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

Wolken Simulation mit der ExaHyPE-Engine

Author:	Lukas Krenz
Supervisor:	Univ.-Prof. Dr. Michael Bader
Advisor:	Leonhard Rannabauer
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.

Munich, January , 2019 Lukas Krenz

Acknowledgments

Quote!

(Booktitle)

Acknowledgements here.

Abstract

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

We use the compressible Navier-Stokes (NS) equations with heat conduction to describe the dynamics of clouds. Due to the diffusive components, the NS-Equations are not hyperbolic. We thus need to modify parts of the numerical scheme.

Furthermore, we extend the equations to the reactive compressible Navier Stokes that include a source term that models chemical reactions.

Contents

CHAPTER 1

Introduction

We make the following contributions:

- We implement an ADER-DG and MUSCL-scheme for the compressible Navier-Stokes equations.
- We describe a coupling of the compressible Navier-Stokes equations with an advection-diffusion-reaction term for two dimensional problems.
- We compare different mesh refinement strategies and evaluate their computational efficiency.
- We investigate the ability of our scheme to simulate classical fluid mechanics test scenarios correctly.
- We describe the simulation of flows with a background state that is in hydrostatic equilibrium. This is useful for problems in atmospheric flow and for problems in meterology. Furthermore, we evaluate our discretization for these scenarios.

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 good accuracy. We start with a description 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 seque into a short description of the MUSCL-Hancock scheme, a second order finite volume scheme.

We conclude this chapter by a description of adaptive mesh refinement (AMR) and finite volume limiting, which combines both numerical methods to achieve a stable, high-order method.

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 $\boldsymbol{x} = (x, z)$ and $\boldsymbol{x} = (x, y, z)$ for the two and three-dimensional case respectively. The variable t corresponds to the currenty (physical) time. We solve the PDE in the conversative form (eq. 2.20)

Dont reference to following chapter!

$$\begin{matrix}\text{mass cons.}\\ \text{momentum cons.}\\ \text{energy cons.}\\ \text{cont. gas}\end{matrix} : \underbrace{\frac{\partial}{\partial t}\begin{pmatrix}\rho\\ \rho\boldsymbol{v}\\ \rho E\\ \rho Z\end{pmatrix}}_{Q} + \boldsymbol{\nabla}\cdot\underbrace{\left(\underbrace{\begin{pmatrix}\rho\boldsymbol{v}\\ \boldsymbol{v}\otimes\rho\boldsymbol{v}+\mathbf{I}p\\ \boldsymbol{v}\cdot(\mathbf{I}\rho E+\mathbf{I}p)\\ \rho\boldsymbol{v}Z\end{pmatrix}}_{F^h(Q)} + \underbrace{\begin{pmatrix}-\varepsilon\boldsymbol{\nabla}\rho\\ \boldsymbol{\sigma}(Q,\boldsymbol{\nabla}Q)\\ \boldsymbol{v}\cdot\boldsymbol{\sigma}(Q,\boldsymbol{\nabla}Q)-\kappa\boldsymbol{\nabla}T\\ -\varepsilon\boldsymbol{\nabla}\rho Z\end{pmatrix}}_{F^v(Q,\boldsymbol{\nabla}Q)}\right)}_{F(Q,\boldsymbol{\nabla}Q)} = \underbrace{\begin{pmatrix}S_\rho\\ S_{\rho\boldsymbol{v}}\\ S_{\rho E}\\ S_{\rho Z}\end{pmatrix}}_{S(Q,x,t)} \tag{2.1}$$

which describes the time evolution of variables $\mathbf{Q}$ with respect to a flux $F(\mathbf{Q}, \nabla\mathbf{Q})$ 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 the energy density and ρZ is the mass fraction of the chemical reactant. The chemical reactant in our case is unburnt gas.

The rows of the equation are the conservation of mass, the conservation of momentum, the conservation of energy and the continuity of the reactant. We split the flux into a hyperbolic $F^h(\mathbf{Q})$ and a viscous part$F^v(\nabla\mathbf{Q})$

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

The hyperbolic flux is given by

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

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 caloric equation of state of an ideal reacting gas

Geopot height correct?

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

The term $q_0\rho Z$ is the chemical energy with the heat release q_0, the term gz is the geopotential height. The pressure is related to the temperature T by the thermic ideal gas law

$$\frac{p}{\rho} = RT, \tag{2.6}$$

where R is the specific gas constant of a fluid.

The viscous flux is

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

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_r. These constants are all fluid dependent and relate to each other 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 then compute the heat conduction coefficient

Cite sth for this!

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

where the Prandtl number Pr depends on the fluid.

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

$$\begin{aligned} |\lambda_c^{max}| &= (\partial \mathbf{F} / \partial \mathbf{Q}) \cdot \mathbf{n} = |\mathbf{v_n}| + c, \\ |\lambda_v^{max}| &= (\partial \mathbf{F} / \partial (\nabla \mathbf{Q} \cdot \mathbf{n})) \cdot \mathbf{n} = \max\left(\frac{4}{3}\frac{\mu}{\rho}, \frac{\gamma\mu}{Pr\,\rho}, \varepsilon\right) \end{aligned} \tag{2.11}$$

with velocity in direction of the normal $\mathbf{v_n}$ and speed of sound $c = \sqrt{\gamma R T}$.

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

Either include gravity in pressure or add source term for energy.

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

where $g = 9.81$ and $\mathbf{k}$ is a unit vector pointing in z-direction.

Some of our scenarios can be described as a pertubation of a background state that is 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)

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

. The background state eq. (2.13) leads to large values and can diminish other parts of the flux due to numerical instabilities [Mül+10]. We can now 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, the

definition of divergence into eq. (2.14) allows us to apply eq. (2.13). Furthermore, the background states have zero derivatives in all directions but in z. We arrive at

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

. We thus cancel parts of the flux with parts of the source term. Note that the time derivative of the background density also vanishes. We do not use this to simplify the implementation of the gradient computation. This form of the equation is similar to equation set 3 of [GR08] but without splitting the energy.

The chemical reactive source term is

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

We use the very simple reaction model

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

where τ is the timescale of the reaction and T_{ign} is the activation temperature.

2.1.1. Boundary conditions

To close the system we need to impose boundary conditions.

Cauchy

For some scenarios we use Cauchy boundary conditions, i.e. we prescribe both $\mathbf{Q}$ and $\boldsymbol{\nabla}\mathbf{Q}$ at the boundary. In most cases, we would like to impose periodic boundary conditions, due to the inner workings of *ExaHyPE* this is not possible. Instead we use the analytical solution of our problems at the boundary, imposing both value and gradients of the conservative variables. Note that this leads to an error when our problem does not posses an exact analytical solution. This is the case for test cases that are analytical solutions to 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

No-Slip

A standard wall boundary condition for viscous fluid is the no-slip condition, where we assume that the fluid has a velocity of zero near the wall. We enfore this by setting

Check if this is the correct physical description!

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

where a superscript of o and i denotes the values outside and inside of the boundary respectively.

Similarly to the no-slip condition, we define the free-slip condition *Free-Slip*

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

where only the velocity in normal direction is zero after the Riemann solver. 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 discontinous 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 and d the number of spatial dimensions. 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 boundary $\partial\Omega$. In our implementation of the discontinous 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 broken 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 a grid cell with coordinates $\mathbf{x}$ and returns the corresponding point $\hat{\mathbf{x}}$ on the reference cell

check if $[0,1]$ detontes correct interval

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

We also define the inverse mapping $\mathcal{M}^{-1}$. This 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 point in time to the reference time $\hat{t} \in (0,1)$

$$\hat{t} = \mathcal{T}(t) = t_0 + (\Delta t)t, \tag{2.25}$$

where Δt is timestep of a cell and t_0 is the beginning of its time interval. The space-time reference cell is then $\hat{C} = [0,1]^{D+1}$

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 by

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

Basis Functions

The next step is the definition of basis functions for the reference cell. We use Langrange polynomials that are defined in the one-dimensional case by

$$\hat{\varphi}_i(x) = \prod_{i \neq j}^{N} \frac{x - x_j}{x_i - x_j}. \tag{2.28}$$

We can then represent functions as a sum over the support points x_j

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

where we used the Einstein summation convention where summation over repeated indices is implied. The support points x_i are chosen such that they coincide with the

Remove this sentence, or sentence before introducing einstein summation!

nodes of the Gauss-Legendre quadrature which we can use to evaluate integrals

$$\int_0^1 f(x)\,dx = \sum_n^N w_n f(x_n), \tag{2.30}$$

where x_n is a quadrature point with weight w_n.

The basis functions are similarly defined in d-dimensions by considering nodes that are the tensor-products of d one-dimensional Gauss-Legendre quadrature. To simplify notation, we order the polynomials by a linearised index, a d-dimensional function is then approximated by

$$f(\mathbf{x}) = \sum_{i=0}^{(N+1)^d-1} f(\mathbf{x}_i)\hat{\varphi}_i(\mathbf{x}) = f(\mathbf{x}_i)\hat{\varphi}_i(\mathbf{x}). \tag{2.31}$$

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

$$\underbrace{\int_0^1 \cdots \int_0^1}_{d \text{ times}} f(\mathbf{x})\,d\mathbf{x} = \sum_n^{(N+1)^D-1} w_n f(\mathbf{x}_n) \tag{2.32}$$

with

$$w_n = \prod_i^d w_i. \tag{2.33}$$

Fix notation, especially multi-dimensional integral.

Check number of total polynomials.

Two important properties of this family of polynomials are

$$\text{Interpolating:} \qquad \hat{\varphi}_i(n_i) = \delta_{ij} \tag{2.34}$$

$$\text{Orthogonality:} \qquad \underbrace{\int_0^1 \cdots \int_0^1}_{d \text{ times}} \hat{\varphi}_i(\hat{\mathbf{x}})\,\hat{\varphi}_j(\hat{\mathbf{x}})\,d\hat{\mathbf{x}} = w_i\delta_{ij} \tag{2.35}$$

We will later need basis functions over a space-cell $v = 0, \ldots, N^{var}, n = 0, \ldots, (N+1)^d - 1, l = 0, \ldots, N$

$$\begin{aligned} \hat{\phi}_n(\hat{\mathbf{x}}) &= \varphi_n(x) \\ \phi_n(\mathbf{x}) &= \hat{\phi}_n(\mathcal{M}(\mathbf{x})) \end{aligned} \tag{2.36}$$

where $\hat{\phi}$ denotes the basis function on the reference cell. Similarly, we define a basis for a space-time element defined over $(0,1)^D \times (t^n, t^{n+1})$

$$\begin{aligned} \hat{\Theta}_{l,n}(\hat{\mathbf{x}}, \hat{t}) &= \varphi_l(t)\phi_n(x) = \varphi_l(t)\varphi_n(x) \\ \Theta_{l,n}(\mathbf{x}, t) &= \hat{\Theta}(\mathcal{M}(\mathbf{x}), \mathcal{T}(t)) \end{aligned} \tag{2.37}$$

Outside of elements, we define the basis functions to be equal to 0.

Bold x, bold t?

We are now ready to derive the predictor. Let t^n be the current simulation time and $t^{n+1} = t^n + \Delta t$ the time at the end of a timestep. 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 the vector of conservative variables $\mathbf{Q}$ with the space-time predictor $\overline{\mathbf{q}}^C$

Variables in basis

Explain in basis functions sec

Predictor

$$\int_{t^n}^{t^{n+1}} \int_C \Theta^C(\mathbf{x},t) \frac{\partial \overline{\mathbf{q}}}{\partial t} \, d\mathbf{x}\, dt + \int_{t^n}^{t^{n+1}} \int_C \Theta^C(\mathbf{x},t) \left(\boldsymbol{\nabla} \cdot \mathbf{F}(\overline{\mathbf{q}}, \boldsymbol{\nabla}\overline{\mathbf{q}})\right) d\mathbf{x}\, dt = \int_{t^n}^{t^{n+1}} \int_C \Theta^C \mathbf{S}(\overline{\mathbf{q}}) \, d\mathbf{x}\, dt \,. \tag{2.38}$$

We integrate the first term 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 discrete solution at time t. This corresponds to upwinding in time [Dum+08]. Note that this neglects the interaction with neighbouring cells. We account for this in the corrector step.

Insert in equation!

$$\begin{aligned} &\int_C \Theta_k^C(\mathbf{x},t^{n+1}) \overline{\mathbf{q}}(\mathbf{x},t^{n+1}) \, d\mathbf{x} - \int_{t^n}^{t^{n+1}} \int_C \frac{\partial}{\partial t} \Theta_k^C(\mathbf{x},t) \overline{\mathbf{q}}(\mathbf{x},t) \, d\mathbf{x}\, dt = \\ &\quad \int_C \Theta_k^C(\mathbf{x},t^n) \overline{\mathbf{q}}(\mathbf{x},t^n) \, d\mathbf{x} + \int_{t^n}^{t^{n+1}} \int_C \Theta_k^C(\mathbf{x},t) \boldsymbol{\nabla} \cdot \mathbf{F}(\overline{\mathbf{q}}, \boldsymbol{\nabla}\overline{\mathbf{q}}) \, d\mathbf{x}\, dt + \int_{t^n}^{t^{n+1}} \int_C \Theta_k^C(\mathbf{x},t) \mathbf{S}(\overline{\mathbf{q}}) \, d\mathbf{x}\, dt \end{aligned} \tag{2.39}$$

All quantities can be expanded in the space-time basis. For example, the expansion of space-time predictor can be written as

$$\overline{\mathbf{q}}^C(\mathbf{x},t) = \underline{\overline{q}}^C_{\nu,l,n} \Theta^C_{\nu,l,n}(\mathbf{x},t). \tag{2.40}$$

We denote the space-time flux as $\overline{F}$ and the space-time source as $\overline{S}$ with coefficient vectors $\underline{\overline{F}}$ and $\underline{\overline{S}}$. This is a part where we notice the computational advantage of the nodal approach: The coefficients correspond to the quantity evaluated at the basis node!

Inserting eq. (2.40) results in a local systems of equations that can be by a fixed point iteration scheme. To show the computational efficiency of our scheme, we follow the approach of [Dum+08] and collect our integrals into matrices. Henceforth, we drop the cell index for all quantities. For the space-time predictor, we introduce an iteration counter i. To close the iterative scheme, we set the initial value for the space-time predictor as

Source is Dominics ADER-DG-document.

$$\underline{\overline{q}}^i(t=0) = \underline{u}^i \tag{2.41}$$

which corresponds to a stationary guess. We arrive at the matrices

This is an expansion only in space!

$$\mathbf{L} \in \mathbb{R}^{N^{var}(N+1)^{d+1} \times N^{var}(N+1)^{d+1}}, \quad \mathbf{r} \in \mathbb{R}^{N^{var}(N+1)^{d+1}}, \quad \boldsymbol{w} \in \mathbb{R}^{N^{var}(N+1)^{d+1}} \tag{2.42}$$

with entries

$$\begin{aligned}\mathbf{L}_{(n\nu l),(n'\nu l')} = &\int_C \Theta_{\nu,l,n}(t^{n+1})\,\Theta_{\nu,l',n'}(t^{n+1})\,dx \\ &- \int_{t^n}^{t^{n+1}} \int_C \left(\frac{\partial}{\partial t}\Theta_{\nu,l,n}(t)\right)\Theta_{\nu,l',n'}(t)\,dx\,dt\end{aligned} \tag{2.43}$$

$$\begin{aligned}\mathbf{r}_{\nu,l,n}(\overline{\mathbf{q}}) = &\int_C \Theta_{\nu,l,n}(t)\phi_{\nu,n'}\underline{u}_{\nu,n'}\,dx \\ &+ \int_{t^n}^{t^{n+1}} \int_C \Theta_{\nu,l,n}(t)\Theta_{\nu,l',n'}(t)S_{\nu,l',n'}(\overline{\mathbf{q}}^i)\,dx\,dt\end{aligned} \tag{2.44}$$

$$\boldsymbol{w}_{\nu,l,n}(\overline{\mathbf{q}}) = \overline{\underline{F}}_{\nu,l',n'}(\overline{\mathbf{q}}) \int_{t^n}^{t^{n+1}} \int_C \left(\boldsymbol{\nabla}\Theta_{\nu,l,n}\right)\Theta_{\nu,l',n'}(t)\,dx\,dt \tag{2.45}$$

Note here that $\boldsymbol{w}$ is constant during a interation step. Using these matrices, we can solve for the space-time-predictor with the iterative scheme

$$\mathbf{L}\underline{\overline{q}}^{i+1} = \mathbf{r}(\mathbf{u}_h) - \boldsymbol{w}(\underline{\overline{q}}^i) \tag{2.46}$$

$$\underline{\overline{q}}^{i+1} = \mathbf{L}^{-1}\left(\mathbf{r}(\mathbf{u}_h) - \boldsymbol{w}(\underline{\overline{q}}^i)\right). \tag{2.47}$$

For details and proof of convergence for the linear case, see [Dum+08].

Preparing for corrector

We then prepare the space-time-predictor for use in the corrector step. Due to the linearity of the Riemann solver, we can exchange time and space integration and thus only need to consider the Riemann problem for time-averaged quantities. We then extrapolate all quantities to the boundary. This is needed for the Riemann solver and for the reconstruction of boundary values. We accomplish this by evaluating the function at the degrees of freedom on the boundary. Thanks to the interpolating property of the nodal basis (eq. 2.34), this reduces to a change of basis which can be precomputed on the reference element. We compute time-averaged and boundary extrapolated versions of the space-time-predictor, the degrees of freedom and their gradient, and the fluxes.

Details

Corrector

The final step of the ADER-DG-scheme is the corrector. For this step, we write the approximation of a function q in a cell as

$$q(\mathbf{x})_{|C_i} = \hat{u}^n_{i,l}\phi_l(\mathbf{x}), \tag{2.48}$$

where the basis, unlike in eq. (2.40), does not depend on the time. We then multiply the system eq. (2.20) by a test function ϕ_i and integrate over the space-time volume $(C \times [t^n, t^{n+1}])$. We arrive yet again at the weak formulation of the PDE

$$\int_{t^n}^{t^{n+1}} \int_C \phi_k \frac{\partial \mathbf{Q}}{\partial t}\,dx\,dt + \int_{t^n}^{t^{n+1}} \int_C \phi_k\left(\boldsymbol{\nabla}\cdot \mathbf{F}(\mathbf{Q}, \boldsymbol{\nabla}\mathbf{Q})\right)dx\,dt = \int_{t^n}^{t^{n+1}} \int_C \phi_k S(\mathbf{Q}, \mathbf{x}, t)\,dx\,dt\,. \tag{2.49}$$

We now replace the solution $\mathbf{Q}$ with the spacetime-predictor $\overline{\mathbf{q}}(\mathbf{x}, t)$ and write it as a polynomial using the representation of eqs. (2.40) and (2.48). Integrate first by parts in time (note basis here not defined over time), flux divergence by parts in space.

Include gradient everywhere!

Fix following equation, sum only needed due to expansion of dsol

Not all quantities are defined over time

$$\sum_{n'=0}^{} (N+1)^{d+1} \left(\int_C \phi_n \phi_{n'} \, dx \right) (\mathbf{u}^{n+1} - \mathbf{u}^n) + \left(\int_{t^n}^{t^{n+1}} \int_{\partial C} \phi_n \, \text{Riemann}(\overline{\mathbf{q}}^-, \overline{\mathbf{q}}^+) \cdot \mathbf{n} \, dS \, dt \right)$$
$$\left(\int_{t^n}^{t^{n+1}} \int_C \nabla \phi^k \cdot F(\mathbf{q}, \nabla \mathbf{q}) \, dx \, dt \right) = \left(\int_{t^n}^{t^{n+1}} \int_C \phi^k \mathbf{S}(\mathbf{q}) \, dx \, dt \right) \tag{2.50}$$

The first term is our mass matrix with entries

$$\mathbf{M}_{kl} = \int_C \phi_n \phi_{n'} \, dx = V \underbrace{w_k \delta_{kl}}_{\text{diagonal}} \tag{2.51}$$

which is, due to the orthogonal basis (eq. 2.35), diagonal and thus trivial to invert. We can then insert the definition of our basis function into our equation and arrive at

$$\mathbf{M} \left(\underline{u}^{t+1} - \underline{u}^t \right) = \Delta t \, (\mathbf{a} - \mathbf{b} + \mathbf{s}) \tag{2.52}$$

with

$$\begin{aligned} a_{v,n} &= \underline{F}_{n'} \int_C \phi_{v,n'} \nabla \phi_{v,n}(t) \, dx \\ b &= \dots \\ s_n &= \underline{S}_{v,n'} \int_C \phi_{v,n} \phi_{v,n'} \, dx \, , \end{aligned} \tag{2.53}$$

where we evaluated the time-integrals by using the fact that our quantities are time-averaged. We can solve this for the new unknows $\mathbf{u}^{n+1}$

$$\begin{aligned} \mathbf{u}^{n+1} &= \mathbf{u}^n + \Delta t (\text{update}) \\ \text{update} &= \mathbf{M}^{-1} \, (\mathbf{a} - \mathbf{b} + \mathbf{s}) \end{aligned} \tag{2.54}$$

which is clearly of the form of a one-step scheme.

Riemann solver & timestep

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

$$\text{Riemann}(q_h^-, \nabla q_h^-; g_h^+, \nabla q_h^+) \cdot \mathbf{n} = \frac{1}{2} \left(F(q_h^+, \nabla q_h^+) + F(q_h^-, \nabla q_h^-) \right) - \frac{1}{2} s_{max} (q_h^+ - q_h^-), \tag{2.55}$$

with a penalty term

$$s_{max} = \max \left(|\lambda_c^{max} \left(q_h^- \right)|, |\lambda_c^{max} \left(q_h^+ \right)| \right) + 2\eta \max \left(|\lambda_v^{max} \left(q_h^- \right)|, |\lambda_v^{max} \left(q_h^+ \right)| \right) \tag{2.56}$$

and

$$\eta = \frac{2N+1}{h\sqrt{\frac{1}{2}\pi}}. \tag{2.57}$$

In this equation, N is the polynomial order and h is the side-length of an element. The penalty term depends on the maximal absolute eigenvalues of both the convective and viscous part of the equations

$$\begin{aligned} |\lambda_c^{max}| &= (\partial \mathbf{F}/\partial \mathbf{Q}) \cdot \mathbf{n}, \\ |\lambda_v^{max}| &= (\partial \mathbf{F}/\partial (\nabla \mathbf{Q} \cdot \mathbf{n})) \cdot \mathbf{n}, \end{aligned} \tag{2.58}$$

in direction of the normal vector $\mathbf{n}$ to the cell face.

This Riemann solver was first published in [GLM08] and used for an ADER-DG scheme in [Dum10]. The timestep per dimension d is restriced to a so-called CFL-type penalty

$$\Delta t \leqslant \mathrm{CFL}\, \alpha(N)\, h \sum_{i=0}^{d-1} \frac{1}{|\lambda_c^{max}| + 2|\lambda_v^{max}| \frac{1}{\alpha(N) h}} \tag{2.59}$$

with N polynomial order and h side length of the cell [Dum10; GLM08]. The eigenvalues are evaluated for the flux Jacobian in direction i. The constant $\alpha(N) \leqslant (2N+1)^{-1}$ is 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 rigoruous stability analysis of the Riemann solver, see [GLM08]. We use a value of 0.7 for the constant CFL.

Paper only describe 1d, explain evaluation of eigenvalues in direction i

Is this the correct paper, or is this in the other part of the series?

2.3. MUSCL-Hancock Finite Volume 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, our discussion and notation follows the one in [Toro9]. Similar to [Toro9] we only describe the two-dimensional case, an extension to three dimensions is straigthforward.

Cite original paper (van Leer)

Source term!

In contrast to the previous DG scheme, we now only store cell averages instead of a full polynomial solution per cell. We achieve higher order then by a reconstruction of a linear function. Let $U_{i,j}$ 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. This is done by assuming a linear function per cell. We use the minmod slope-limiter

$$\mathrm{minmod}(a,b) = \begin{cases} 0.0 & \mathrm{sign}(a) \neq \mathrm{sign}(b) \\ a & |a| < |b| \\ b & \text{otherwise} \end{cases} \tag{2.60}$$

to avoid unphyiscally steep gradients and stabilize the system [LeVo02]. This function then be used to compute the slopes

$$\begin{aligned} s^x_{i,j} &= \Delta x \,\text{minmod}\left(u_{i+1,j} - u_{i,j},\, u_{i,j} - u_{i-1,j}\right), \\ s^y_{i,j} &= \Delta y \,\text{minmod}\left(u_{i,j+1} - u_{i,j},\, u_{i,j} - u_{i,j-1}\right), \end{aligned} \tag{2.61}$$

where Δx and Δy are the inverse cell sizes in x and y direction respectively.

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

$$\begin{aligned} u^{-x}_{i,j} &= u_{i,j} - \frac{1}{2} s^x_{i,j}, \qquad & u^{+x}_{i,j} &= u_{i,j} + \frac{1}{2} s^x_{i,j}, \\ u^{-y}_{i,j} &= u_{i,j} - \frac{1}{2} s^y_{i,j}, \qquad & u^{+y}_{i,j} &= u_{i,j} + \frac{1}{2} s^y_{i,j}. \end{aligned} \tag{2.62}$$

We also use the slopes defined in eq. (2.61) to estimate the gradient of $u_{i,j}$ in each cell by the block matrix

Notation!

$$\nabla u_{i,j} = \left(s^x_{i,j} \middle| s^y_{i,j} \right), \tag{2.63}$$

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. 2.61). To achieve second order in time, we evolve the boundary extrapolated values in time by

Time Evolution

Correct index for gradient of cell average

Actually compare with implementation. Currently gradient is ignored here!

Describe splliting of F into F_x, F_y

Double check update

$$\begin{aligned} \forall_{k \in [-x,+x,-y,+y]} : \hat{u}^k_{ij} = u^k_{i,j} &+ \frac{\Delta t}{2\Delta x} \left[F_x(u^{-x}_{i,j}, \nabla u_{i,j}) - F_x(u^{+x}_{i,j}, \nabla u_{i,j}) \right] \\ &+ \frac{\Delta t}{2\Delta y} \left[F_y(u^{-y}_{i,j}, \nabla u_{i,j}) - F_y(u^{+y}_{i,j}, \nabla u_{i,j}) \right]. \end{aligned} \tag{2.64}$$

Note that we do not need to consider neighboring cells for this step.

Finally, the update can be described by

$$\begin{aligned} u_{i,j}(t^{n+1}) = u_{i,j}(t^n) &\\ &+ \frac{\Delta t}{\Delta x} \text{Riemann}\left(\hat{u}^{+x}_{i-1,j}, \nabla u_{i-1,j};\ \hat{u}^{-x}_{i,j}, \nabla u_{i,j} \right) \\ &- \frac{\Delta t}{\Delta x} \text{Riemann}\left(\hat{u}^{+x}_{i,j}, \nabla u_{i,j};\ \hat{u}^{-x}_{i+1,j}, \nabla u_{i+1,j} \right) \\ &+ \frac{\Delta t}{\Delta y} \text{Riemann}\left(\hat{u}^{+y}_{i,j-1}, \nabla u_{i,j-1};\ \hat{u}^{-y}_{i,j}, \nabla u_{i,j} \right) \\ &- \frac{\Delta t}{\Delta y} \text{Riemann}\left(\hat{u}^{+y}_{i,j}, \nabla u_{i,j};\ \hat{u}^{-y}_{i,j+1}, \nabla u_{i,j+1} \right). \end{aligned} \tag{2.65}$$

We again use the Rusanov-type Riemann solver eq. (2.55). The timestep is subject to a penalty of the form

$$\Delta t \leqslant \mathrm{CFL}\, \frac{h}{|\lambda_c^{\max}| + 2|\lambda_\nu^{\max}|\frac{1}{h}}, \tag{2.66}$$

with a constant value of CFL that is typically close to 0.7.

2.4. Adaptive Mesh Refinement & Finite Volume Limiting

This section is concerned with two different but related concepts:adaptive mesh refinement AMR and finite volume limiting. 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.

Grid, spacefilling curve, and so on.

2.4.1. Adaptive Mesh Refinement

The goal of AMR is increasing the computational efficiency by using a higher spatial resolution in areas of interest. A simple example for this would be using finer mesh cells for the simulation of a wave, while using a low resolution for areas where the wave does not pass through. Of course, the difficult part is defining areas of interest. 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.

Let $f(x) : \mathbb{R}^{N_{\text{vars}}} \to \mathbb{R}$ be a function that maps the discrete solution of a cell to an arbitrary indicator variable. Furthermore, we assume that f is expanded in the space-basis (eq. 2.48). Then the total variation (TV) of this function is defined by

Cite sth for this?

$$\mathrm{TV}\,[f(x)] = \int_\Omega |\nabla f\,(x)\,|. \tag{2.67}$$

This integral can be evaluated efficiently on the reference cell (eqs. 2.23 and 2.48) with Gaussian quadrature (eq. 2.32). In the following, we divide this total variation by the volume of each cell, i.e. we use the volume of the corresponding reference cells as criterion.

A simple way of determining outliers is by how many standard deviations it differs from the mean value. We thus want to compute the mean and variance over all grid cells. To minimize intranodal communications and to simplify the implementation, we do this by performing only pairwise merge operations. The simplest way to compute this would be by using the textbook definitions of (population) mean μ and variance σ^2 of a random variable X

Gradient on reference cell, Gaussian quad., inline

Definition for FV schemes (maybe just mapping to DG-Cells)

$$\begin{aligned} \mu[X] &= \mathbb{E}\,[X] \\ \sigma^2[X] &= \mathbb{E}\left[X^2\right] - \mathbb{E}\,[X]^2\,. \end{aligned} \tag{2.68}$$

The 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

$$\sigma = \sqrt{\frac{n-1}{n}\overline{\sigma^2}}. \tag{2.69}$$

Chebychev's inequality

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 $\mu_\Sigma/n_\Sigma, m_{\text{total}}/n_\Sigma - 1, n_\Sigma$

$$\mathbb{P}(|X - \mu| \geqslant c\sigma) \leqslant \frac{1}{c^2}, \tag{2.70}$$

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

$$\text{evaluate-refinement}(\mathbf{Q}, \mu, \sigma) = \begin{cases} \text{refine} & \text{if } \mathrm{TV}(\mathbf{Q}) \geqslant \mu + T_{\text{refine}}\sigma \\ \text{delete} & \text{if } \mathrm{TV}(\mathbf{Q}) \leqslant \mu + T_{\text{delete}}\sigma \\ \text{keep} & \text{otherwise} \end{cases} \tag{2.71}$$

which refines a cell when its total variations differs by a multiple of the standard deviation. In this equation, the constants T_r and T_d can be used to tailor the trade-off

between the quality of approximation and computational cost. They can be chosen freely, as long as $T_d < T_r$ holds. Chebychev's inequality then guarantees that only a subset 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.

2.4.2. 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 polynomical approximation can lead to wrong or unphyiscal 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 eliminate discontinuities [HW08].

It doesnt work like this exactly.

A relatively recent way of dealing with this problem is the finite volume subcell 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 valid floating point number. We also check whether it is physically admissible. In our case we use the criterion

$$\text{is-admissible}(\mathbf{Q}) = \begin{cases} \text{true} & \rho > 0 \wedge p > 0 \wedge Z > 0 \\ \text{false} & \text{otherwise.} \end{cases} \tag{2.72}$$

The positivity of the pressure implies positivity of energy.

Mention dmp if I end up using it.

If a solution violates our criteria, we recompute the timestep using 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 $N_s = 2N + 1$ subcells. This number of subcells has the advantage that we do not loose spatial resolution. Similarly, we can use the same timestep size for both cell types. The reason for this is that the CFL-conditions for both schemes (eqs. 2.59 and 2.66) coincide for our choice of N_s.

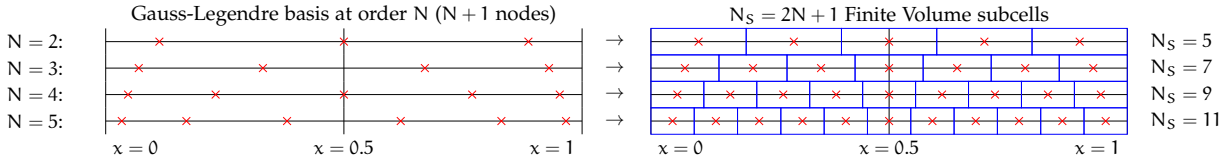

Figure 2.1.: Conversion from DG-dofs to fv-dofs. Image taken from [Dum+18b].

For the sake of brevity, we omit technical details. We refer the interested reader to [DL16].

Cite source for subcells figure 2.1. See guidebook. NOTE THAT IT IS MODIFIED. Also: Fix size, currently font is too small

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 accesible 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

Jinja? Or sth else?

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. In many cases it thus suffices 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`. These implementation 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 exists 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. 2.46). 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. (2.55) and (2.59).
- Computing the gradient for the MUSCL-Hancock scheme (**musclhancock.cpph**) and passing it to the flux (**rusanov.cpph**).

Furthermore, 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 for a cell the observables from its degrees of freedoms and cell 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(Q_cell, Δx_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 $(\mathrm{TV}[\mathtt{f}(\mathbf{Q})], 0, 1)$, where $\mathtt{f}(\mathbf{Q})$ returns an arbitary indicator value per cell. Finally, reduceGlobalObservables is implemented exactly as described in algorithm 1.

Use Q, TV macro here! Same for algorithm for cell.

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

- Add 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 concurrently. 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, updateTimeStepSizes.

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

- Reduce global observables per node in the mappings `FinaliseMeshRefinement`, `FusedTimeStep`, `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 call the method updateG.

- If we use MPI, we also need to reduce the observables over all ranks. *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 methodsof the DG-solver:

  Next and current?!

  - 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. Together with the aforementioned resetting of the observables, this again corresponds to algorithm 2.

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

CHAPTER 4

Scenarios

Unless otherwise noted, we use the constants

$$\begin{aligned} \gamma &= 1.4, \\ \mathrm{Pr} &= 0.7, \\ c_v &= 1.0 \end{aligned} \tag{4.1}$$

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 definition of energy.

4.1. A Manufactured Solution

Following [Dum10], the scenario can be described in primitive variables by setting

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

We set the constants to $\left(p_0 = 0.1, p_b = \gamma^{-1}, \rho_0 = 0.5, \rho_b = 1, \boldsymbol{v}_0 = 0.25(1,1)^\intercal, \boldsymbol{k} = \pi/5(1,1)^\intercal\right)$.

We derive a source term by inserting this solution into the PDE by using the symbolic math toolkit *Sage* [The18]. This procedure is called method of manufactured solutions [Roa02].

4.2. Classical Scenarios

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

4.2.1. Taylor-Green Vortex

$$\begin{aligned} \rho(x,t) &= 1 \\ \boldsymbol{v}(x,t) &= \exp(-2\mu t)\begin{pmatrix} \sin(x)\cos(y) \\ -\cos(x)\sin(y) \end{pmatrix} \\ p(x,t) &= \exp(-4\mu t)\frac{1}{4}\left(\cos(2x)+\cos(2y)\right)+C \end{aligned} \tag{4.3}$$

$C = 100/\gamma$ [Dum+16]

Constants!

As initial conditions we use eq. (4.3) at time $(t = 0)$ and as boundary conditions we impose the solution and the corresponding gradients. Note that this solution is not exactly correct, as it assumes an incompressible flow. Due to the low mach number a compressible flow approaches this solution.

4.2.2. Lid-Driven Cavity Flow

The lid-driven cavity flow is a very simple testing scenario for the Navier-Stokes equations. We use a constant pressure of (what exactly) and a constant density $(\rho = 1.0)$ for the entire domain. The flow is then driven entirely by the boundary conditions. They consist in four no-slip walls where the upper wall has a constant velocity $\boldsymbol{v} = (1.0, 0)$. This scenario is surprisingly difficult for an ADER-DG scheme as the boundary conditions lead to a discontinuity at the top left corner. There, the velocity of the wall is piece-wise constant. Therefore, this scenario is a good choice to test the stability of the finite volume limiter.

4.2.3. ABC-Flow

We use the Arnold–Beltrami–Childress (ABC) flow as a simple example of a three-dimensional scenario [TD16]. This scenario has, unlike the three-dimensional version of the Taylor-Green vortex, at least for the incompressible case an analytical solution

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

Thus, we can simulate this without periodic boundary conditions.

Analytical solution!

4.2.4. Radialsymmetric CJ Detonation Wave

We investigate the radialsymmetric CJ detonation wave scenario from [HLW00]. This scenario for the reactive Euler-equations is set up in a 2D box of size $[-1,1] \times [-1,1]$ surrounded by free-slip walls (eq. 2.19). Inside a circle of radius 0.3 we have totally unburnt gas, outside of this circle we have burned gas. Let $\alpha = \operatorname{atan2}(y,x)$ be the angle in polar coordinates. We use the initial conditions

$$\begin{aligned} \rho_u &= 0.887565 \\ \boldsymbol{v}_u &= -0.577350 \begin{pmatrix} \cos(\alpha) \\ \sin(\alpha) \end{pmatrix} \\ p_u &= 0.191709 \\ Z_u &= 1 \end{aligned} \tag{4.5}$$

for the burnt gas and

$$\begin{aligned} \rho_b &= 1.4 \\ \boldsymbol{v}_b &= 0 \\ p_b &= 1 \\ Z_b &= 0 \end{aligned} \tag{4.6}$$

for the unburnt gas.

Similar to [HLW00], we use the constants

$$\begin{aligned} \gamma &= 1.4, \\ \mathrm{Pr} &= 0.7, \\ c_v &= 2.5, \\ c_p &= 3.5, \\ R &= 1.0. \end{aligned} \tag{4.7}$$

What about viscous effects?

4.3. Atmospheric Scenarios

The scenario is described in terms of potential temperature θ

$$\theta = T \frac{p_0}{p}^{R/c_p}, \tag{4.8}$$

where $p_o = 10 \times 10^5$ Pa is the reference pressure. Solving for T leads to

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

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

$$\theta = \bar{\theta} + \theta'. \tag{4.10}$$

The initial conditions for pressure and density can be computed directly from the PDE once few assumptions have been given. We assume that $\bar{\theta}$ is constant over the entire domain. Inserting the definition of potential temperature, given by eq. (4.9), into the equation of state (see eq. (2.5)) leads us to

$$p(z) = \rho(z) R \left(\bar{\theta} \frac{p}{p_0} \right)^{R/c_p}, \tag{4.11}$$

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

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

or equivalently

$$\rho(z) = \frac{p_0^{\frac{R}{c_p}} p^{\frac{1}{\gamma}}(z)}{R\bar{\theta}}. \tag{4.13}$$

We want to consider flow in hydrostatic equilibrium, i.e. flow where the force originating from the pressure-gradient is exactly balanced by gravitation. 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)

$$\begin{aligned} \frac{d}{dz} p(z) &= -g\rho(z) = -\frac{g p_0^{\frac{R}{c_p}} p^{\frac{1}{\gamma}}(z)}{R\bar{\theta}} \\ p(0) &= p_0 \end{aligned} \tag{4.14}$$

where we use the reference pressure p_0 as the pressure on ground level. This ODE can be now simply solved by separation of variables. We then obtain after simplifying the solution

Refer to hydrostatic balance eq. in methods chapter!eq. (2.13)

$$p(z) = \left(\left(1 - \frac{1}{\gamma} \right) \left(C - \frac{g p_0^{\frac{R}{c_p}} z}{R\bar{\theta}} \right) \right)^{\frac{c_p}{R}}, \tag{4.15}$$

with constant of integration

$$C = \frac{c_p p_0^{\frac{R}{c_p}}}{R}. \tag{4.16}$$

We can now compute the initial conditions for a scenario where the background potential energy is in hydrostatic equilbrium with a pertubation that is not in equilbrium. We first compute the background pressure with background potential temperature $\overline{\theta} = \text{const.}$ using eq. (4.15). Next, we compute the pertubated potential temperature θ and use eq. (4.9) to convert it to temperature. The density $\rho(x)$ can now be evaluated with eq. (2.6). Finally, we compute the energy by inserting pressure and density into the equation of state (2.5).

This leaves us with one problem: The adiabatic no-slip boundary conditions defined by eq. (2.18) on page 5 are no longer valid. Concretely, we need to impose a viscous heat flux to ensure that the atmosphere stays in hydrostatic balance [GR08]

Flux correct? Fix notation

$$F^{\text{visc}} = \kappa \frac{\partial \overline{T}}{\partial z} = \frac{R\overline{\theta}\left(\frac{\overline{p}(z)}{p_0}\right)^{\frac{R}{c_p}} \frac{d}{dz}\overline{p}(z)}{c_p \overline{p}(z)}. \tag{4.17}$$

The background pressure can be reconstructed by eq. (4.15), its derivative is given by eq. (4.14).

We use the following set of fluid dependent constants for these scenarios

$$\begin{aligned} \gamma &= 1, \\ \text{Pr} &= 0.71, \\ R &= 287.058, \\ p_0 &= 10000, \end{aligned} \tag{4.18}$$

the other constants are evaluated following eq. (2.9).

4.3.1. Two Bubbles

As a simple test case we consider a scenario where we have one large warm air bubble in the middle, and a small cold bubble on the top. This scenario was first published by Robert [Rob93]. We use a domain of 1 km × 1 km. The initial conditions of the scenario are given by a pertubation of a hydrostatic equlibrium. We use a background potential temperature of $\overline{\theta} = 300$.

Let

$$r^2 = (x - x_0)^2 + (z - z_0)^2 \tag{4.19}$$

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

$$\theta' = \begin{cases} A & r \leqslant a \\ A \exp\left(-(r-a)^2/s^2\right) & r > a, \end{cases} \tag{4.20}$$

where A denotes the maximal pertubation, a is the size of the bubble and s is the decay rate. We have two bubbles, with the values

$$\begin{array}{lllll} \text{warm:} & A = 0.5 & a = 150\,\text{m} & s = 50\,\text{m} & x_c = 500\,\text{m} \quad z_c = 300\,\text{m}, \\ \text{cold:} & A = -0.15 & a = 0\,\text{m} & s = 50\,\text{m} & x_c = 560\,\text{m} \quad z_c = 640\,\text{m}. \end{array} \tag{4.21}$$

Similar to [Mül+10], we use a constant viscosity of $\mu = 0.1$ to regularize the solution.

4.3.2. Cosine Bubble

A similar scenario is the cosine bubble as described in [GR08]. In this scenario the flow is driven by one warm bubble. We use the same procedure as before to compute the initial conditions, again with $\overline{\theta} = 300$ but with a pertubation of the form

$$\theta' = \begin{cases} A/2\,[1 + \cos(\pi r)] & r \leqslant a \\ 0 & r > a \end{cases} \tag{4.22}$$

, where r again is the radius to the center (eq. 4.19) and constants

$$A = 0.5 \quad a = 250\,\text{m} \quad x_c = 500\,\text{m} \quad z_c = 350\,\text{m}. \tag{4.23}$$

Similar to the previous scenario, we use a constant viscosity of $\mu = 0.1$.

Are we? Check [GR08]!

CHAPTER 5

Results

5.1. Convergence Test

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

Order	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	4.44	4.44	4.67
5	5.37	5.34	5.28
6	5.43	5.42	5.30

Before computing the error we first need to define the L_p norms for a $p > 0$ by

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

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 $\hat{f}(\mathbf{x}, t)$. The error at a time t is thus defined as

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

We first observe that for our approximation space Ω we can split up the error as a sum over all cells

$$\text{Total-Error}(t, p) = \sum_{C \in \Omega} \text{Total-Error}_C(t, p) \tag{5.3}$$

where Total-Error_C is the error in the cell C.

We evaluate the cell-wise error with Gaussian quadrature using eq. (2.27)

Describe how to integrate the error for cells, and effect of volume

$$\|f_K(x)\|_p = \left(V \sum_i |f(\mathbf{x}_i, t) - \hat{f}(\mathbf{x}_i, t)|^p w_i \right)^{1/p}, \tag{5.4}$$

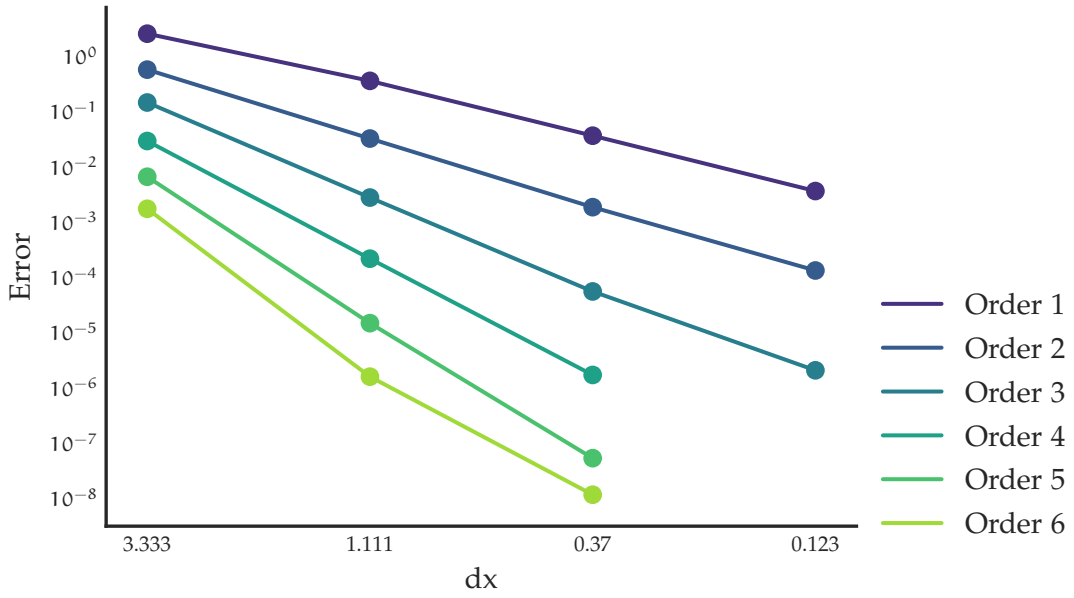

Figure 5.1.: L_2-Error vs. Grid Size

where $V = \Delta x \Delta y$ is the volume of each two-dimensional cell. The sum runs over all quadrature nodes x_i with weights w_i. We use ten nodes.

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

We compute the error for the manufactured-solution scenario (section 4.1). Looking at the plot for the L_2 error (refer to plot here) shows that our method converges for all tested polynomial orders.

Preliminary results used here!

To compute the numerical order of convergence, we perform a linear regression of the logarithm of the error vs. the logarithm of the meshsize. The size of the slope is then the convergence order. We can see (table 5.1) that our numerical convergence rate increases with the polynomial order. We do not achieve the optimal theoretical order of convergence of $N + 1$. In [Dum10], which use 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 odd order only achieve a numerical convergence order of $N + 1/2$.

5.2. Accurate CFD

Show:

- Taylor-Green High Order
- ABC Medium Order

5.3. Accurate Clouds

- Cosine Bubble
- Two Bubbles

5.4. Time-to-solution: AMR vs Static Mesh

To show the effectiveness of the AMR, we compare the time to solution of a simulation using AMR and another simulation with a fully refined mesh. In detail, we compare

AMR We use a coarse mesh with just 9×9 cells. Each cell can be refined up to three times. That is, the cells have sidelengths of roughly $(111.1\,\mathrm{m}, 37.04\,\mathrm{m}, 12.35\,\mathrm{m})$. As refinement thresholds we use $T_{\text{refine}} = 2.5$ and $T_{\text{delete}} = -0.5$ in eq. (2.71).

Fine We use a uniform grid where all cells have a sidelength corresponding to the finest mesh of the AMR-test case. In addition, we disable the reduction of global observables.

We use an ADER-DG-method of order 4 and thus have a minimal effective resolution of ca. 3.09 m.

To ensure a fair comparison, we ran both configurations with the same optimisation option.

We use a single node of the Haswell-nodes of the Supermuc (Phase 2). The node has two xxxCPUs with 28 cores each. Both CPUs currently run on 1.8 GHz We use the TBB parallelisation of *ExaHyPE*. To find a reasonable parameter for the number of background threads, we ran a grid search. We ran three AMR testcases for a simulation time of 1 min. The best median performance was achieved by using seven background threads.

name cpus

Noisy, show results

5.5. Reactive Euler & Navier Stokes

- With viscous effects

- Without viscous effects

CHAPTER 6

Conclusion

Conclusion here.

APPENDIX A

PDE: Eigenvalues & Stuff

In this appendix we collect the derivation of the two-dimensional eigenvalues and spatial derivatives of variables. We only present results for two-dimensions, the results for three-dimensions follows trivially.

In x-direction

$$\boldsymbol{\nabla} Q_n = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ -\frac{j_x j_y}{\rho^2} & \frac{j_y}{\rho} & \frac{j_x}{\rho} & 0 & 0 \\ -\frac{1}{2}\left(2\,g_y - \frac{j_x^2}{\rho^2} - \frac{j_y^2}{\rho^2}\right)(\gamma-1) - \frac{j_y^2}{\rho^2} & -\frac{(\gamma-1)j_x}{\rho} & -\frac{(\gamma-1)j_y}{\rho} + \frac{2\,j_y}{\rho} & \gamma-1 & -(\gamma-1)q_0 \\ -\frac{\left(2\,g_y - \frac{j_x^2}{\rho^2} - \frac{j_y^2}{\rho^2}\right)(\gamma-1)j_y}{2\,\rho} - \frac{(E+pvar)j_y}{\rho^2} & -\frac{(\gamma-1)j_x j_y}{\rho^2} & -\frac{(\gamma-1)j_y^2}{\rho^2} + \frac{E+pvar}{\rho} & \frac{\gamma j_y}{\rho} & -\frac{(\gamma-1)j_y q_0}{\rho} \\ -\frac{Z j_y}{\rho^2} & 0 & \frac{Z}{\rho} & 0 & \frac{j_y}{\rho} \end{pmatrix} \tag{A.1}$$

Heat flux[1]:

$$\begin{aligned} a &= -\frac{E}{\rho^2} + \frac{j_x^2 + j_y^2}{\rho^3} \\ b &= -\frac{j_x}{\rho^2} \\ c &= -\frac{j_y}{\rho^2} \\ \frac{\partial T}{\partial x} &= \frac{1}{c_v}\left(a\frac{\partial \rho}{\partial x} + b\frac{\partial j_x}{\partial x} + c\frac{\partial j_y}{\partial x} - q_0\frac{\rho\frac{\partial}{\partial x}Z - Z\frac{\partial}{\partial x}\rho}{\rho^2}\right) \\ \frac{\partial T}{\partial y} &= \frac{1}{c_v}\left(a\frac{\partial \rho}{\partial y} + b\frac{\partial j_x}{\partial y} + c\frac{\partial j_y}{\partial y} - q_0\frac{\rho\frac{\partial}{\partial y}Z - Z\frac{\partial}{\partial y}\rho}{\rho^2} - g\right) \end{aligned} \tag{A.2}$$

[1] a, b, c terms 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. 18).

[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. 15).

[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 p. 16).

[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. 6, 9, 10, 12).

[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 p. 23).

[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 p. 6).

[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. 16).

[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. 2, 6, 12, 22, 29).

[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. 12).

[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. 5, 26, 27).

[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 p. 2).

[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 p. 24).

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

[LeV02] R. J. LeVeque. *Finite Volume Methods for Hyperbolic Problems*. Cambridge University Press, 2002. DOI: 10.1017/cbo9780511791253 (see p. 13).

[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. 4, 27).

[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. 22).

[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 p. 26).

[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 p. 23).

[The18] The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 8.3)*. http://www.sagemath.org. 2018 (see p. 22).

[Toro9] E. F. Toro. *Riemann Solvers and Numerical Methods for Fluid Dynamics*. Springer Berlin Heidelberg, 2009. DOI: 10.1007/b79761 (see p. 12).

[Was04] L. Wasserman. *All of Statistics*. Springer New York, 2004. DOI: 10.1007/978-0-387-21736-9 (see p. 15).

[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. 17).