1\|\|_\infty$	EOC_2	EOC_∞
16	3.52e-04	1.59e-03	—	—
32	1.76e-05	1.24e-04	4.32	3.68
64	1.07e-06	8.17e-06	4.04	3.93
128	6.69e-08	5.34e-07	4.01	3.93
256	4.17e-09	3.34e-08	4.01	4.00
512	2.60e-10	2.09e-09	4.00	4.00
mean			4.07	3.91

DOF	$\|\|B_2\|\|_2$	$\|\|B_2\|\|_\infty$	EOC_2	EOC_∞
16	3.52e-04	1.59e-03	—	—
32	1.76e-05	1.24e-04	4.32	3.68
64	1.07e-06	8.17e-06	4.04	3.93
128	6.69e-08	5.34e-07	4.01	3.93
256	4.17e-09	3.34e-08	4.01	4.00
512	2.60e-10	2.09e-09	4.00	4.00
mean			4.07	3.91

DOF	$\|\|B_3\|\|_2$	$\|\|B_3\|\|_\infty$	EOC_2	EOC_∞
16	4.75e-04	2.71e-03	—	—
32	3.22e-05	1.84e-04	3.88	3.88
64	1.45e-06	1.09e-05	4.47	4.08
128	8.64e-08	6.80e-07	4.07	4.01
256	5.36e-09	4.25e-08	4.01	4.00
512	3.34e-10	2.66e-09	4.00	4.00
mean			4.09	3.99

DOF	$\|\|\psi\|\|_2$	$\|\|\psi\|\|_\infty$	EOC_2	EOC_∞
16	2.84e-04	1.04e-03	—	—
32	9.49e-06	5.33e-05	4.90	4.29
64	4.67e-07	2.51e-06	4.34	4.41
128	2.82e-08	1.48e-07	4.05	4.08
256	1.75e-09	9.16e-09	4.02	4.02
512	1.09e-10	5.71e-10	4.00	4.00
mean			4.26	4.16

DOF	$\|\|\rho_1\|\|_2$	$\|\|\rho_1\|\|_\infty$	EOC_2	EOC_∞
16	1.01e-04	4.51e-04	—	—
32	5.69e-06	2.51e-05	4.16	4.17
64	2.73e-07	1.42e-06	4.38	4.15
128	1.68e-08	9.12e-08	4.02	3.96
256	1.02e-09	5.43e-09	4.04	4.07
512	6.36e-11	3.37e-10	4.00	4.01
mean			4.12	4.07

DOF	$\|\|\rho_2\|\|_2$	$\|\|\rho_2\|\|_\infty$	EOC_2	EOC_∞
16	2.03e-04	9.02e-04	—	—
32	1.14e-05	5.01e-05	4.16	4.17
64	5.46e-07	2.83e-06	4.38	4.15
128	3.36e-08	1.82e-07	4.02	3.96
256	2.04e-09	1.09e-08	4.04	4.07
512	1.27e-10	6.73e-10	4.00	4.01
mean			4.12	4.07

Table 5.8: EOC for the conservative variables using the entropy-stable DGSEM with polynomial degree $N_p = 3$ and different degrees of freedom in a single dimension for the multi-component ideal MHD equations.

5.2.2 Mass, Momentum, Energy and Entropy Conservation

Since we do not only want to use the multi-component Euler equations but also want to calculate applications with the multi-component ideal MHD equations, we have to show the correctness of our method for these equations as well. Since the ideal MHD equations are basically just an extension of the Euler equations, we have to show here again the conservation properties of our method for mass, momentum, energy, and entropy while using our fourth-order entropy conservative DGSEM. However, we would like to briefly mention here that the ideal GLM-MHD equations no longer represent a conservation law.

Setup

Similar to the Euler case, we work again with a weak shock wave testcase, which is just stable enough to not break but can still prove the conservation properties of our method. The initial condition is basically an ideal GLM-MHD adapted version of the initial conditions of the Euler equations in the previous section which look as follows

$$(\rho, p, v_1, v_2, v_3, B_1, B_2, B_3, \psi, Y_1, Y_2)$$

$$= \begin{cases} (1.1691, 1.245, 0.1882\phi_c, 0.1882\phi_s), 0.0, 1.0, 1.0, 1.0, 0.0, \frac{1}{3}, \frac{2}{3}), & r \leq 0.5 \\ (1.0, 1.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 0.0, \frac{1}{3}, \frac{2}{3}), & r > 0.5 \end{cases}$$

with $\phi_c = \cos(\arctan(x_2/x_1))$, $\phi_s = \sin(\arctan(x_2/x_1))$ and the radius $r = |\vec{x}|_2 = \sqrt{x_1^2 + x_2^2}$ to the center of the blast wave while we use the following parameters regarding the components

$$\begin{aligned} (\gamma_1, \gamma_2) &= (2.0, 4.0) \\ \left(\tilde{R}_1, \tilde{R}_2\right) &= (1.0, 2.0)\,. \end{aligned} \tag{5.3}$$

The simulation takes place in the spatial domain $\Omega = [-2, 2] \times [-2, 2]$ using periodic boundary conditions.

Solver

We start the simulation with a CFL condition of $CFL = 0.5$ and the GLM speed $GLM = 0.5$ for the temporal domain $T = [0.0, 2.0]$. Again, we use the five-stage, fourth-order low-storage method of Carpenter and Kennedy for the

time integration as well as the fourth-order entropy-conservative DGSEM with polynomial degree $N_p = 3$ for the spatial integration.

Result

We begin with the mass and momentum conservation, where we calculate the difference in total mass and total momentum accumulated over the entire domain at time $T_{end} = 2.0$ as well as at time $T_{start} = 0.0$ which lies within machine precision, see table 5.9.

	ρv_1	ρv_2	ρv_3	ρ_1	ρ_2	ρe
$\lvert\Delta U\rvert$	3.2E-17	1.8E-17	7.6E-18	5.3E-14	1.1E-13	3.6E-16

Table 5.9: Difference in accumulated total mass, momentum, and energy over the whole domain at time $T_{end} = 2$ to the accumulated total mass, momentum, and energy over the entire domain at time $T_{start} = 0$ in 2D calculated with the fourth-order DGSEM and $DOF = 128^2$ degrees of freedom.

Since we want to verify that our scheme really is entropy-conservative or entropy-stable, we analyse the total entropy change throughout the simulation over the entire domain. For an entropy-conservative scheme, we do not expect the entropy to change during the simulation, whereas for an entropy-stable scheme we do expect an entropy change, see table 5.10.

	EC	ES
$\lvert\Delta S\rvert$	9.9E-19	7.1E-05

Table 5.10: Total entropy change over the entire domain at time $T_{end} = 2$ in 2D calculated with the fourth-order entropy-conservative DGSEM and the entropy-stable DGSEM using $DOF = 128^2$ degrees of freedom.

5.3 Shock Capturing Scheme

In the following section we will show that our shock capturing scheme is able to ensure robust simulations in a shock dominated environment and demonstrate

that the entropy stabilization property yields an even more robust simulation.

5.3.1 Sod's shock-tube problem

One should assume that a test case, which is widely used as a benchmark for the single-component Euler equations should also be a useful test case for the multi-component Euler equations as long as the test case gets adapted properly. A quick and easy way to adapt the single-component case to the multi-component case is to simply define the left side density and the right side density as individual components in the multi-component system and equip them with different heat capacity ratios. This kind of Sod's shock-tube for multi-component Euler equations has already been used in several publications [7, 91] with the objective to compare different numerical methods to a reference solution. Sod's shock-tube is a highly appreciated 1D Riemann problem for Euler equations, since it is able to clearly visualize three different characteristics of the system, namely the rarefaction wave, the contact discontinuity, and the shock discontinuity.

Setup

The initial conditions of this example are given by

$$
\begin{aligned}
&(\rho, p, v, Y_L, Y_R) \\
&= \begin{cases} (1, 1, 0, 1, 0), & x \leq 0.5 \qquad \text{(left side)} \\ (0.125, 0.1, 0, 0, 1), & x > 0.5 \qquad \text{(right side)} \end{cases}
\end{aligned}
$$

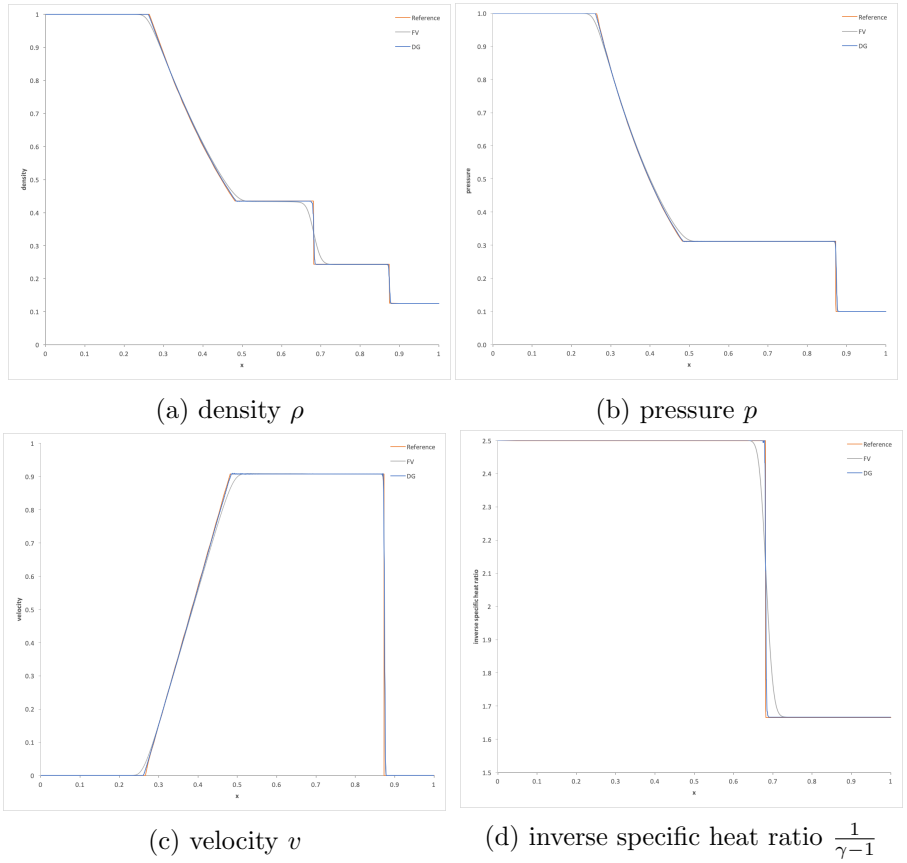
with the following parameters regarding the components

$$
\begin{aligned}
\left(\gamma_L, \gamma_R\right) &= (1.4, 1.6) \\
(\tilde{R}_L, \tilde{R}_R) &= (1, 1).
\end{aligned}
$$

On the boundary we set the initial solution using Dirichlet boundary conditions.

Solver

The goal of this example is to compare our fourth-order DGFV hybrid method to the basic first-order FV method as well as the reference solution. To have



(a) density ρ

(b) pressure p

(c) velocity v

(d) inverse specific heat ratio $\frac{1}{\gamma-1}$

Figure 5.1: Comparison of the fourth-order entropy-stable DGFV hybrid solution (blue) to the first-order FV solution (grey) and digitally extracted reference solution from Gouasmi et al. [7] (orange) at time $T_{end} = 0.2$ with $DOF = 2048$ degrees of freedom in space.

a fair comparison, we choose the same degrees of freedom for both methods, namely $DOF = 2048$. We use the stable but also very dissipative LLF surface flux for both methods as well as the entropy-conservative flux as volume flux for the DG hybrid method. Since this test case consists of shocks, contact discontinuities, and rarefraction waves, we are obliged to use the shock-capturing scheme for our DGSEM solution which is embedded into our DGFV hybrid method. All calculations have been made with a time step size constrained of

$CFL = 0.3$. The simulation is calculated to the endtime of $T_{end} = 0.2$.

Result

As can be seen, both methods, the FV method and the DGFV hybrid method, work quite well. It is obvious that the FV method is more dissipative than the DGFV hybrid method, despite using the same degrees of freedom, whereas the DGFV hybrid method is very much on point with the reference solution using this resolution. The overall picture clearly shows an accuracy advantage for the DGFV hybrid method compared to the first-order FV method for the same degrees of freedom.

5.3.2 Medium Blast Wave

The medium blast wave is a good example to test the shock capturing capabilities in a domain full of shocks used by Hennemann and Gassner [8] which we adapt to the multi-component case. It consists of a radially symmetric blast wave expanding from the center of a domain where a high amount of density and pressure is concentrated. The boundary line between the concentrated gas in the center and the ambient gas with a smaller density and pressure concentration can be described as a shock front which is expanding towards the outer boundary of the domain. Since the outer boundary conditions have been chosen to be periodic, the shock front reflects back into the domain and starts to form a complex shock pattern, which leads to several new shock interactions.

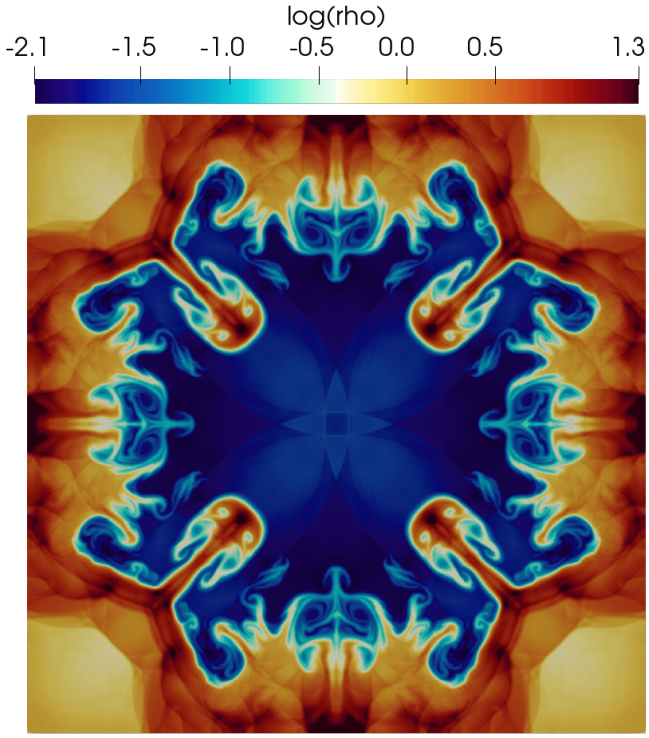


Figure 5.2: Distribution of the quantity $log(\rho)$ for the fourth-order entropy-stable DGSEM at time $T_{end} = 12.5$ using $DOF = 1024$ degrees of freedom in each spatial dimension resulting in overall $DOF = 1048576$ degrees of freedom.

Setup

For the initial condition, we choose the radius of the concentrated gas in the center to be $r = \sqrt{x_1^2 + x_2^2} \leq 0.5$, using the euclidic norm. Outside of this radius, we set the ambient gas to be at rest in both directions and set the ambient pressure and density to a low value. Inside, we set the gas to move towards the outside while having a slightly higher density concentration as well as a way higher pressure in this area. We set the simulation domain in space to $\Omega = [-2.0, 2.0]^2$. Now, we set the initial conditions to be

$$(\rho, p, v_1, v_2, Y_1, Y_2)$$

$$= \begin{cases} (1, 0.001, 0, 0, 0, 1), & r > 0.5 \quad \text{(ambient)} \\ (1.1691, 1.245, 0.1882\cos(\phi), 0.1882\sin(\phi), 1, 0), & r \leq 0.5 \quad \text{(center)} \end{cases}$$

with $\phi = \arctan(x_1^2 + x_2^2)$ and the following parameters regarding the components

$$\begin{aligned} \left(\gamma_1, \gamma_2\right) &= (1.2, 1.4) \\ (\tilde{R}_1, \tilde{R}_2) &= (0.2, 0.4). \end{aligned}$$

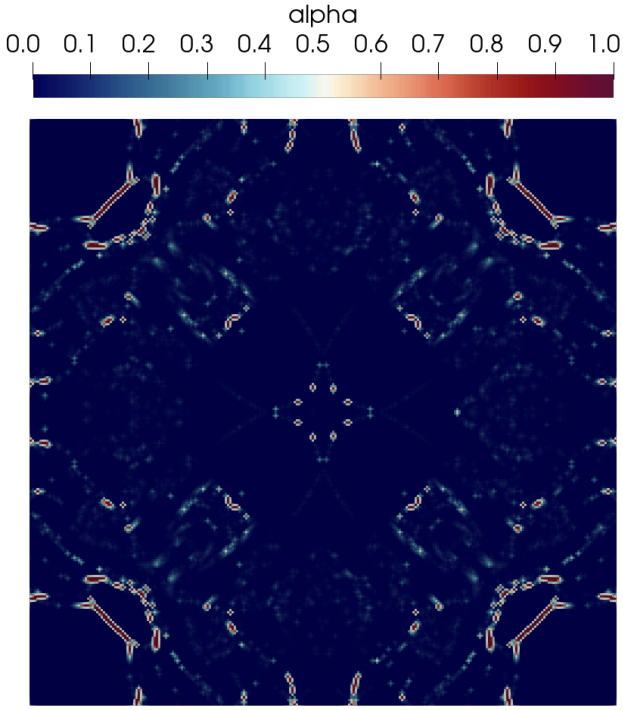


Figure 5.3: Distribution of the blending coefficient α for the fourth-order entropy-stable DGFV hybrid scheme at time $T_{end} = 12.5$ using $DOF = 1024$ degrees of freedom in each spatial dimension resulting in overall $DOF = 1048576$ degrees of freedom.

Solver

We set the simulation domain in time to $T = [0, 12.5]$. Moreover, the domain in space Ω is subdivided into $N_q = 256^2$ quadrilateral elements each representing the solution with a polynomial of degree $N_p = 3$ leading to a fourth-order scheme with $DOF = 1048576$ degrees of freedom in space, whereas in time we use the SSPRK54 scheme with a time step size constrained by $CFL = 0.8$.

As has been mentioned above, this simulation has to deal with a lot of shocks and is therefore not usable without a proper shock capturing scheme. To deal with this problem, we use our shock capturing scheme with a maximum blending coefficient of $\alpha_{max} = 0.5$ for the entropy-stable DGFV hybrid scheme. The reason for the relatively low maximum blending coefficient of $\alpha_{max} = 0.5$ lies in the increased robustness of the entropy-stable DGFV hybrid scheme which in turn leads to less dissipative solutions.

Result

Figure 5.2 shows the endsolution at time $T = 12.5$ for the entropy-stable DGFV hybrid scheme. In Figure 5.3 it can be seen that the entropy-stable DGFV hybrid scheme is able to stabilize the simulation with quite a small blending coefficient of $\alpha_{max} = 0.5$.

5.4 Positivity-Preserving Limiter

Although the entropy-stable DGFV hybrid scheme with shock-capturing is very robust in most applications, it can be necessary to have a positivity-preserving limiter in cases where densities or pressure are too close to zero and might become negative due to oscillations resulting from the high-order scheme. To eliminate these oscillations, it may be necessary to change the blending coefficient a posteriori to a value which guarantees positive densities and pressure. To show the benefits of this a posteriori limiter, we will simulate the Kelvin-Helmholtz instability without shock-capturing (which means we are using $\alpha = 0$ leading to a pure DGSEM method) and show that the standard DGFV hybrid scheme as well as the entropy-stable DGFV hybrid scheme are not able to run the simulation until the end. Afterwards, we will run the simulation again

with the positivity-preserving limiter activated and show that this a posteriori limiter is robust enough to keep the simulation alive until the end.

5.4.1 Kelvin-Helmholtz Instability

The following Kelvin-Helmholtz test case, taken from Rueda-Ramírez and Gassner [10] and adapted to the multi-component case, is a good example for the underlying high-order DGSEM, since it is quite challenging due to the very low resolution as well as due to the fact that the setup in inviscid with an effective Reynolds number of $Re = \infty$. Due to the fact that the initial flow is subsonic with a Mach number of $Ma \leq 0.6$, no shocks are developed whereas compressibility is still a relevant feature in the simulation. While pressure is uniformelly distributed across the entire domain, we set the density as well as the velocities varying across the domain. More precisely, we distribute the density in a way that generates several layers of different densities which move in opposite directions. For example, this phenomenon is known as a weather phenomenon, where two different layers of air are mixed due to different velocities and directions of the flow.

Setup

For the initial condition, we choose

$$(\rho, p, v_1, v_2, Y_1, Y_2) = \ (1 + \tfrac{3}{2}\phi, 1, \tfrac{1}{2}(\phi - 1), \tfrac{1}{10} sin(2\pi x, 0.5, 0.5)$$

with $\phi = \tanh(15x_2 + 7.5) - \tanh(15x_2 - 7.5)$ and the following parameters regarding the components

$$\begin{aligned} \left(\gamma_1, \gamma_2\right) &= (1.2, 1.67) \\ (\tilde{R}_1, \tilde{R}_2) &= (0.2, 4.0). \end{aligned}$$

The spatial domain is in 2D and has the size $\Omega = [-1.0, 1.0]^2$.

Solver

We want to run this example for quite a long time period to show the difference in stability for both, the standard and entropy-stable DGSEM, namely

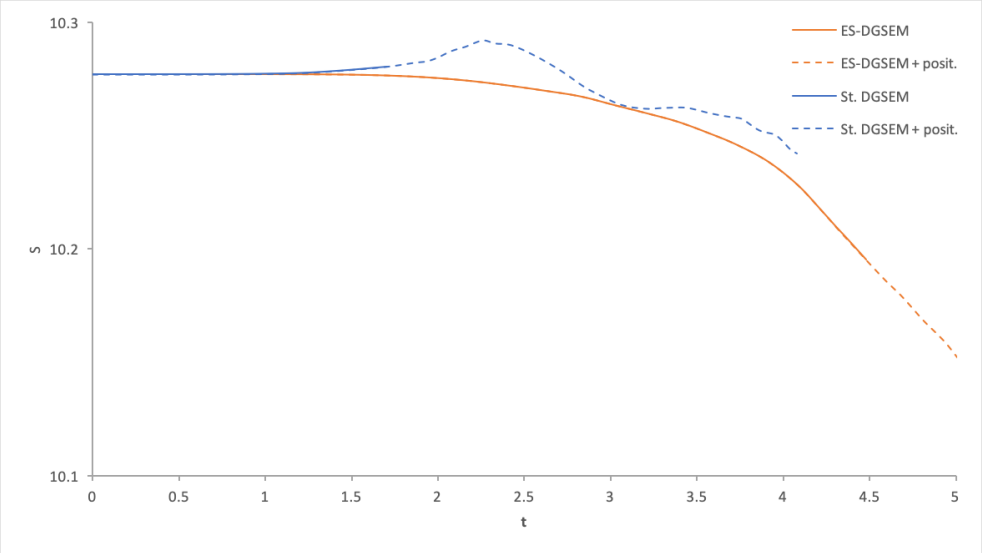


Figure 5.4: Zoomed in section of the total entropy evolution of the standard DGSEM (blue) and the entropy stable DGSEM (orange) over t=[0,5] in the entire simulation domain calculated without positivity-preserving (solid) and with positivity-preserving (dashed).

for $T = [0, 25]$ with a time step constraint of $CFL = 0.5$ and the SSPRK54 time integration scheme. Furthermore, we divide the domain into $N_q = 32^2$ quadrilateral elements and use a fourth-order DGSEM with a polynomial degree $N_p = 3$ which leads to $DOF = 16384$ degrees of freedom in space. First, we start the simulation without any positivity limiter or shock-capturing and track the entropy development for both of our schemes. Afterwards, both schemes are started again with the activated a posteriori positivity limiter. The surface flux for both frameworks is the LLF flux.

Result

As can be seen in Figure 5.4, the standard DGSEM as well as the entropy-stable DGSEM without positivity-preserving (see the continuous lines) abort before the end time $T = 25$. As expected, the total entropy over the whole domain

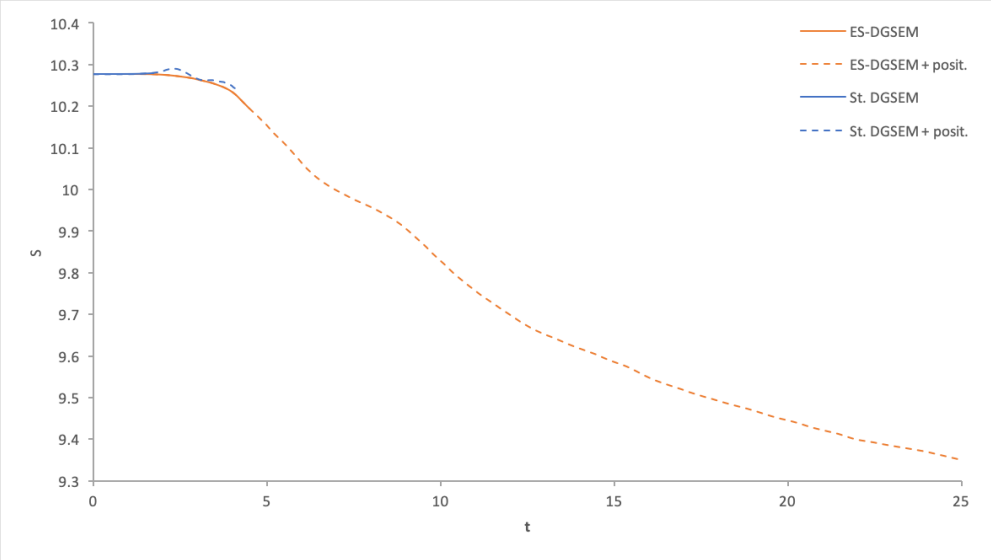


Figure 5.5: Total entropy evolution of the standard DGSEM (blue) and the entropy-stable DGSEM (orange) over time in the entire simulation domain for t=[0,25] calculated without positivity-preserving (solid) and with positivity-preserving (dashed).

starts to increase for the standard DGSEM, which is an unphysical property and therefore leads to an early termination of the simulation at time $t \approx 1.7$. If we compare this result with the entropy-stable DGSEM simulation, we can clearly see the advantage of the entropy stability property. The overall total entropy starts to decrease and is therefore able to carry the simulation about three times further until the termination time $t \approx 4.5$. Subsequently, we start the simulation again with the now activated positivity limiter and watch the simulations run further than before. The standard DGSEM is now able to run the simulation until $t \approx 4.0$ which is about two times further than without the positivity limiter, but still not as far as the entropy-stable DGSEM without the positivity limiter. When we observe the dashed line of the standard DGSEM, we can see that the simulation could be carried on although the total entropy has been increasing even more, which shows that the positivity limiter is able to keep the solution alive without switching to a full first-order scheme until $t \approx 2.2$. Shortly after, the positivity limiter has to use way more amount of the

FV method which leads to a decreasing total entropy but is not able to rescue the solution anymore, which might be the consequence of the prior unphysical behaviour of the scheme. On the contrary, the entropy-stable DGSEM with positivity limiter is able to run to the end of the simulation $t = 25$ while the total entropy is monotonic decreasing throughout the whole simulation as expected.

5.5 Adaptive Mesh Refinement

Despite increasingly powerful hardware, efficiency is still a hot topic in the world of fluid dynamics. A great tool to increase computational efficiency in simulations with huge areas where nothing special happens, is the adaptive mesh refinement (AMR). With AMR it is possible to refine areas where it is needed and coarsen the other areas instead of refining the whole domain. A good example to demonstrate these AMR capabilities is again a Kelvin-Helmholtz instability where huge areas do not have to be refined necessarily.

5.5.1 Kelvin-Helmholtz Instability

The following Kelvin-Helmholtz test case is based on Rueda-Ramírez and Gassner [10] and adapted to the multi-component case. It is a good example to show the AMR capabilities of our code, while using three different components. The overall dynamics of the simulation are similar to the previous one in section 5.3.1 but with a quite different initial condition.

Setup

The initial condition for this test case looks as follows

$$(\rho, p, v_1, v_2, Y_1, Y_2, Y_3)$$

$$= \begin{cases} (1, 2.5, -0.5, v_2, 0.1, 0.1, 0.8), & x_2 \geq 0.75 \quad \text{(top)} \\ (2, 2.5, 0.5, v_2, 0.05, 0.9, 0.05), & 0.25 < x_2 < 0.75 \quad \text{(center)} \\ (1, 2.5, -0.5, v_2, 0.8, 0.1, 0.1), & x_2 \leq 0.25 \quad \text{(bottom)} \end{cases}$$

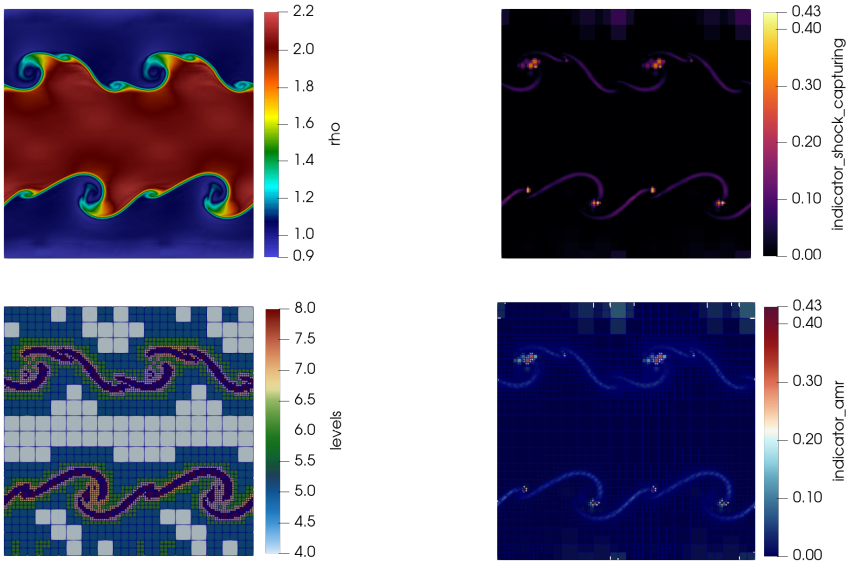


Figure 5.6: Kelvin-Helmholtz simulation using the entropy-stable DGSEM with AMR at $T_{end} = 0.8$. Showing the density distribution (top-left), shock-capturing indicator (top-right), mesh refinement levels (bottom-left), and amr indicator (bottom-right) calculated with overall $DOF = 16384$ degrees of freedom in space in the beginning.

with $v_2 = w_0 \cdot \sin(4\pi x_1) \cdot (\exp(-(x_2-0.25)^2/(2\sigma^2)) + \exp(-(x_2-0.75)^2/(2\sigma^2)))$, $w_0 = 0.1$ and $\sigma = \frac{0.05}{\sqrt{2}}$ and the following parameters regarding the components

$$\begin{aligned}(\gamma_1, \gamma_2, \gamma_3) &= (1.2, 1.4, 1.67)\\ (\tilde{R}_1, \tilde{R}_2, \tilde{R}_3) &= (0.2, 0.4, 4.0).\end{aligned}$$

The spatial domain has the size $\Omega = [0.0, 1.0]^2$.

Solver

We run this example with the entropy-stable DGSEM for $T = [0.0, 0.8]$ with a time step size condition of $CFL = 0.5$ and the SSPRK54 time integration scheme. Initially, we divide the domain into $N_q = 32^2$ quadrilateral elements and use a fourth-order DGFV hybrid method with a polynomial degree $N_p = 3$

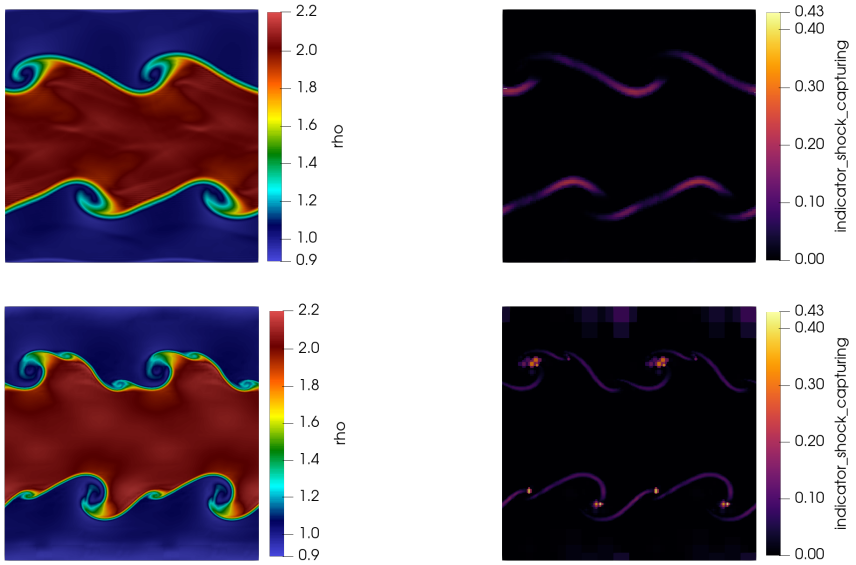


Figure 5.7: Entropy-stable DGFV hybrid method without AMR (top) and with AMR (bottom) at time $T_{end} = 0.8$ showing the density distribution (left) and the shock-capturing indicator (right). Those simulations have been calculated in roughly the same amount of time (≈ 1150 seconds).

which leads to $DOF = 16384$ degrees of freedom in space. Since we now use AMR, our simulation will choose where to refine further and where to coarsen the mesh with a maximal refinement level of $N_q = 256^2$ and a minimum refinement level of $N_q = 16^2$. The choosen surface flux is the LLF flux. Despite that, another simulation has been started without AMR with $N_q = 128^2$ which takes about an equal amount of time for the whole simulation, namely ≈ 1150 seconds.

Result

As can be seen in Figure 5.6, the shock-capturing indicator as well as the AMR indicator are fairly good at following the mixing layer of the components. Comparing this result to the same simulation without AMR, which takes roughly the same amount of time, it can be seen in Figure 5.7 that the AMR simulation

shows way more scales than the non AMR simulation.

5.6 Chemical Networks

When a chemical reaction takes place during a simulation, a component might lose some of its total mass and another one gains some. Therefore, chemical networks have to be seen as a kind of source term. To show that the chemical reaction network works properly, we will run a 1D simulation with five components and two chemical reactions and compare it to a reference solution taken from [84].

5.6.1 1D Detonation Waves

The following 1D detonation wave test case is a great example to verify that the chemical reaction networks work properly. It consists of five components and two chemical reactions. Both reactions have its own ignition temperature, which means that the chemical reactions only take place above different temperature thresholds. Not only does the first chemical reaction start earlier because of the lower temperature threshold, it also has a higher reaction rate leading to much faster reactions and therefore a faster component transfer. This stiff problem is quite challenging since it is not trivial to capture the correct speeds of all waves.

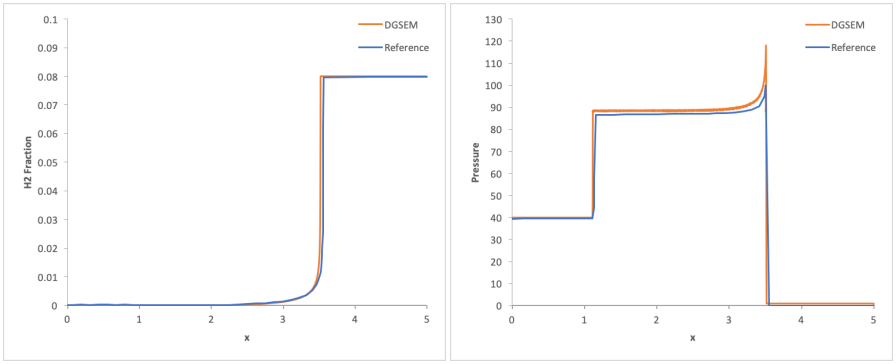
Setup

It follows, that the example consists of two one-way reactions

$$H_2 + O_2 \rightarrow 2OH, \tag{I}$$
$$2OH + H_2 \rightarrow 2H_2O, \tag{II}$$

with five components as in [83] and [84]. Thereby, the fifth component N_2 is used as a dilute catalyst. We set the initial conditions to be

$$(\rho, p, v, Y_{H_2}, Y_{O_2}, Y_{OH}, Y_{H_2O}, Y_{N_2}) = \begin{cases} (2, 40, 10, 0, 0, 0.17, 0.63, 0.2), & x \leq 0.5 \\ (1, 1, 0, 0.08, 0.72, 0, 0, 0.2), & x > 0.5 \end{cases}$$



(a) H_2 fraction y_{H_2} (b) pressure p

Figure 5.8: Comparison of the standard DGFV solution (orange) to the digital extracted reference solution (blue) from [84] for time $T_{end} = 0.25$ calculated with $DOF = 8192$ degrees of freedom in each spatial dimension.

with the following parameters regarding the reaction model and components

$$\left(\gamma_{H_2}, \gamma_{O_2}, \gamma_{OH}, \gamma_{H_2O}, \gamma_{N_2}\right) = (1.4, 1.4, 1.4, 1.4, 1.4) \tag{5.4}$$

$$\left(q_{H_2}, q_{O_2}, q_{OH}, q_{H_2O}, q_{N_2}\right) = (0, 0, -50, -100, 0) \tag{5.5}$$

$$\left(\tilde{R}_{H_2}, \tilde{R}_{O_2}, \tilde{R}_{OH}, \tilde{R}_{H_2O}, \tilde{R}_{N_2}\right) = (4.1242, 0.2598, 0.4, 0.4615, 0.2968) \tag{5.6}$$

$$\left(Da^I, T^I_{ign}\right) = (1 \times 10^5, 1.5) \tag{5.7}$$

$$\left(Da^{II}, T^{II}_{ign}\right) = (2 \times 10^4, 10). \tag{5.8}$$

The example is solved on the interval $\Omega = [0.0, 5.0]$ in space whereat the boundaries are set to be a solid wall.

Solver

This example has been run with the standard DGFV hybrid method and a time step size condition of $CFL = 0.8$ using the SSPRK54 time integration scheme for $T = [0.0, 0.25]$ in time. We divide the domain into $N_q = 2048$ quadrilateral elements and use a fourth-order DGFV hybrid method with a polynomial degree $N_p = 3$ which leads to $DOF = 8192$ degrees of freedom in space. Since we have to solve a chemical network additionaly, we also activate KROME using the chemistry callback implemented into Trixi.jl.

Result

As we can see in Figure 5.8, our scheme is able to capture all waves at correct speed. It looks even more sharp then the reference solution which might is mainly due to the inaccurate extraction of the reference data. Overall, we see that our reactive terms for the multi-component Euler equations solver work properly.

6 Applications

Now that we have proven the functionality and robustness of our method in the previous chapter, we want to simulate more applied and at the same time more complex examples. In this chapter, we will apply the entropy-conservative fluxes for the multi-component Euler and ideal MHD equations as well as chemical networks in conjunction with our DGFV hybrid method in several test examples. We will see that our high-order method can not only simulate these examples very accurately, but also gives very good results even in difficult situations.

We start with a typical shock-bubble interaction problem, which has already been studied in Gouasmi et al. [7] using an entropy-stable high-order method for the multi-component Euler equations. We will show that we can not only obtain comparable results with our method, but we can even increase the difficulty of the test example by letting the individual densities be close to zero without obtaining positivity-preserving issues. We then switch to a test example involving chemical reactions and show that our method also provides good and accurate results with these while our AMR and shock-capturing scheme is doing a good job. As a final multi-component reactive Euler use case, we will simulate a strong detonation with multi-step reactions and several different components and show that our high-order method also provides very good and accurate results here.

Subsequently, we will also apply our specially derived entropy-stable high-order method to the multi-component ideal GLM-MHD equations. Since the literature does not have many application-ready test cases for these equations, we will consider a specially created application example, which will also include chemical reactions.

6.1 Shock-Bubble Interaction

In this section, we will simulate the commonly [92, 93] used shock-bubble interaction problem for multi-component compressible flows and show the capabilities of our high-order entropy-stable DGFV hybrid method. This test case deals with an interaction of a shock wave with a cylindrical gas inhomogeneity, or in other words, a shock wave hits a helium-air bubble inside a tube with a inflow on the left boundary and an outflow on the right boundary whereas the top and bottom boundaries are reflective.

We will show that we are not only able to simulate this test case similar to Gouasmi et al. [7] with our high-order entropy-stable DGFV hybrid method, but that we are also able to simulate this test case without any modification of the initial condition due to the positivity-preserving properties of our method.

Setup

The shock-bubble interaction test case consists of two components, namely air and a helium-air mixture, leading to the following initial conditions

$$(\rho, p, v_1, v_2, Y_1, Y_2) = \begin{cases} (1.6861, 159060, -113.5243, 0, 1, 0), & x_1 > 2.75 \\ (1.225\frac{\tilde{R}_1}{\tilde{R}_2}, 101325, 0, 0, 0, 1), & r < 0.25 \\ (1.225, 101325, 0, 0, 1, 0), & else \end{cases} \tag{6.1}$$

with $r = ||\vec{x}||_2$ and the parameters

$$(\gamma_1, \gamma_2) = (1.4, 1.648) \tag{6.2}$$

$$(\tilde{R}_1, \tilde{R}_2) = (0.287, 1.578). \tag{6.3}$$

Therefore, we can choose our equations in the elixir file as:

```
05 equations = CompressibleEulerMulticomponentEquations2D(
06                            gammas = (1.4, 1.648),
07                            gas_constants = (0.287, 1.578),
08                            heat_of_formations = (0.0, 0.0))
```

and also write our initial condition function:

```
09 function initial_condition_shock_bubble(x, t,
10              equations::CompressibleEulerMulticomponentEquations2D)
11    inicenter = SVector(2.25, 0.0)
12    x_norm = x[1] - inicenter[1]
13    y_norm = x[2] - inicenter[2]
14    r = sqrt(x_norm^2 + y_norm^2)
15    delta = eps() #delta = 0.03
16    if x[1] > 2.75
17       rho1 = 1.6861 - delta
18       rho2 = delta
19       v1 = -113.5243
20       v2 = 0.0
21       p = 159060
22    elseif r < 0.25
23       rho1 = delta
24       rho2 = 1.225 * gas_constants[1]/gas_constants[2] - delta
25       v1 = 0.0
26       v2 = 0.0
27       p = 101325
28    else
29       rho1 = 1.225 - delta
30       rho2 = delta
31       v1 = 0.0
32       v2 = 0.0
33       p = 101325
34    end
35    return prim2cons(SVector(v1, v2, p, rho1, rho2), equations)
36 end
37 initial_condition = initial_condition_shock_bubble
```

The domain in space is $\Omega = [0.0, 4.45] \times [-0.445, 0.445]$ and $T = [0.0, 0.0067691]$ in time with reflective boundary conditions at the top and bottom boundary as well as inflow and outflow boundary conditions on the left and right boundary.

```
38 boundary_condition = BoundaryConditionDirichlet(initial_condition)
39 boundary_conditions = Dict( :x_neg => boundary_condition,
40                              :x_pos => boundary_condition,
41                              :y_neg => boundary_condition_slip_wall,
42                              :y_pos => boundary_condition_slip_wall)
```

Solver

Now that we know our setup, we can start to set our solver by using the fourth-order entropy-stable DGFV hybrid scheme with the local Lax-Friedrichs surface flux and the entropy-conservative volume flux by Gouasmi et al. [7]. We choose the shock-capturing indicator by Hennemann and Gassner with the shock-indicator variable being density times pressure.

```
43 surface_flux = flux_lax_friedrichs
44 volume_flux = flux_ec_gouasmi
45 polydeg = 3
46 basis = LobattoLegendreBasis(polydeg)
47 indicator_sc = IndicatorHennemannGassner(equations, basis,
48                                           alpha_max = 1.0,
49                                           alpha_min = 0.0,
50                                           alpha_smooth = true,
51                                           variable = density_pressure)
52 volume_integral = VolumeIntegralShockCapturingHG(indicator_sc;
53                     volume_flux_dg = volume_flux,
54                     volume_flux_fv = surface_flux)
55 solver = DGSEM(polydeg = polydeg, surface_flux = surface_flux,
56                volume_integral = volume_integral)
```

We also have to set the P4estMesh suitable to our spatial domain:

```
coordinates_min = (0.0, -0.445)
coordinates_max = (4.45, 0.445)
trees_per_dimension = (5, 1)
mesh = P4estMesh(trees_per_dimension,
                 polydeg=1, initial_refinement_level=8,
                 coordinates_min = coordinates_min,
                 coordinates_max = coordinates_max,
                 periodicity = false)
```

Now we can insert everything needed into our semidiscretization:

```
semi = SemidiscretizationHyperbolic(mesh, equations,
       initial_condition, solver,
       boundary_conditions = boundary_conditions)
```

Now we just need to hand over the time span to the DifferentialEquation.jl ODE problem:

```
tspan = (0.0, 0.0067691)
ode = semidiscretize(semi, tspan)
```

Additionally, we want to analyze and save our solution every few time steps:

```
summary_callback = SummaryCallback()
analysis_interval = 1000
analysis_callback = AnalysisCallback(semi,
                                     interval = analysis_interval)
alive_callback = AliveCallback(analysis_interval = analysis_interval)
save_solution = SaveSolutionCallback(interval = 1000,
                                     save_initial_solution = true,
                                     save_final_solution = true)
```

The last important callback is the stepsize callback where we set our CFL condition to $CFL = 0.8$:

```
78 stepsize_callback = StepsizeCallback(cfl = 0.8)
```

Then we collect all just now defined callbacks together:

```
79 callbacks = CallbackSet(summary_callback,
80                          analysis_callback,
81                          alive_callback,
82                          save_solution,
83                          stepsize_callback)
```

Finally we can start our simulation with our SSPRK54 time integration scheme:

```
84 sol = solve(ode, SSPRK54(), dt=1.0,
85             save_everystep = false, callback = callbacks);
86 summary_callback()
```

Result

Our goal was to test the entropy-conservative flux by Gouasmi et al. [7] by trying to reproduce the shock-bubble interaction test results. However, let us first compare the standard DGFV hybrid solution with the entropy-stable DGFV hybrid solution in Figure 6.1. As we can see, the pure first-order FV method is way more dissipative than the fourth-order DGFV hybrid method using the same degrees of freedom. We calculated the shock-bubble interaction problem exactly as stated in Gouasmi et al.[7] with a positivity-preserving coefficient, namely delta = 0.03, added to the species which would otherwise be zero and subtracted from the other species leading to the same amount of total density. By doing this, we receive pretty good results for the fourth-order DGFV hybrid method in Figure 6.3 similar to Gouasmi et al. [7]. This means that our entropy-stable fourth-order DGFV hybrid method works well for this particular test case. In contrast to the used method by Gouasmi et al., we are able to preserve the positivity of the partial densities as well as the positivity of pressure, so that we can also simulate this test case by setting the positivity preserving coefficient to machine precision with delta = eps. Although, this

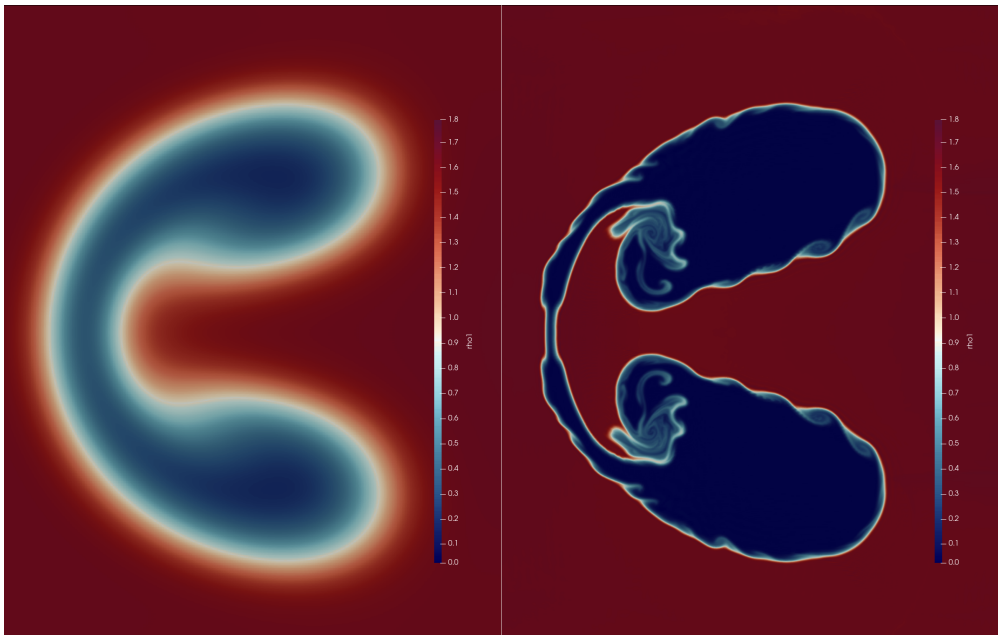


Figure 6.1: Comparison of the first density component ρ_1 for the first-order FV solution (left) and fourth-order entropy-stable DGFV hybrid solution (right) with the DGFV hybrid scheme.

simulation was not able to run stable with the chosen shock-capturing indicator variable, namely density times pressure, we were able to run the simulation with the much more intrusive shock-capturing indicator variable where we multiply both individual densities together with the pressure. As we see in Figure 6.2, we were able to run the test case with densities being near machine precision, however, some of the features that were still there in the previous simulation are no longer. This can be explained by the fact that the shock-capturing procedure with our chosen indicator was able to keep the simulation alive, but had to pay a high price for it. In short, this means that we had to do a lot of shock-capturing which led to a lot of reliance on the first-order FV method. One way to solve this problem is to find a suitable shock-capturing indicator variable that manages to keep the simulation alive while not switching too much to first-order, or in other words, being smarter about marking the cells that need to be blended.

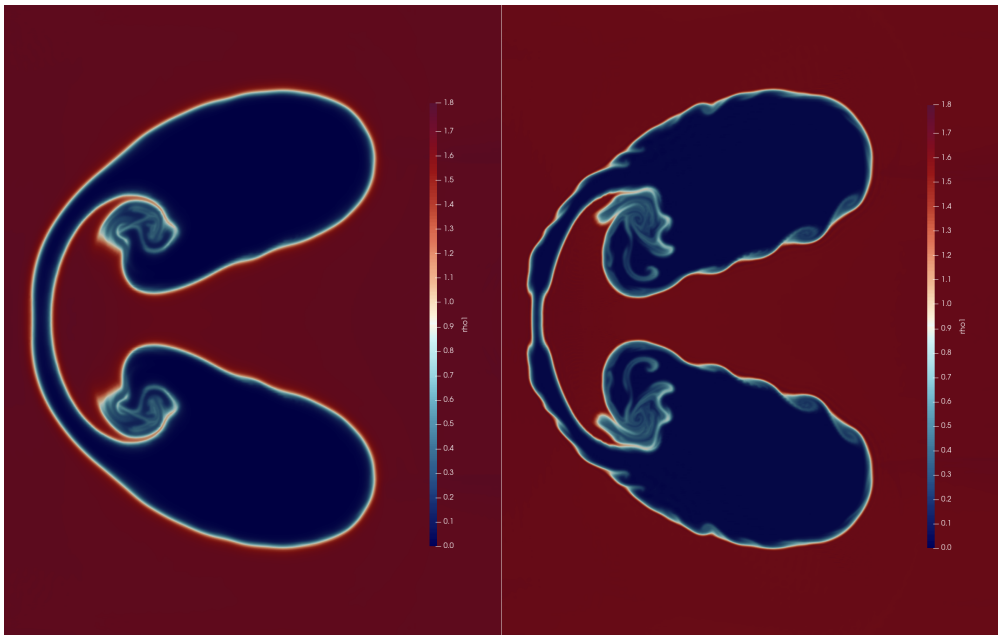


Figure 6.2: Comparison of the first density component ρ_1 for the entropy-stable DGFV hybrid method using the positivity-preserving coefficient delta = eps (left) and delta = 0.03 (right).

6.2 Detonation Diffraction Problems

In this section, we want to test our method for applications with chemical reactions and show that our high-order method does not break even with different geometries and diffraction angles. We will even use AMR, which, as we will see, works wonderfully for use cases like these. We will now study three similar but also slightly different detonation diffraction problems, which also have different complexities due to their different geometries/angles. These test cases have been used already in [94, 95].

The following use cases have a similar configuration, namely that all use the left edge as inflow while all other walls are set to be reflective. A shock wave now moves from the left edge to the right edge, and hits a sharp corner which has three different angles depending on the testcase. These sharp corners cause the

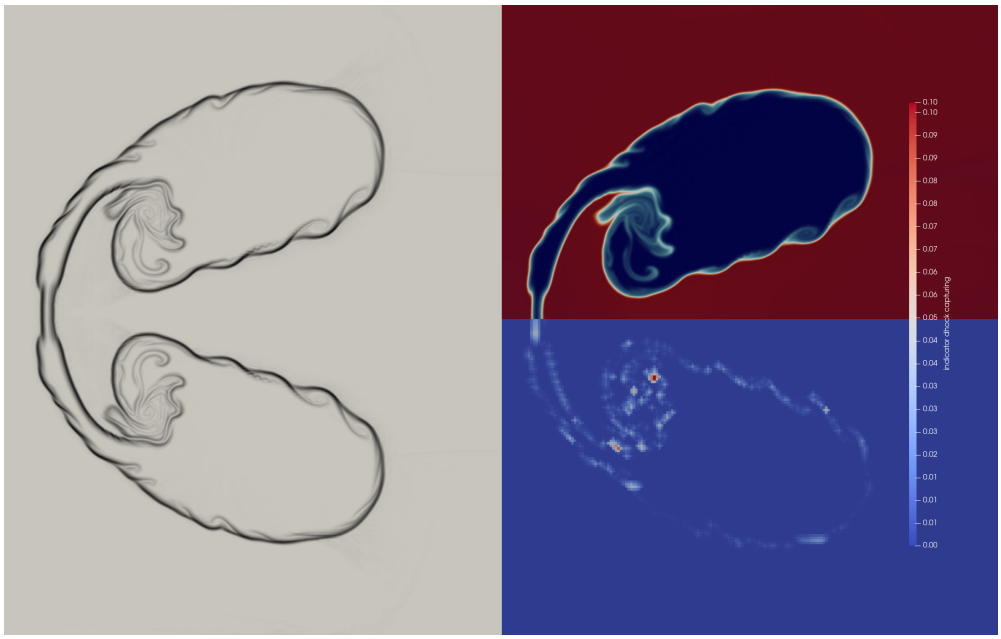


Figure 6.3: An entropy-stable DGFV hybrid method simulation of the shock-bubble interaction problem with positivity-preserving coefficient delta = 0.03. Showing a schlieren plot of the overall density (left), a plot of the first density ρ_1 (top-right) as well as the shock-capturing indicator (bottom-right).

shock wave to be diffracted, which can lead to positivity problems, especially with high-order methods. Another complexity is that the chemical reaction caused by the shock wave starts the production of a second component which our procedure must also master.

6.2.1 90 Degree Corner Diffraction

Let us now start with a grid containing a ninety degree corner. We start with one species which flows into the domain from the top left, which, depending on the temperature, starts to convert into another species. The critical points of this simulation are the shock capturing of the shock wave at the ninety degree corner as well as the chemical reaction mechanism. To be able to simulate such a test case in Trixi.jl, we also have to create a fitting mesh with *HOHQMesh*

which is able to run with the *P4estMesh* capability.

Setup

The example consists of one naive reaction

$$a \to b \tag{6.4}$$

with two components using the Arrhenius form with $\tilde{K} = 2566.4$. We set the initial conditions to be

$$(\rho, \rho e, v_1, v_2, Y_a, Y_b) = \begin{cases} (11, 970, 6.18, 0, 1, 0), & x < 0.5 \\ (1, 55, 0, 0, 1, 0), & x \geq 0.5 \end{cases} \tag{6.5}$$

with the following parameters regarding the reaction model and components

$$(\gamma_A, \gamma_B) = (1.2, 1.2) \tag{6.6}$$
$$(q_A, q_B) = (50, 0) \tag{6.7}$$

and the ignition temperature $\tilde{T} = 50$. The example is solved on the domain $\Omega = [0.0, 5.0] \times [0.0, 5.0] \setminus C$ with $C = [0.0, 1.0] \times [0.0, 2.0]$ in space and $T = [0.0, 0.6]$ in time. The boundary conditions are set to be reflective everywhere except on the upper left at $x = 0$ with

$$(\rho, \rho e, v_1, v_2, Y_a, Y_b) = (11, 970, 6.18, 0, 1, 0), \quad x = 0.0. \tag{6.8}$$

Therefore, we need to use our multi-component Euler equations with the included chemistry solver which we set up as follows:

```
05 equations = CompressibleEulerMulticomponentEquations2D(
06                            gammas = (1.2, 1.2),
07                            gas_constants = (0.287, 0.287),
08                            heat_of_formations = (50.0, 0.0))
```

stating the multi-component specific parameters as well as the chemical reaction parameters. Now we need to state the initial condition:

```
09 function initial_condition_diffraction_90(x, t,
10              equations::CompressibleEulerMulticomponentEquations2D)
11    if x[1] < 0.5
12       rho1 = 11.0
13       rho2 = eps()
14       rho = rho1 + rho2
15       rho_v1 = rho * 6.18
16       rho_v2 = rho * 0.0
17       rho_e = 970.0
18    else
19       rho1 = 1.0
20       rho2 = eps()
21       rho = rho1 + rho2
22       rho_v1 = rho * 0.0
23       rho_v2 = rho * 0.0
24       rho_e = 55.0
25    end
26    return SVector(rho_v1, rho_v2, rho_e, rho1, rho2)
27 end
28 initial_condition = initial_condition_diffraction_90
```

and set this function as initial condition. Since we know the boundary conditions, we have to define them following:

```
29 boundary_condition_dirichlet = BoundaryConditionDirichlet(
30                                initial_condition)
31 boundary_conditions = Dict( :B1 => boundary_condition_slip_wall,
32                              :B2 => boundary_condition_slip_wall,
33                              :B3 => boundary_condition_slip_wall,
34                              :B4 => boundary_condition_slip_wall,
35                              :B5 => boundary_condition_slip_wall,
36                              :B6 => boundary_condition_dirichlet)
```

with $B1, B2, \ldots, B6$ being the six walls we created with *HOHQMesh* (for more details we refer to section 4.4). We have to activate our chemical network (see section 4.3) by adding:

```
37 chemistry_term = chemical_reaction_network
```

into the elixir, which integrates the chemical network solver *KROME* into Trixi.jl.

Solver

Now we have to state our solver, which in our case will be the standard DGFV hybrid scheme with a LLF surface flux. We set the polynomial degree to $N_p = 3$, initialize the shock-capturing scheme with the indicator ρp and set our desired maximum and minimum alpha:

```
38 surface_flux = flux_lax_friedrichs
39 volume_flux = flux_central
40 polydeg = 3
41 basis = LobattoLegendreBasis(polydeg)
42 indicator_sc = IndicatorHennemannGassner(equations, basis,
43                                           alpha_max = 1.0,
44                                           alpha_min = 0.0,
45                                           alpha_smooth = true,
46                                           variable = density_pressure)
47 volume_integral = VolumeIntegralShockCapturingHG(indicator_sc;
48                    volume_flux_dg = volume_flux,
49                    volume_flux_fv = surface_flux)
50 solver = DGSEM(polydeg = polydeg, surface_flux = surface_flux,
51                volume_integral = volume_integral)
```

We also need to load our with *HOHQMesh* designed mesh file:

```
52 mesh_file =joinpath("examples/p4est_2d_dgsem","diffraction_90.inp")
53 mesh = P4estMesh{2}(mesh_file)
```

and wrap everything up by handing it over to the semidiscretization in Trixi.jl:

```
semi = SemidiscretizationHyperbolic(mesh, equations,
       initial_condition, solver,
       boundary_conditions = boundary_conditions,
       source_terms = nothing,
       chemistry_terms = chemistry_term)
```

The last thing our ODE problem needs is the time span with the end time $T_{end} = 0.6$:

```
tspan = (0.0, 0.6)
ode = semidiscretize(semi, tspan)
```

For analysis purposes, we set the following functions and callbacks:

```
summary_callback =SummaryCallback()
analysis_interval = 100
analysis_callback = AnalysisCallback(semi,
                                     interval = analysis_interval)
alive_callback = AliveCallback(analysis_interval = analysis_interval)
save_solution = SaveSolutionCallback(interval = 100,
                                     save_initial_solution = true,
                                     save_final_solution = true)
```

and start to initialize our AMR indicator:

```
amr_indicator = IndicatorHennemannGassner(semi,
                                          alpha_max = 1.0,
                                          alpha_min = 0.0,
                                          alpha_smooth = true,
                                          variable = density_pressure)
```

together with the AMR controller:

```
74 amr_controller = ControllerThreeLevel(semi, amr_indicator,
75                                       base_level = 0,
76                                       med_level = 1, med_threshold = 0.05,
77                                       max_level = 3, max_threshold = 0.1)
```

and AMR callback:

```
78 amr_callback = AMRCallback(semi, amr_controller,
79                            interval = 5,
80                            adapt_initial_condition = true,
81                            adapt_initial_condition_only_refine = true)
```

For a more detailed explaination we refer to section 4.5 which explains a quite similar example. Now we need to define the last two callbacks, namely the step size callback with a $CFL = 0.8$ and the chemistry callback:

```
82 stepsize_callback = StepsizeCallback(cfl = 0.8)
83 chemistry_callback = KROMEChemistryCallback())
```

Afterwards, we wrap all callbacks together:

```
84 callbacks = CallbackSet(summary_callback,
85                          analysis_callback,
86                          alive_callback,
87                          save_solution,
88                          amr_callback,
89                          stepsize_callback,
90                          chemistry_callback)
```

Last but not least, we set our ODE solver with our desired SSPRK54 time integration scheme as well as a dummy time step size which will be overwritten in the callbacks:

```
91 sol = solve(ode, SSPRK54(), dt=1.0,
92             save_everystep = false, callback = callbacks);
93 summary_callback()
```

Result

In the following, we show the solution computed with the fourth-order DGFV hybrid method as well as the first-order FV method of pressure and density. As we will see, the choice of the shock-capturing and AMR indicator influences the shock tracking for the DGFV hybrid simulations as we see in Figure 6.4 and Figure 6.5. Here, it can be seen that the shock indicator with the total

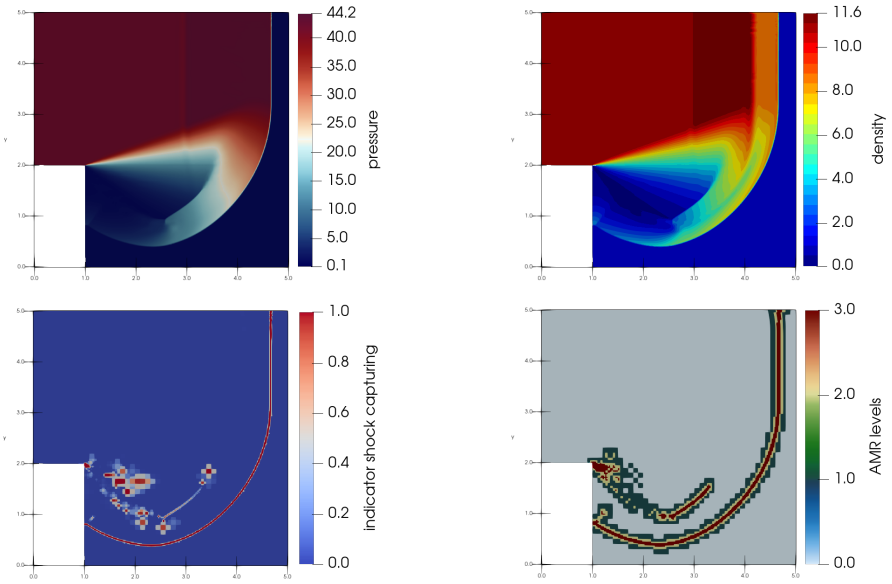


Figure 6.4: Ninety degrees detonation diffraction problem simulated with the fourth-order DGFV hybrid method. The top left and top right images show the pressure as well as the total density at the final time T = 0.6. The bottom left and bottom right images show the activation of the shock-capturing indicator (in this case density times pressure) as well as the active AMR level.

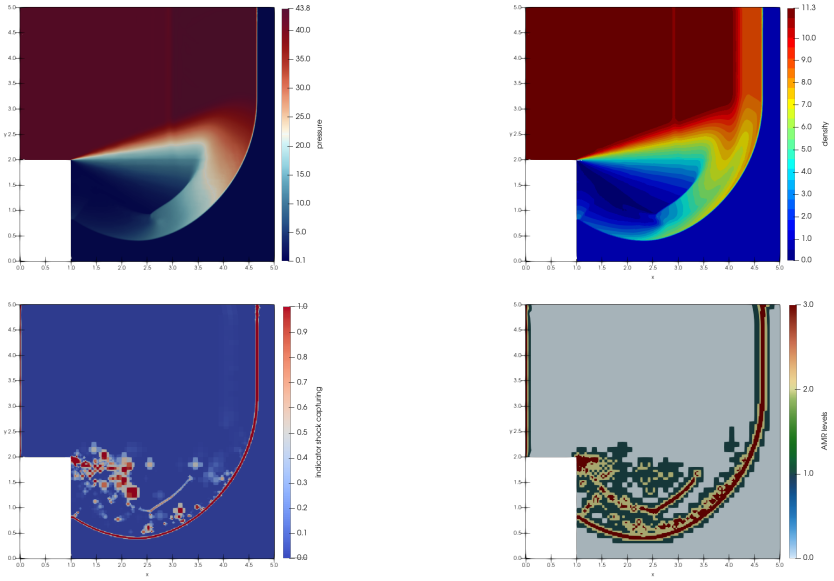


Figure 6.5: Ninety degrees detonation diffraction problem simulated with the fourth-order DGFV hybrid method. The top left and top right images show the pressure as well as the total density at the final time T = 0.6. The bottom left and bottom right images show the activation of the shock-capturing indicator (in this case multiplied individual densities times pressure) as well as the active AMR level.

density times pressure identifies the shocks more cleanly but leads to oscillations in some areas, while the somewhat more sensitive shock indicator with the multiplied individual densities times pressure does reduce the oscillations in this test case. Although the first indicator looks cleaner than the second indicator, it does not lead to similar accurate solutions. Therefore, we will continue to use the multiplied individual densities times pressure indicator for our detonation diffraction test cases below. We want to mention, that this indicator as well as the first mentioned indicator is not suitable for all test cases. The density times pressure indicator is particularly unsuitable if, for example, the total density is constant, but the individual densities differ greatly. An alternative here would be, for example, to use a single density times pressure as indicator.

Furthermore, we see an accuracy increase for the fourth-order DGFV hybrid method compared to the first-order FV method as can be seen in Figure 6.6.

This is even better illustrated in Figure 6.7, where the two methods are

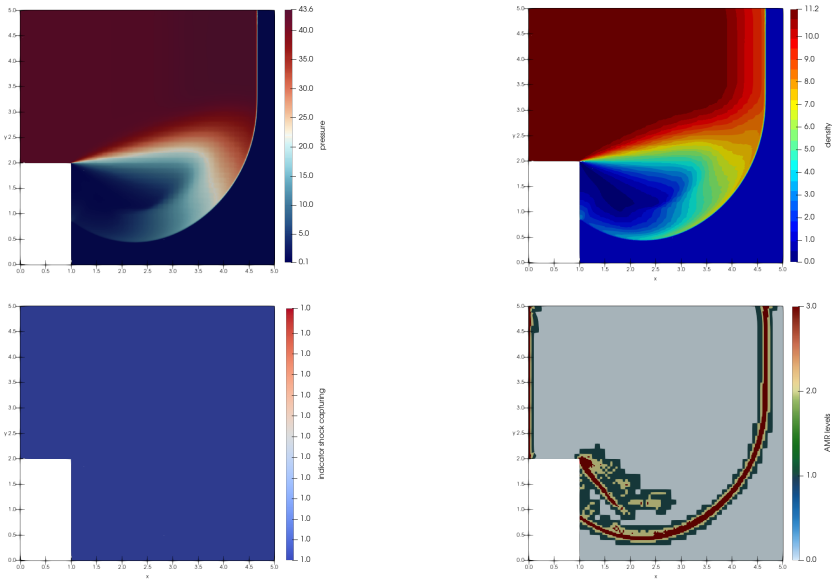


Figure 6.6: Ninety degrees detonation diffraction problem simulated with the first-order FV method. The top left and top right images show the pressure as well as the total density at the final time T = 0.6. The bottom left and bottom right images show the activation of the shock-capturing indicator (which is set to $\alpha = 1.0$ everywhere) as well as the active AMR level.

compared. The AMR levels show quite well that the rear part of the shock front in Figure 6.6 is dissipated so much that even the more sensitive AMR indicator does not have to be activated at this point. In general, the first order FV method appears to be more dissipative, as expected. For the sake of completeness, we would like to mention the mesh inprint, which is due to the subcell implementation of the FV method in Trixi.jl.

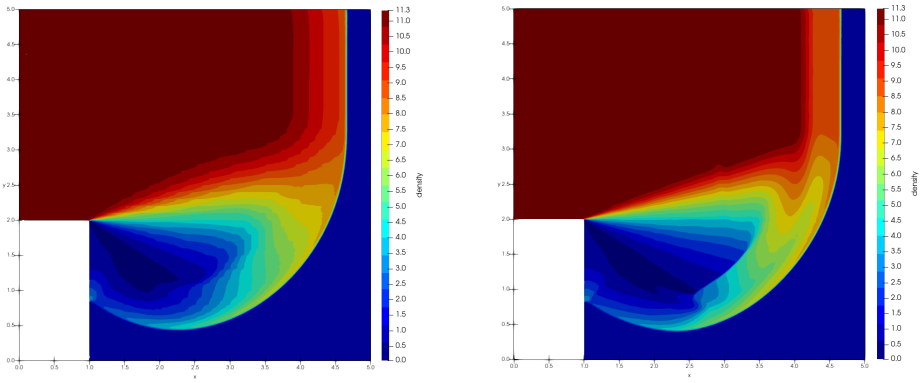


Figure 6.7: Comparison of the ninety degrees detonation diffraction problem simulated with the first-order FV method (left) and fourth-order DGFV hybrid method (right).

6.2.2 120 Degree Corner Diffraction

The one hundret and twenty degree corner diffraction is quite similar to the ninety degree corner example except for a few parameters. First, we need to build a new mesh which has a one hundret and twenty degree diffraction corner and which is twenty percent wider in x_1-direction as in the previous testcase. Afterwards, we are able to apply the adapted initial conditions as well as other parameters like the end time differing from the ninety degree testcase.

Setup

Since the domain size as well as the diffraction corner angle differ from the ninety degree detonation diffraction test case, we have to adapt the *HOHQMesh* file described in subsection 4.4.2 to look as follows:

```
\begin{END_POINTS_LINE}
   name = B1
   xStart = [0.0,2.0,0.0]
   xEnd = [1.15,2.0,0.0]
\end{END_POINTS_LINE}
\begin{END_POINTS_LINE}
   name = B2
   xStart = [1.15,2.0,0.0]
   xEnd = [0.0,0.0,0.0]
\end{END_POINTS_LINE}
\begin{END_POINTS_LINE}
   name = B3
   xStart = [0.0,0.0,0.0]
   xEnd = [6.0,0.0,0.0]
\end{END_POINTS_LINE}
\begin{END_POINTS_LINE}
   name = B4
   xStart = [6.0,0.0,0.0]
   xEnd = [6.0,5.0,0.0]
\end{END_POINTS_LINE}
\begin{END_POINTS_LINE}
   name = B5
   xStart = [6.0,5.0,0.0]
   xEnd = [0.0,5.0,0.0]
\end{END_POINTS_LINE}
\begin{END_POINTS_LINE}
   name = B6
   xStart = [0.0,5.0,0.0]
   xEnd = [0.0,2.0,0.0]
\end{END_POINTS_LINE}
```

Now, we have to adjust the initial conditions to be

$$(\rho, \rho e, v_1, v_2, Y_1, Y_2) = \begin{cases} (11, 970, 6.18, 0, 1, 0), & x_1 < 0.6 \text{ and } x_2 \geq 2.0 \\ (1, 55, 0, 0, 1, 0), & x_1 \geq 0.6 \text{ or } x_2 < 2.0 \end{cases} \tag{6.9}$$

which leads to the following adjustment of the elixir:

```
09 function initial_condition_diffraction_120(x, t,
10              equations::CompressibleEulerMulticomponentEquations2D)
11    if x[1] < 0.6 || x[2] >= 2.0
12       rho1 = 11.0
13       rho2 = eps()
14       rho = rho1 + rho2
15       rho_v1 = rho * 6.18
16       rho_v2 = rho * 0.0
17       rho_e = 970.0
18    else
19       rho1 = 1.0
20       rho2 = eps()
21       rho = rho1 + rho2
22       rho_v1 = rho * 0.0
23       rho_v2 = rho * 0.0
24       rho_e = 55.0
25    end
26    return SVector(rho_v1, rho_v2, rho_e, rho1, rho2)
27 end
28 initial_condition = initial_condition_diffraction_120
```

The simulation will run a bit longer since the end time will be $T_{end} = 0.68$:

```
59 tspan = (0.0, 0.68)
60 ode = semidiscretize(semi, tspan)
```

Solver

Due to the robustness of the DGFV hybrid scheme, we do not have to adjust the solver compared to the ninety degree example for this particular testcase. The only thing we need to change is the used mesh file:

```
52 mesh_file = joinpath("examples/p4est_2d_dgsem","diffraction_120.inp")
53 mesh = P4estMesh{2}(mesh_file)
```

Result

As we can see in Figure 6.8, the fourth-order DGFV hybrid method delivers a very sharp and accurate solution. Compared to the first order FV method

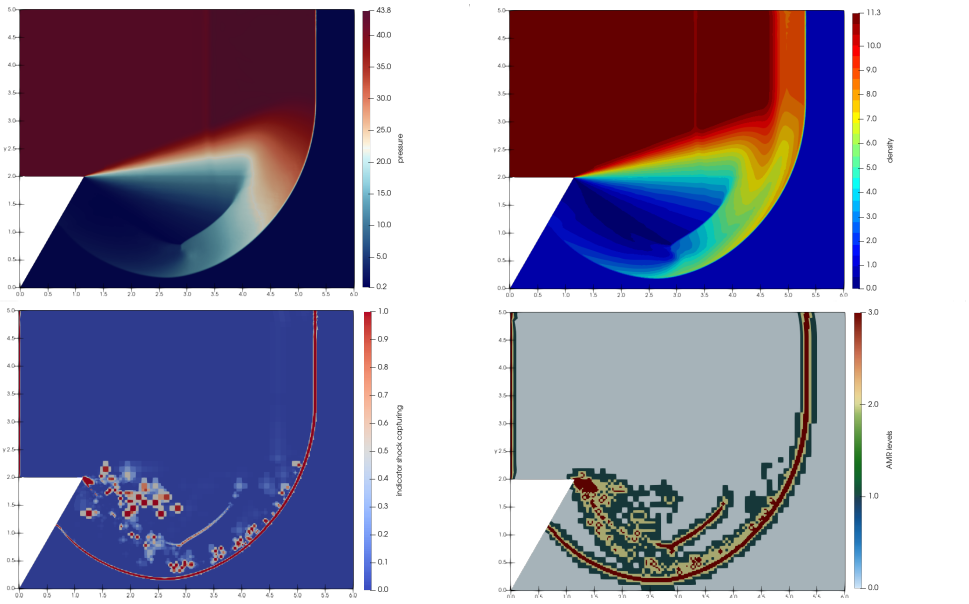


Figure 6.8: One hundret and twenty degrees detonation diffraction problem simulated with the fourth-order DGFV hybrid method. The top left and top right images show the pressure as well as the total density at the final time T = 0.68. The bottom left and bottom right images show the activation of the shock-capturing indicator (in this case multiplied individual densities times pressure) as well as the active AMR level.

in Figure 6.9, we obtain much more accurate solutions with the fourth-order DGFV hybrid method which are on par with the solutions in the literature [94, 95].

6.2.3 135 Degree Corner Diffraction

Another quite similar diffraction test case is the one hundret and thirty five degree corner diffraction. Again, we have to build a new mesh, which is forty percent wider than the mesh in the ninety degree example but with an even

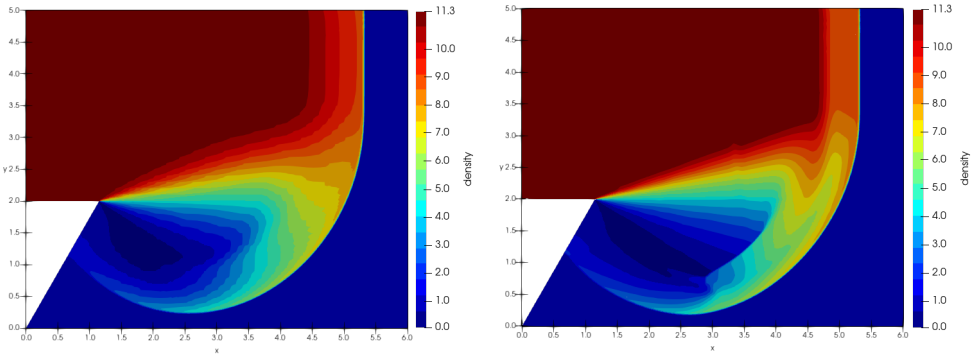


Figure 6.9: Comparison of the one hundret and twenty degrees detonation diffraction problem simulated with the first-order FV method (left) and fourth-order DGFV hybrid method (right).

more difficult diffraction corner of one hundret and thirty five degrees. Furthermore, the initial condition has to be adapted slightly due to the changes of the mesh geometry.

Setup

Again, the domain size as well as the diffraction corner angle differ from the ninety degree and the one hundret and twenty degree detonation diffraction test case, which means that we have to adjust the *HOHQMesh* file as follows:

```
\begin{END_POINTS_LINE}
   name = B1
   xStart = [0.0,2.0,0.0]
   xEnd = [2.0,2.0,0.0]
\end{END_POINTS_LINE}
\begin{END_POINTS_LINE}
   name = B2
   xStart = [2.0,2.0,0.0]
   xEnd = [0.0,0.0,0.0]
\end{END_POINTS_LINE}
   name = B3
   xStart = [0.0,0.0,0.0]
   xEnd = [7.0,0.0,0.0]
\end{END_POINTS_LINE}
\begin{END_POINTS_LINE}
   name = B4
   xStart = [7.0,0.0,0.0]
   xEnd = [7.0,5.0,0.0]
\end{END_POINTS_LINE}
\begin{END_POINTS_LINE}
   name = B5
   xStart = [7.0,5.0,0.0]
   xEnd = [0.0,5.0,0.0]
\end{END_POINTS_LINE}
\begin{END_POINTS_LINE}
   name = B6
   xStart = [0.0,5.0,0.0]
   xEnd = [0.0,2.0,0.0]
\end{END_POINTS_LINE}
```

Furthermore, we have to adjust the initial condition

$$(\rho, \rho e, v_1, v_2, Y_1, Y_2) = \begin{cases} (11, 970, 6.18, 0, 1, 0), & x_1 < 1.5 \text{ and } x_2 \geq 2.0 \\ (1, 55, 0, 0, 1, 0), & x_1 \geq 1.5 \text{ or } x_2 < 2.0 \end{cases} \tag{6.10}$$

which has to be adjusted in the elixir:

```
09 function initial_condition_diffraction_135(x, t,
10              equations::CompressibleEulerMulticomponentEquations2D)
11    if x[1] < 1.5 || x[2] >= 2.0
12       rho1 = 11.0
13       rho2 = eps()
14       rho = rho1 + rho2
15       rho_v1 = rho * 6.18
16       rho_v2 = rho * 0.0
17       rho_e = 970.0
18    else
19       rho1 = 1.0
20       rho2 = eps()
21       rho = rho1 + rho2
22       rho_v1 = rho * 0.0
23       rho_v2 = rho * 0.0
24       rho_e = 55.0
25    end
26    return SVector(rho_v1, rho_v2, rho_e, rho1, rho2)
27 end
28 initial_condition = initial_condition_diffraction_135
```

The simulation time is identical to the one hundret and twenty degree example.

Solver

Again, the DGFV hybrid method is robust enough to run this third example without the need to change any parameter of the solver. We just have to exchange the existing mesh file with the new mesh:

```
52 mesh_file = joinpath("examples/p4est_2d_dgsem","diffraction_135.inp")
53 mesh = P4estMesh{2}(mesh_file)
```

Result

Similar to the two other test cases, Figure 6.10 shows that we are able to produce at least similar solutions to the literature [94, 95]. Furthermore, the

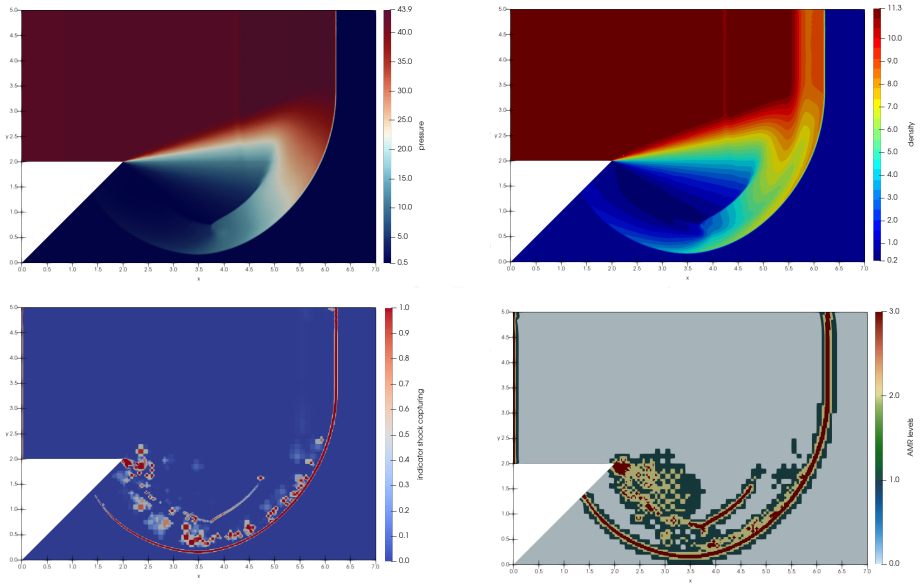


Figure 6.10: One hundret and thirty five degrees detonation diffraction problem simulated with the fourth-order DGFV hybrid method. The top left and top right images show the pressure as well as the total density at the final time T = 0.68. The bottom left and bottom right images show the activation of the shock-capturing indicator (in this case multiplied individual densities times pressure) as well as the active AMR level.

fourth-order DGFV hybrid method delivers again a way more accurate and less dissipative solution than the first-order FV method. The high-order method appears to have no oscillations or other disturbances and shines with sharp contours compared to the first order FV solution 6.10.

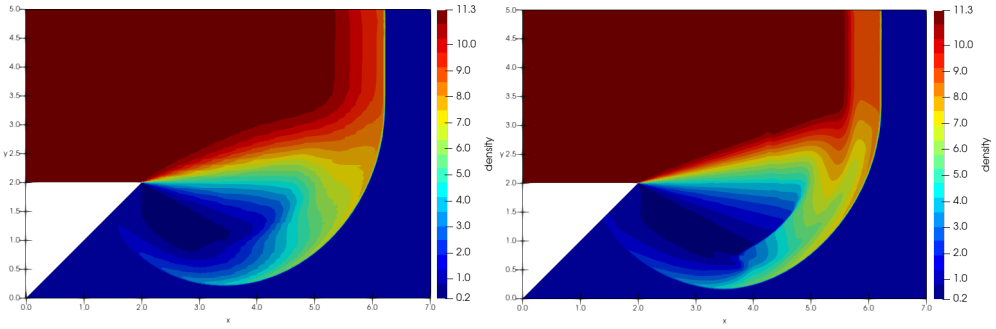


Figure 6.11: Comparison of the one hundret and thirty five degrees detonation diffraction problem simulated with the first-order FV method (left) and fourth-order DGFV hybrid method (right).

6.3 Strong Detonation with Multi-Step Reaction

In this example, we consider a two-dimensional asymmetric detonation problem with a multi-step reaction mechanism taken from [83, 84]. The domain is split into three parts (zone A, zone B and zone C) where, zone A and zone B consist of burnt gas (in this case OH and H_2O) and zone C consists of unburnt gas (H_2 and O_2). A fifth gas component, namely N_2, is used in all zones as a catalyst. During the simulation, a shock wave moves from left to right through the channel of width 6 and height 2. This shock wave sweeps through the contact surfaces AC between zone C and zone A as well as BC between zone C and zone B and initiates a chemical reaction chain. This means, that higher temperatures are caused by the shock wave, which leads to the start of a reaction of the two unburnt gas components in zone C resulting in the gas component OH which then leads to another reaction with H_2 and the production of gas component H_2O. Furthermore, the first reaction is not only way easier to activate due to the smaller ignition temperature, but also a much

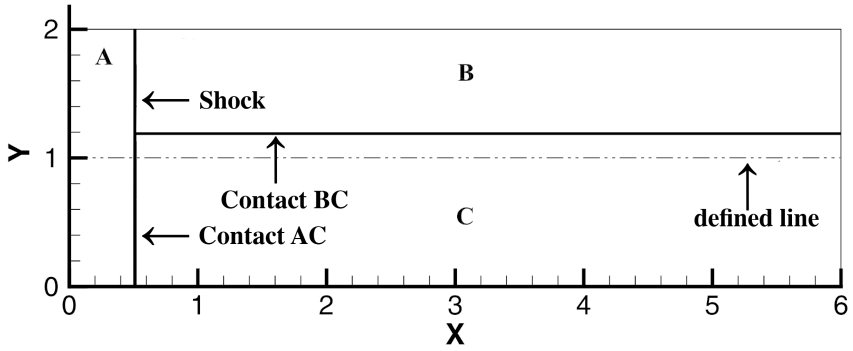


Figure 6.12: Computational domain split into three parts, zone A, zone B, and zone C with the contact surfaces AC and BC for the strong detonation with multi-step reaction test case adapted from [84].

faster reaction due to the higher activation energy than the second reaction.

Setup

It follows that the example consists of two one-way reactions

$$H_2 + O_2 \rightarrow 2OH, \tag{I}$$
$$2OH + H_2 \rightarrow 2H_2O, \tag{II}$$

with 5 components as in [83] or [84]. Thereby the fifth component N_2 is used as a dilute catalyst. We set the initial conditions to be

$$(\rho, p, v_1, v_2, Y_{H_2}, Y_{O_2}, Y_{OH}, Y_{H_2O}, Y_{N_2})$$
$$= \begin{cases} (2, 40, 10, 0, 0, 0.17, 0.63, 0.2), & x_1 \leq 0.5 \qquad \text{(zone-A)} \\ (1, 1, 0, 0, 0, 0, 0.17, 0.63, 0.2), & x_1 > 0.5, x_2 \geq 1.2 \qquad \text{(zone-B)} \\ (1, 1, 0, 0, 0.08, 0.72, 0, 0, 0.2), & x_1 > 0.5, x_2 < 1.2 \qquad \text{(zone-C)} \end{cases}$$

with the following parameters regarding the reaction model and components

$$\left(\gamma_{H_2}, \gamma_{O_2}, \gamma_{OH}, \gamma_{H_2O}, \gamma_{N_2}\right) = (1.4, 1.4, 1.4, 1.4, 1.4) \tag{6.11}$$

$$\left(q_{H_2}, q_{O_2}, q_{OH}, q_{H_2O}, q_{N_2}\right) = (0, 0, -50, -100, 0) \tag{6.12}$$

$$\left(W_{H_2}, W_{O_2}, W_{OH}, W_{H_2O}, W_{N_2}\right) = (2, 32, 17, 18, 28) \tag{6.13}$$

$$\left(Da^I, T^I_{ign}\right) = (1 \times 10^5, 2) \tag{6.14}$$

$$\left(Da^{II}, T^{II}_{ign}\right) = (2 \times 10^4, 10). \tag{6.15}$$

The example is solved on the interval $\Omega = [0.0, 6.0] \times [0.0, 2.0]$ in space and $T = [0.0, 0.1]$ in time whereat the upper and lower boundaries are set to be a solid wall.

As we have seen in the overview, the example consists of five components, which means that we have to use one of the multi-component equations as stated in chapter 2. Since we are not dealing with magnetic fields but have to solve a chemical network, we can calmly fall back on the multi-component Euler equations with the included chemistry solver:

```
05 equations = CompressibleEulerMulticomponentEquations2D(
06            gammas = (1.4, 1.4, 1.4, 1.4, 1.4),
07            gas_constants = (4.1242, 0.2598, 0.4891, 0.4615, 0.2968),
08            heat_of_formations = (0.0, 0.0, -50.0, -100.0, 0.0))
```

Now that we know all three zones of the domain with their associated values density, pressure, velocity, and mass fraction, we can start to build up the initial condition.

```
09 function initial_condition_strong_detonation(x, t,
10              equations::CompressibleEulerMulticomponentEquations2D)
11    if x[1] <= 0.5
12       v1 = 10.0
13       v2 = 0.0
14       p = 40.0
15       H2 = eps()
16       O2 = eps()
17       OH = 0.17 * 2.0
18       H2O = 0.63 * 2.0
19       N2 = 0.2 * 2.0
20    elseif x[1] > 0.5 && x[2] >= 1.2
21       v1 = 0.0
22       v2 = 0.0
23       p = 1.0
24       H2 = eps()
25       O2 = eps()
26       OH = 0.17
27       H2O = 0.63
28       N2 = 0.2
29    elseif x[1] > 0.5 && x[2] < 1.2
30       v1 = 0.0
31       v2 = 0.0
32       p = 1.0
33       H2 = 0.08
34       O2 = 0.72
35       OH = eps()
36       H2O = eps()
37       N2 = 0.2
38    end
39    prim_rho = SVector{5, real(equations)}(H2, O2, OH, H2O, N2)
40    prim_velocity_pressure = SVector{3, real(equations)}(v1, v2, p)
41    return prim2cons(vcat(prim_velocity_pressure,
                            prim_rho), equations)
42 end
43 initial_condition = initial_condition_strong_detonation
```

Now we set Dirichlet boundary conditions for the left and right boundary by using the predefined function *BoundaryConditionDirichlet*, which takes the initial condition as an input as well as wall boundary conditions for the top and bottom boundary by using the boundary function *boundary_condition_slip_wall* which already exists in Trixi.jl:

```
44 boundary_condition_dirichlet = BoundaryConditionDirichlet(
45                                 initial_condition)
46 boundary_conditions= Dict( :x_neg => boundary_condition_dirichlet,
47                             :x_pos => boundary_condition_dirichlet,
48                             :y_neg => boundary_condition_slip_wall,
49                             :y_pos => boundary_condition_slip_wall)
```

Additionally, we have to activate our chemical network:

```
50 chemistry_term = chemical_reaction_network
```

Solver

For this example, both the FV method as well as the DGFV hybrid method have been used. In the following, we will focus on how to setup the DGFV hybrid method for this example. First of all, we have to choose appropriate flux functions. In this case, we use the standard DGFV hybrid method with a central volume flux and a dissipative LLF surface flux. Due to the nature of DG methods, we can easily choose the order of the method which depends on the polynomial degree. In this instance, we use the polynomial degree of three which results in a fourth-order method. Given the fact that high-order methods lead to spurious oscillations near shocks, it is necessary to use a shock-capturing method. As described in chapter 3, we use a DGFV hybrid method which blends a stable first-order FV solution with the fourth-order DG solution. To do so, we first need to set a shock indicator which we choose to be the indicator of Hennemann and Gassner. Since this test case is highly unstable, we choose the parameter to be as stabilizing as possible by allowing the maximum blending of one hundret percent Finite Volume as well as propagating the blending factor to the neighboring elements. Experiments have shown, that the indicator variable

density times pressure works just fine for this test case. In the end, we have to choose the correct volume integral, which in this case will be the shock capturing volume integral by Hennemann and Gassner. Finally, we can set our solver to be the DGFV hybrid solver with our chosen polynomial degree, surface flux, and volume integral:

```
surface_flux = flux_lax_friedrichs
volume_flux = flux_central
polydeg = 3
basis = LobattoLegendreBasis(polydeg)
indicator_sc = IndicatorHennemannGassner(equations, basis,
                                         alpha_max = 1.0,
                                         alpha_min = 0.0,
                                         alpha_smooth = true,
                                         variable = density_pressure)
volume_integral = VolumeIntegralShockCapturingHG(indicator_sc;
                    volume_flux_dg = volume_flux,
                    volume_flux_fv = surface_flux)
solver = DGSEM(polydeg = polydeg, surface_flux = surface_flux,
               volume_integral = volume_integral)
```

To build the mesh, we first have to state the domain size:

```
coordinates_min = (0.0, 0.0)
coordinates_max = (6.0, 2.0)
```

Since we use *P4estMesh* in this example, which is based on a tree datastructure, we also have to state the number of trees in each dimension. To get nice quadrilateral elements, we can choose the following:

```
trees_per_dimension = (3, 1)
```

Now we can construct the *P4estMesh*, where we are allowed to choose the refinement level as well as the polynomial degree of the mesh. Since we use non periodic boundary conditions, we have to set this parameter to false:

```
68 mesh = P4estMesh(trees_per_dimension,
69              polydeg = 1,
70              initial_refinement_level = 9,
71              coordinates_min = coordinates_min,
72              coordinates_max = coordinates_max,
73              periodicity = false)
```

Now with all these modules, we can set the space discretization:

```
74 semi = SemidiscretizationHyperbolic(mesh,
75              equations,
76              initial_condition,
77              solver,
78              boundary_conditions,
79              source_terms=nothing,
80              chemistry_terms=chemistry_term)
```

For the time discretization, we still have to choose appropriate parameters like the time span and the CFL condition of the simulation:

```
81 tspan = (0.0, 0.1)
82 ode = semidiscretize(semi, tspan)
83 stepsize_callback = StepsizeCallback(cfl=0.8)
```

Since we have to solve the chemical network as an additional ODE, we have to call *KROME* with a specific callback:

```
84 chemistry_callback = KROMEChemistryCallback()
```

We then collect all callbacks we want to use:

```
85 callbacks = CallbackSet(summary_callback,
86               analysis_callback,
87               alive_callback,
88               save_solution,
89               stepsize_callback,
90               chemistry_callback)
```

Now we can choose an appropriate ODE solver to be the fourth-order SSP Runge-Kutta method:

```
91 sol = solve(ode, SSPRK54(),
92              dt=1.0,
93              save_everystep=false,
94              callback=callbacks,
95              maxiters=1e7);
```

For this example, both the FV method as well as the DGFV hybrid method have been used with the same degrees of freedom ($DOF = 6144 \times 2048 = 12582912$) resulting in a mesh of the size of $NQ = 6144 \times 2048 = 12582912$ quadrilateral elements on a uniform *P4estMesh* for the first-order FV method as well as $NQ = 1536 \times 512 = 786432$ quadrilateral elements for the fourth-order DGFV hybrid method and a CFL number of $CFL = 0.8$. For time integration, the *SSPRK54* method of the DifferentialEquations.jl package has been used. Although this example of a strong detonation can be classified to be pretty stiff, no positivity limiter was needed. Furthermore, a LLF surface flux as well as a standard central volume flux was used.

Result

In the following, we show a comparison of the solution computed with the first-order FV method as well as the fourth-order DGFV hybrid method for the quantities temperature and OH-fraction.

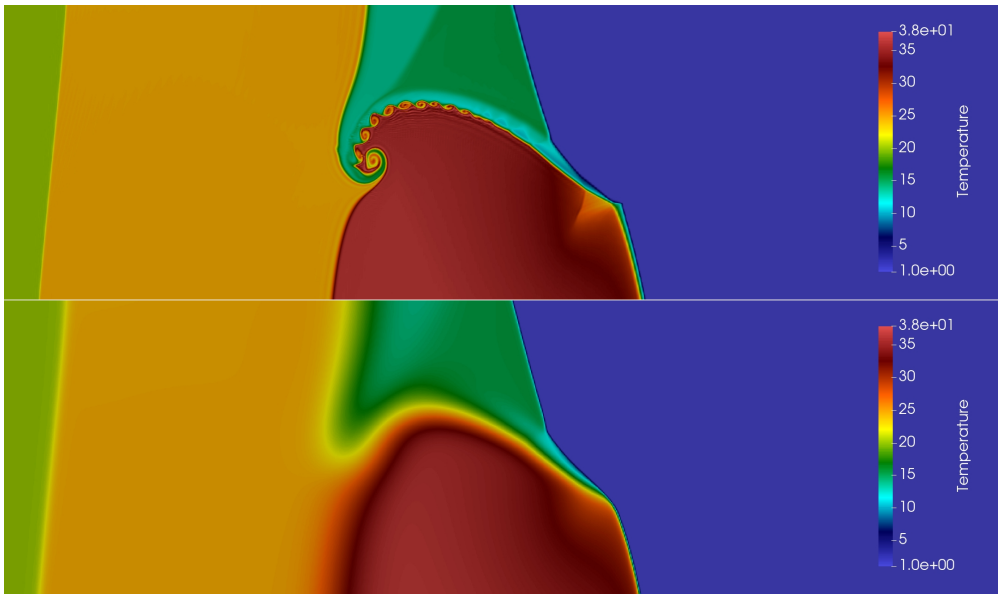


Figure 6.13: Zoom for temperature calculated with the fourth-order DGFV hybrid method (top) and first-order FV method (bottom).

As can be seen in Figure 6.13, both simulations look pretty similar due to the identical degrees of freedom whereby the DGFV hybrid method looks much sharper and produces Kelvin-Helmholtz like instabilities at the mixing layer. The same phenomenon can be observed in more detail for the OH-fraction in Figure 6.14. Moreover, the first-order FV method looks way more dissipative than the fourth-order DGFV hybrid scheme which does not seem to produce any kind of instabilities.

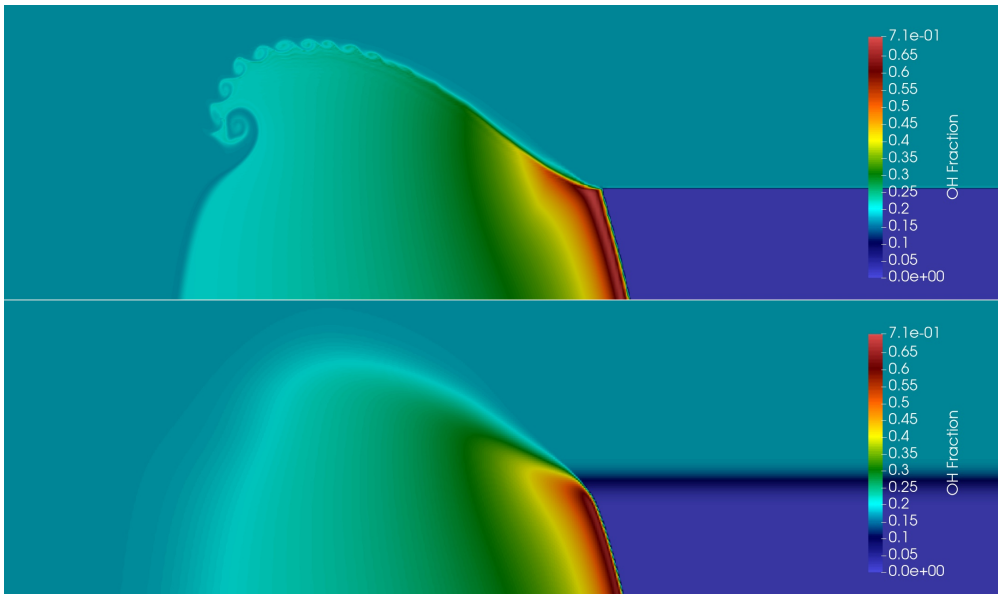


Figure 6.14: Zoom for OH-fraction calculated with the fourth-order DGFV hybrid method (top) and first-order FV method (bottom).

6.4 Reactive Multi-Component MHD Rotor Problem

As we have seen, our DGFV hybrid method already works quite well for applications based on the multi-component Euler equations. Now, we want to go one step further and apply our method to applications based on the multi-component magnetohydrodynamic equations. Since there are hardly any useful test cases in the literature to simulate ideal MHD equations with multi-components, we have taken a problem from [31], which was introduced by [96], and adapted it to a multi-component example. We want to show that our method also works well for ideal MHD equations with multi-components and with the addition of chemical reactions.

The test case consists of a dense disc embedded in a static magnetized homogeneous medium. Due to the rapid spinning of the disc, the magnetic field winds up which then leads to strong toroidal Alfvén waves which spread into the ambient fluid. We adapt this test case in a way, that the disc in the middle of the domain consists of one species with the actual gas properties as supposed in the single-component test case, whereby the ambient fluid is another species which consists of a helium-air mixture similar to the shock-bubble interaction problem. Finally, we add a chemical reaction that causes one component to slowly transform into the other component when a certain temperature is reached.

Setup

The multi-component ideal MHD rotor test case is pretty similar to the single-component case, except that we take one component for the disc and one component for the outer medium. We can write this down as the following initial condition

$$(\rho, p, v_1, v_2, v_3, B_1, B_2, B_3, \psi, Y_1, Y_2)$$

$$= \begin{cases} (10, 1, -20\Delta y, 20\Delta x, 0, \frac{5}{\sqrt{4}}, 0, 0, 0, 1, 0), & r \leq r_0 \\ (1 + 9f(r), 1, -20\Delta y f(r), 20\Delta x f(r), 0, \frac{5}{\sqrt{4}}, 0, 0, 0, 0.5, 0.5), & r \in (r_0, r_1) \\ (1, 1, 0, 0, 0, \frac{5}{\sqrt{4}}, 0, 0, 0, 0, 1), & r \geq r_1 \end{cases}$$

with $f(r) = \frac{r_1 - r}{r_1 - r_0}$, the radius $r = \sqrt{(x_1 - 0.5)^2 + (x_2 - 0.5)^2}$, the distances to the center$\Delta x_1 = x_1 - 0.5$, $\Delta x_2 = x_2 - 0.5$, the inner radius $r_0 = 0.1$, the outer

radius $r_1 = 0.115$ as well as the following parameters regarding the components

$$\left(\gamma_a, \gamma_b\right) = (1.4, 1.648) \tag{6.16}$$
$$(\tilde{R}_a, \tilde{R}_b) = (0.287, 1.578) \tag{6.17}$$
$$(q_a, q_b) = (0, 0) \tag{6.18}$$
$$(Da, T_{ign}) = (1.0, 1.0). \tag{6.19}$$

The test case is solved on the two-dimensional domain $\Omega = [0.0, 1.0] \times [0.0, 1.0]$ in space and $T = [0.0, 0.15]$ in time. The boundaries are set to be outflow everywhere. Additionally, the example consists of a one-way reaction mechanism

$$a \to b, \tag{6.20}$$

where the species inside the disc transforms to the species outside the disc when the ignition temperature is reached.

We start to create our elixir similar to the shock-bubble interaction test case but for the multi-component ideal GLM-MHD equations:

```
05 equations = IdealGlmMhdMulticomponentEquations2D(
06                            gammas = (1.4, 1.648),
07                            gas_constants = (0.287, 1.578),
08                            heat_of_formations = (0.0, 0.0))
```

We write an initial condition function based on the information above:

```
09 function initial_condition_rotor(x, t,
10                     equations::IdealGlmMhdMulticomponentEquations2D)
11     delta = 0.1
12     dx = x[1] - 0.5
13     dy = x[2] - 0.5
14     r = sqrt(dx^2 + dy^2)
15     f = (0.115 - r)/0.015
16     if r <= 0.1
17         rho1 = 10.0 - delta
18         rho2 = delta
19         v1 = -20.0*dy
20         v2 = 20.0*dx
21     elseif r >= 0.115
22         rho1 = delta
23         rho2 = 1.0 - delta
24         v1 = 0.0
25         v2 = 0.0
26     else
27         rho1 = 0.5 + 0.5*9.0*f
28         rho2 = 0.5 + 0.5*9.0*f
29         v1 = -20.0*f*dy
30         v2 = 20.0*f*dx
31     end
32     v3 = 0.0
33     p = 1.0
34     B1 = 5.0/sqrt(4.0*pi)
35     B2 = 0.0
36     B3 = 0.0
37     psi = 0.0
38     return prim2cons(SVector(v1, v2, v3, p, B1, B2, B3, psi,
                                    rho1, rho2), equations)
39 end
40 initial_condition = initial_condition_rotor
41 chemistry_term = chemical_reaction_network
```

and initialize our initial condition as well as our chemistry term.

Solver

In this example, the DGFV hybrid method has been used with an LLF flux for the surface flux as well as our entropy-conservative flux for the volume flux. We chose a polynomial degree of three $N_p = 3$ leading to a fourth-order method. Again, we chose the Hennemann-Gassner shock-indicator with the multiplied partial densities times pressure indicator variable:

```
42 surface_flux = (flux_lax_friedrichs, flux_nonconservative_powell)
43 volume_flux = (flux_ec, flux_nonconservative_powell)
44 polydeg = 3
45 basis = LobattoLegendreBasis(polydeg)
46 indicator_sc = IndicatorHennemannGassner(equations, basis,
47                                           alpha_max = 1.0,
48                                           alpha_min = 0.0,
49                                           alpha_smooth = true,
50                                        variable = densities_pressure)
51 volume_integral = VolumeIntegralShockCapturingHG(indicator_sc;
52                     volume_flux_dg = volume_flux,
53                     volume_flux_fv = surface_flux)
54 solver = DGSEM(polydeg = polydeg, surface_flux = surface_flux,
55                volume_integral = volume_integral)
```

For the mesh, we need to state the domain size as well as the right mesh type:

```
56 coordinates_min = (0.0, 0.0)
57 coordinates_max = (1.0, 1.0)
58 mesh = TreeMesh(coordinates_min, coordinates_max,
59 initial_refinement_level=6,
60 n_cells_max=1_000_000)
```

So that we can take all these modules and set the space discretization:

```
semi = SemidiscretizationHyperbolic(mesh,
             equations,
             initial_condition,
             solver,
             source_terms=nothing,
             chemistry_terms=chemistry_term)
```

After the space discretization, we also need to state the time discretization:

```
tspan = (0.0, 0.15)
ode = semidiscretize(semi, tspan)
```

For analysis purposes, we also initialize a few analysis callbacks:

```
summary_callback = SummaryCallback()
analysis_interval = 100
analysis_callback = AnalysisCallback(semi,
                                     interval = analysis_interval)
alive_callback = AliveCallback(analysis_interval = analysis_interval)
save_solution = SaveSolutionCallback(interval = 100,
                                     save_initial_solution = true,
                                     save_final_solution = true)
```

We also want to use AMR and therefore have to initialize our AMR indicator similar to our shock-capturing scheme:

```
amr_indicator = IndicatorHennemannGassner(semi,
                                          alpha_max = 1.0,
                                          alpha_min = 0.0,
                                          alpha_smooth = true,
                                          variable =
densities_pressure)
```

together with the AMR controller, where we choose refinement level 5 as the base level and refinement level 9 as the maximum refinement level:

```
82 amr_controller = ControllerThreeLevel(semi, amr_indicator,
83                   base_level = 5,
84                   max_level = 9, max_threshold = 0.1)
```

and the AMR callback:

```
85 amr_callback = AMRCallback(semi, amr_controller,
86                 interval = 5,
87                 adapt_initial_condition = true,
88                 adapt_initial_condition_only_refine = true)
```

The last callbacks are the stepsize callback, the GLM speed callback as well as the chemistry callback:

```
89 stepsize_callback = StepsizeCallback(cfl = 0.5)
90 glm_speed_callback = GlmSpeedCallback(glm_scale=0.5, cfl=cfl)
91 chemistry_callback = KROMEChemistryCallback())
```

Afterwards we wrap all callbacks together:

```
92 callbacks = CallbackSet(summary_callback,
93                          analysis_callback,
94                          alive_callback,
95                          save_solution,
96                          chemistry_callback,
97                          amr_callback,
98                          stepsize_callback,
99                          glm_speed_callback)
```

Last but not least, we set our ODE solver with our desired *SSPRK54* time integration scheme as well as a dummy time step size which will be overwritten in the callbacks:

```
100 sol = solve(ode, SSPRK54(), dt=1.0,
101             save_everystep = false, callback = callbacks);
102 summary_callback()
```

Result

In the following, we will compare results of the adapted multi-component GLM-MHD rotor test case with the reactive adapted multi-component GLM-MHD rotor test case. First of all, let us take a look at Figure 6.15 and compare how

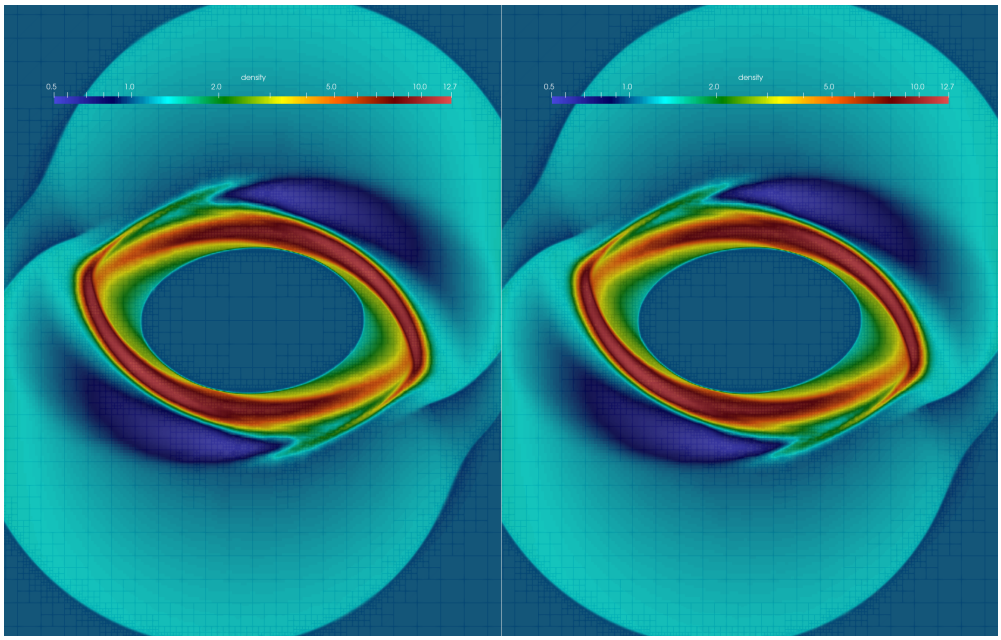


Figure 6.15: Comparison of the log scaled density for the adapted multi-component GLM-MHD rotor test case with a standard DGFV hybrid method (left) and the entropy-stable DGFV hybrid method (right).

the MHD rotor solution looks like for the multi-component test case with a

standard DGFV hybrid method compared to the entropy-stable DGFV hybrid method with our just derived entropy-conservative volume flux for the ideal GLM-MHD equations. Even though the solutions look similar, one can see slight advantages for the entropy-stable method, since it provides sharper edges as well as fewer artifacts. Second, we want to see if our scheme is able to run

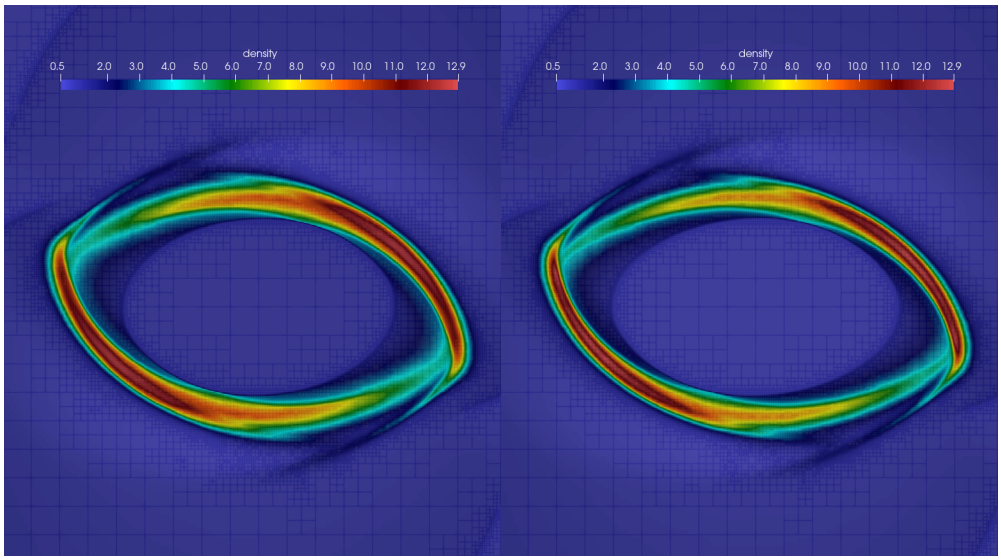


Figure 6.16: Comparison of the density for the adapted multi-component GLM-MHD rotor test case (left) calculated with the entropy-stable DGFV hybrid method and the adapted reactive multi-component GLM-MHD rotor test case (right) calculated with the standard DGFV hybrid method and reaction rate $Da = 1.0$.

a simulation together with a relatively weak chemical reaction. Therefore, we take a look at Figure 6.17 to compare the non-reactive multi-component solution to the reactive multi-component solution. Although slight differences are visible, the solution does not differ that much. This follows from the fact that the chemical reaction only starts at a temperature of 1 and the reaction rate is relatively low with K=1. Nevertheless, in this case our method is good enough to stabilize the simulation sufficiently and to provide an oscillation-free solution. Last, we want to compare the standard DGFV hybrid method with the entropy-stable hybrid method for this test case with activated reaction

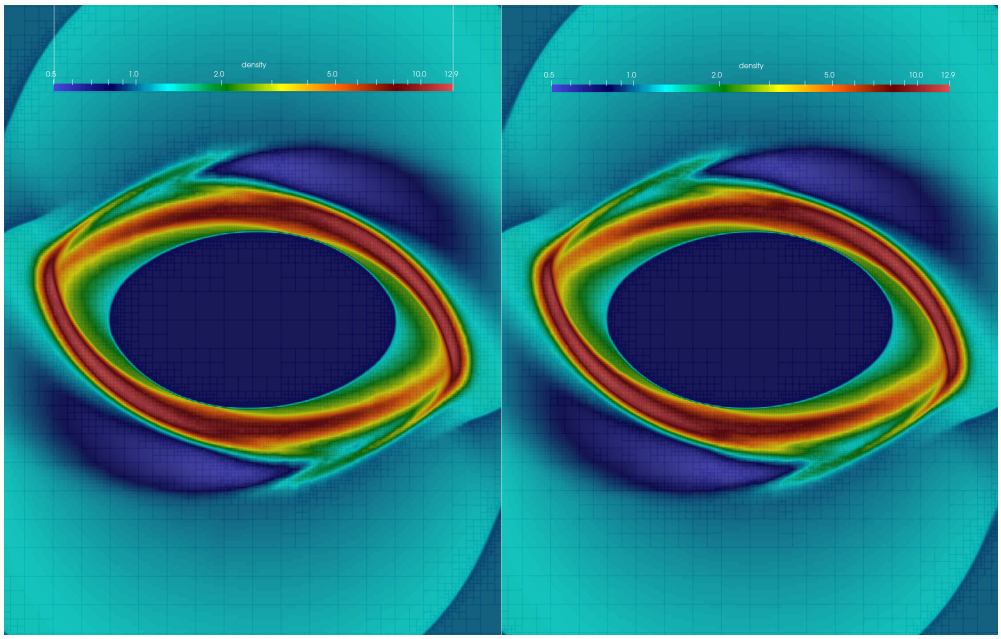


Figure 6.17: Comparison of the log-scaled density for the adapted multi-component GLM-MHD rotor test case (left) calculated with the standard DGFV hybrid method and reaction rate $Da = 1.0$, and the adapted reactive multi-component GLM-MHD rotor test case (right) calculated with the entropy-stable DGFV hybrid method and reaction rate $Da = 1.0$.

terms and reaction rate $Da = 1.0$. We can see that the entropy-stable results look more favorable, since it is better in resolving structures and contains fewer artifacts.

7 Conclusion

With this work, we have pursued the goal of developing an entropy-stable high-order discontinuous Galerkin spectral element method for multi-component equations such as the multi-component Euler as well as the multi-component ideal GLM-MHD equations. It was essential that it can handle hard shocks, is able to maintain the positivity of the individual densities as well as the pressure, even when these partial densities are close to machine precision or even chemical reactions are taking place. For this purpose, we first discussed the special properties of the equations in chapter 2 and then, based on these findings, developed our method in chapter 3. We have incorporated several features into our numerical method to solve the special problems of our underlying equations, such as the shock-capturing blending scheme or the positivity preserving limiter, to name a few.

Since we also want to show how research can be made accessible to others, one of the difficulties was to commit to a specific simulation framework and to make the existing resources and data structures usable for our purposes. Often, there is no way around the fact that certain features have to be integrated into the simulation framework that are essential for our own work. This can lead to some additional work, but the extra features can be useful for other users and developers of the simulation framework. Therefore, in chapter 4 we have focused our attention on the simulation framework and briefly explained the individual packages used. Here, one will notice that it can also be an advantage to make ones work available to others, since we were also using other peoples packages as we could see for example with the package DifferentialEquations.jl. or KROME.jl

Now, that the main ideas have been settled, we could begin with the validation of our method in chapter 5. The scheme we developed was put through its paces, with each integrated feature having to pass a functional test. After this necessary and important chapter was successfully completed, we could finally

get down to business. We were able to put the capabilities of our method to the test by calculating complex use cases in chapter 6.

7.1 Accomplishments

In order to reflect the success of this work, we will now revisit the research questions posed at the beginning and explain whether we were able to solve them.

(1) How well does the high-order DGFV hybrid method perform for multi-component equations applications?

As expected, and seen in chapter 6, we were able to achieve very good and accurate results with our high-order DGFV method that could not be achieved at the same resolution or with the same number of degrees of freedom with the first-order FV method. In addition, with the blending method used, in most cases there is no need to worry about the robustness of the method, which is a major advantage for the high-order DGFV hybrid method.

(2) Is it possible to construct an entropy-stable high-order DG method for the multi-component ideal MHD equations?

As we have seen in the publication of Gouasmi et al. [7], it is possible to determine an appropriate entropy-conservative flux for the multi-component Euler equations using a suitable entropy choice. Based on this entropy choice, we were able to determine an entropy-conservative flux for the multi-component ideal GLM-MHD equations and verified this flux in chapter 5.

(3) How well does the high-order DG method perform when you allow chemical networks in addition to the multi-component equations?

Although chemical networks bring an additional complexity to the simulation that can lead to problems especially with high-order methods, we could show in chapter 5 and chapter 6 that our high-order method is robust enough to perform these simulations. Due to the lower dissipation and thus sharper resolution of the high-order DG method, it is also easy to keep the correct propagation of detonation waves.

(4) What do we need to add to the simulation framework Trixi.jl so that we can use it for our thesis?

Although the simulation framework Trixi.jl already contained some useful features and procedures that have been useful for our work, it was still necessary to integrate some additional features into the framework. On the one hand, the Euler equations or ideal MHD equations already existed in Trixi.jl, on the other hand, the multi-component variants of these did not yet exist. Additionally, there was already a shock-capturing method for cartesian grids, but had to be extended also for curvilinear grids. Furthermore, a positivity-preserving limiter by Zhang and Shu already existed, but we additionally integrated the positivity-preserving limiter by Rueda-Ramírez and Gassner and adapted it to multi-component equations. As a last innovation, we have also integrated an external ODE solver for chemical networks called KROME, which could then simply be added to the elixirs.

7.2 Conclusion and Outlook

We conclude that the DGFV hybrid method copes very well with the multi-component equations and chemical networks. It is very robust and achieves superior results compared to low-order methods. Moreover, it is possible to derive entropy-conservative fluxes for the multi-component equations, with which it is possible to obtain an entropy-stable DGFV hybrid method and thus even more robust simulations and better.

Even though this work shows what the DGFV hybrid scheme is already capable of, there are still some areas that could be worked on. For example, one could look for a better variable for the shock-capturing indicator. A desired variable might allow us to run simulations more robustly while activating the blending only when it is indispensable. This way it would be possible to use less of the first-order scheme and further improve the accuracy of the scheme.

Based on this, it would also be possible to exchange the first-order FV scheme for a second-order FV scheme. Since blending the first-order solution sometimes seems too dissipative, this idea would lead to a drastic increase in accuracy without losing much in terms of robustness.

Currently KROME.jl is only a wrapper of the actual code written in Fortran. Therefore, the workflow is a bit more cumbersome than it should be. An idea would be to include the chemical reaction solver by another package or Julia intern library, which might improve the workflow of Trixi.jl.

List of Figures

List of Tables

Erklärung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Dissertation selbstständig und ohne die Benutzung anderer als der angegebenen Hilfsmittel und Literatur angefertigt habe. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten und nicht veröffentlichten Werken dem Wortlaut oder dem Sinn nach entnommen wurden, sind als solche kenntlich gemacht. Ich versichere an Eides statt, dass diese Dissertation noch keiner anderen Fakultät oder Universität zur Prüfung vorgelegen hat; dass sie - abgesehen von unten angegebenen Teilpublikationen und eingebundenen Artikeln und Manuskripten - noch nicht veröffentlicht worden ist sowie, dass ich eine Veröffentlichung der Dissertation vor Abschluss der Promotion nicht ohne Genehmigung des Promotionsausschusses vornehmen werde. Die Bestimmungen dieser Ordnung sind mir bekannt. Darüber hinaus erkläre ich hiermit, dass ich die Ordnung zur Sicherung guter wissenschaftlicher Praxis und zum Umgang mit wissenschaftlichem Fehlverhalten der Universität zu Köln gelesen und sie bei der Durchführung der Dissertation zugrundeliegenden Arbeiten und der schriftlich verfassten Dissertation beachtet habe und verpflichte mich hiermit, die dort genannten Vorgaben bei allen wissenschaftlichen Tätigkeiten zu beachten und umzusetzen. Ich versichere, dass die eingereichte elektronische Fassung der eingereichten Druckfassung vollständig entspricht.

Teilpublikationen:

Christof Martin Czernik

Digital unterschrieben von Christof Martin Czernik
Datum: 2022.12.05 11:58:09 +01'00'

Dormagen, den 18.09.2022

Ort, Datum

Unterschrift

Curriculum Vitae

Persönliche Daten:

Name	Christof Martin Czernik
Geburtstag	19.11.1993
Geburtsort	Dormagen, Deutschland
Nationalität	Deutsch

Bildungsweg:

2019-2022	Promotionsstudium in Mathematik Universität zu Köln
2017-2019	Masterstudium in Wirtschaftsmathematik Universität zu Köln
2013-2017	Bachelorstudium in Wirtschaftsmathematik Universität zu Köln
2010-2013	Allgemeine Hochschulreife Bertha-von-Suttner Gesamtschule Dormagen

Bibliography

[1] Jan S Hesthaven. *Numerical methods for conservation laws: From analysis to algorithms*. SIAM, 2017.

[2] Chi-Wang Shu. "High-order finite difference and finite volume WENO schemes and discontinuous Galerkin methods for CFD". In: *International Journal of Computational Fluid Dynamics* 17.2 (2003), pp. 107–118.

[3] David Gottlieb and Chi-Wang Shu. "On the Gibbs phenomenon and its resolution". In: *SIAM review* 39.4 (1997), pp. 644–668.

[4] Mark H Carpenter and Travis C Fisher. "High-order entropy stable formulations for computational fluid dynamics". In: *21st AIAA Computational Fluid Dynamics Conference*. 2013, p. 2868.

[5] Travis C Fisher and Mark H Carpenter. "High-order entropy stable finite difference schemes for nonlinear conservation laws: Finite domains". In: *Journal of Computational Physics* 252 (2013), pp. 518–557.

[6] Travis C Fisher, Mark H Carpenter, Jan Nordström, Nail K Yamaleev, and Charles Swanson. "Discretely conservative finite-difference formulations for nonlinear conservation laws in split form: Theory and boundary conditions". In: *Journal of Computational Physics* 234 (2013), pp. 353–375.

[7] Ayoub Gouasmi, Karthik Duraisamy, and Scott M Murman. "Formulation of Entropy-Stable schemes for the multicomponent compressible Euler equations". In: *Computer Methods in Applied Mechanics and Engineering* 363 (2020), p. 112912.

[8] Sebastian Hennemann, Andrés M Rueda-Ramirez, Florian J Hindenlang, and Gregor J Gassner. "A provably entropy stable subcell shock capturing approach for high order split form DG for the compressible Euler equations". In: *Journal of Computational Physics* 426 (2021), p. 109935.

[9] Ranocha, Hendrik, Schlottke-Lakemper, Michael, Winters, Andrew R, Faulhaber, Erik, Chan, Jesse, and Gassner, Gregor J. "Adaptive numerical simulations with Trixi. jl: A case study of Julia for scientific computing". In: *arXiv preprint arXiv: 2108.06476* (2021).

[10] Andrés M Rueda-Ramirez and Gregor J Gassner. "A subcell finite volume positivity-preserving limiter for DGSEM discretizations of the Euler equations". In: *arXiv preprint arXiv:2102.06017* (2021).

[11] Randall J LeVeque and Randall J Leveque. *Numerical methods for conservation laws*. Vol. 214. Springer, 1992.

[12] Jan S Hesthaven and Tim Warburton. *Nodal discontinuous Galerkin methods: algorithms, analysis, and applications*. Springer Science & Business Media, 2007.

[13] Eleuterio F Toro. *Riemann solvers and numerical methods for fluid dynamics: a practical introduction*. Springer Science & Business Media, 2013.

[14] Michael S Mock. "Systems of conservation laws of mixed type". In: *Journal of Differential equations* 37.1 (1980), pp. 70–88.

[15] Peter Lax and Burton Wendroff. "Systems of conservation laws". In: *Communications on Pure and Applied mathematics* 13.6 (1960), pp. 217–237.

[16] Eitan Tadmor. "The numerical viscosity of entropy stable schemes for systems of conservation laws. I". In: *Mathematics of Computation* 49.179 (1987), pp. 91–103.

[17] Peter D Lax. *Hyperbolic systems of conservation laws and the mathematical theory of shock waves*. SIAM, 1973.

[18] Peter D Lax. "Hyperbolic systems of conservation laws II". In: *Communications on pure and applied mathematics* 10.4 (1957), pp. 537–566.

[19] Ulrik Skre Fjordholm. *High-order accurate entropy stable numercial schemes for hyperbolic conservation laws*. ETH Zurich, 2013.

[20] Peter D Lax. *Hyperbolic partial differential equations*. Vol. 14. American Mathematical Soc., 2006.

[21] Niklas Wintermeyer. "A novel entropy stable discontinuous Galerkin spectral element method for the shallow water equations on GPUs". PhD thesis. Universität zu Köln, 2018.

[22] Randall J LeVeque et al. *Finite volume methods for hyperbolic problems*. Vol. 31. Cambridge university press, 2002.

[23] Sergei Konstantinovich Godunov. "A difference scheme for numerical solution of discontinuous solution of hydrodynamic equations". In: *Math. Sbornik* 47 (1959), pp. 271–306.

[24] Vladimir Vasil'evich Rusanov. *Calculation of interaction of non-steady shock waves with obstacles*. NRC, Division of Mechanical Engineering, 1962.

[25] Peter D Lax. "Weak solutions of nonlinear hyperbolic equations and their numerical computation". In: *Communications on pure and applied mathematics* 7.1 (1954), pp. 159–193.

[26] Philip L Roe. "Approximate Riemann solvers, parameter vectors, and difference schemes". In: *Journal of computational physics* 43.2 (1981), pp. 357–372.

[27] Amiram Harten, Peter D Lax, and Bram van Leer. "On upstream differencing and Godunov-type schemes for hyperbolic conservation laws". In: *SIAM review* 25.1 (1983), pp. 35–61.

[28] Claus-Dieter Munz, Michael Dumbser, and Sabine Roller. "Linearized acoustic perturbation equations for low Mach number flow with variable density and temperature". In: *Journal of Computational Physics* 224.1 (2007), pp. 352–364.

[29] Andreas Dedner, Friedemann Kemm, Dietmar Kröner, C-D Munz, Thomas Schnitzer, and Matthias Wesenberg. "Hyperbolic divergence cleaning for the MHD equations". In: *Journal of Computational Physics* 175.2 (2002), pp. 645–673.

[30] Wenlong Dai and Paul R Woodward. "On the divergence-free condition and conservation laws in numerical simulations for supersonic magnetohydrodynamical flows". In: *The Astrophysical Journal* 494.1 (1998), p. 317.

[31] Dominik Derigs, Andrew R Winters, Gregor J Gassner, Stefanie Walch, and Marvin Bohm. "Ideal GLM-MHD: about the entropy consistent nine-wave magnetic field divergence diminishing ideal magnetohydrodynamics equations". In: *Journal of Computational Physics* 364 (2018), pp. 420–467.

[32] Rémi Abgrall. "How to prevent pressure oscillations in multicomponent flow calculations: a quasi conservative approach". In: *Journal of Computational Physics* 125.1 (1996), pp. 150–160.

[33] Rémi Abgrall and Smadar Karni. "Computations of compressible multifluids". In: *Journal of computational physics* 169.2 (2001), pp. 594–623.

[34] Bernard Larrouturou. "How to preserve the mass fractions positivity when computing compressible multi-component flows". In: *Journal of computational physics* 95.1 (1991), pp. 59–84.

[35] Smadar Karni. "Multicomponent flow calculations by a consistent primitive algorithm". In: *Journal of Computational Physics* 112.1 (1994), pp. 31–43.

[36] Stephen R Turns et al. *Introduction to combustion*. Vol. 287. McGraw-Hill Companies New York, NY, USA, 1996.

[37] Bertil Gustafsson. *High order difference methods for time dependent PDE*. Vol. 38. Springer Science & Business Media, 2007.

[38] Ami Harten and Stanley Osher. "Uniformly high-order accurate nonoscillatory schemes. I". In: *Upwind and High-Resolution Schemes*. Springer, 1997, pp. 187–217.

[39] Xu-Dong Liu, Stanley Osher, and Tony Chan. "Weighted essentially non-oscillatory schemes". In: *Journal of computational physics* 115.1 (1994), pp. 200–212.

[40] Panagiotis Tsoutsanis, Antonis F Antoniadis, and Karl W Jenkins. "Improvement of the computational performance of a parallel unstructured WENO finite volume CFD code for Implicit Large Eddy Simulation". In: *Computers & Fluids* 173 (2018), pp. 157–170.

[41] Jason E Hicken and David W Zingg. "Summation-by-parts operators and high-order quadrature". In: *Journal of Computational and Applied Mathematics* 237.1 (2013), pp. 111–125.

[42] Christopher A Kennedy and Andrea Gruber. "Reduced aliasing formulations of the convective terms within the Navier–Stokes equations for a compressible fluid". In: *Journal of Computational Physics* 227.3 (2008), pp. 1676–1700.

[43] David A Kopriva. *Implementing spectral methods for partial differential equations: Algorithms for scientists and engineers*. Springer Science & Business Media, 2009.

[44] Olivier Desjardins, Guillaume Blanquart, Guillaume Balarac, and Heinz Pitsch. "High order conservative finite difference scheme for variable density low Mach number turbulent flows". In: *Journal of Computational Physics* 227.15 (2008), pp. 7125–7159.

[45] Travis C Fisher and Mark H Carpenter. "High-order entropy stable finite difference schemes for nonlinear conservation laws: Finite domains". In: *Journal of Computational Physics* 252 (2013), pp. 518–557.

[46] Mark H Carpenter, Travis C Fisher, Eric J Nielsen, and Steven H Frankel. "Entropy stable spectral collocation schemes for the Navier–Stokes equations: Discontinuous interfaces". In: *SIAM Journal on Scientific Computing* 36.5 (2014), B835–B867.

[47] Mark H Carpenter, Travis C Fisher, Eric J Nielsen, and Steven H Frankel. "Entropy stable spectral collocation schemes for the Navier–Stokes equations: Discontinuous interfaces". In: *SIAM Journal on Scientific Computing* 36.5 (2014), B835–B867.

[48] Travis Calob Fisher. "High-order L2 stable multi-domain finite difference method for compressible flows". PhD thesis. Purdue University, 2012.

[49] Travis C Fisher and Mark H Carpenter. "High-order entropy stable finite difference schemes for nonlinear conservation laws: Finite domains". In: *Journal of Computational Physics* 252 (2013), pp. 518–557.

[50] Gregor J Gassner, Andrew R Winters, and David A Kopriva. "Split form nodal discontinuous Galerkin schemes with summation-by-parts property for the compressible Euler equations". In: *Journal of Computational Physics* 327 (2016), pp. 39–66.

[51] Nao Shima, Yuichi Kuya, Yoshiharu Tamaki, and Soshi Kawai. "Preventing spurious pressure oscillations in split convective form discretization for compressible flows". In: *Journal of Computational Physics* 427 (2021), p. 110060.

[52] Hendrik Ranocha. "Entropy conserving and kinetic energy preserving numerical methods for the Euler equations using summation-by-parts operators". In: *Spectral and high order methods for partial differential equations ICOSAHOM 2018* 134 (2020), pp. 525–535.

[53] Eitan Tadmor. "The numerical viscosity of entropy stable schemes for systems of conservation laws. I". In: *Mathematics of Computation* 49.179 (1987), pp. 91–103.

[54] Farzad Ismail and Philip L Roe. "Affordable, entropy-consistent Euler flux functions II: Entropy production at shocks". In: *Journal of Computational Physics* 228.15 (2009), pp. 5410–5436.

[55] Andrés M Rueda-Ramirez, Sebastian Hennemann, Florian J Hindenlang, Andrew R Winters, and Gregor J Gassner. "An entropy stable nodal discontinuous Galerkin method for the resistive MHD equations. Part II: Subcell finite volume shock capturing". In: *Journal of Computational Physics* 444 (2021), p. 110580.

[56] Johannes Markert, Gregor Gassner, and Stefanie Walch. "A sub-element adaptive shock capturing approach for discontinuous Galerkin methods". In: *Communications on Applied Mathematics and Computation* (2021), pp. 1–43.

[57] Michael Schlottke-Lakemper, Andrew R Winters, Hendrik Ranocha, and Gregor J Gassner. "A purely hyperbolic discontinuous Galerkin approach for self-gravitating gas dynamics". In: *Journal of Computational Physics* 442 (2021), p. 110467.

[58] Johannes Markert, Stefanie Walch, and Gregor Gassner. "A discontinuous Galerkin solver in the flash multiphysics framework". In: *Monthly Notices of the Royal Astronomical Society* 511.3 (2022), pp. 4179–4200.

[59] Per-Olof Persson and Jaime Peraire. "Sub-cell shock capturing for discontinuous Galerkin methods". In: *44th AIAA Aerospace Sciences Meeting and Exhibit*. 2006, p. 112.

[60] Xiangxiong Zhang and Chi-Wang Shu. "Maximum-principle-satisfying and positivity-preserving high-order schemes for conservation laws: survey and new developments". In: *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* 467.2134 (2011), pp. 2752–2776.

[61] Raymond J Spiteri and Steven J Ruuth. "A new class of optimal high-order strong-stability-preserving time discretization methods". In: *SIAM Journal on Numerical Analysis* 40.2 (2002), pp. 469–491.

[62] Marsha J Berger and Joseph Oliger. "Adaptive mesh refinement for hyperbolic partial differential equations". In: *Journal of computational Physics* 53.3 (1984), pp. 484–512.

[63] Marsha J Berger and Phillip Colella. "Local adaptive mesh refinement for shock hydrodynamics". In: *Journal of computational Physics* 82.1 (1989), pp. 64–84.

[64] Lucas Friedrich, Andrew R Winters, David C Del Rey Fernández, Gregor J Gassner, Matteo Parsani, and Mark H Carpenter. "An entropy stable h/p non-conforming discontinuous Galerkin method with the summation-by-parts property". In: *Journal of Scientific Computing* 77.2 (2018), pp. 689–725.

[65] David A Kopriva and John H Kolias. "A conservative staggered-grid Chebyshev multidomain method for compressible flows". In: *Journal of computational physics* 125.1 (1996), pp. 244–261.

[66] David A Kopriva, Stephen L Woodruff, and M Yousuff Hussaini. "Computation of electromagnetic scattering with a non-conforming discontinuous spectral element method". In: *International journal for numerical methods in engineering* 53.1 (2002), pp. 105–122.

[67] Tan Bui-Thanh and Omar Ghattas. "Analysis of an hp-nonconforming discontinuous Galerkin spectral element method for wave propagation". In: *SIAM Journal on Numerical Analysis* 50.3 (2012), pp. 1801–1826.

[68] Florian Hindenlang, Gregor J Gassner, Christoph Altmann, Andrea Beck, Marc Staudenmaier, and Claus-Dieter Munz. "Explicit discontinuous Galerkin methods for unsteady problems". In: *Computers & Fluids* 61 (2012), pp. 86–93.

[69] Gregor Gassner and David A Kopriva. "A comparison of the dispersion and dissipation errors of Gauss and Gauss–Lobatto discontinuous Galerkin spectral element methods". In: *SIAM Journal on Scientific Computing* 33.5 (2011), pp. 2560–2579.

[70] David A Kopriva and Edwin Jimenez. "An assessment of the efficiency of nodal discontinuous Galerkin spectral element methods". In: *Recent Developments in the Numerics of Nonlinear Hyperbolic Conservation Laws*. Springer, 2013, pp. 223–235.

[71] T. Grassi, S. Bovino, D. R. G. Schleicher, J. Prieto, D. Seifried, E. Simoncini, and F. A. Gianturco. "KROME - a package to embed chemistry in astrophysical simulations". In: *Monthly Notices of the Royal Astronomical Society* 439.3 (Feb. 2014), pp. 2386–2419. DOI: 10.1093/mnras/stu114. URL: https://doi.org/10.1093%2Fmnras%2Fstu114.

[72] Gregor J Gassner, Florian Hindenlang, and Claus-Dieter Munz. "A Runge-Kutta based discontinuous Galerkin method with time accurate local time stepping". In: *Adaptive High-Order Methods in Computational Fluid Dynamics*. World Scientific, 2011, pp. 95–118.

[73] Andrew R Winters and David A Kopriva. "High-order local time stepping on moving DG spectral element meshes". In: *Journal of Scientific Computing* 58.1 (2014), pp. 176–202.

[74] Lucas Friedrich, Gero Schnücke, Andrew R Winters, David C Fernández, Gregor J Gassner, and Mark H Carpenter. "Entropy stable space–time discontinuous Galerkin schemes with summation-by-parts property for hyperbolic conservation laws". In: *Journal of Scientific Computing* 80.1 (2019), pp. 175–222.

[75] Sigal Gottlieb and Chi-Wang Shu. "Total variation diminishing Runge-Kutta schemes". In: *Mathematics of computation* 67.221 (1998), pp. 73–85.

[76] Richard Courant, Kurt Friedrichs, and Hans Lewy. "Über die partiellen Differenzengleichungen der mathematischen Physik". In: *Mathematische annalen* 100.1 (1928), pp. 32–74.

[77] Robert I McLachlan and G Reinout W Quispel. "Splitting methods". In: *Acta Numerica* 11 (2002), pp. 341–434.

[78] Gilbert Strang. "On the construction and comparison of difference schemes". In: *SIAM journal on numerical analysis* 5.3 (1968), pp. 506–517.

[79] T. Loman, Y. Ma, V. Ilin, S. Gowda, N. Korsbo, N. Yewale, C. V. Rackauckas, and S. A. Isaacson. "Catalyst: Fast Biochemical Modeling with Julia". In: *bioRxiv* (2022). DOI: 10.1101/2022.07.30.502135. eprint: https://www.biorxiv.org/content/early/2022/08/02/2022.07.30.502135.full.pdf. URL: https://www.biorxiv.org/content/early/2022/08/02/2022.07.30.502135.

[80] Christopher Rackauckas et al. *SciML/DifferentialEquations.jl: v7.2.0*. Version v7.2.0. June 2022. DOI: 10.5281/zenodo.6780348. URL: https://doi.org/10.5281/zenodo.6780348.

[81] *Trixi.jl docs stable*. URL: %5Curl%7Bhttps://trixi-framework.github.io/Trixi.jl/stable/%7D.

[82] *Platform specific instructions for official binaries for Julia*. URL: https://julialang.org/downloads/platform/.

[83] Jian-Hang Wang, Shucheng Pan, Xiangyu Y Hu, and Nikolaus A Adams. "A split random time stepping method for stiff and non-stiff chemically reacting flows". In: *arXiv preprint arXiv:1802.04116* (2018).

[84] Bin Zhang, Hong Liu, Fang Chen, and Jian Hang Wang. "The equilibrium state method for hyperbolic conservation laws with stiff reaction terms". In: *Journal of Computational Physics* 263 (2014), pp. 151–176.

[85] *Trixi.jl docs stable - overview semidiscretizations*. URL: https://trixi%20-framework.github.io/Trixi.jl/stable/overview/Semidiscretizations.

[86] *Trixi Framework - Mesh generation with HOHQMesh*. URL: https://trixi-framework.github.io/HOHQMesh/.

[87] Travis C Fisher, Mark H Carpenter, Jan Nordström, Nail K Yamaleev, and Charles Swanson. "Discretely conservative finite-difference formulations for nonlinear conservation laws in split form: Theory and boundary conditions". In: *Journal of Computational Physics* 234 (2013), pp. 353–375.

[88] Gregor J Gassner. "A skew-symmetric discontinuous Galerkin spectral element discretization and its relation to SBP-SAT finite difference methods". In: *SIAM Journal on Scientific Computing* 35.3 (2013), A1233–A1253.

[89] Gregor J Gassner. "A kinetic energy preserving nodal discontinuous Galerkin spectral element method". In: *International Journal for Numerical Methods in Fluids* 76.1 (2014), pp. 28–50.

[90] Dominik Derigs, Andrew R Winters, Gregor J Gassner, and Stefanie Walch. "A novel high-order, entropy stable, 3D AMR MHD solver with guaranteed positive pressure". In: *Journal of Computational Physics* 317 (2016), pp. 223–256.

[91] Pooya Movahed and Eric Johnsen. "A solution-adaptive method for efficient compressible multifluid simulations, with application to the Richtmyer –Meshkov instability". In: *Journal of Computational Physics* 239 (2013), pp. 166–186.

[92] James J Quirk and Smadar Karni. "On the dynamics of a shock–bubble interaction". In: *Journal of Fluid Mechanics* 318 (1996), pp. 129–163.

[93] Antonio Marquina and Pep Mulet. "A flux-split algorithm applied to conservative models for multicomponent compressible flows". In: *Journal of Computational Physics* 185.1 (2003), pp. 120–138.

[94] Cheng Wang, Xiangxiong Zhang, Chi-Wang Shu, and Jianguo Ning. "Robust high order discontinuous Galerkin schemes for two-dimensional gaseous detonations". In: *Journal of Computational Physics* 231.2 (2012), pp. 653–665.

[95] Juntao Huang and Chi-Wang Shu. "Positivity-preserving time discretizations for production–destruction equations with applications to non- equilibrium flows". In: *Journal of Scientific Computing* 78.3 (2019), pp. 1811–1839.

[96] Dinshaw S Balsara and Daniel S Spicer. "A staggered mesh algorithm using high order Godunov fluxes to ensure solenoidal magnetic fields in magnetohydrodynamic simulations". In: *Journal of Computational Physics* 149.2 (1999), pp. 270–292.