DEPARTMENT OF INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Master's Thesis in Informatics

Cloud Simulation with the ExaHyPE-Engine

Lukas Krenz

DEPARTMENT OF INFORMATICS

TECHNICAL UNIVERSITY OF MUNICH

Master's Thesis in Informatics

Cloud Simulation with the ExaHyPE-Engine

Wolkensimulationen mit der ExaHyPE-Engine

Author:	Lukas Krenz
Supervisor:	Univ.-Prof. Dr. Michael Bader
Advisor:	Leonhard Rannabauer, M.Sc.
Submission Date:	February 15, 2019

I confirm that this master's thesis in informatics is my own work and I have documented all sources and material used.

Garching, January , 2019 Lukas Krenz

Acknowledgments

I want to thank Prof. Michael Bader for the chance to work on this exciting topic. I am particularly grateful to my advisor Leonhard Rannabauer, whose guidance made this thesis not only possible but also enjoyable. Additionally, I wish to acknowledge the support of Dominic Charrier, who was of great help with the implementation in *ExaHyPE*.

My gratitude extends to my friends and family, who supported me during the entire thesis. Special thanks go to my friend William who proofread this document.

Abstract

This thesis is concerned with the simulation of clouds using the ExaHyPE framework. ExaHyPE is an engine for hyperbolic partial differential equations that uses the ADER-DG scheme.

We use the compressible Navier-Stokes equations for computational fluid dynamics. Due to the diffusive components, the Navier-Stokes equations are not hyperbolic. We thus need to modify parts of the numerical scheme. In addition, we extend the equation set to the reactive compressible Navier-Stokes equations which include a source term that models chemical reactions.

We develop a gradient-based, global adaptive mesh refinement (AMR) indicator that finds cells with unusually large gradients.

Our implementation is evaluated for standard two and three dimensional fluid dynamics test cases, for flows over a realistic background atmosphere and for a reactive detonation wave. Additionally, we perform a convergence test and a time-to-solution benchmark that evaluates our proposed AMR criterion.

Contents

CHAPTER 1

Introduction

Climate modeling

Simulations are an important part of weather and climate forecasts. Both types of forecasts are difficult problems that are limited by a lack of computational power. Especially in a time where the realization of exascale supercomputers is imminent, scalable numerical model are becoming more and more important. Current climate and weather models are not ready for the exascale era [Sch+18]. While [Sch+18] proposes a spatial resolution of 1 km as a goal, current state of the art simulations operate on a resolution that is an order of magnitude coarser.

Of course, a finer grid is not useful if the computational effort is too large. Thus they propose a compute rate of one simulated years per wall-clock day. An example of a recent supercomputing effort in this domain [Fuh+18] achieved a performance of 0.043 simulated years per wall-clock day with a resolution of 930 m for a near-global simulation on the Piz Daint supercomputer. They used the COSMO 5.0 model which is based on a finite-difference approach.

Discontinuous Galerkin

The discontinuous Galerkin (DG) methods are exciting numerical schemes as they are relatively easy to scale. In addition, they can be of high order and can perform local grid refinement [HW08]. These advantages are only some reasons why they have been investigated for weather and climate modeling [Mül+10; GR08]. Recently, ADER discontinuous Galerkin methods has been proposed, for example in [Dum+08]. These methods are scalable and of arbitrary order in both time and space. With an appropriate limiter, they are also stable, even in the vicinity of discontinuities [DL16]. They are thus promising numerical methods, even for the exascale era.

The goal of this thesis is to evaluate the ADER-DG method of [Dum+08] for testing scenarios, which have been devised as benchmark examples for meteorological applications [Rob93; GR08]. We implement this with the *ExaHyPE*-engine, which is "An Exascale Hyperbolic PDE Engine".

We make the following contributions:

- In section 2.1, we describe a coupling of the compressible Navier-Stokes equations for two and three dimensional problems. In addition, we adapt the equation set for scenarios with gravity and present a coupling with an advection-reaction term.

- We describe an ADER-DG (section 2.2) and MUSCL-Hancock-scheme (section 2.3) for problems with diffusive terms. Moreover, we describe a combination of both methods in section 2.4.1, where we use the MUSCL-Hancock-scheme in cases where the ADER-DG-method is unstable.
- In section 2.4.2, we introduce an adaptive mesh refinement (AMR) method that checks whether a local cell is an outlier. As indicator function, we use the total variation of the discrete solution, and as outlier detection, we use the mean and standard variance of this function.
- In chapter 3 we show how we implemented our proposed method using the *ExaHyPE*-engine.
- We present seven scenarios that stem from different domains in chapter 4.
- We check the performance of our proposed method in chapter 5.
  - In section 5.1 we perform a convergence test.
  - In section 5.2 we investigate the ability of our scheme to simulate classical fluid mechanics (CFD) test scenarios correctly.
  - We evaluate the simulation quality of atmospheric flows in section 5.3.
  - For these scenarios, we perform a time-to-solution test for our AMR strategy (section 5.4).
  - Finally, we evaluate the implementation of the reactive terms in section 5.5.

CHAPTER 2

An ADER-DG Scheme for the Navier-Stokes Equations

This chapter describes the numerical and physical background that is needed to simulate reacting fluids with high accuracy. We start with a description of the arbitrary high order derivatives (ADER) discontinuous Galerkin (DG) method, with a focus on equations that contain both advective and diffusive terms. We then segue into a short description of the MUSCL-Hancock scheme, which is a second order finite volume method.

We conclude this chapter with a description of finite volume limiting and adaptive mesh refinement (AMR). The limiting combines both numerical methods to achieve a stable, high-order method and the AMR allows us to perform high quality simulations with a smaller computational cost.

2.1. Reactive Compressible Navier Stokes Equations

Fluid motion can be described by the compressible Navier Stokes equations. We follow the description in [Dum10] for the Navier Stokes part. The coupling with the advection-diffusion-reaction equation follows [HD11] which describes this equation set for the one-dimensional case. In the following, we will denote the spatial coordinates as $\mathbf{x} = (x, z)$ and $\mathbf{x} = (x, y, z)$ for the two and three-dimensional case respectively. The variable t corresponds to the current physical time. We solve the PDE in the conservative form

$$\underbrace{\frac{\partial}{\partial t}\begin{pmatrix}\rho\\ \rho\boldsymbol{v}\\ \rho E\\ \rho Z\end{pmatrix}}_{\mathbf{Q}} + \nabla\cdot\underbrace{\left(\underbrace{\begin{pmatrix}\rho\boldsymbol{v}\\ \boldsymbol{v}\otimes\rho\boldsymbol{v}+\mathbf{I}p\\ \boldsymbol{v}\cdot(\mathbf{I}\rho E+\mathbf{I}p)\\ \rho\boldsymbol{v}Z\end{pmatrix}}_{\mathbf{F}^h(\mathbf{Q})} + \underbrace{\begin{pmatrix}0\\ \sigma(\mathbf{Q},\nabla\mathbf{Q})\\ \boldsymbol{v}\cdot\sigma(\mathbf{Q},\nabla\mathbf{Q})-\kappa\nabla T\\ 0\end{pmatrix}}_{\mathbf{F}^v(\mathbf{Q},\nabla\mathbf{Q})}\right)}_{\mathbf{F}(\mathbf{Q},\nabla\mathbf{Q})} = \underbrace{\begin{pmatrix}S_\rho\\ S_{\rho\boldsymbol{v}}\\ S_{\rho E}\\ S_{\rho Z}\end{pmatrix}}_{\mathbf{S}(\mathbf{Q},\mathbf{x},t)} \tag{2.1}$$

which describes the time evolution of variables $\mathbf{Q}$ with respect to a flux $\mathbf{F}(\mathbf{Q}, \nabla\mathbf{Q})$ and a source $\mathbf{S}(\mathbf{Q})$. The vector of conserved quantities is given by

$$\mathbf{Q} = (\rho, \rho\boldsymbol{v}, \rho E, \rho Z)\,, \tag{2.2}$$

where ρ is the density, $\rho\boldsymbol{v}$ is the two or three-dimensional momentum, ρE is the energy density and ρZ is the mass fraction of the chemical reactant. In our case, the chemical reactant is unburnt gas. The rows of eq. (2.1) are the conservation of mass, the conservation of momentum, the conservation of energy and the continuity equation of the reactant. We split the flux into a hyperbolic $\mathbf{F}^h(\mathbf{Q})$ and a viscous part $\mathbf{F}^v(\mathbf{Q}, \nabla\mathbf{Q})$

$$\mathbf{F}(\mathbf{Q}, \nabla\mathbf{Q}) = \mathbf{F}^h(\mathbf{Q}) + \mathbf{F}^v(\mathbf{Q}, \nabla\mathbf{Q}). \tag{2.3}$$

Hyperbolic flux

The hyperbolic flux is given by

$$\mathbf{F}^h(\mathbf{Q}) = \begin{pmatrix} \rho\boldsymbol{v} \\ \boldsymbol{v} \otimes \rho\boldsymbol{v} + \mathbf{I}p \\ \boldsymbol{v} \cdot (\mathbf{I}\rho E + \mathbf{I}p) \\ \rho\boldsymbol{v}Z \end{pmatrix}, \tag{2.4}$$

which are the Euler equations coupled with an advection equation. In this equation, $\mathbf{a} \otimes \mathbf{b} = \mathbf{a}\mathbf{b}^\intercal$ is the Kronecker or outer product of two vectors $\mathbf{a}$ and $\mathbf{b}$. The pressure p is given by the equation of state of an ideal reacting gas

$$p = (\gamma - 1)\left(\rho E - \frac{1}{2}(\boldsymbol{v} \cdot \rho\boldsymbol{v}) - q_0\rho Z - gz\right). \tag{2.5}$$

The term $q_0\rho Z$ is the chemical energy with a heat release q_0 [HLW00], the term gz is the geopotential height with the gravity of Earth g [GR08]. The pressure is related to the temperature T by the thermal equation of state

$$p = \rho R T, \tag{2.6}$$

where R is the specific gas constant of a fluid.

Viscous flux

The viscous flux is

$$\mathbf{F}^v(\mathbf{Q}, \nabla\mathbf{Q}) = \begin{pmatrix} 0 \\ \boldsymbol{\sigma}(\mathbf{Q}, \nabla\mathbf{Q}) \\ \boldsymbol{v} \cdot \boldsymbol{\sigma}(\mathbf{Q}, \nabla\mathbf{Q}) - \kappa\nabla T \\ 0 \end{pmatrix}, \tag{2.7}$$

where $\boldsymbol{\sigma}$ and $(\kappa\nabla T)$ denote the stress tensor and heat flux respectively. The viscous effects of the fluid are modeled by the stress tensor

$$\boldsymbol{\sigma}(Q, \nabla Q) = \mu\left((2/3\nabla \cdot \boldsymbol{v}) - (\nabla\boldsymbol{v} + \nabla\boldsymbol{v}^\intercal)\right). \tag{2.8}$$

We further introduce the ratio of specific heats γ and the heat fraction for constant volume c_v and constant volume c_p. These constants are all fluid dependent and relate to each other [Dum10] and the gas constant by

$$\begin{aligned} c_v &= \frac{1}{\gamma - 1} R, \\ c_p &= \frac{\gamma}{\gamma - 1} R, \\ R &= c_p - c_v, \\ \gamma &= \frac{c_p}{c_v}. \end{aligned} \tag{2.9}$$

We can compute the heat conduction coefficient

$$\kappa = \frac{\mu\gamma}{Pr} \frac{1}{\gamma - 1} R = \frac{\mu c_p}{Pr}, \tag{2.10}$$

where the Prandtl number Pr depends on the fluid. The temperature gradient ∇T in terms of the conservative variables can be found in appendix A.

The maximal absolute eigenvalues for the normal Jacobians of both convective and viscous part in direction of a normal vector $\mathbf{n}$ are

$$\begin{aligned} |\lambda_c^{max}| &= |v_n| + c, \\ |\lambda_v^{max}| &= \max\left(\frac{4}{3}\frac{\mu}{\rho}, \frac{\gamma\mu}{Pr\,\rho}\right) \end{aligned} \tag{2.11}$$

with velocity in direction of the normal v_n and speed of sound $c = \sqrt{\gamma R T}$. The Jacobians of both fluxes can be found in appendix A.

Gravity

We also support two different kind of optional source terms. The first source term is gravity, given by

$$S_{\rho v} = -g e_z \cdot \rho v, \tag{2.12}$$

where e_z is the unit vector pointing in z-direction [GR08].

Some of our scenarios can be described as a perturbation of a background state that is initially in hydrostatic balance

$$\frac{\partial}{\partial z}\overline{p}(z) = -g\overline{\rho}(z), \tag{2.13}$$

where $\overline{p}$ and $\overline{\rho}$ is the background pressure and density. We now focus on the following part of the momentum equation (neglecting viscous effects) of eq. (2.1) in combination with gravitational source term (eq. 2.12)

$$\frac{\partial \rho v}{\partial t} + \nabla \cdot (v \otimes \rho v + I p) = -g k \cdot \rho. \tag{2.14}$$

The background state eq. (2.13) can be comparatively large and can thus diminish other parts of the flux due to numerical instabilities [Mül+10]. To avoid this, we split the pressure as $p = \overline{p}(z) + p'(\mathbf{x}, t)$ and the density as $\rho = \overline{\rho}(z) + \rho'(\mathbf{x}, t)$. Inserting this splitting and the definition of divergence into eq. (2.14) allows us to apply eq. (2.13). The derivatives of the background states are non-zero only for the z-direction. We arrive at

$$\frac{\partial \rho \boldsymbol{v}}{\partial t} + \boldsymbol{\nabla} \cdot \left(\boldsymbol{v} \otimes \rho \boldsymbol{v} + \mathbf{I} p' \right) = -g \mathbf{k} \rho'. \tag{2.15}$$

We thus cancel parts of the flux with parts of the source term. This form of the equation is inspired by equation set 3 of [GR08]. They additionally use a splitting of the energy and use the perturbations directly as conservative variables.

Chemical reactions

The chemical reactive source term is

$$S_{\rho Z} = -K(T)\rho Z. \tag{2.16}$$

We use the simple reaction model

$$K(T) = \begin{cases} -\frac{1}{\tau} & T > T_{ign} \\ 0 & \text{otherwise} \end{cases} \tag{2.17}$$

where τ is the timescale of the reaction and T_{ign} is the activation temperature [HD11; HLW00].

Boundary conditions

To close the system, we need to impose boundary conditions. Many scenarios are described with periodic boundary conditions. This is not possible with *ExaHyPE*. Instead, we use the analytical solution of our problems at the boundary, imposing both the conservative variables $\mathbf{Q}$ and its gradient $\boldsymbol{\nabla}\mathbf{Q}$. This type of boundary condition is called Cauchy condition. Note that this leads to an error when our problem does not possess an exact analytical solution. This is the case for test cases that are exact solutions for the incompressible Navier Stokes equations but do not satisfy the compressible equation set.

As a physical boundary condition, we limit ourselves to wall boundary condition. A standard wall boundary condition for viscous fluid is the no-slip condition, where we assume that when the fluid is close to a wall, it has a velocity of zero relative to the wall. We enforce this by setting

$$\begin{aligned} \rho^o &= \rho^i, \\ \rho \boldsymbol{v}^o &= 2\rho \boldsymbol{v}^w - \rho \boldsymbol{v}^i, \\ \rho E^0 &= \rho E^i, \\ \rho Z^0 &= \rho Z^i, \\ (\nabla Q)^o &= (\nabla Q^o), \end{aligned} \tag{2.18}$$

where a superscript of o and i denotes the values outside and inside of the domain respectively. The term ρv^w denotes the velocity of the wall.

Similarly to the no-slip condition, we define the free-slip condition

$$\rho v_d^o = \begin{cases} -\rho v_d^i & d = \text{normal} \\ \rho v_d^i & \text{otherwise} \end{cases} \tag{2.19}$$

where only the velocity in normal direction is zero next to the wall. The other variables are extrapolated in the same manner as in eq. (2.18).

We manually set the energy component of the numerical flux to zero for both no- and free-slip conditions.

2.2. The ADER-DG Method

We describe the arbitrary derivative discontinuous Galerkin (ADER-DG) method in this chapter. Our description of the main method follows [Dum+08; Dum10; Dum+18a], our notation follows primarily [Dum+18a]. Let N^{var} be the number of variables, d the number of spatial dimensions and $N > 0$ the degree of the basis functions. We also call N the order of the method. We discuss the approximation of systems of partial differential equations (PDE) of the form

$$\frac{\partial}{\partial_t} Q + \nabla \cdot F(Q, \nabla Q) = S(x, t, Q), \tag{2.20}$$

which in contrast to hyperbolic conservation laws of the form

$$\frac{\partial}{\partial_t} Q + \nabla \cdot F(Q) = S(x, t, Q) \tag{2.21}$$

contain the gradient $\nabla Q \in \mathbb{R}^{N^{var} \times d}$ of the so called vector of conservative variables $Q \in \mathbb{R}^{N^{var}}$ [Dum10]. They are therefore not hyperbolic but rather parabolic or elliptic.

We discuss the solution of a conservation law eq. (2.20) with two or three dimensional domain Ω and its boundary $\partial\Omega$. In our implementation of the discontinuous Galerkin (DG) framework, we approximate this solution in the space

$$\Omega = \bigcup_k C_k \tag{2.22}$$

of disjoint quadrilateral cells C. Note that we do not distinguish between the approximation space and the domain; the use should be clear given its surrounding context. For each cell we denote its center by cell-center(C) and its size by $\Delta x, \Delta y$ and Δz.

The ADER-DG-scheme is a predictor-corrector method. We first compute a local time evolution of each cell in a predictor step and then add the influence of the neighbors in the corrector step.

Reference cell

For reasons of computational efficiency, we describe the scheme in terms of a reference cell. We use the quad cell $\hat{C} = (0,1)^d$ as space reference cell. The mapping, which takes a point on the reference cell with coordinates $\hat{\mathbf{x}}$ and and returns its coordinates $\mathbf{x}$ in a grid cell C is

$$\mathbf{x} = \mathcal{M}(\hat{\mathbf{x}}) = \text{cell-center}(C) + \begin{pmatrix} \Delta x & & \\ & \Delta y & \\ & & \Delta z \end{pmatrix} \begin{pmatrix} \hat{x} - 0.5 \\ \hat{y} - 0.5 \\ \hat{z} - 0.5 \end{pmatrix}. \tag{2.23}$$

The symbol $\mathcal{M}^{-1}$ denotes the inverse mapping. It can be used to define a function $f(\mathbf{x})$ acting on a point $\mathbf{x}$ of a regular cell in terms of a function $\hat{f}(\mathbf{x})$ that acts on the corresponding point of the reference cell by the relation

$$f(\mathbf{x}) = f\left(\mathcal{M}^{-1}(\mathbf{x})\right) = \hat{f}(\hat{\mathbf{x}}). \tag{2.24}$$

Similar to this, we define a mapping that maps a reference time point $\hat{t}$ to simulation time t

$$t = \mathcal{T}(\hat{t}) = t^k + (\Delta t)\hat{t}, \tag{2.25}$$

where the reference time interval is $[0,1]$ and the time interval of a cell is $[t^k, t^{k+1}]$. The time at the beginning of a timestep with index k and timestep size Δt is denoted by t^k, the time at the end of a step is $t^{k+1} = t^k + \Delta t$.

The Jacobian determinant of $\mathcal{M}$ is simply the volume V of the cell

$$V = \det(\text{Jacobian}\,\mathcal{M}) = \Delta x\,\Delta y\,\Delta z, \tag{2.26}$$

the Jacobian determinant of $\mathcal{T}$ is Δt. This is useful for evaluating derivatives by the chain rule and, similarly, integrals by substitution. For example, we can evaluate a volume integral over a cell by

$$\int_C f(\mathbf{x})\,d\mathbf{x} = V \int_{\hat{C}} \hat{f}(\hat{\mathbf{x}})\,d\hat{\mathbf{x}}\,. \tag{2.27}$$

Basis functions

The next step is the definition of basis functions for our cells. We use Lagrange polynomials that are defined in the one-dimensional case for the reference interval $(0,1)$ by

$$\hat{\varphi}_i(x) = \prod_{\substack{0 \leqslant j \leqslant N \\ i \neq j}} \frac{x - \hat{x}_j}{x_i - \hat{x}_j}. \tag{2.28}$$

We can then approximate functions as a sum over the support points $\hat{x}_j$

$$f(x) = \sum_{i=0}^{N} f(\hat{x}_i)\hat{\varphi}_i(x) = f(\hat{x}_i)\hat{\varphi}_i(x), \tag{2.29}$$

where we used the Einstein summation convention which implies summation over repeated free indices. This convention is used from here on. The support points $\hat{x}_i$ are chosen such that they coincide with the nodes of the Gauss-Legendre quadrature of order $N+1$ which we can use to evaluate integrals

$$\int_0^1 f(\hat{x})\,d\hat{x} = \sum_{i=0}^{N} w_i f(\hat{x}_i), \tag{2.30}$$

where w_i is the quadrature weight corresponding to the node $\hat{x}_i$.

The basis functions are similarly defined in d-dimensions by considering nodes that are the tensor-products of d one-dimensional Gauss-Legendre quadrature nodes

$$\hat{\phi}_n(\mathbf{x}) = \prod_{i=0}^{d} \hat{\varphi}_{n_i}(x_i). \tag{2.31}$$

They are indexed by a multi-index $\mathbf{n} = (n_1, \ldots, n_d)$ that is linearized such that $n = (0, \ldots, (N+1)^d)$. A d-dimensional function is then approximated by

$$f(\mathbf{x}) = \sum_{i=0}^{(N+1)^d-1} f(\hat{\mathbf{x}}_i)\hat{\varphi}_i(\mathbf{x}) = f(\hat{\mathbf{x}}_i)\hat{\varphi}_i(\mathbf{x}). \tag{2.32}$$

Similarly to the one-dimensional case, we can evaluate integrals by

$$\underbrace{\int_0^1 \cdots \int_0^1}_{d \text{ times}} f(\hat{\mathbf{x}})\,d\hat{\mathbf{x}} = \sum_{n=0}^{(N+1)^d-1} w_n f(\hat{\mathbf{x}}_n) \tag{2.33}$$

with

$$w_n = \prod_{i}^{d} w_{n_i}, \tag{2.34}$$

where the quadrature weights w_n are the product of all relevant one-dimensional quadrature weights, using the aforementioned (linearized) multi-index n.

Two important properties of this family of polynomials are

$$\text{Interpolating:} \qquad \hat{\varphi}_i(\hat{x}_j) = \delta_{ij}, \qquad \hat{\phi}_n(\hat{\mathbf{x}}_m) = \delta_{nm}, \tag{2.35}$$

$$\text{Orthogonality:} \qquad \int_0^1 \hat{\varphi}_i(\hat{x}_i)\hat{\varphi}_j(\hat{x}_j)\,d\hat{x}_j = w_i\delta_{ij}, \qquad \underbrace{\int_0^1 \cdots \int_0^1}_{d \text{ times}} \hat{\phi}_n(\hat{\mathbf{x}})\,\hat{\phi}_m(\hat{\mathbf{x}})\,d\hat{\mathbf{x}} = w_n\delta_{nm}, \tag{2.36}$$

where we made use of the Kronecker delta

$$\delta_{ij} = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}. \tag{2.37}$$

Cell basis

We define the basis functions over a space-cell with an index $n = 0, \ldots, (N+1)^d - 1$ as

$$\phi_n(\mathbf{x}) = \hat{\phi}_n\left(\mathcal{M}^{-1}(\mathbf{x})\right). \tag{2.38}$$

In addition to the space cells, we also introduce space-time cells $C \times [t^k, t^{k+1}]$ with corresponding reference cell $(0,1)^d \times [0,1]$. A basis for these cells is given by

$$\begin{aligned} \hat{\psi}_{l,n}(\hat{\mathbf{x}}, \hat{t}) &= \varphi_l(t)\phi_n(x) \\ \psi_{l,n}(\mathbf{x}, t) &= \hat{\psi}\left(\mathcal{M}^{-1}(\mathbf{x}), \mathcal{T}^{-1}(t)\right) \end{aligned} \tag{2.39}$$

where the first index $l = 0, \ldots, N$ runs over the time. Outside of elements, we define the basis functions to be equal to zero.

We introduce an additional index $\nu = (0, \ldots, N^{var} - 1)$ that corresponds to a variable. The complete basis functions are then defined by

$$\begin{aligned} \psi_{\nu,n,l}(\mathbf{x}, t) &= \boldsymbol{e}_\nu \otimes \psi_{n,l}(\mathbf{x}, t) \\ \phi_{\nu,n}(\mathbf{x}) &= \boldsymbol{e}_\nu \otimes \phi_n(\mathbf{x}) \end{aligned} \tag{2.40}$$

where $\boldsymbol{e}_\nu \in \mathbb{R}^{N^{var}}$ is vector with components

$$(\boldsymbol{e}_\nu)_{\nu'} = \delta_{\nu\nu'}, \tag{2.41}$$

and $\mathbf{A} \otimes \mathbf{B}$ denotes the tensor product between two tensors $\mathbf{A}$ and $\mathbf{B}$.

Expansion of quantities

All quantities can be expanded in the space-time basis. For example, the expansion of the space-time predictor $\overline{\mathbf{q}}^C$ can be written as

$$\overline{\mathbf{q}}^C(\mathbf{x}, t) = \underline{\overline{q}}^C_{\nu',l',n'} \psi^C_{\nu',l',n'}(\mathbf{x}, t), \tag{2.42}$$

with coefficients $\overline{\underline{q}}$. The space-time predictor is the result of the predictor step. We denote the space-time flux as $\overline{\mathbf{F}}$ and the space-time source as $\overline{\mathbf{S}}$ with coefficient vectors $\overline{\underline{F}}$ and $\overline{\underline{S}}$. Similarly, we define the space degrees of freedoms by an expansion in the space basis, for example for the space predictor

$$\mathbf{q}^C(\mathbf{x}) = \underline{q}^C_{\nu',n'}\phi^C_{\nu',n'}(\mathbf{x}), \tag{2.43}$$

where the basis does not depend on the time, unlike in eq. (2.42). We need the space-expansion for the predictor $\mathbf{q}$, the discrete solution $\mathbf{u}$, the source $\mathbf{S}$ and the flux $\mathbf{F}$, with coefficients $\underline{q}, \underline{\mathbf{u}}, \underline{S}$ and $\underline{F}$ respectively. Each coefficient index generally has the same range as the corresponding basis index. For example, the indices n and n' both run over the entire space-basis.

Here, the computational advantage of the nodal approach is evident: The coefficients correspond to the quantity evaluated at the basis node, due to the interpolation property (eq. 2.35).

Predictor

We are now ready to derive the predictor. We first multiply the conservation law by a test function of the same function space as the basis, integrate over a space-time-element C and replace all terms with their discrete approximations. In particular, we replace the vector of conservative variables $\mathbf{Q}$ with the space-time predictor $\overline{\mathbf{q}}^C$ and arrive at

$$\begin{aligned}\int_{t^k}^{t^{k+1}}\int_C \psi^C_{\nu,l,n}(\mathbf{x},t)\frac{\partial\overline{\mathbf{q}}}{\partial t}\,d\mathbf{x}\,dt + \int_{t^k}^{t^{k+1}}\int_C \psi^C_{\nu,l,n}(\mathbf{x},t)\left(\boldsymbol{\nabla}\cdot\overline{\mathbf{F}}(\overline{\mathbf{q}},\boldsymbol{\nabla}\overline{\mathbf{q}})\right)d\mathbf{x}\,dt \\ = \int_{t^k}^{t^{k+1}}\int_C \psi^C_{\nu,l,n}\overline{\mathbf{S}}(\mathbf{x},t,\overline{\mathbf{q}})\,d\mathbf{x}\,dt\,.\end{aligned} \tag{2.44}$$

Henceforth, we drop the cell index for all quantities and the spatial argument for the basis and test functions. We integrate the first term of eq. (2.44) by parts in time and the flux divergence in space. Here we do not use the Riemann solver for the flux boundary term but rather use the known discrete solution at time t. This corresponds to upwinding in time [Dum+08]. Note that this neglects the interaction with neighboring cells. The cell-local scheme is then

$$\begin{aligned}\int_C \psi_{\nu,l,n}(t^{k+1})\overline{\mathbf{q}}^{i+1}(t^{k+1})\,d\mathbf{x} - \int_{t^k}^{t^{k+1}}\int_C \frac{\partial}{\partial t}\psi_{\nu,l,n}(t)\overline{\mathbf{q}}^{i+1}(t)\,d\mathbf{x}\,dt = \int_C \psi_{\nu,l,n}(t^k)\mathbf{u}(t^k)\,d\mathbf{x} \\ + \int_{t^k}^{t^{k+1}}\int_C \psi_{\nu,l,n}(t)\boldsymbol{\nabla}\cdot\overline{\mathbf{F}}(\overline{\mathbf{q}}^i,\boldsymbol{\nabla}\overline{\mathbf{q}}^i)\,d\mathbf{x}\,dt \\ + \int_{t^k}^{t^{k+1}}\int_C \psi_{\nu,l,n}(t)\overline{\mathbf{S}}(\overline{\mathbf{q}}^i)\,d\mathbf{x}\,dt\,.\end{aligned} \tag{2.45}$$

Here, we introduced the iteration counter i as a subscript for the space-time predictor. Using the expansions in space and time (eqs. 2.42 and 2.43) results in a local systems of

equations that can be solved by a fixed-point iteration scheme. We set the initial value for the space-time predictor as

$$\underline{\overline{q}}^{0}_{v,l,n} = \underline{u}_{v,n} \tag{2.46}$$

which corresponds to a stationary initial guess [Dum+08]. For an overview of possible initial guesses, see [Dum+18a].

In practice, we precompute all integrals that contain only basis functions and their derivatives for the reference cell. The computation of the integrals for general space-time cells can then be performed by applying eq. (2.27). For details and proof of convergence for the case of linear PDEs, see [Dum+08].

Preparing for corrector

It is useful to compute time-averaged versions of our quantities. For example, this can be done for the space-time predictor by

$$\begin{aligned} q(x) &= \frac{1}{\Delta t}\int_{t^k}^{t^{k+1}} \overline{q}(x,t), dt \\ q_{v,n}(x) &= \underline{\overline{q}}_{v,l,n} w_l. \end{aligned} \tag{2.47}$$

We then extrapolate the space predictor and its gradient, and the fluxes to the boundary. In our case we have a $d-1$ dimensional basis for all $2d$ faces. We accomplish this by evaluating the function at the degrees of freedom at the boundary. Due to the interpolating property of the nodal basis (eq. 2.35), this reduces to a change of basis which can be precomputed on the reference element. We use the extrapolated values later for the Riemann solver and for the reconstruction of boundary values.

Corrector

The final step of the ADER-DG-scheme is the corrector. We multiply the system eq. (2.20) by a space test function and integrate over the space-time control volume. This results, yet again, in the weak formulation of the PDE

$$\begin{aligned} \int_{t^k}^{t^{k+1}} \int_C \phi_{v,n} \frac{\partial Q}{\partial t} \, dx \, dt + \int_{t^k}^{t^{k+1}} \int_C \phi_{v,n} \left(\nabla \cdot F(Q, \nabla Q)\right) dx \, dt \\ = \int_{t^k}^{t^{k+1}} \int_C \phi_{v,n} S(Q, x, t) \, dx \, dt \, . \end{aligned} \tag{2.48}$$

We now insert our discrete approximations and apply the space-basis expansion (eq. 2.43) to the discrete solution. The first term can be integrated by parts in time. This has the convenient result that only the boundary terms remain because the basis is constant in time. We then integrate the flux divergence by parts in space and insert the Riemann

solver to connect with the neighboring elements. Finally, we need to solve

$$\begin{aligned}\left(\int_C \phi_{\nu,n}\phi_{\nu,n'}\,dx\right)(\underline{u}_{\nu,n'}^{t^{k+1}} - \underline{u}_{\nu,n'}^{t^k}) + \left(\int_{t^k}^{t^{k+1}}\int_{\partial C}\phi_{\nu,n}\,\text{Riemann}(\overline{q}^-, \nabla\overline{q}^-; \overline{q}^+, \nabla\overline{q}^+)\cdot n\,dS\,dt\right)\\ - \left(\int_{t^k}^{t^{k+1}}\int_C \nabla\phi_{\nu,n}\cdot\overline{F}(q, \nabla q)\,dx\,dt\right)\\ = \left(\int_{t^k}^{t^{k+1}}\int_C \phi_{\nu,n}\overline{S}(q)\,dx\,dt\right)\end{aligned} \tag{2.49}$$

for the coefficients $\underline{u}_{\nu,n}^{t^{k+1}}$ of the new solution. Here the superscripts $\pm$ denote values that are extrapolated to the boundaries. Again, the computational effort can be vastly reduced by collecting the integrals into matrices and precomputing as much as possible. Using the orthogonality of the basis functions (eq. 2.36) on the first term of eq. (2.49), it becomes clear that the correction is a simple, explicit computation. Note that we evaluate the Riemann problem for the time-averaged space-predictor in our implementation. This is justified by the linearity of our numerical flux [Dum+08; Dum10].

Riemann solver & timestep

As a Riemann solver we use a simple Rusanov-flux that is adapted for diffusive problems

$$\begin{aligned}\text{Riemann}(Q^-, \nabla Q^-; Q^+, \nabla Q^+)\cdot n = \frac{1}{2}\left(F(Q^+, \nabla Q^+) + F(Q^-, \nabla Q^-)\right)\\ - \frac{1}{2}s_{max}(Q^+ - Q^-),\end{aligned} \tag{2.50}$$

with a penalty term

$$s_{max} = \max\left(|\lambda_c^{max}(Q^-)|, |\lambda_c^{max}(Q^+)|\right) + 2\eta\max\left(|\lambda_\nu^{max}(Q^-)|, |\lambda_\nu^{max}(Q^+)|\right) \tag{2.51}$$

and

$$\eta = \frac{2N+1}{h\sqrt{\frac{1}{2}\pi}}. \tag{2.52}$$

In this equation h is the side-length of an element and a superscript of $\pm$ denotes the state of the right or left side. The penalty term depends on the maximal absolute eigenvalues of the Jacobian matrices in normal direction of both convective and viscous fluxes

$$\begin{aligned}|\lambda_c^{max}| &= \max \quad \text{eigenvalues}\left(\frac{\partial F}{\partial Q}\cdot n\right),\\ |\lambda_\nu^{max}| &= \max \quad \text{eigenvalues}\left(\frac{\partial F}{\partial(\nabla Q\cdot n)}\cdot n\right),\end{aligned} \tag{2.53}$$

in direction of the normal vector $\mathbf{n}$. This Riemann solver was first published in [GLM08] and used for an ADER-DG scheme in [Dum10].

The timestep is restricted to a so-called CFL-type penalty

$$\Delta t \leqslant \mathrm{CFL}\, \alpha(N)\, h \sum_{i=1}^{d} \frac{1}{|\lambda_c^{max}|_i + 2|\lambda_\nu^{max}|_i \frac{1}{\alpha(N)h}}, \tag{2.54}$$

where the eigenvalues are evaluated for the flux Jacobian in direction i [Dum10; GLM08]. The constant $\alpha(N) \leqslant (2N+1)^{-1}$ can be obtained from von Neumann analysis on a simple model problem and depends on the approximation order [Dum+08]. We use this constant for both hyperbolic and diffusive penalty terms to ensure stability. For a more detailed and rigorous stability analysis of the Riemann solver, see [GLM08]. We use a value of 0.7 for the constant CFL.

2.3. The MUSCL-Hancock Scheme

We discussed a modern, high-order DG method in the previous chapter. While this scheme is efficient, it can become unstable, especially in case of discontinuities. A simple alternative is a finite volume scheme. We use the MUSCL-Hancock method[Lee79]. Our discussion and notation follows the one in [Toro9]. Similar to [Toro9] we only describe the two-dimensional case without source terms, an extension to three dimensions is straightforward.

In contrast to the previous DG scheme, we now only store averages instead of a full polynomial solution for each cell. We achieve higher order by reconstructing a linear function from a cell and its neighbors. Let $\mathbf{u}_{i,j}$ denote the averages per cell at spatial coordinates i, j.

Boundary extrapolated values

We first compute the slope of the linear function connecting the cells. We use the minmod slope-limiter

$$\mathrm{minmod}(a,b) = \begin{cases} 0.0 & \mathrm{sign}(a) \neq \mathrm{sign}(b) \\ a & |a| < |b| \\ b & \text{otherwise} \end{cases} \tag{2.55}$$

to avoid unphyiscally steep gradients and thus to stabilize the scheme [LeV02]. This function can be used to compute the slopes

$$\begin{aligned} \mathbf{s}_{i,j}^x &= \Delta x\, \mathrm{minmod}\left(\mathbf{u}_{i+1,j} - \mathbf{u}_{i,j},\, \mathbf{u}_{i,j} - \mathbf{u}_{i-1,j}\right), \\ \mathbf{s}_{i,j}^y &= \Delta y\, \mathrm{minmod}\left(\mathbf{u}_{i,j+1} - \mathbf{u}_{i,j},\, \mathbf{u}_{i,j} - \mathbf{u}_{i,j-1}\right), \end{aligned} \tag{2.56}$$

where Δx and Δy are the inverse cell sizes in x and y direction respectively. Here, we apply the minmod function element-wise.

We use the slope to reconstruct the value at the cell boundaries. In the following, $\mathbf{u}_{i,j}^{\pm x}$ is the average of the left (+) or right (−) cell boundary. Similar, $\mathbf{u}_{i,j}^{\pm y}$ is the value at the top and bottom cell boundary. These so called boundary extrapolated values are given by

$$\begin{aligned} \mathbf{u}_{i,j}^{-x} &= \mathbf{u}_{i,j} - \frac{1}{2}\mathbf{s}_{i,j}^{x}, \qquad \mathbf{u}_{i,j}^{+x} = \mathbf{u}_{i,j} + \frac{1}{2}\mathbf{s}_{i,j}^{x}, \\ \mathbf{u}_{i,j}^{-y} &= \mathbf{u}_{i,j} - \frac{1}{2}\mathbf{s}_{i,j}^{y}, \qquad \mathbf{u}_{i,j}^{+y} = \mathbf{u}_{i,j} + \frac{1}{2}\mathbf{s}_{i,j}^{y}. \end{aligned} \tag{2.57}$$

We also use the slopes defined in eq. (2.56) to estimate the gradient of $\mathbf{u}_{i,j}$ in each cell by the block matrix

$$\nabla \mathbf{u}_{i,j} = \left(\mathbf{s}_{i,j}^{x} \middle| \mathbf{s}_{i,j}^{y} \right), \tag{2.58}$$

which is of course an abuse of notation insofar as it is not the gradient of the constant cell value but rather of the linear reconstruction (eq. 2.56). To achieve second order in time, we evolve the boundary extrapolated values

Time evolution

$$\begin{aligned} \forall (k \in \{-x, +x, -y, +y\}): \quad \hat{\mathbf{u}}_{ij}^{k} = \mathbf{u}_{i,j}^{k} \\ + \frac{\Delta t}{2\Delta x} \left[\mathbf{F}_x(\mathbf{u}_{i,j}^{-x}, \nabla \mathbf{u}_{i,j}) - \mathbf{F}_x(\mathbf{u}_{i,j}^{+x}, \nabla \mathbf{u}_{i,j}) \right] \\ + \frac{\Delta t}{2\Delta y} \left[\mathbf{F}_y(\mathbf{u}_{i,j}^{-y}, \nabla \mathbf{u}_{i,j}) - \mathbf{F}_y(\mathbf{u}_{i,j}^{+y}, \nabla \mathbf{u}_{i,j}) \right], \end{aligned} \tag{2.59}$$

where $\mathbf{F}_x$ and $\mathbf{F}_y$ denote the x and y part of the flux respectively. Note that we do not need to consider neighboring cells for this step.

Finally, the update can be described by

$$\begin{aligned} \mathbf{u}_{i,j}(t^{n+1}) = \mathbf{u}_{i,j}(t^{n}) \\ + \frac{\Delta t}{\Delta x} \text{Riemann}\left(\hat{\mathbf{u}}_{i-1,j}^{+x}, \nabla \mathbf{u}_{i-1,j};\ \hat{\mathbf{u}}_{i,j}^{-x} \ , \nabla \mathbf{u}_{i,j} \right) \\ - \frac{\Delta t}{\Delta x} \text{Riemann}\left(\hat{\mathbf{u}}_{i,j}^{+x} \ , \nabla \mathbf{u}_{i,j} \ ;\ \hat{\mathbf{u}}_{i+1,j}^{-x}, \nabla \mathbf{u}_{i+1,j} \right) \\ + \frac{\Delta t}{\Delta y} \text{Riemann}\left(\hat{\mathbf{u}}_{i,j-1}^{+y}, \nabla \mathbf{u}_{i,j-1};\ \hat{\mathbf{u}}_{i,j}^{-y} \ , \nabla \mathbf{u}_{i,j} \right) \\ - \frac{\Delta t}{\Delta y} \text{Riemann}\left(\hat{\mathbf{u}}_{i,j}^{+y} \ , \nabla \mathbf{u}_{i,j} \ ;\ \hat{\mathbf{u}}_{i,j+1}^{-y}, \nabla \mathbf{u}_{i,j+1} \right). \end{aligned} \tag{2.60}$$

This results in a scheme that has an order of convergence of two for both space and time. We treat the source term by a simple splitting scheme. The interested reader is referred to the discussions in [LeV02; Toro09].

We again use the Rusanov-type Riemann solver (eq. 2.50). The timestep is subject to a penalty of the form

$$\Delta t \leqslant \text{CFL} \frac{h}{|\lambda_c^{\max}| + 2|\lambda_\nu^{\max}| \frac{1}{h}}, \tag{2.61}$$

with a constant value of CFL = 0.9.

For reasons of simplicity, we only prescribe the solution at the boundary and then estimate the gradient with eq. (2.58). This has the effect that the scheme becomes unstable in the case of strong viscous effect next to the boundaries.

2.4. Finite Volume Limiting & Adaptive Mesh Refinement

This section is concerned with two different but related concepts: finite volume limiting and adaptive mesh refinement. Both have in common that they change the structure of the grid. Instead of a regular grid where each cell has the same size, we now have different cell sizes and even different numerical schemes.

2.4.1. Finite Volume Limiting

Higher order DG methods cannot cope with discontinuous solutions. Even worse, in the case of non-linear fluxes, discontinuities can appear even from smooth initial data. In addition, oscillations stemming from the high-order polynomial approximation can lead to wrong or unphysical data. This is called Gibbs phenomenon. A classical way of dealing with this problem is the usage of so called limiters. The idea is to smooth out steep gradients and thus lessen the effect of discontinuities [HW08].

A relatively recent way of dealing with this problem is the finite volume sub-cell limiter. Our discussion follows the one in [DL16]. This limiting is an a posteriori limiting, which means that we first evaluate a timestep with an unlimited ADER-DG-method and then check whether our solution is "correct". In our case, we check whether the solution is a finite floating point number. We also check whether it is physically admissible with the criterion

$$\text{is-admissible}(\mathbf{Q}) = \begin{cases} \text{true} & \rho > 0 \wedge p > 0 \wedge Z \geqslant 0 \\ \text{false} & \text{otherwise.} \end{cases} \tag{2.62}$$

The positivity of the pressure implies positivity of energy.

To combat the Gibbs phenomenon, we can make use of the discrete maximum principle (DMP) [DL16]. This criterion marks cells as troubled that have a numerical solution that is significantly higher or lower than the solutions of its neighbors, which are denoted by the set $\mathcal{N}$. Here, a cell is defined to be its own neighbor. We check for each cell C_i whether

$$\forall \mathbf{x} \in C_i : \min_{\mathbf{y} \in \mathcal{N}_i} \left(\mathbf{u}^{t^k}(\mathbf{y}, t^k) \right) - \delta \leqslant \mathbf{u}^{t^{k+1}}(\mathbf{x}, t^{k+1}) \leqslant \max_{\mathbf{y} \in \mathcal{N}_i} \left(\mathbf{u}(\mathbf{y}, t^k) \right) + \delta \tag{2.63}$$

with tolerance

$$\delta = \max \left(\varepsilon_0, \varepsilon \left(\max_{\mathbf{y} \in \mathcal{N}} \left(\mathbf{u}^{t^k}(\mathbf{x}, t^k) \right) - \min_{\mathbf{y} \in \mathcal{N}} \left(\mathbf{u}^{t^k}(\mathbf{x}, t^k) \right) \right) \right) \tag{2.64}$$

holds. The parameters ε and ε_0 can be set freely. The former controls by which factor the local jump can differ from its neighborhood, the latter is needed for zero jumps [DL16]. Note that this criterion is only a heuristic and is susceptible to false positives.

If a solution violates any of our criteria, we recompute the last timestep for the offending cells with a limited finite volume method. In our case we use the MUSCL-Hancock-method (section 2.3). The recomputation starts by subdividing the infringing cell into $N_s = 2N + 1$ sub-cells. This number of sub-cells has the advantage that we do not lose spatial resolution and that we can use the same timestep size for both cell types. The reason for this is that the CFL-conditions for both schemes (eqs. 2.54 and 2.61) coincide for our choice of N_s. Figure 2.1 depicts the mapping from DG-cells to finite volume sub-cells. For the sake of brevity, we omit technical details and refer the interested reader to [DL16].

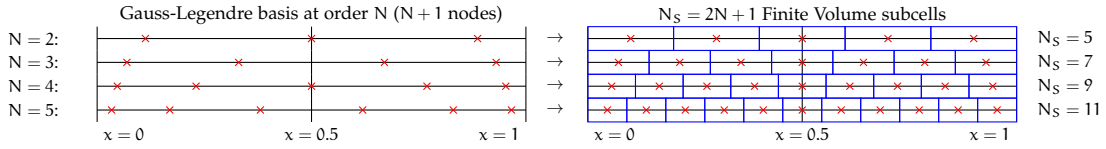


Figure 2.1.: Conversion from DG degrees of freedoms to finite volume degree of freedoms. Image taken from [Dum+18b]. We changed the typeface of the figure.

2.4.2. Adaptive Mesh Refinement

The goal of adaptive mesh refinement (AMR) is to increase the computational efficiency by using a coarse spatial resolution in areas with low information content and a higher resolution in areas of interest. A simple example for this would be using smaller mesh cells for the simulation of a wave, while using larger cells for areas where the wave does not pass through. Of course, the difficult part is recognizing interesting areas. We use a global, gradient based refinement strategy. We first describe the computation of our indicator variable and then describe a simple way how one can detect outliers.

Total variation

Let $f(\mathbf{x}) : \mathbb{R}^{N_{vars}} \to \mathbb{R}$ be a function that maps the discrete solution of a cell to an arbitrary indicator variable. We assume that f is expanded in the space-basis (eq. 2.43). The total variation (TV) of this function is defined by

$$\mathrm{TV}\left[f(\mathbf{x})\right] = \| \int_C |\nabla f(\mathbf{x})| \, d\mathbf{x} \|_1 \tag{2.65}$$

for each cell. The operator $\| \cdot \|_1$ denotes the discrete l_1 norm in this equation. This integral can be evaluated efficiently on the reference cell (eqs. 2.23 and 2.43) with Gaussian quadrature (eq. 2.33).

Outliers

A simple way of determining outliers is to compute the mean and standard deviation

Algorithm 1 Merging two sets of reduced mean and variance [CGL82]

function REDUCE-VARIANCE(G_0, G_1)
  if $n_0 = 0$ **then**
    return μ_1, σ_1^2, n_1
  if $n_1 = 0$ **then**
    return μ_0, σ_0^2, n_0
  $\Delta \leftarrow \mu_1 - \mu_0$
  $n_\Sigma \leftarrow n_0 + n_1$
  $m_a \leftarrow \sigma_0^2(n_0 - 1)$
  $m_b \leftarrow \sigma_1^2(n_1 - 1)$
  $m_\Sigma \leftarrow {}^{m_a + m_b + (\Delta^2 n_0 n_1)}/_{n_\Sigma}$
  $\mu_\Sigma \leftarrow \mu_0 n_0 + \mu_1 n_1$
  return ${}^{\mu_\Sigma}/_{n_\Sigma}, {}^{m_{\text{total}}}/_{n_\Sigma - 1}, n_\Sigma$

for a set. Any value that differs by a certain multitude of the standard deviation from the mean can be marked as an outlier. We thus want to compute the mean and variance over all grid cells. For computational efficiency and to simplify the implementation, we do this by performing only pairwise merge operations. The simplest way of computing this is by the textbook definitions of mean μ and population variance σ^2 of a random variable X

$$\begin{aligned} \mu[X] &= \mathbb{E}[X], \\ \sigma^2[X] &= \mathbb{E}\left[X^2\right] - \mathbb{E}[X]^2, \end{aligned} \tag{2.66}$$

where $\mathbb{E}[X]$ denotes the expectation of a random variable X. This naive algorithm is numerically unstable when the variance is orders of magnitudes smaller than the mean, as this can lead to catastrophic cancellations. We thus compute the mean and variance with the parallel algorithm of [CGL82]. To do this, we need to store three variables per computation unit: the mean μ, the current sample variance $\overline{\sigma^2}$ and the number of processed elements n. We collect these items in the vector $G = (\mu, \overline{\sigma^2}, n)$. We can then merge a pair of observed variables with algorithm 1. This algorithm computes an estimate of the population variance using Bessel's correction, i.e. it returns the sample variance times ${}^{n}/_{n-1}$. In our case, the compute the variance over all grid cells. This is why we compute the population standard deviation by

$$\sigma = \sqrt{\frac{n-1}{n}\overline{\sigma^2}}. \tag{2.67}$$

Chebyshev's inequality

$$\mathbb{P}(|X - \mu| \geqslant c\sigma) \leqslant \frac{1}{c^2}, \tag{2.68}$$

where $\mathbb{P}$ is the probability of an event, holds for arbitrary distributions with mean μ and standard deviation σ [Was04]. This motivates the following refinement criterion

$$\text{evaluate-refinement}(\mathbf{Q}, \mu, \sigma) = \begin{cases} \text{refine} & \text{if } \mathrm{TV}(\mathbf{Q}) \geqslant \mu + \mathsf{T}_{\text{refine}}\sigma \\ \text{delete} & \text{if } \mathrm{TV}(\mathbf{Q}) < \mu + \mathsf{T}_{\text{delete}}\sigma \\ \text{keep} & \text{otherwise} \end{cases} \tag{2.69}$$

which refines a cell when its total variations differs by a multiple of the standard deviation. In this equation, the constants $\mathsf{T}_{\text{refine}}$ and $\mathsf{T}_{\text{delete}}$ can be used to tailor the trade-off between the quality of approximation and computational cost. They can be chosen freely, as long as $\mathsf{T}_{\text{refine}} < \mathsf{T}_{\text{delete}}$. Chebyshev's inequality (eq. 2.68) then guarantees that only a subset of cells is marked for refinement. Note that this inequality provides only a loose bound. If one would be willing to assume a distribution of the indicator variable, tighter bounds can be derived [Was04].

CHAPTER 3

Implementation

This chapter is organized as follows: We first give a short overview of the general structure of the *ExaHyPE* code and then describe the changes done over the course of this project. *ExaHyPE* implements an ADER-DG solver with MUSCL-Hancock limiting and AMR, as described in chapter 2. It uses *Intel® Threading Building Blocks* (TBB) for shared memory parallelism and the *Message Passing Interface* (MPI) for distributed memory parallelism. In the following, we define a "user" as someone who implements the logic of a PDE but does not change the implementation of *ExaHyPE* itself. This is in contrast with a "developer" whose scope is the entirety of the source code. The goal of our implementation is that all new features should be accessible to users as well.

3.1. ExaHyPE

ExaHyPE is based on the space-filling-curves library *Peano* [WM11]. The workflow from the perspective of a user is:

- Generate glue-code using a *Python*-based Toolkit that inserts parameters into templates. The parameters are configured in a setting file. The user can configure the number of dimensions, numerical order and type of solver (ADER-DG, MUSCL-Hancock, or limiting-ADER-DG). Depending on the solver type, a different implementation is generated. In case of a limiting solver, a finite-volume and a DG implementation is generated. An implementation consists of
  - An `AbstractSolver` that is a subclass of `FiniteVolumesSolver`, `ADERDGSolver` or `LimitingADERDGSolver`. The solver types all have a shared superclass `Solver` which describes a common interface.
  - A class that inherits from the aforementioned abstract class that implements the PDE logic.
- The user then needs to modify the generated `Solver` class. The PDE is specified in the methods flux and eigenvalues. We implement the grid refinement criterion in refinementCriterion and the limiting criterion in isPhysicallyAdmissible. To close the system, the user needs to prescribe initial and boundary conditions with

adjustPointSolution, boundaryConditions and boundaryValues. The methods boundaryConditions first calls boundaryValues and then solves the Riemann-problem at the boundary. Thus, it suffices in many cases to only override the latter method.

- Finally, the user can specify more options such as mesh size and spacing, simulation time, parallelization options, plotters, and optimization parameters. After compiling the program, it can be run with the setting file as first and only parameter.

From the perspective of a developer, the situation is more complicated. We will thus only give the necessary minimum of information that is needed to understand the modifications conducted by us. *ExaHyPE* is structured as a *Peano* project and follows the basic structure:

- A runner iterates over the grid and calls an adapter for so-called events. An example for an event is entering a cell.
- An adapter calls one or multiple mappings.
- A mapping then performs the actual program logic. For our purposes, we only need to consider the mappings (with corresponding adapters):
  - `FinalizeMeshRefinement`
  - `FusedTimeStep`
- Often, a mapping calls a method defined in `Solver`. The implementation of this method can differ between different solvers.
- The actual numerical logic is outsourced to kernels.

Furthermore, the degrees of freedom and some meta-information are stored for each cell in a heap data-structure. The information for each DG-cell is defined in the file **ADERDGCellDescrition.def**. Similar files exist for the finite volume solver but are not relevant for us. These files are converted to C++ files by the software *DaStGen* [Bun+08].

3.2. Implementing Numerics

We needed to modify the existing implementation of the mappings, solvers and kernels to be able to solve problems with diffusive fluxes. These changes are:

- Modifying the space-time-predictor (defined in file **spaceTimePredictorNonlinear.cpph**).

  - The function aderPicardLoopNonlinear implements the space-time-predictor loop (eq. 2.45). We pass the gradient of the conserved variables to the flux method of the `Solver`. Because we need the time-average of the gradient later, we return it.
  - In the function aderExtrapolatorNonlinear, which implements the boundary extrapolation, we also compute the extrapolated gradient.
  - In the function spaceTimePredictorNonlinear, we also return the extrapolated gradient.

- Storing the extrapolated and time-averaged gradient for each cell for use in the boundary conditions and the corrector step.
- Using the gradient for the computation of the boundary conditions. They are defined in the file **boundaryConditions.cpph**.
- Adding the diffusive penalty terms for the Riemann solver (**riemannSolverNonlinear.cpph**) and CFL-condition (**stableTimeStepSize.cpph**). This corresponds to eqs. (2.50) and (2.54).
- Computing the gradient for the MUSCL-Hancock scheme (**musclhancock.cpph**) and passing it to the flux (**rusanov.cpph**).

Additionally, we needed to change the templates for the glue code and needed to modify some function signatures to accompany the other changes.

3.3. Implementing Global Observables

The implementation of the global observables is quite simple. We use three functions that need to be defined by the user. They adhere to the interface:

- The function mapGlobalObservables computes the observables for a cell from its degrees of freedoms and its size. It returns a vector of size N^{gobs}.
- The function reduceGlobalObservables updates the current partially reduced vector of global observables "reducedGlobalObservables" with another vector of partially reduced observables "curGlobalObservables".
- The function resetGlobalObservables takes no arguments and returns a vector of size N^{gobs}. This vector should be the identity of the reduction, i.e. a value a with the property that for all b $\mathrm{reduce}(a, b) = b$.

Algorithm 2 Reducing global observables

observables ← RESETGLOBALOBSERVABLES
for cell ∈ cells **do**
  curObservables ← MAPGLOBALOBSERVABLES($\mathbf{Q}_{\text{cell}}, \Delta x_{\text{cell}}$)
  observables ← REDUCEGLOBALOBSERVABLES(observables, curObservables)
return observables

We then compute the observables by algorithm 2 This results in a simple, yet powerful interface, which can be used to implement all reductions that can be performed in a pair-wise manner. Examples are computing the minimum, maximum, sum, mean and variance.

For the variance of the total variation, we store the partial mean, sample variance, and count as global observables. The vector $(-1, -1, 0)$ is then the base-case for the iteration algorithm 2 and is thus returned by the function resetGlobalObservables. The method mapGlobalObservables is defined to return $(\text{TV}[f(\mathbf{Q})], 0, 1)$, where $f(\mathbf{Q})$ returns an arbitrary indicator value per cell. Finally, reduceGlobalObservables is implemented exactly as described in algorithm 1.

For the actual implementation, we follow algorithm 2 but include distributed and shared memory parallelization in the following way:

- Adding vectors "_globalObservables" and "_nextGlobalObservables" to all solvers. The former stores the observables of the previous timestep, the latter the ones of the current timestep. We can thus use the old observables for mesh refinement and compute a new reduction at the same time. Before the first timestep, we reset both current and next observables. At the beginning of each timestep, we overwrite the previous data with the current data, and reset "_nextGlobalObservables" with resetGlobalObservables. This is done in the methods startNewTimeStep, startNewTimeStepFused, updateTimeStepSizesFused and updateTimeStepSizes.

- Reducing global observables in solver with method reduceGlobalObservables that takes a "cellInfo" and a "solverNumber". The solver number is always 0 because we only use one solver at a given time. The cell information stores pointers to the degrees of freedoms. We implement this method for all solvers. This has the advantage that we have a shared interface for all solver types.

- Reducing global observables per node in the mappings `FinaliseMeshRefinement`, `FusedTimeStep` and `UpdateAndReduce`.
  - We initialize the observables in the method beginIteration with resetGlobalObservables.

  - We reduce the observables in leaveCell by calling the solver method reduceGlobalObservables with the "cellInfo" of the current cell as argument. For the mapping `UpdateAndReduce`, this is done in the method enterCell instead.
  - If we use TBB, we merge the partially reduced observables in mergeWithWorkerThread with reduceGlobalObservables using the current reduced observables and the partially reduced observables from the worker. This is not necessary for the mapping `FinaliseMeshRefinement`.
  - We update the global observables of the solver in the method endIteration, which calls the method updateGlobalObservables.

- Reducing the observables over all ranks, if using MPI. *ExaHyPE* already reduces the timestep size. We extend these already existing MPI messages to also include the global observables "_globalObservables". For this, we changed the following methods of the DG-solver:
  - compileMessageForMaster and sendDataToMaster to send data from the workers to the master rank.
  - mergeWithWorkerData to merge the received worker data with the current master data.
  - compileMessageForWorker and sendDateToWorker to send the reduced data from the master to the workers.
  - mergeWithMasterData to overwrite the current data of the workers with the reductions from the previous timestep.

  The solvers update the variable "_nextGlobalObservables" during the MPI-reduction. This again corresponds to algorithm 2 together with the aforementioned resetting of the observables,

The global observables interface is defined for all solvers, the MPI-reduction only for the ADER-DG-solver.

CHAPTER 4

Scenarios

In this chapter, we introduce various scenarios that we will later use to to evaluate the performance of our method. Table 4.1 shows an overview of our scenarios. They all have a different focus:

- The manufactured solution is an exact solution for our equation set and can be thus used to reliably evaluate the quality of our implementation.
- The Taylor-Green vortex, the ABC-flow and the lid-driven cavity flow are standard CFD testing scenarios for which we have a reference solution.
- The bubble scenarios are simple examples for flows over a realistic background atmosphere.
- The CJ detonation wave uses the reactive advection term to model a detonation.

Unless otherwise noted, we use the constants

$$\gamma = 1.4, \quad \mathrm{Pr} = 0.7, \quad c_v = 1.0. \tag{4.1}$$

The other constants follow from the definition of eq. (2.9).

We describe some scenarios in terms of the primitive variables $(\rho, \boldsymbol{v}, p)$, these can be converted to the conservative variables by the definition of momentum and the equation

Table 4.1.: Overview of scenarios

Name	Analytic Solution	Gravity	Coupling	Limiting
Manufactured solution [Dum10]	✓	✘	✘	✘
Taylor-Green vortex [Dum+16]	✓	✘	✘	✘
ABC-flow [TD16]	✓	✘	✘	✘
Lid driven cavity [FDZ17]	✘	✘	✘	✓
Cosine bubble [GR08]	✘	✓	✘	✘
Two bubbles [Rob93; Mül+10]	✘	✓	✘	✘
CJ detonation [HLW00; HD11]	✘	✘	✓	✓

of state eq. (2.5). All scenarios, with the exception of the CJ-detonation wave, use only the compressible Navier-Stokes equations and do not use the advection equation. This corresponds to setting $\rho Z = 0$ and $q_0 = 0$.

4.1. A Manufactured Solution

The compressible Navier Stokes equations have no interesting exact solution. This is why we follow the method of manufactured solutions [Roa02] and create our own. Following [Dum10], we assume that

$$\begin{aligned} p(\mathbf{x}, t) &= p_0 \cos(\mathbf{k}\mathbf{x} - \omega t) + p_b, \\ \rho(\mathbf{x}, t) &= \rho_0 \sin(\mathbf{k}\mathbf{x} - \omega t) + \rho_b, \\ \boldsymbol{v}(\mathbf{x}, t) &= \boldsymbol{v}_0 \sin(\mathbf{k}\mathbf{x} - \omega t). \end{aligned} \tag{4.2}$$

We use the constants

$$\begin{aligned} p_u &= 1/10, & p_b &= 1/\gamma, \\ \rho_0 &= 1/2, & \rho_b &= 1, \\ \boldsymbol{v}_0 &= 1/4, \begin{pmatrix} 1 \\ 1 \end{pmatrix}, & \mathbf{k} &= \pi/5 \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \\ \omega &= 2\pi. \end{aligned} \tag{4.3}$$

We derive a source term by inserting this solution into the PDE (eq. 2.1) by using the symbolic math toolkit *Sage* [The18]. Our computation of the source term can be found at [Kre18]. In combination with the derived source term, eq. (4.2) then is an exact solution of our PDE.

Note that we only consider the compressible Navier Stokes equations here, without the reactive advection equation.

4.2. Classical Scenarios

In addition to the convergence test, we use standard CFD scenarios to evaluate our simulations.

Taylor-Green vortex

An interesting testing scenario is the Taylor-Green vortex, which exhibits a complicated flow from a simple initial condition. It is an analytic solution for the incompressible Navier-Stokes equations which agrees relatively well with our equation set at low Mach numbers. We can thus use it to evaluate the performance of our solver. The

solution is

$$\begin{aligned} \rho(\mathbf{x}, t) &= 1, \\ \boldsymbol{v}(\mathbf{x}, t) &= \exp(-2\mu t) \begin{pmatrix} \sin(x)\cos(y) \\ -\cos(x)\sin(y) \end{pmatrix}, \\ p(\mathbf{x}, t) &= \exp(-4\mu t)\, 1/4\, (\cos(2x) + \cos(2y)) + C, \end{aligned} \tag{4.4}$$

where $C = 100/\gamma$ is a constant that governs the speed of sound and thus the Mach number [Dum+16].

We use a viscosity of $\mu = 0.1$ and Cauchy boundary conditions.

ABC-flow

We use the Arnold–Beltrami–Childress (ABC) flow as a simple example of a three-dimensional scenario [TD16]. Unlike the three-dimensional Taylor-Green vortex, the ABC-flow has the analytical solution

$$\begin{aligned} \rho(\mathbf{x}, t) &= 1, \\ \boldsymbol{v}(\mathbf{x}, t) &= \exp(-1\mu t) \begin{pmatrix} \sin(z) + \cos(y) \\ \sin(x) + \cos(z) \\ \sin(y) + \cos(x) \end{pmatrix}, \\ p(\mathbf{x}, t) &= -\exp(-2\mu t)\, (\cos(x)\sin(y) + \sin(x)\cos(z) + \sin(z)\cos(y)) + C \end{aligned} \tag{4.5}$$

in the incompressible limit. Here, $C = 100/\gamma$ is an arbitrary constant. We use a viscosity of $\mu = 0.01$ and Cauchy boundary conditions.

Lid-driven cavity

The lid-driven cavity flow is another simple testing scenario for the Navier-Stokes equations. The flow is driven entirely by the boundary conditions. They consist in four no-slip walls (eq. 2.18) where the upper wall has a constant velocity $\boldsymbol{v}^w = (1.0, 0)$. We want to simulate a flow with a Mach number of $M = 0.1$ and a Reynolds number of $Re = 100$ [FDZ17]. This leads us to the constant initial conditions of

$$\begin{aligned} \rho(\mathbf{x}) &= 1, \\ \boldsymbol{v}(\mathbf{x}) &= 0, \\ p(\mathbf{x}) &= 100/\gamma. \end{aligned} \tag{4.6}$$

This scenario is surprisingly difficult for an ADER-DG scheme as the boundary conditions can lead to a discontinuity at the corners, because the wall speed is discontinuous [FDZ17]. Therefore, this scenario is a good choice to test the stability of our scheme.

CJ detonation wave

We investigate the radialsymmetric Chapman-Jouget (CJ) detonation wave scenario from [HLW00]. This scenario for the reactive Navier Stokes equations is set up in a 2D box of size $[-1, 1] \times [-1, 1]$ surrounded by no-slip walls (eq. 2.18). We use the reactive source term (eqs. 2.16 and 2.17) and set the characteristic timescale to the non-stiff case

$\tau = 0.1$ and the threshold temperature to $T_{ign} = 0.26$. Inside a circle of radius 0.3 we have completely unburnt gas, outside of this circle we have burned gas. Let $\alpha = \text{atan2}(z, x)$ be the angle in polar coordinates. We use the initial conditions

$$\begin{aligned} \rho_u &= 0.887565 & \rho_b &= 1.4 \\ \boldsymbol{v}_u &= -0.577350 \begin{pmatrix} \cos(\alpha) \\ \sin(\alpha) \end{pmatrix} & \boldsymbol{v}_b &= \begin{pmatrix} 0 \\ 0 \end{pmatrix} \\ p_u &= 0.191709 & p_b &= 1 \\ Z_u &= 1 & Z_b &= 0 \end{aligned} \tag{4.7}$$

where a subscript of u and b denotes the unburnt and burnt state respectively. Note that this scenario is radialsymmetric.

Similar to [HLW00], we use the constants

$$\gamma = 1.4, \qquad Pr = 0.75, \qquad c_v = 2.5, \qquad c_p = 3.5, \qquad R = 1.0. \tag{4.8}$$

The Prandtl number is inspired from the viscous scenarios of [HD11]. We consider this scenario for the inviscid case and also for a viscosity of $\mu = 0.01$.

4.3. Atmospheric Scenarios

The following scenarios are all described in terms of a perturbation of a balanced atmosphere. This balanced atmosphere is called hydrostatic balance and is described by eq. (2.13). Before we are able to describe individual scenarios, we first need to describe how we can initialize the background atmosphere.

For this, the potential temperature θ

$$\theta = T \left(\frac{p_0}{p} \right)^{R/c_p}, \tag{4.9}$$

is useful. The constant $p_o = 10 \times 10^5$ Pa is the reference pressure. Solving this equation for T leads to

$$T = \theta (\frac{p}{p_0})^{R/c_p}. \tag{4.10}$$

To allow for an easier description of the initial conditions we split the potential temperature into a background state $\overline{\theta}$ and a perturbation θ'

$$\theta = \overline{\theta} + \theta'. \tag{4.11}$$

The initial conditions for pressure and density can be computed directly from the PDE given a few assumptions. We assume that $\overline{\theta}$ is constant over the entire domain [GR08].

Inserting the definition of potential temperature, given by eq. (4.10), into the equation of state (eq. 2.5) leads us to

$$p(z) = \rho(z) R \left(\overline{\theta} \frac{p}{p_0} \right)^{R/c_p}, \tag{4.12}$$

which we solve for $p(z)$. After some algebra and simplifying, we arrive at

$$p(z) = p_0 \left(\frac{R \overline{\theta} \rho(z)}{p_0} \right)^{\gamma}, \tag{4.13}$$

or equivalently

$$\rho(z) = \frac{p_0^{\frac{R}{c_p}} p^{\frac{1}{\gamma}}(z)}{R \overline{\theta}}. \tag{4.14}$$

We want to consider flow in hydrostatic equilibrium, i.e. a flow for which the pressure-gradient and gravitational force are exactly in balance [Mül+10]. Inserting this assumption in the momentum equation in z-direction and using eq. (2.5) once again, we arrive at the ordinary differential equation (ODE)

$$\begin{aligned} \frac{d}{dz} p(z) &= -g\rho(z) = -\frac{g p_0^{\frac{R}{c_p}} p^{\frac{1}{\gamma}}(z)}{R \overline{\theta}}, \\ p(0) &= p_0, \end{aligned} \tag{4.15}$$

where we use the reference pressure p_0 as the pressure on ground level. This ODE is the hydrostatic equilibrium (similar to eq. (2.13)) in terms of the potential temperature and can be solved by separation of variables. After simplifying, we obtain

$$p(z) = \left(\left(1 - \frac{1}{\gamma} \right) \left(C - \frac{g p_0^{\frac{R}{c_p}} z}{R \overline{\theta}} \right) \right)^{\frac{c_p}{R}}, \tag{4.16}$$

with constant of integration

$$C = \frac{c_p p_0^{\frac{R}{c_p}}}{R}. \tag{4.17}$$

We can now compute the initial conditions for a scenario with a background potential energy that is in hydrostatic equilibrium and a perturbation that is not. We first compute the background pressure with background potential temperature $\overline{\theta} = \text{const.}$ using eq. (4.16). Next, we compute the perturbed potential temperature θ and use eq. (4.10) to convert it to temperature. The density $\rho(x)$ can now be evaluated with the thermal equation of state eq. (2.6). Finally, we compute the energy by inserting pressure and density into the equation of state (2.5).

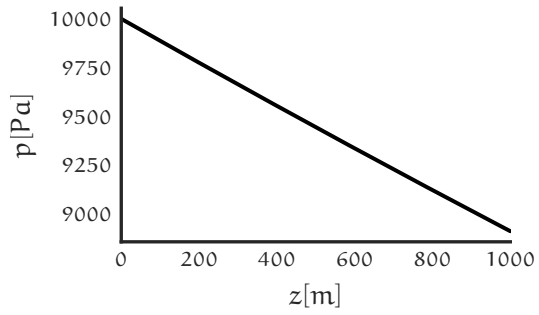


Figure 4.1.: Hydrostatic background pressure for constants eq. (4.19)

Hydrostatic wall

This leaves us with one problem: The adiabatic free-slip boundary conditions defined by eq. (2.19) are no longer valid. Concretely, we need to impose the viscous heat flux

$$F^{\nu}_{\rho E} = \kappa \frac{\partial \overline{T}}{\partial z} = \kappa \frac{R\overline{\theta}\left(\frac{\overline{p}(z)}{p_0}\right)^{\frac{R}{c_p}} \frac{d}{dz}\overline{p}(z)}{c_p \overline{p}(z)} \tag{4.18}$$

to ensure that the atmosphere stays in hydrostatic balance [GRo8]. The background pressure can be reconstructed by eq. (4.16), its derivative is given by eq. (4.15). We set the other parts of the viscous flux to zero.

All scenarios use the gravitational source term (eq. 2.12) and the splitting of density and pressure (eq. 2.15). We use the following set of constants for these scenarios

$$\gamma = 1.4, \qquad Pr = 0.71, \qquad R = 287.058, \qquad p_0 = 10000, \qquad g = 9.81, \tag{4.19}$$

the other constants are evaluated according to eq. (2.9). The background pressure for these constants is depicted in figure 4.1.

Cosine bubble

As a simple test case, we consider the cosine bubble scenario which describes a background atmosphere that is perturbed by a large warm air bubble in the middle of the domain [GRo8]. We use a background potential temperature of $\overline{\theta} = 300$ and a domain of $1\,\mathrm{km} \times 1\,\mathrm{km}$. The background atmosphere is assumed to be in hydrostatic equilibrium. Let

$$r^2 = (x - x_0)^2 + (z - z_0)^2 \tag{4.20}$$

denote the difference between spatial positions x, z and the center of a bubble x_c, z_c. The perturbation is given by the function

$$\theta' = \begin{cases} A/2\,[1 + \cos(\pi r)] & r \leqslant a, \\ 0 & r > a, \end{cases} \tag{4.21}$$

where A denotes the maximal perturbation and a is the size of the bubble. We use the constants

$$A = 0.5 \quad a = 250\,\mathrm{m} \quad x_c = 500\,\mathrm{m} \quad z_c = 350\,\mathrm{m}. \tag{4.22}$$

We use a constant viscosity of $\mu = 0.1$.

Two bubbles

The two bubbles scenario consists of a small cold air bubble that is on top of a large warm air bubble [Rob93; Mül+10]. We use the same procedure as before to compute the initial conditions, again with $\overline{\theta} = 300$ but with a perturbation of the form

$$\theta' = \begin{cases} A & r \leqslant a, \\ A \exp\left(-\frac{(r-a)^2}{s^2}\right) & r > a, \end{cases} \tag{4.23}$$

where s is the decay rate and r is the radius to the center (eq. 4.20). We have two bubbles, with constants

$$\begin{array}{llllll} \text{warm:} & A = 0.5, & a = 150\,\mathrm{m}, & s = 50\,\mathrm{m}, & x_c = 500\,\mathrm{m}, & z_c = 300\,\mathrm{m}, \\ \text{cold:} & A = -0.15, & a = 0\,\mathrm{m}, & s = 50\,\mathrm{m}, & x_c = 560\,\mathrm{m}, & z_c = 640\,\mathrm{m}. \end{array} \tag{4.24}$$

This scenario is interesting because we have small scale details that are important for the resulting flow. Similar to [Mül+10] and to the previous scenario, we use a constant viscosity of $\mu = 0.1$ to regularize the solution.

CHAPTER 5

Results

We discussed various scenarios in the previous chapter. In this chapter, we use them to gauge the quality of our numerical scheme. We start with a convergence test, followed by three standard testing scenarios. Next, we simulate two different cloud scenarios and check whether our AMR criterion is useful.

5.1. Convergence Test

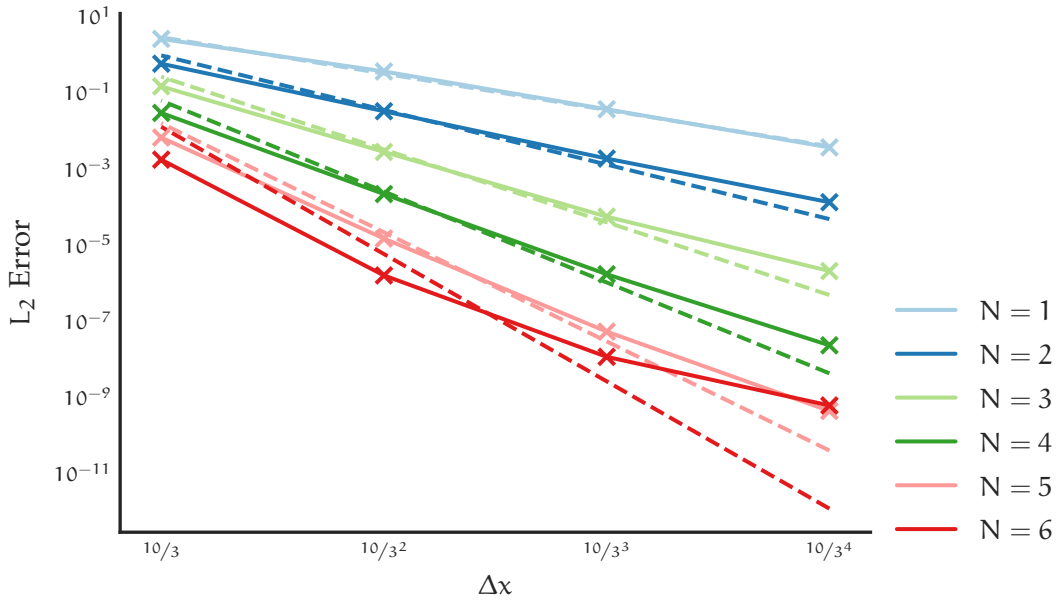


Figure 5.1.: L_2-Error vs. Grid Size. The dashed lines are the optimal order of convergence $N + 1$. The line is estimated by a linear regression fit and setting the slope to the ideal order.

For the convergence test, we run our manufactured solution scenario (section 4.1) for all orders 1 to 6 and for number of cells $3^2, 9^2, 27^2$ and 81^2.

Table 5.1.: Numerical order of convergence of ADER-DG method

Polynomial Order N	L_1	L_2	L_∞
1	2.03	2.00	1.92
2	2.56	2.55	2.55
3	3.43	3.40	3.44
4	4.27	4.27	4.36
5	5.00	5.02	5.08
6	4.46	4.50	4.65

In the following, we describe the error for one variable. We define the total error as the sum of the error of all conservative variables. Before computing the error, we first need to define the L_p norms for a $p > 0$ by

$$\|f(\mathbf{x})\|_p = \left(\int_\Omega |f(\mathbf{x})|^p \, d\mathbf{x} \right)^{1/p}. \tag{5.1}$$

We are interested in the total error which we define as the norm of the difference between an analytical solution $f(\mathbf{x}, t)$ and our approximation $\mathbf{u}(\mathbf{x}, t)$. The error at a time t is thus defined as

$$\text{Error}(t, p) = \|f(\mathbf{x}, t) - \mathbf{u}(\mathbf{x}, t)\|_p. \tag{5.2}$$

We first observe that for our approximation space Ω we can split up the error as a sum over all cells. We evaluate the cell-wise error with Gaussian quadrature (eqs. 2.27 and 2.33)

$$\text{Error}(t, p) = \left(\sum_{C_i \in \Omega} \left(V \sum_{\hat{\mathbf{x}}_j \in \hat{C}_i} |f^{C_i}(\hat{\mathbf{x}}_j, t) - \mathbf{u}^{C_i}(\hat{\mathbf{x}}_j, t)|^p \, w_j \right) \right)^{1/p}. \tag{5.3}$$

Here, a superscript of C_j denotes the restriction of the function to a grid cell. To increase the quadrature quality, we evaluate both analytical and approximate solution at the quadrature nodes of a scheme with 10 quadrature nodes $\hat{x}_j$ with weights w_j. We then use these nodes to compute the approximation error per cell.

For the limiting case of $p \to \infty$ we use the maximum of the point-wise error over all nodal points.

To compute the numerical order of convergence, we perform a linear regression of the logarithm of the error vs. the logarithm of the mesh size. The size of the slope is then the convergence order. The plot for the L_2 error (figure 5.1) shows that our method converges for all tested polynomial orders. Note that the error for the finest grid with order 6 is worse than the error for order 5. For all other mesh sizes and orders,

a higher order implies a smaller error. Comparing our errors with the optimal order of convergence $N + 1$, we notice that while our scheme exhibits the theoretical order of convergence of 2 for a polynomial order $N = 1$, larger orders exhibit diminishing returns.

The same is true for other error norms (table 5.1). This means, that we do not achieve the optimal theoretical order of convergence in general. In [Dum10], which uses a similar numerical method and the same scenario, but a different grid, Dumbser achieves the optimal order for most polynomial orders. Some DG-methods of even polynomial order only achieve a numerical convergence order of $N + 1/2$. The exact theoretical reason for this is unclear but this phenomenon is similar to the behavior of other DG-schemes. We refer the interested reader to the discussion of error estimates for nonlinear problems and for multi-dimensional problems with DG-schemes of [HW08].

5.2. Simulation of CFD Scenarios

After we established that our scheme converges, we evaluate it using standard CFD testing scenarios that either have an analytical solution or for which we have a reference solution.

Taylor-Green vortex

We used an ADER-DG-scheme of order 5 with a mesh of size 27×27 cells to simulate the Taylor-Green Vortex (eq. 4.4). The simulation ran for 10 s. Our solution resulted in the expected streamlines (figure 5.2). Furthermore, we compare our results with the analytical solution for the incompressible case. We achieved a small error (figure 5.3) for both pressure and velocity. The small errors are expected and not only stem from numerical inaccuracies but primarily from the fact that we simulated a compressible fluid. Note that the error is small over the entire domain, and not only close to the boundary, where we imposed the analytical solution. Overall, we notice a very good agreement with both the analytical solution and the results of [Dum+16].

ABC-flow

We used an ADER-DG-scheme of order 2 with a mesh consisting of $27 \times 27 \times 27$ cells to simulate the ABC-flow (eq. 4.5). We ran the simulation for 1 s. Our solution for the pressure shows an excellent agreement (figure 5.4) with the analytical solution. We thus show that our implementation can also handle three-dimensional scenarios.

Lid-driven cavity

We used a MUSCL-Hancock-limited ADER-DG-scheme of order 3 with a mesh of size 27×27 and a simulation time of $t = 30$ s for the lid-driven cavity (eq. 4.6). The streamlines (figure 5.5a) agree with the expected solution but have two smaller, secondary vortices in the bottom corners. They do not appear in the plots of [Dum10] but do appear in [GGS82]. Additionally, we show the velocity along both axes in figure 5.5b. They too agree with the reference solution of [GGS82].

Surprisingly, the limiter was not activated at all. We noticed that we need the limiter when we use a lower speed of sound and thus have a flow with larger Mach number.

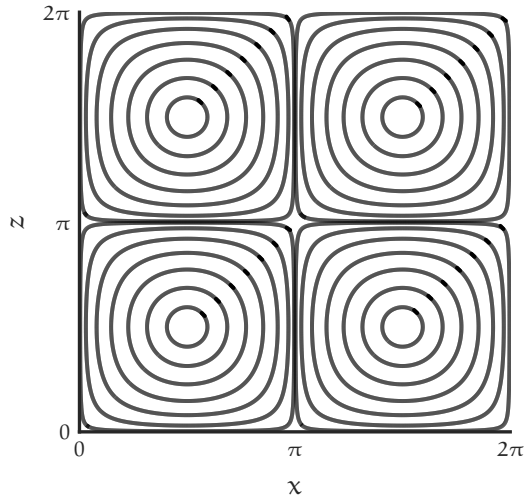
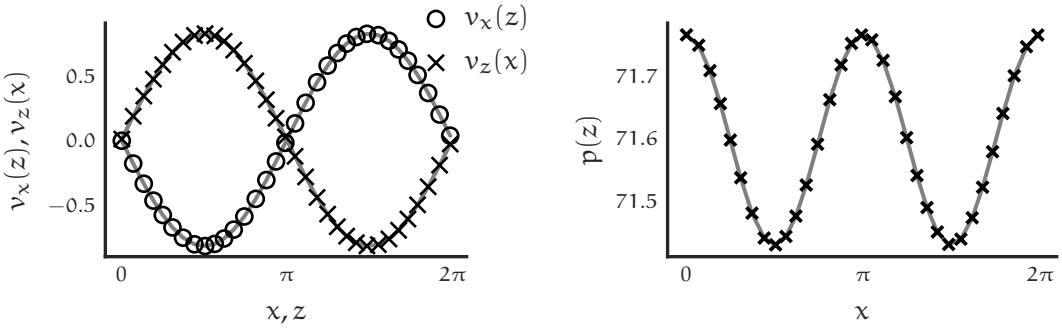


Figure 5.2.: Streamlines for Taylor-Green vortex at $t = 10\,s$



(a) Position vs. velocity

(b) Position vs. pressure

Figure 5.3.: Comparison of Taylor-Green simulation with analytical solution (eq. 4.4) at $t = 10\,s$. The former is represented by the scatter plot, the latter by the gray line. Note that we only show every 5th point. The other variable of the slice is held constant at a value of zero.

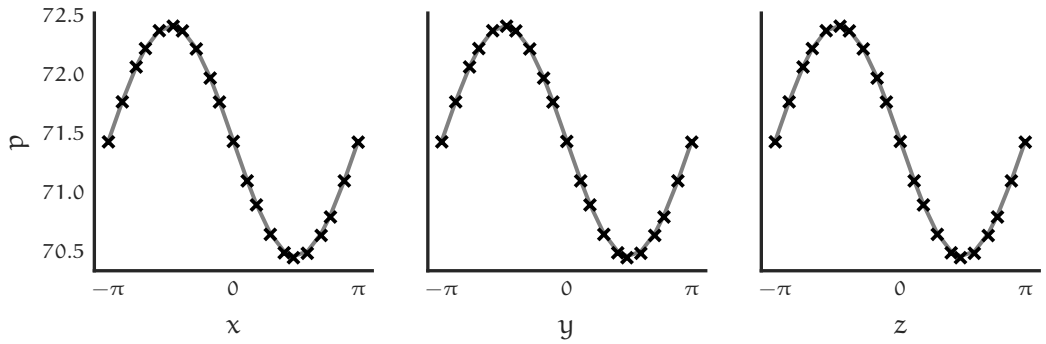
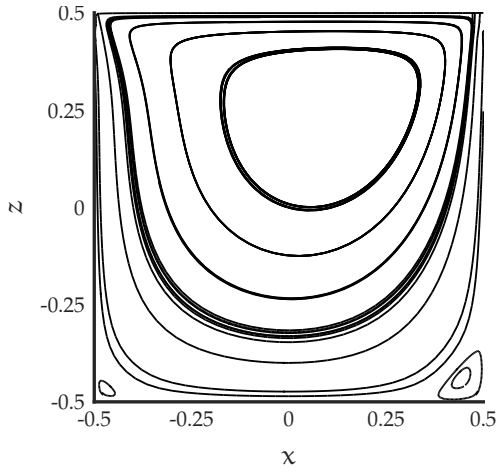
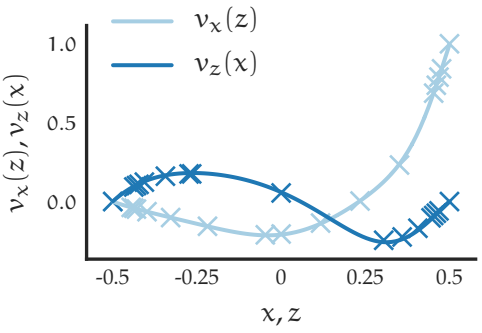


Figure 5.4.: Pressure for the ABC-flow at $t = 1\,s$. Each column represents a one-dimensional slice, all other coordinates are held constant at a value of zero. We compare the analytical solution (eq. 4.5) with our approximation. The former is represented by the gray line, the latter by the scatter plot. We only show every 4th point.



(a) Streamlines of the cavity flow.



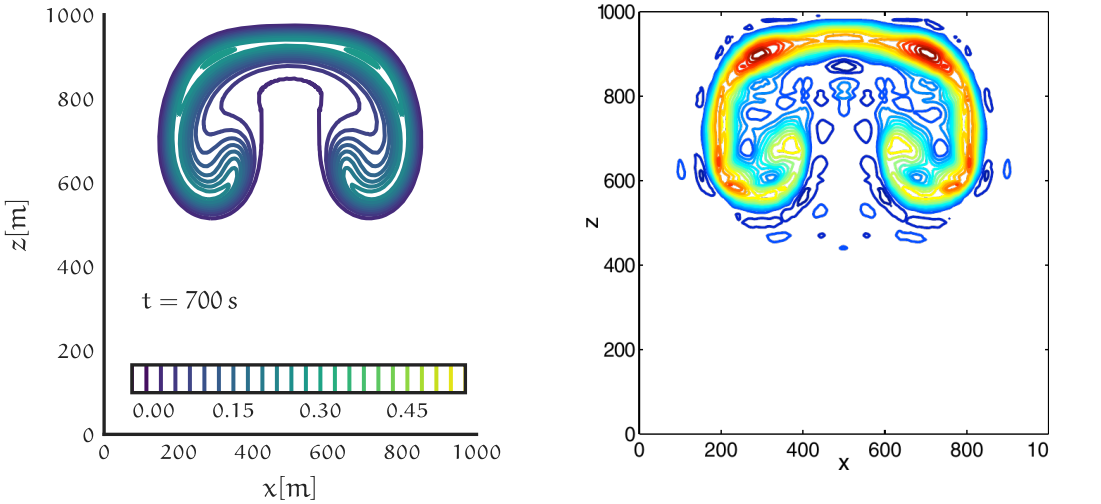
(b) Plot of velocity slices. Lines are our solution, scatter plot shows reference solution [GGS82]. For each slice, the other variable is held constant at a value of zero.

Figure 5.5.: Lid-Driven Cavity solution at $t = 30\,s$

5.3. Simulation of Clouds

We discussed the capabilities of our scheme regarding standard CFD testing scenario. In this section, we consider our bubble scenarios. We ran our first scenario, the cosine bubble (eq. 4.22), with an ADER-DG-scheme of order 4 and the adaptive mesh criterion eq. (2.69) with thresholds of $T_{refine} = 2.5$ and $T_{delete} = -0.5$ and the total variation of the potential temperature θ as indicator variable. Our coarsest level had 3^3 cells, our finest level had 3^4 cells. We thus had a resolution of ca. 37.037 m up to 12.3457 m.

Cosine bubble



(a) Our result

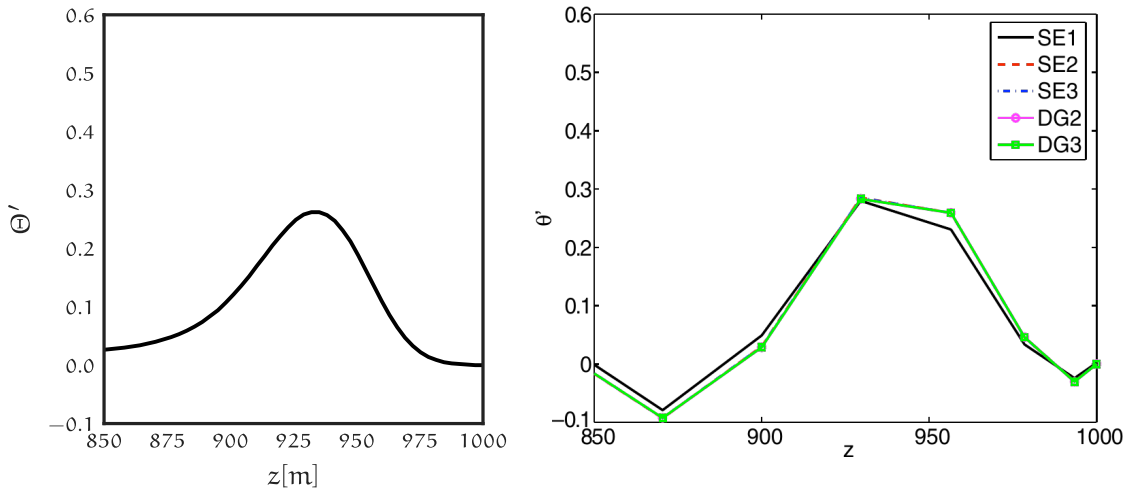
(b) Figure taken from [GR08], resolution of 20 m, N = 10, DG, modified Navier-Stokes

Figure 5.6.: Contours for the cosine bubble results. Contours are $-0.05, -0.025, \ldots 0.525$, excluding zero.

The coarse structure of our results (figure 5.6a) agrees with the reference solution (figure 5.6b of [GR08] but the fine details differ. In contrast to us, they used a scheme of polynomial order 10 and a uniform spatial resolution of 20 m. Moreover, we used a physical viscosity of 0.1, which explains why we have fewer spurious details despite using a lower order. This becomes clearer by comparing the potential temperature perturbation. Our solution has a smooth distribution inside the cloud (figure 5.7a) while the results of [GR08] show sharp peaks (figure 5.7b). In addition, our solution does not contain negative values for the perturbation. Despite the different parameters, the maximum perturbation and its position matches for both results.

We ran the second scenario, the two bubbles scenario (eq. 4.24), with the same AMR settings as the first one. The results in (figure 5.8) are comparable to the reference solution [Mül+10] for the first seven minutes. Note that the final plot at 7 min looks

Two bubbles

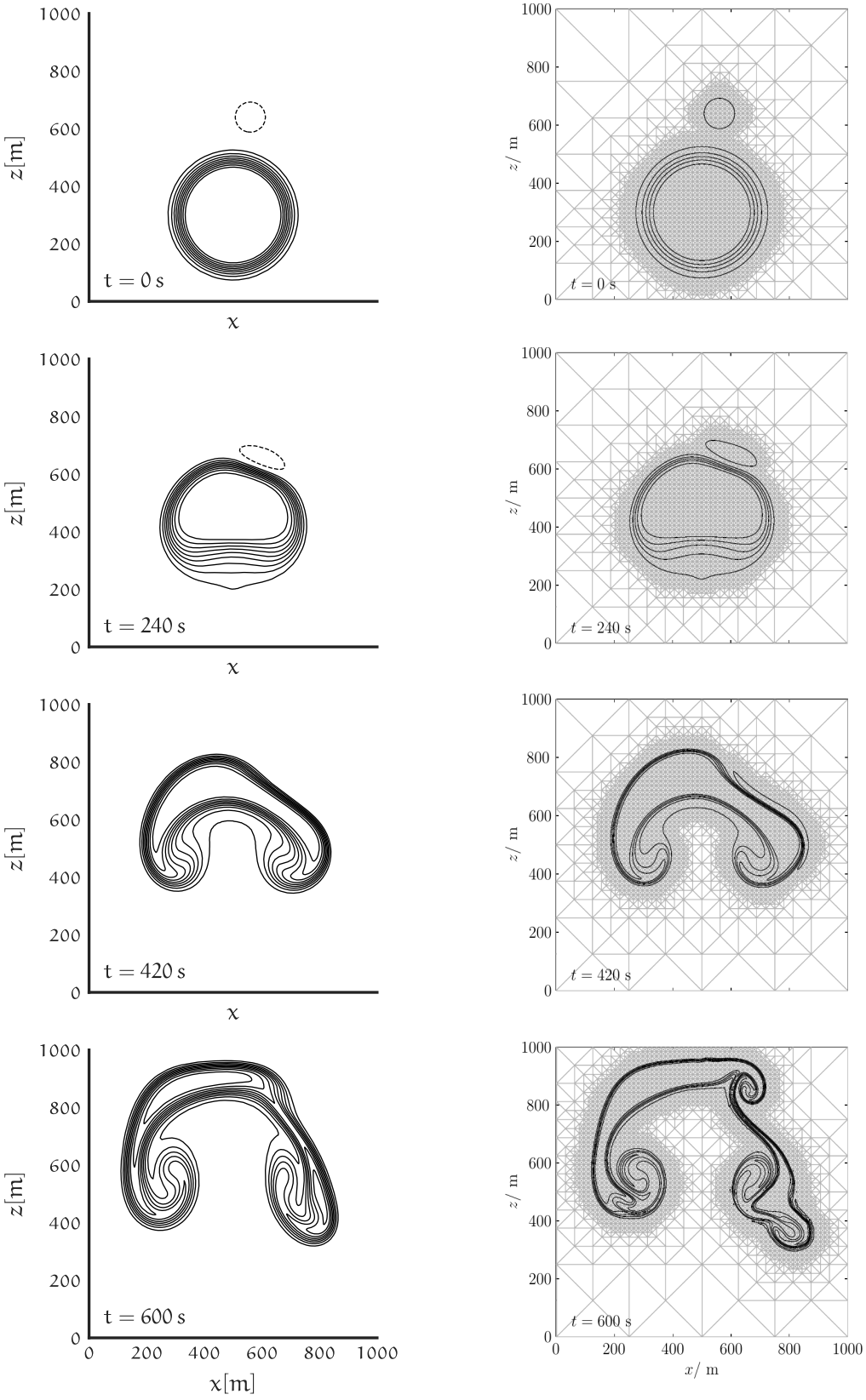


(a) Our result

(b) Figure taken from [GRo8], resolution of 20 m, N = 10, DG3 uses modified Navier-Stokes

Figure 5.7.: Height z vs. perturbation of potential temperature for the cosine bubble scenario at $t = 700\,s$ and $x = 500\,m$.

different in our simulation. A possible source for this is that we use a different equation set than [Mül+10]. While we use a general purpose set with small modifications for atmospheric flows, they use a equation set that is fully focused on this scenario. Furthermore, we use the full Navier-Stokes viscous fluxes instead of an approximation.



(a) Our AMR solution.

(b) Reference solution from [Mül+10].

Figure 5.8.: Contours of the potential temperature for two bubble scenario. On the left is our solution, the right plots are taken from [Mül+10]. We rescaled the reference plots and changed their arrangement and axes placement. Contour values for potential temperature perturbation are $-0.05, 0.05, 0.1, \ldots 0.45$.

5.4. AMR vs. Static Mesh

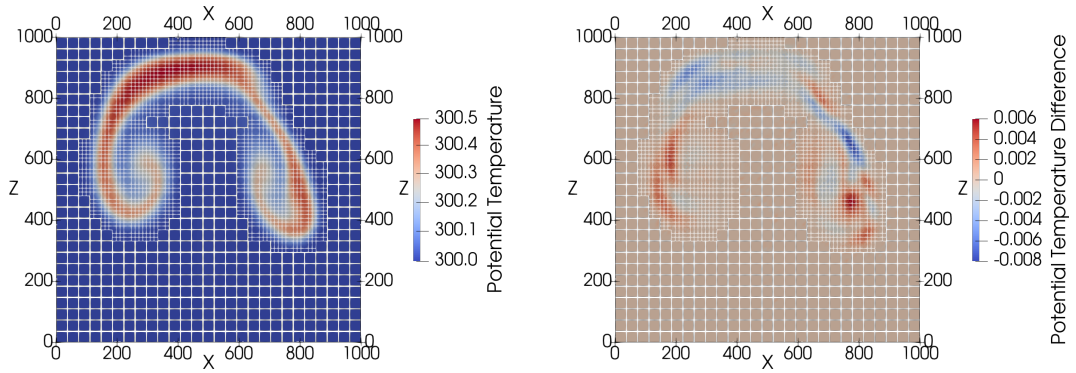
To show the effectiveness of the AMR criterion, we compare the two settings:

AMR We used a coarse mesh with just 27×27 cells. Each could be refined once. That is, the cells had side lengths of roughly 37.04 m and 12.35 m. As refinement thresholds we used $T_{\text{refine}} = 2.5$ and $T_{\text{delete}} = -0.5$ in eq. (2.69). They are the exact same settings that we used for the bubble scenarios in the previous section.

Fine We used a uniform grid where all cells had a side length corresponding to the finest mesh of the AMR-test case. In addition, we disabled the reduction of global observables.

While our AMR criterion is able to cope with an additional, coarser level, the current implementation in *ExaHyPE* is unstable in this case. This is why we only report an AMR run with one level of dynamic refinement.

For both settings, we used an ADER-DG-method of order 4 and thus had a minimal effective resolution of ca. 3.09 m. Note that the AMR criterion manages to track the bubble correctly (figure 5.9a) and introduces only small errors (figure 5.9b) compared to the solution with the fully refined mesh.



(a) Potential temperature of AMR solution.

(b) Potential temperature of fully refined solution minus AMR solution.

Figure 5.9.: AMR-grid at $t = 700$ s for the two bubbles scenario.

Time to solution test

We now want to compare the time to solution for both settings. To ensure a fair comparison, we ran both configurations with the same optimization options:

```
"architecture": "hsw",
"shared_memory":
```

```
    "cores": 28,
    "background_job_consumers": 6,
    "properties_file": "sharedmemory.properties",
    "autotuning_strategy": "dummy"
},
"optimisation": {
    "fuse_algorithmic_steps": true,
    "fuse_algorithmic_steps_factor": 0.99,
    "spawn_predictor_as_background_thread": true,
    "spawn_amr_background_threads": true,
    "spawn_update_as_background_thread": true
}
```

We ran the simulation without plotters to focus on the performance of the numerics and the mesh refinement routines. We used a single node of the Supermuc Phase 2. The node has two Intel® Haswell Xeon Processor E5-2697 v3 CPUs with 14 cores each. Both CPUs ran at a frequency of 1.8 GHz.

We solely used the TBB parallelization of *ExaHyPE*. To find a reasonable parameter for the number of background job consumers, we ran a grid search. We ran three AMR test cases for a simulation time of 1 min. The test cases were identical in setting to the aforementioned AMR run, but with two levels of refinement. The best median performance was achieved with six background threads. Note that the results were noisy and it is unclear whether the chosen number of background threads is optimal.

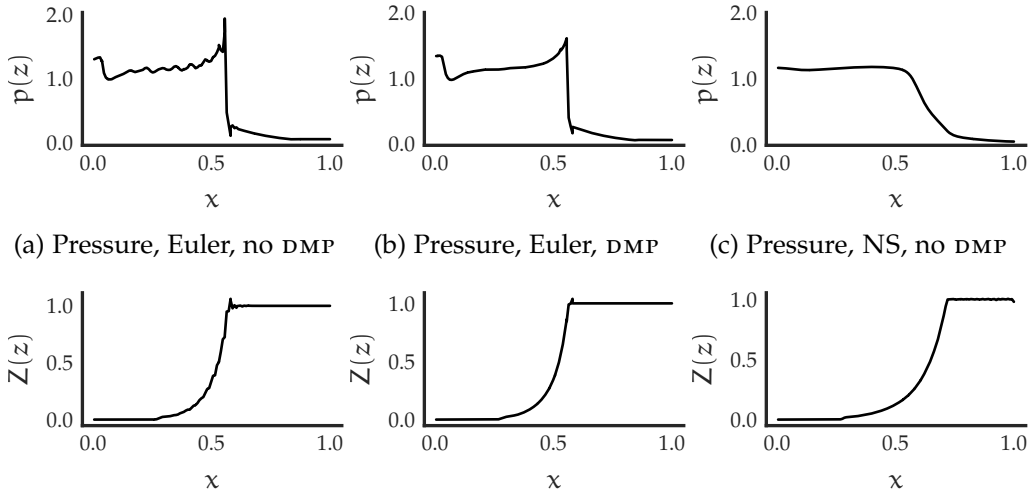
The AMR-run took 98 654 s (1 d 3 h 24 min 14 s) while the fully refined grid took 169 071 s (1 d 22 h 57 min 51 s). In other words, the latter method took over 71% longer.

To summarize, our AMR criterion saved a significant amount of computational effort while preserving the fine details of a fully refined grid.

5.5. Simulation of Reactive Detonation Waves

As a final benchmark, we used the CJ detonation wave (eq. 4.7) to evaluate the reactive Navier-Stokes equations. We used a limiting ADER-DG-scheme with an order of 3. For runs without viscosity we used a mesh size of 81×81, for the runs with a viscosity of $\mu = 0.01$ we used a coarser mesh consisting in 27×27 cells. We chose the smaller grid because our boundary conditions became unstable for finer grids. A reason for this might be the fact that we are unable to prescribe the gradient at the boundary in our MUSCL-Hancock implementation. For some runs, we used the DMP settings

```
"limiter": {
    "dmp_observables": 3,
```



(a) Pressure, Euler, no DMP (b) Pressure, Euler, DMP (c) Pressure, NS, no DMP

(d) Z, Euler, no DMP (e) Z, Euler, DMP (f) Z, NS, no DMP

Figure 5.10.: Slices along the x-axis for our three configurations. The coordinate $z = 0$ is held constant. We only show one half of the slice because the scenario is symmetric.

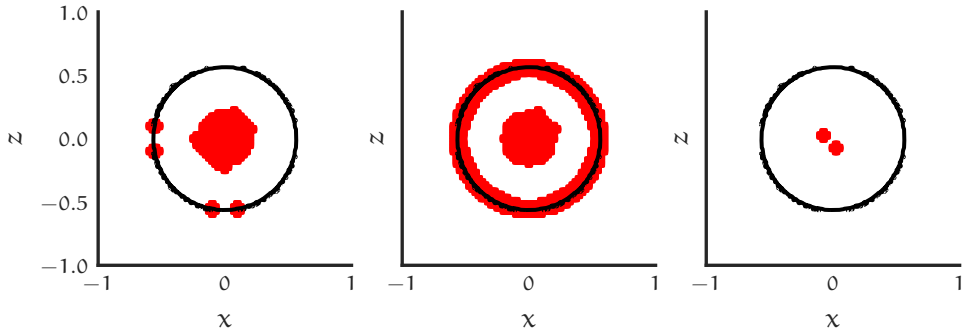
```
    "dmp_relaxation_parameter": 0.05,
    "dmp_difference_scaling": 1e-3,
    "helper_layers": 1
}
```

where the `dmp_relaxation_parameter` refers to ε_0 and `dmp_difference_scaling` refers to ε in eq. (2.63). The large value of $\varepsilon_0 = 0.05$ is necessary, as otherwise the full domain is refined. We use ρ, p and Z as DMP variables. The results for runs with and without DMP can be seen in figure 5.10. Note, that oscillations appeared for the Euler equations without the DMP-criterion. This is not the case for the Navier Stokes equations.

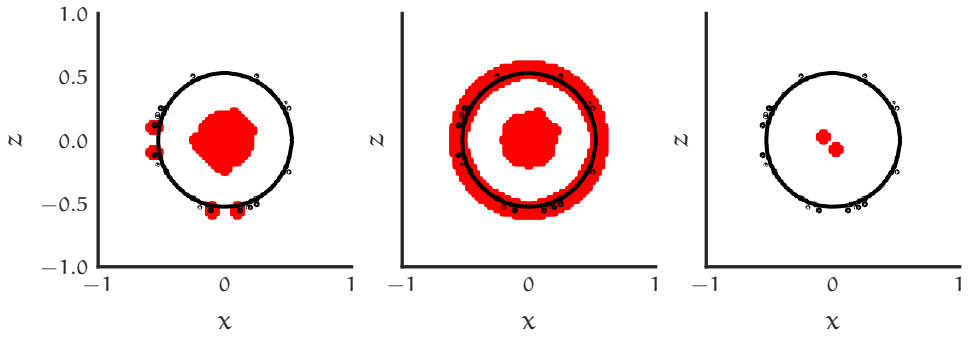
This scenario shows the differences between the viscous Navier-Stokes equations and the Euler equations clearly: The viscous effects smooth out the otherwise sharp curves. Furthermore, they have the effect that fewer cells are limited (figure 5.11). Note that the limiter activation are not symmetric which is unexpected for a radialsymmetric scenario. The contours for pressure and mass fraction look similar for all considered settings.

It is difficult to compare these results with our references because they both use different settings. In contrast to [HLW00] we use a non-stiff source term and viscous effects; [HD11] includes viscosity but is limited to one-dimensional scenarios. Comparing with figure 5.12 we can see that our simulation captures the overall behavior of the

reaction similar to the stiff simulations of [HLW00]. The viscous effects have a similar smoothing effect as in [HD11].



(a) Contour is $p = 0.5$



(b) Contour is $Z = 0.5$

Figure 5.11.: Contour plots for the detonation scenario at $t = 0.4$. From left to right: Euler without DMP, Euler with DMP, Navier Stokes without DMP. Red color indicates that a cell is limited.

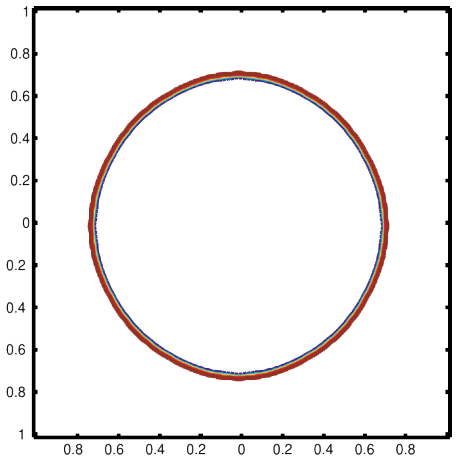


Figure 5.12.: Reference solution for mass fraction with stiff source. Image taken from [HLWoo].

CHAPTER 6

Conclusion

We implemented a MUSCL-Hancock-limited ADER-DG-scheme with AMR for the reactive Navier Stokes equations in this thesis. We showed that the scheme works and is capable of simulation different scenarios:

- We showed in section 5.1 that our scheme converges for all considered orders for the Navier-Stokes equations.
- We evaluated our method for standard CFD scenarios (section 5.2) and achieved competitive results for all used scenarios. This is true for both two-dimensional scenarios (Taylor-Green vortex and lid-driven cavity) and for the three-dimensional ABC-flow.
- Furthermore, our method allows us to simulate flows in hydrostatic equilibrium correctly (see section 5.3).
- The results of section 5.4 showed that our AMR-criterion is able to save a significant portion of the computational effort compared to a uniform fine grid. It is able to do this without degrading the quality of the solution, even for a scenario that contains small-scale details.
- Our results of section 5.5 showed that our implementation is capable of simulating chemical reactions with a limited scheme, at least for a source term that is not stiff.

Of course, there are some things that can be improved and which need further work, such as:

- The current treatment of the boundary conditions is not stable for some scenarios. Especially the viscous walls for the MUSCL-Hancock-method are not correct, due to limitations of *ExaHyPE*.
- The CFL-condition (eqs. 2.54 and 2.61) has the effect that our method needs a large number of timesteps. Even though our AMR-criterion is able to reduce the number of needed cells drastically, we could get another large improvement by utilizing local time-stepping.

- Our implementation of the CJ detonation wave cannot handle realistic detonations because we do not have an implementation of stiff source terms. An implementation of this is straight-forward but includes changes to the space-time predictor and its initial guess, and also to the MUSCL-Hancock-scheme.

APPENDIX A

Details for Navier-Stokes Equations

This appendix contains the Jacobians of the flux and the gradient of the temperature. We only present results for two-dimensions, the results for three-dimensions follow trivially. To simplify the notation, we denote the unit vector pointing in x-direction by $\boldsymbol{e}_x$ and the vector in z-direction by $\boldsymbol{e}_z$.

The Jacobian of the hyperbolic flux eq. (2.4) are in x and z direction:

$$\frac{\partial \mathbf{F}}{\partial \mathbf{Q}} \cdot \boldsymbol{e}_x = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ -\frac{1}{2}\left(2gz - \frac{j_x^2}{\rho^2} - \frac{j_z^2}{\rho^2}\right)(\gamma-1) - \frac{j_x^2}{\rho^2} & -\frac{(\gamma-1)j_x}{\rho} + \frac{2j_x}{\rho} & -\frac{(\gamma-1)j_z}{\rho} & \gamma-1 & -(\gamma-1)q_0 \\ -\frac{j_x j_z}{\rho^2} & \frac{j_z}{\rho} & \frac{j_x}{\rho} & 0 & 0 \\ -\frac{\left(2gz - \frac{j_x^2}{\rho^2} - \frac{j_z^2}{\rho^2}\right)(\gamma-1)j_x}{2\rho} - \frac{(E+p)j_x}{\rho^2} & -\frac{(\gamma-1)j_x^2}{\rho^2} + \frac{E+p}{\rho} & -\frac{(\gamma-1)j_x j_z}{\rho^2} & \frac{\gamma j_x}{\rho} & -\frac{(\gamma-1)j_x q_0}{\rho} \\ -\frac{Zj_x}{\rho^2} & \frac{Z}{\rho} & 0 & 0 & \frac{j_x}{\rho} \end{pmatrix}, \tag{A.1}$$

$$\frac{\partial \mathbf{F}}{\partial \mathbf{Q}} \cdot \boldsymbol{e}_z = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ -\frac{j_x j_z}{\rho^2} & \frac{j_z}{\rho} & \frac{j_x}{\rho} & 0 & 0 \\ -\frac{1}{2}\left(2gz - \frac{j_x^2}{\rho^2} - \frac{j_z^2}{\rho^2}\right)(\gamma-1) - \frac{j_z^2}{\rho^2} & -\frac{(\gamma-1)j_x}{\rho} & -\frac{(\gamma-1)j_z}{\rho} + \frac{2j_z}{\rho} & \gamma-1 & -(\gamma-1)q_0 \\ -\frac{\left(2gz - \frac{j_x^2}{\rho^2} - \frac{j_z^2}{\rho^2}\right)(\gamma-1)j_z}{2\rho} - \frac{(E+p)j_z}{\rho^2} & -\frac{(\gamma-1)j_x j_z}{\rho^2} & -\frac{(\gamma-1)j_z^2}{\rho^2} + \frac{E+p}{\rho} & \frac{\gamma j_z}{\rho} & -\frac{(\gamma-1)j_z q_0}{\rho} \\ -\frac{Zj_z}{\rho^2} & 0 & \frac{Z}{\rho} & 0 & \frac{j_z}{\rho} \end{pmatrix}. \tag{A.2}$$

The Jacobians for the viscous flux are

$$\frac{\partial \mathbf{F}}{\partial\left(\boldsymbol{\nabla}\mathbf{Q}\cdot\boldsymbol{e}_x\right)}\cdot\boldsymbol{e}_x = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ \frac{4\,\mu j_x}{3\,\rho^2} & -\frac{4\,\mu}{3\,\rho} & 0 & 0 & 0 \\ \frac{\mu j_z}{\rho^2} & 0 & -\frac{\mu}{\rho} & 0 & 0 \\ l+\frac{4\,\mu j_x^2}{3\,\rho^3}+\frac{\mu j_z^2}{\rho^3} & \frac{(\gamma-1)\kappa j_x}{R\rho^2}-\frac{4\,\mu j_x}{3\,\rho^2} & \frac{(\gamma-1)\kappa j_z}{R\rho^2}-\frac{\mu j_z}{\rho^2} & -\frac{(\gamma-1)\kappa}{R\rho} & \frac{(\gamma-1)\kappa q_0}{R\rho} \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \tag{A.3}$$

$$\frac{\partial \mathbf{F}}{\partial\left(\boldsymbol{\nabla}\mathbf{Q}\cdot\boldsymbol{e}_z\right)}\cdot\boldsymbol{e}_z = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ \frac{\mu j_x}{\rho^2} & -\frac{\mu}{\rho} & 0 & 0 & 0 \\ \frac{4\,\mu j_z}{3\,\rho^2} & 0 & -\frac{4\,\mu}{3\,\rho} & 0 & 0 \\ l+\frac{\mu j_x^2}{\rho^3}+\frac{4\,\mu j_z^2}{3\,\rho^3} & \frac{(\gamma-1)\kappa j_x}{R\rho^2}-\frac{\mu j_x}{\rho^2} & \frac{(\gamma-1)\kappa j_z}{R\rho^2}-\frac{4\,\mu j_z}{3\,\rho^2} & -\frac{(\gamma-1)\kappa}{R\rho} & \frac{(\gamma-1)\kappa q_0}{R\rho} \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}, \tag{A.4}$$

with

$$l = \kappa\left(\frac{(2\,g z\rho + q_0 Z - E)(\gamma-1)}{R\rho^2} - \frac{\left(2\,g z\rho^2 + 2\,q_0 Z\rho + j_x^2 + j_z^2 - 2\,E\rho\right)(\gamma-1)}{R\rho^3}\right). \tag{A.5}$$

The gradient of the temperature in conservative variables is[1]

$$\begin{aligned} \frac{\partial T}{\partial x} &= \frac{1}{c_v}\left(a\frac{\partial \rho}{\partial x} + b\frac{\partial j_x}{\partial x} + c\frac{\partial j_z}{\partial x} - q_0\frac{\rho\frac{\partial}{\partial x}Z - Z\frac{\partial}{\partial x}\rho}{\rho^2}\right), \\ \frac{\partial T}{\partial z} &= \frac{1}{c_v}\left(a\frac{\partial \rho}{\partial z} + b\frac{\partial j_x}{\partial z} + c\frac{\partial j_z}{\partial z} - q_0\frac{\rho\frac{\partial}{\partial z}Z - Z\frac{\partial}{\partial z}\rho}{\rho^2} - g\right), \end{aligned} \tag{A.6}$$

with

$$\begin{aligned} a &= -\frac{E}{\rho^2} + \frac{j_x^2 + j_z^2}{\rho^3}, \\ b &= -\frac{j_x}{\rho^2}, \\ c &= -\frac{j_z}{\rho^2}. \end{aligned} \tag{A.7}$$

[1] The form of terms a, b, c are from personal communication with Michael Dumbser.

Bibliography

[Bun+08] H. Bungartz, W. Eckhardt, M. Mehl, and T. Weinzierl. "DaStGen—A Data Structure Generator for Parallel C++ HPC Software." In: *Computational Science – ICCS 2008*. Springer Berlin Heidelberg, 2008, pp. 213–222. DOI: 10.1007/978-3-540-69389-5_25 (see p. 21).

[CGL82] T. F. Chan, G. H. Golub, and R. J. LeVeque. "Updating Formulae and a Pairwise Algorithm for Computing Sample Variances." In: *COMPSTAT 1982 5th Symposium held at Toulouse 1982*. Physica-Verlag HD, 1982, pp. 30–41. DOI: 10.1007/978-3-642-51461-6_3 (see p. 18).

[DL16] M. Dumbser and R. Loubère. "A simple robust and accurate a posteriori sub-cell finite volume limiter for the discontinuous Galerkin method on unstructured meshes." In: *Journal of Computational Physics* 319 (Aug. 2016), pp. 163–199. DOI: 10.1016/j.jcp.2016.05.002 (see pp. 1, 16, 17).

[Dum+08] M. Dumbser, D. S. Balsara, E. F. Toro, and C.-D. Munz. "A unified framework for the construction of one-step finite volume and discontinuous Galerkin schemes on unstructured meshes." In: *Journal of Computational Physics* 227.18 (Sept. 2008), pp. 8209–8253. DOI: 10.1016/j.jcp.2008.05.025 (see pp. 1, 7, 11–14).

[Dum+16] M. Dumbser, I. Peshkov, E. Romenski, and O. Zanotti. "High order ADER schemes for a unified first order hyperbolic formulation of continuum mechanics: Viscous heat-conducting fluids and elastic solids." In: *Journal of Computational Physics* 314 (June 2016), pp. 824–862. DOI: 10.1016/j.jcp.2016.02.015 (see pp. 25, 27, 34).

[Dum+18a] M. Dumbser, F. Fambri, M. Tavelli, M. Bader, and T. Weinzierl. "Efficient Implementation of ADER Discontinuous Galerkin Schemes for a Scalable Hyperbolic PDE Engine." In: *Axioms* 7.3 (Sept. 2018), p. 63. DOI: 10.3390/axioms7030063 (see pp. 7, 12).

[Dum+18b] M. Dumbser, F. Guercilena, S. Köppel, L. Rezzolla, and O. Zanotti. "Conformal and covariant Z4 formulation of the Einstein equations: Strongly hyperbolic first-order reduction and solution with discontinuous Galerkin schemes." In: *Physical Review D* 97.8 (Apr. 2018). DOI: 10.1103/physrevd.97.084053 (see p. 17).

[Dum10] M. Dumbser. "Arbitrary high order PNPM schemes on unstructured meshes for the compressible Navier–Stokes equations." In: *Computers & Fluids* 39.1 (Jan. 2010), pp. 60–76. DOI: 10.1016/j.compfluid.2009.07.003 (see pp. 3, 5, 7, 13, 14, 25, 26, 34).

[FDZ17] F. Fambri, M. Dumbser, and O. Zanotti. "Space–time adaptive ADER-DG schemes for dissipative flows: Compressible Navier–Stokes and resistive MHD equations." In: *Computer Physics Communications* 220 (Nov. 2017), pp. 297–318. DOI: 10.1016/j.cpc.2017.08.001 (see pp. 25, 27).

[Fuh+18] O. Fuhrer, T. Chadha, T. Hoefler, G. Kwasniewski, X. Lapillonne, D. Leutwyler, D. Lüthi, C. Osuna, C. Schär, and T. C. Schulthess. "Near-global climate simulation at 1km resolution: establishing a performance baseline on 4888 GPUs with COSMO 5.0." In: *Geoscientific Model Development* 11.4 (May 2018), pp. 1665–1681. DOI: 10.5194/gmd-11-1665-2018 (see p. 1).

[GGS82] U. Ghia, K. Ghia, and C. Shin. "High-Re solutions for incompressible flow using the Navier-Stokes equations and a multigrid method." In: *Journal of Computational Physics* 48.3 (Dec. 1982), pp. 387–411. DOI: 10.1016/0021-9991(82)90058-4 (see pp. 34, 36).

[GLM08] G. Gassner, F. Lörcher, and C.-D. Munz. "A Discontinuous Galerkin Scheme based on a Space-Time Expansion II. Viscous Flow Equations in Multi Dimensions." In: *Journal of Scientific Computing* 34.3 (2008), pp. 260–286. DOI: 10.1007/s10915-007-9169-1 (see p. 14).

[GR08] F. Giraldo and M. Restelli. "A study of spectral element and discontinuous Galerkin methods for the Navier–Stokes equations in nonhydrostatic mesoscale atmospheric modeling: Equation sets and test cases." In: *Journal of Computational Physics* 227.8 (Apr. 2008), pp. 3849–3877. DOI: 10.1016/j.jcp.2007.12.009 (see pp. 1, 4–6, 25, 28, 30, 37, 38).

[HD11] A. Hidalgo and M. Dumbser. "ADER Schemes for Nonlinear Systems of Stiff Advection–Diffusion–Reaction Equations." In: *Journal of Scientific Computing* 48.1-3 (2011), pp. 173–189. DOI: 10.1007/s10915-010-9426-6 (see pp. 3, 6, 25, 28, 42, 43).

[HLW00] C. Helzel, R. J. Leveque, and G. Warnecke. "A Modified Fractional Step Method for the Accurate Approximation of Detonation Waves." In: *SIAM Journal on Scientific Computing* 22.4 (Jan. 2000), pp. 1489–1510. DOI: 10.1137/s1064827599357814 (see pp. 4, 6, 25, 27, 28, 42–44).

[HW08] J. S. Hesthaven and T. Warburton. *Nodal Discontinuous Galerkin Methods*. Springer New York, 2008. DOI: 10.1007/978-0-387-72067-8 (see pp. 1, 16, 34).

[Kre18] L. Krenz. *Cloud simulation with the ExaHyPE-Engine*. `https://github.com/krenzland/cloudSuperComputing/blob/df30445/notebooks/ManufacturedSolution.ipynb`. Version Git commit: df304454bd15196120d14a3ca93ff164cc7e20d7. 2018 (see p. 26).

[Lee79] B. van Leer. "Towards the ultimate conservative difference scheme. V. A second-order sequel to Godunov's method." In: *Journal of Computational Physics* 32.1 (July 1979), pp. 101–136. DOI: 10.1016/0021-9991(79)90145-1 (see p. 14).

[LeV02] R. J. LeVeque. *Finite Volume Methods for Hyperbolic Problems*. Cambridge University Press, 2002. DOI: 10.1017/cbo9780511791253 (see pp. 14, 15).

[Mül+10] A. Müller, J. Behrens, F. X. Giraldo, and V. Wirth. "An adaptive discontinuous Galerkin method for modeling cumulus clouds." In: (2010) (see pp. 1, 6, 25, 29, 31, 37–39).

[Roa02] P. J. Roache. *Code Verification by the Method of Manufactured Solutions*. Tech. rep. 1. 2002, p. 4. DOI: 10.1115/1.1436090 (see p. 26).

[Rob93] A. Robert. "Bubble Convection Experiments with a Semi-implicit Formulation of the Euler Equations." In: *Journal of the Atmospheric Sciences* 50.13 (July 1993), pp. 1865–1873. DOI: 10.1175/1520-0469(1993)050<1865:bcewas>2.0.co;2 (see pp. 1, 25, 31).

[Sch+18] T. C. Schulthess, P. Bauer, O. Fuhrer, T. Hoefler, C. Schar, and N. Wedi. "Reflecting on the goal and baseline for exascale computing: a roadmap based on weather and climate simu-lations." In: *Computing in Science & Engineering* (2018), pp. 1–1. DOI: 10.1109/mcse.2018.2888788 (see p. 1).

[TD16] M. Tavelli and M. Dumbser. "A staggered space–time discontinuous Galerkin method for the three-dimensional incompressible Navier–Stokes equations on unstructured tetrahedral meshes." In: *Journal of Computational Physics* 319 (Aug. 2016), pp. 294–323. DOI: 10.1016/j.jcp.2016.05.009 (see pp. 25, 27).

[The18] The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 8.3)*. `http://www.sagemath.org`. 2018 (see p. 26).

[Toro9] E. F. Toro. *Riemann Solvers and Numerical Methods for Fluid Dynamics*. Springer Berlin Heidelberg, 2009. DOI: 10.1007/b79761 (see pp. 14, 15).

[Was04] L. Wasserman. *All of Statistics*. Springer New York, 2004. DOI: 10.1007/978-0-387-21736-9 (see p. 19).

[WM11] T. Weinzierl and M. Mehl. "Peano—A Traversal and Storage Scheme for Octree-Like Adaptive Cartesian Multiscale Grids." In: *SIAM Journal on Scientific Computing* 33.5 (Jan. 2011), pp. 2732–2760. DOI: 10.1137/100799071 (see p. 20).