WESTFÄLISCHE
WILHELMS-UNIVERSITÄT
MÜNSTER

Numerical methods for kinetic equations

Masterarbeit
zur Erlangung des akademischen Grades
Master of Science

Westfälische Wilhelms-Universität Münster
Fachbereich Mathematik und Informatik
Institut für Numerische und Angewandte Mathematik

Betreuung:
Prof. Dr. Mario Ohlberger

Eingereicht von:
Tobias Leibner

Münster, September 2015

Abstract

Kinetic equations play an important role in many physical applications. Prominent examples are the Boltzmann equation of gas dynamics and the radiative transfer equation. In general, analytic solutions are not available and thus numerical solutions have to be found. However, due to their high dimensionality, kinetic equations cause a great amount of computational cost which may effectively make it impossible to get a sufficiently accurate solution using standard numerical solvers. Thus, methods that can find approximate solutions with less effort have to be used.

A popular approach is to express the solution in terms of the first moments of the kinetic equation. This eliminates the dependency on the velocity variable and reduces the computational cost significantly. However, the resulting hyperbolic system of equations still has to be solved in several dimensions. Furthermore, many moments and thus a large system of equations may be necessary to get a reasonable approximation to the true solution. Hence, efficient solvers are still required to solve the problem in reasonable time.

The goal of this thesis was the implementation of an efficient and generic solver for hyperbolic systems of equations in the C++ software framework DUNE. The implementation was tested against problems with known solution and existing solvers for the moment models.

Contents

1 Kinetic equations

Kinetic equations play an important role in many physical applications. One of the most prominent examples is the Boltzmann equation which was derived by the Austrian physicist Ludwig Boltzmann in 1872 [9] and still forms the basis for the kinetic theory of gases. The Boltzmann equation or similar kinetic equations proved to be applicable not only to classical gases but also to electron transport in solids and plasmas, neutron transport in nuclear reactors, photon transport in superfluids and radiative transfer, among others [16, 52]. More recently, kinetic equations were also derived in the context of biological modelling, e.g. for studying cell movement or wolf migration [32, 41].

The kinetic equations we regard have the form

$$\partial_t p(t,\mathbf{x},\mathbf{v}) + \mathbf{v}\cdot\nabla p(t,\mathbf{x},\mathbf{v}) = \mathcal{L}p(t,\mathbf{x},\mathbf{v}), \tag{1.1}$$

where ∇p denotes the gradient of p with respect to the spatial variable $\mathbf{x}\in\mathbb{R}^d$ and $\partial_t p$ the partial derivative with respect to the time variable $t\in\mathbb{R}_+ := [0,\infty]$. Here and in the remainder of the thesis we will use lower-case bold notation for vectors and upper-case bold notation for matrices to distinguish them from scalar quantities more easily. Typically, p is a density (e.g. the density of particles in a domain or a probability density) describing the distribution of particles (gas molecules, photons, cells, etc.) at time t with respect to position $\mathbf{x}\in\Omega\subset\mathbb{R}^d$ and velocity $\mathbf{v}\in V\subset\mathbb{R}^d$. $\mathcal{L}$ is a (possibly nonlinear) operator modelling velocity changes of the particles. If $\mathcal{L}=0$, this equation is just describing transport of particles where each particle has fixed velocity that does not change over time. Thus, kinetic equations are also often referred to as transport equations.

Together with the kinetic equation we usually have an initial condition

$$p(0,\mathbf{x},\mathbf{v}) = p_0(\mathbf{x},\mathbf{v}) \tag{1.2}$$

for a given p_0.

Often, Ω is not the whole space $\mathbb{R}^d$ but only a bounded domain. In that case, we also impose boundary conditions

$$p(t,\mathbf{x},\mathbf{v}) = p_b(t,\mathbf{x},\mathbf{v}) \quad \text{for} \quad \mathbf{v}\cdot\mathbf{n}(\mathbf{x}) < 0 \tag{1.3}$$

where $\mathbf{n}$ is the outer normal to the boundary $\partial\Omega$ of the spatial domain. Note that we only prescribe boundary conditions for velocities that correspond to particles entering the domain. The number of particles leaving the domain is fully determined by the particle density and the kinetic equation and thus cannot be prescribed if the problem shall be well-posed [23].

In principle, standard numerical solvers for partial differential equations (PDEs), such as finite difference methods, can be used to solve a given kinetic equation directly. However, as the density p depends on time, space and velocity, the problem must be solved in $\mathbb{R}^{2d+1}$ (or at least in a bounded domain therein), where d is the spatial dimension. For $d = 3$ in a typical application this results in a dimension of 7 which causes high computational cost and may de facto render it impossible to solve the problem with sufficient accuracy. Thus, methods that can find approximate solutions with less effort have to be used. A variety of Monte Carlo methods has been developed for this purpose. However, they are very inefficient when macroscopic time and space scales are regarded but the particle processes take place on the microscopic scale (e.g. [8, 58, 67]). Sometimes, multiscale methods can be used to obtain limit equations in that case (e.g. [19, 41, 42, 43]).

In this thesis, we focus on a different approach, the so-called moment models. These models transfer the kinetic equation to a coupled system of PDEs that is independent of the velocity variable (see Section 2). This reduces the dimension of the problem from $2d + 1$ to $d + 1$ and thus reduces computational cost significantly. In Section 3, we investigate the resulting systems of PDEs. Numerical methods for the solution of moment models are regarded in Section 4. Even with reduced dimension efficient solvers are still required to solve the problem in reasonable time. One goal of this thesis is the implementation of an efficient solver in the C++ software framework DUNE (see Section 5). Numerical tests of the implementation and a comparison with existing solvers for moment models can be found in Section 6.

In the following, we will regard some examples to illustrate the occurrence of kinetic equations in applications and to get a better idea of what the abstract operator $\mathcal{L}$ may look like in practice.

Example 1.1 (Boltzmann Transport Equation)**.** We want to describe the distribution of molecules within a gas. For that purpose, we would like to derive an equation for the particle density $p(t, \mathbf{x}, \mathbf{v})$. We will follow the outlines in [66, 77] here.

If we regard a small volume of the domain, there are three reasons why the distribution of particles in this volume would change:

- Molecules leave or enter the volume due to the motion of molecules (diffusion).
- Due to the influence of an external force $\mathbf{F}$, molecules change their velocity.
- Collisions between molecules can lead to velocity changes.

Thus, we have three contributions to the time derivative of p:

$$\partial_t p = \partial_t p|_{\text{diff}} + \partial_t p|_{\text{force}} + \partial_t p|_{\text{coll}}.$$

The diffusion of molecules can easily be modelled: If a molecule at position $\mathbf{x}$ has velocity $\mathbf{v}$, it will be at position $\mathbf{x} + \mathbf{v}dt$ after a short time dt. Thus,

$$p(t + dt, \mathbf{x}, \mathbf{v}) = p(t, \mathbf{x} - dt\mathbf{v}, \mathbf{v})$$

and by Taylor expansion for infinitely small dt

$$p(t,\mathbf{x},\mathbf{v}) + dt\partial_t p(t,\mathbf{x},\mathbf{v}) = p(t,\mathbf{x},\mathbf{v}) - dt\mathbf{v}\cdot\nabla p(t,\mathbf{x},\mathbf{v})$$

which gives

$$\partial_t p|_{\text{diff}}(t,\mathbf{x},\mathbf{v}) = -\mathbf{v}\cdot\nabla p(t,\mathbf{x},\mathbf{v})$$

for the diffusion-induced change in p.

For an external force $\mathbf{F}$, the velocity of a particle changes according to

$$\partial_t\mathbf{v} = \frac{1}{m}\mathbf{F}$$

where m is the molecule mass. As in the diffusion case we can assume that a particle with velocity $\mathbf{v} - dt\partial_t\mathbf{v}$ will have velocity $\mathbf{v}$ at time $t + dt$, so we get

$$\partial_t|_{\text{force}}p(t,\mathbf{x},\mathbf{v}) = -\partial_t\mathbf{v}\cdot\nabla_{\mathbf{v}}p(t,\mathbf{x},\mathbf{v}) = -\frac{1}{m}\mathbf{F}\cdot\nabla_{\mathbf{v}}p(t,\mathbf{x},\mathbf{v}).$$

For the collision part, Boltzmann assumed that only binary collisions need to be considered (dilute gas), that the effect of the external force and container walls on the collision rate is negligible and that velocity and position are uncorrelated. Under the additional assumption that the gas molecules are hard spheres the collision term takes the form [10]

$$\partial_t|_{\text{coll}}p(t,\mathbf{x},\mathbf{v}) = \int\limits_{\mathbb{R}^3}\int\limits_{S^2} |\mathbf{n}\cdot(\mathbf{v}-\mathbf{v}_2)|\left(p(\mathbf{v}')p(\mathbf{v}_2') - p(\mathbf{v})p(\mathbf{v}_2)\right)\mathrm{d}\mathbf{n}\mathrm{d}\mathbf{v}_2$$

where we omitted the $(t,\mathbf{x})$ dependency of p on the right-hand side. Here, S^2 is the three-dimensional unit sphere and $\mathbf{n}\in S^2$ is the unit vector parallel to the line segment joining the molecule centres at the time of collision. The new velocities $\mathbf{v}', \mathbf{v}_2'$ can be calculated from the old velocities $\mathbf{v}, \mathbf{v}_2$ and the angle of collision $\mathbf{n}$. Actually, the collision term consists of two parts [10]: a loss term with the integrand $-|\mathbf{n}\cdot(\mathbf{v}-\mathbf{v}_2)|p(\mathbf{v})p(\mathbf{v}_2)$ that models collisions involving molecules with velocity $\mathbf{v}$ that have a different velocity after the collision, and a gain term with the integrand $|\mathbf{n}\cdot(\mathbf{v}-\mathbf{v}_2)|p(\mathbf{v}')p(\mathbf{v}_2')$ that models collisions where molecules obtain the velocity $\mathbf{v}$.

Taken together, we get the Boltzmann transport equation

$$\partial_t p + \mathbf{v}\cdot\nabla p = -\frac{1}{m}\mathbf{F}\cdot\nabla_{\mathbf{v}}p + \int\limits_{\mathbb{R}^3}\int\limits_{S^2} |\mathbf{n}\cdot(\mathbf{v}-\mathbf{v}_2)|\left(p(\mathbf{v}')p(\mathbf{v}_2') - p(\mathbf{v})p(\mathbf{v}_2)\right)\mathrm{d}\mathbf{n}\mathrm{d}\mathbf{v}_2. \tag{1.4}$$

Example 1.2 (Mesenchymal motion)**.** Mesenchymal motion is a form of cellular movement through tissues which consist of fibre networks. An example is the migration of tumour cells through collagen networks during metastasis. Cells migrate within fibre networks and change their directions according to the orientational distribution of fibres. Moreover, cells actively remodel the tissue by excreting degrading enzymes (e.g. protease) to generate sufficient space to migrate in (see Figure 1 and [86]). In order to derive a mathematical model for mesenchymal motion, we make the ansatz of a kinetic equation.

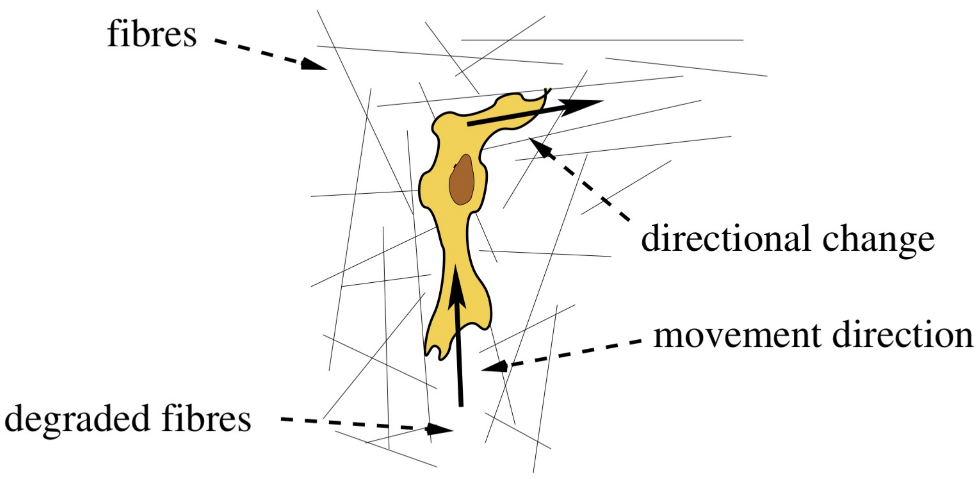


Figure 1: Scheme of mesenchymal motion. The cell movement is guided by the surrounding tissue fibres. At the same time, the cells remodel the fibre network by expressing proteases. From [83].

Let $\Omega \subset \mathbb{R}^d$ be an arbitrary domain. Let $V = [s_1, s_2] \cdot S^{d-1}$ be the set of all possible velocities the cells can have. We are interested in an equation for the cell density $p(t, \mathbf{x}, \mathbf{v})$ at time $t \in \mathbb{R}_+$ and position $\mathbf{x} \in \Omega$ with velocity $\mathbf{v} \in V$.

Let $q(t, \mathbf{x}, \boldsymbol{\theta})$ be a probability density that describes the distribution of fibre directions $\boldsymbol{\theta} \in S^{d-1}$. As q is a probability density, we have

$$\int\limits_{S^{d-1}} q(t, \mathbf{x}, \boldsymbol{\theta}) \, \mathrm{d}\boldsymbol{\theta} = 1$$

for every $(t, \mathbf{x}) \in \mathbb{R}_+ \times \Omega$. We assume a "run and tumble" model for the mesenchymal motion, i.e. cells move at a constant velocity ("run") until they reorient ("tumble") and choose a new velocity. We assume that these reorientation processes happen with a constant rate μ and take no time (i.e. the cells "jump" to a new velocity). We further assume that the new directions of movement follow the distribution $q(t, \mathbf{x}, \boldsymbol{\theta})$, whereas the new speed is chosen randomly from $[s_1, s_2]$. So if we define

$$\hat{\mathbf{v}} = \frac{\mathbf{v}}{\|\mathbf{v}\|} \quad \text{for} \quad \mathbf{v} \in V \quad \text{and} \quad \omega = \int\limits_V q(t, \mathbf{x}, \hat{\mathbf{v}}) \, \mathrm{d}\mathbf{v},$$

where $\|\mathbf{v}\|$ is the Euclidean norm of $\mathbf{v}$, then ω is a probability density on V and gives the distribution of newly chosen velocities. Thus, we get the kinetic equation

$$p_t(t, \mathbf{x}, \mathbf{v}) + \mathbf{v} \cdot \nabla p(t, \mathbf{x}, \mathbf{v}) = -\mu p(t, \mathbf{x}, \mathbf{v}) + \mu \frac{q(t, \mathbf{x}, \hat{\mathbf{v}})}{\omega} \int\limits_V p(t, \mathbf{x}, \mathbf{v}') \, \mathrm{d}\mathbf{v}' \tag{1.5}$$

where the first term on the right-hand side describes the fraction of cells that originally moved with velocity $\mathbf{v}$ and choose a new velocity different from $\mathbf{v}$ at a rate μ, and the second term on the right-hand side accounts for the cells that choose $\mathbf{v}$ as their new velocity.

Furthermore, cells remodel the tissue by secreting proteases and other enzymes that cut fibres along their way. Fibres that are orthogonal to the cell's movement direction are cut, while fibres that align with the cell's movement direction remain intact. Under these assumptions, Hillen [40] derives the following equation for the tissue modifications:

$$q_t(t, \mathbf{x}, \boldsymbol{\theta}) = \kappa \left(\Pi(t, \mathbf{x}, \boldsymbol{\theta}) - A(t, \mathbf{x}) \right) q(t, \mathbf{x}, \boldsymbol{\theta}) \int\limits_V p(t, \mathbf{x}, \mathbf{v}) \, \mathrm{d}\mathbf{v}. \tag{1.6}$$

Here, κ is a factor describing cutting efficiency and the difference $\Pi(t, \mathbf{x}, \boldsymbol{\theta}) - A(t, \mathbf{x})$ is a measure of alignment between the mean direction of cell movement and the fibre direction $\boldsymbol{\theta}$. If the fibre direction is "less parallel" to the cell movement than the average of the fibre distribution, it is degraded. Otherwise its probability weight is increased. For details see [40].

We could regard the coupled partial differential equations (1.5) and (1.6) simultaneously but in the following we concentrate on the kinetic equation (1.5) and assume $q(t, \mathbf{x}, \boldsymbol{\theta})$ is given whenever we refer to equation (1.5).

2 Moment methods for kinetic equations

2.1 General idea

The moment models are motivated by the fact that in many applications we are interested in the spatial distribution of particles, while the velocity distribution is not that important. Therefore, we would like to reduce the dimension of the problem and derive an equation for the spatial particle distribution

$$\bar{p}(t,\mathbf{x}) = \int_V p(t,\mathbf{x},\mathbf{v})\mathrm{d}\mathbf{v}.$$

This can essentially be done by integrating over the velocity component, but it turns out that the integration does not give a closed equation for $\bar{p}$. Instead, we have to regard a system of equations obtained by multiplication by appropriate functions of the velocity and subsequent integration.

Definition 2.1. Let $L^1_+(V)$ be the set of all $L^1(V)$ functions that are positive almost everywhere. Let $p \in L^1_+(V)$ and $\boldsymbol{\ell} = (\ell_1, \dots, \ell_d) \in \mathbb{N}_0^d$ be a multi-index. Define the $\boldsymbol{\ell}$-th *moment* of p as

$$p^{(\boldsymbol{\ell})}(t,\mathbf{x}) := \int_V \mathbf{v}^{\boldsymbol{\ell}} p(t,\mathbf{x},\mathbf{v})\mathrm{d}\mathbf{v}.$$

We say the moment $p^{(\boldsymbol{\ell})}$ has order k if $|\boldsymbol{\ell}| = k$. Here we use the multi-index notation $\mathbf{v}^{\boldsymbol{\ell}} = v_1^{\ell_1} v_2^{\ell_2} \dots v_d^{\ell_d}$ and $|\boldsymbol{\ell}| = \ell_1 + \dots + \ell_d$.

Definition 2.2. Let $\mathbb{M}(V)$ be a linear space of real-valued functions on V and let $B = \{e_i \,|\, i \in \mathbb{N}\}$ be a basis of this space. Define the i-th moment of p with respect to the basis B as

$$p^{(i)}(t,\mathbf{x}) := \int_V e_i(\mathbf{v}) p(t,\mathbf{x},\mathbf{v})\mathrm{d}\mathbf{v}.$$

We say the moment $p^{(i)}$ has order k if e_i is a polynomial of order k.

Usually, the space $\mathbb{M}(V)$ is chosen to be the space $\mathbb{P}(V)$ of polynomials over V. We will always assume that $1 \in \mathbb{M}$ (i.e. the constant function with value 1) and that $e_0 = 1$. This ensures that the zeroth moment is the spatial particle distribution $\bar{p}$.

By multiplying the kinetic equation by the respective basis functions and integrating over V, we obtain a system of infinitely many equations for the moments of p. For instance, for the mesenchymal motion kinetic equation (1.5) in dimension $d = 2$, using the standard polynomial basis $\mathbf{v}^{\boldsymbol{\ell}}$ we get the system of equations

$$\partial_t p^{(\ell_1,\ell_2)} = -\partial_{x_1} p^{(\ell_1+1,\ell_2)} - \partial_{x_2} p^{(\ell_1,\ell_2+1)} - \mu p^{(\ell_1,\ell_2)} + \mu \int_V \mathbf{v}^{(\ell_1,\ell_2)} \frac{q}{\omega} \mathrm{d}\mathbf{v}\, p^{(0,0)} \tag{2.1}$$

for $(\ell_1, \ell_2) \in \mathbb{N}_0^2$, where we interchanged differentiation and integration. If $\mathbb{M}(V) \neq \mathbb{P}(V)$, we should require $v_j e_i \in \mathbb{M}(V)$ for all $j = 1, \dots, d$ and for all $i \in I$ in addition to the assumptions above to get a system of equations depending only on the moments.

To obtain a finite set of equations, we truncate the system at order N, i.e. use only equations for the moments with order not greater than N (or for moments corresponding to a subset $J \subset I$ in the general case). However, because of the factor $\mathbf{v}$ in the transport term of the kinetic equation, the equation for a moment with order $|\ell| = k$ always depends on moments of order $k+1$, as can be seen exemplarily in (2.1). Hence, if we truncate the moment equations at a finite order, the resulting system of equations is always underdetermined. In order to get a unique solution, we therefore need to close the resulting system by making additional assumptions.

Example 2.3 ([72, Section 2])**.** If we regard the system (2.1) in one spatial dimension ($d = 1$) and with velocity space $V = [-1, 1]$, and truncate at order 1, we get the two equations

$$\partial_t p^{(0)} = -\partial_x p^{(1)}, \tag{2.2a}$$

$$\partial_t p^{(1)} = -\partial_x p^{(2)} - \mu p^{(1)} + \mu p^{(0)} \int_{-1}^{1} v \frac{q}{\omega} \mathrm{d}v \tag{2.2b}$$

for the three variables $p^{(0)}$, $p^{(1)}$, $p^{(2)}$ (recall $\int \frac{q}{w} \mathrm{d}v = 1$, so the μ terms cancel in equation (2.2a)). To close the system, we assume that the underlying density is linear in v, i.e.

$$p(t, x, v) = a(t, x)v + b(t, x). \tag{2.3}$$

Using the definitions of the moments $p^{(0)}$, $p^{(1)}$, we get

$$\begin{aligned} p^{(0)}(t, x) &= \int_{-1}^{1} a(t, x)v + b(t, x)\mathrm{d}v = 2b(t, x), \\ p^{(1)}(t, x) &= \int_{-1}^{1} a(t, x)v^2 + b(t, x)v\mathrm{d}v = \frac{2}{3}a(t, x). \end{aligned}$$

Hence $p = \frac{1}{2}p^{(0)} + \frac{3}{2}p^{(1)}v$ and therefore

$$p^{(2)} = \int_{-1}^{1} v^2 \left(\frac{1}{2}p^{(0)} + \frac{3}{2}p^{(1)}v \right) \mathrm{d}v = \frac{1}{3}p^{(0)}. \tag{2.4}$$

Plugging (2.4) into (2.2) gives a closed system of equations for the two moments $p^{(0)}$ and $p^{(1)}$. However, when closing the system of moment equations like that, we cannot guarantee that the resulting moments are physically reasonable. For example, the assumed density (2.3), and thus the zeroth moment $p^{(0)}$, can become negative.

Example 2.3 shows that, before dealing with moment closures, we should examine which moments are reasonable, so we can check the results of our closure later on. This is known as the problem of realizability which we investigate in the next section.

2.2 Realizability

We restrict ourselves to the one-dimensional case and a monomial basis for now. The question whether a vector $\mathbf{p} \in \mathbb{R}^{N+1}$ can occur as the vector of moments of a density p is known as the truncated Hausdorff moment problem [74]:

Definition 2.4. Let $\mathbf{p} = (p^{(0)}, p^{(1)}, \dots, p^{(N)})^T \in \mathbb{R}^{N+1}$ and $[v_L, v_R] \subset \mathbb{R}$ be an interval. The *truncated Hausdorff moment problem* with data $\mathbf{p}$ entails finding a density $p \in L^1_+(V)$ such that

$$\int\limits_{v_L}^{v_R} v^j p(v) \mathrm{d}v = p^{(j)}, \; j = 0, \dots, N. \tag{2.5}$$

If the truncated Hausdorff moment problem is solvable for $\mathbf{p} \in \mathbb{R}^{N+1}$, we call $\mathbf{p}$ *realizable.* The set of all realizable vectors in $\mathbb{R}^{N+1}$ is the *realizability domain* $\mathcal{R}_N$.

The realizability domain can be completely characterized by the definiteness of a set of Hankel matrices built by the components of the moment vector $\mathbf{p}$. For two matrices $\mathbf{A}, \mathbf{B}$ we will use the notation $\mathbf{A} > \mathbf{B}$ if $\mathbf{A} - \mathbf{B}$ is positive definite.

Theorem 2.5. *For a given* $\mathbf{p} \in \mathbb{R}^{N+1}$ *we define the matrices*

$$\mathbf{A}(k) := (p^{(i+j)})_{i,j=0}^k, \; \mathbf{B}(k) := (p^{(i+j+1)})_{i,j=0}^k, \; \mathbf{C}(k) := (p^{(i+j)})_{i,j=1}^k.$$

The moment problem (2.5) *is solvable*

- *for* $N = 2k+1$ *if and only if*

$$v_R \mathbf{A}(k) > \mathbf{B}(k) > v_L \mathbf{A}(k),$$

- *for* $N = 2k$ *if and only if* $\mathbf{A}(k) > 0$ *and*

$$(v_L + v_R)\mathbf{B}(k-1) > v_L v_R \mathbf{A}(k-1) + \mathbf{C}(k).$$

Proof. See [22], [72]. □

Note that similar conditions for the realizability of moments with respect to any other basis of the polynomial space $\mathbb{P}(V)$ can be derived by applying a change of basis first and then using Theorem 2.5.

To check for the positive definiteness of a matrix, we can use the well-known Sylvester's criterion.

Theorem 2.6 (Sylvester's criterion)**.** *Let* $\mathbf{M} \in \mathbb{C}^{n \times n}$ *be hermitian. Then*

$$\mathbf{M} > 0 \quad \textit{if and only if} \quad \det(\mathbf{M}_k) > 0 \textit{ for all } k = 1, \dots, n,$$

where $\mathbf{M}_k$ *is the* k*-th leading principal minor of* $\mathbf{M}$.

Proof. See e.g. [26]. □

If we use Theorem 2.5 together with Theorem 2.6, we get a set of inequalities that describe the realizability domain. These inequalities are usually termed *realizability conditions.*

Example 2.7. Let $[v_L, v_R] = [-1, 1]$. We get the following realizability conditions for

- $N = 0$: $p^{(0)} > 0$,
- $N = 1$: $p^{(0)} > p^{(1)} > -p^{(0)}$,
- $N = 2$: $p^{(0)} > 0$, $p^{(0)}p^{(2)} - (p^{(1)})^2 > 0$, $p^{(0)} > p^{(2)}$, see Figure 2.

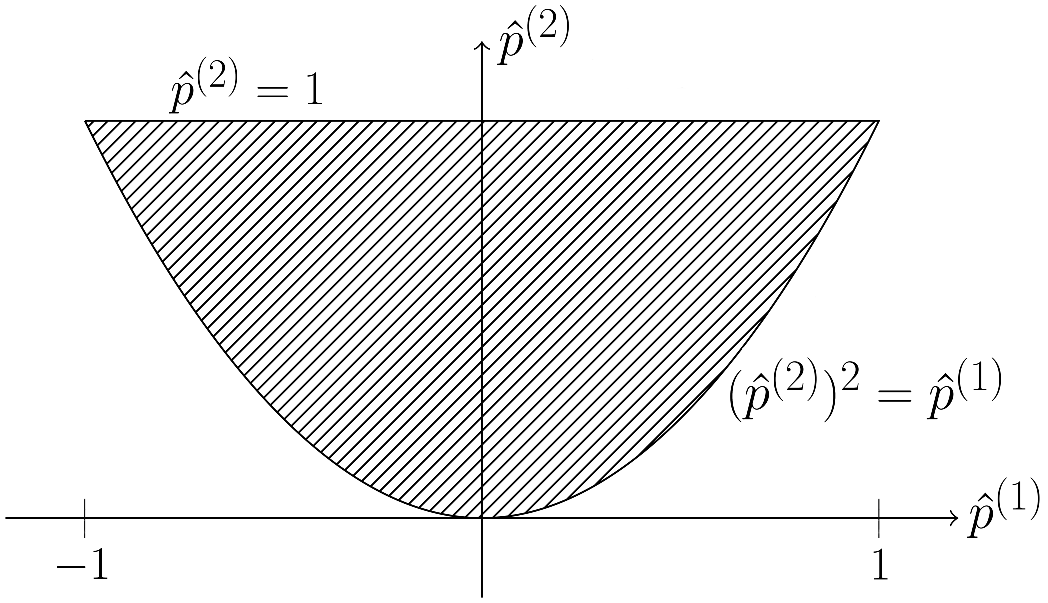


Figure 2: Realizability domain for $N = 2$. With respect to the normalized moments $\hat{p}^{(j)} = \frac{p^{(j)}}{p^{(0)}}$ the realizability domain is given by the inequalities $(\hat{p}^{(1)})^2 < \hat{p}^{(2)} < 1$. Modified from [64].

In the following, we collect some properties of the realizability domain that are important in defining appropriate moment closures (see Section 2.3).

Lemma 2.8. *$\mathcal{R}_N$ is a convex cone (i.e. $c_1\mathbf{p} + c_2\mathbf{q} \in \mathcal{R}_N$ for all $\mathbf{p}$, $\mathbf{q} \in \mathcal{R}_N$, $0 \le c_1$, $c_2 \in \mathbb{R}$).*

Proof. This follows from the fact that L^1_+ is a convex cone. If p, $q \in L^1_+(V)$ are densities that realize the moments $\mathbf{p}$ and $\mathbf{q}$, respectively, then $c_1p + c_2q \in L^1_+(V)$ realizes $c_1\mathbf{p} + c_2\mathbf{q}$. □

Lemma 2.9. *$\mathcal{R}_N \subset \mathbb{R}^{N+1}$ is open.*

Proof. See [1, 44]. □

By Lemma 2.9, we know that vectors on $\partial\mathcal{R}_N$ are not realizable by any density in $L^1_+(V)$. However, if we allow densities that consist of linear combinations of Dirac delta distributions, we can still represent these vectors as moments.

Definition 2.10. We call a density of the form

$$p(v) = \sum_{i=1}^{m} a_i \delta(v - v_i)$$

atomic. For $\mathbf{p} \in \mathcal{R}_N$, a representing atomic density is called *minimal* if m is minimal, i.e. if there is no atomic density that is a linear combination of less than m Dirac delta distributions and still represents $\mathbf{p}$.

Theorem 2.11. *a) For all $\mathbf{p} \in \mathcal{R}_N$ there exists a minimal representing atomic density.*

b) For $\mathbf{p} \in \partial\mathcal{R}_N$, there exists a unique representing atomic density.

c) There exists an efficient and robust algorithm to determine the minimal representing measure.

Proof. See [22]. □

Altogether, we now have completely characterized the realizability domain in one dimension. Generalizing this to higher dimensions is not trivial and subject to ongoing research. In fact, there seems to be no similar complete characterization of the realizability domain for dimension $d > 1$ yet. Partial results have been obtained, among others, in [3, 21, 20, 53, 54] but did not lead to explicit realizability conditions. In [64], realizability conditions for moments of order up to $N = 2$ in three space dimensions have been derived.

Despite the lack of general realizability conditions, closures that guarantee realizability can be obtained by using the fact that on the level of densities realizability essentially equals positivity. We investigate some popular closures in the next section.

2.3 Moment closures

In this section, we regard some of the most popular closure approaches. For simplicity, we assume $\mathbb{M}(V) = \mathbb{P}(V)$. We use the notation

$$\langle g \rangle := \int_V g(\mathbf{v}) \mathrm{d}\mathbf{v} \quad \text{for } g \in L^1(V). \tag{2.6}$$

We collect a basis of the space $\mathbb{P}_N(V)$ of polynomials on V of degree at most N in a vector $\mathbf{m}$ and the corresponding moments in the vector $\mathbf{p}$ such that $\langle \mathbf{m} p \rangle = \mathbf{p}$. The length of these vectors (i.e. the dimension of $\mathbb{P}_N(V)$) is $m := \binom{N+d}{d}$, where the parentheses represent the binomial coefficient. Note that the moments in $\mathbf{p}$ depend on the temporal and spatial variables $(t, \mathbf{x})$ whereas the polynomials in $\mathbf{m}$ depend solely on the velocity variable $\mathbf{v}$. We further denote the set of moments that are realizable with respect to the basis $\mathbf{m}$ as $\mathcal{R}_{\mathbf{m}}$.

The system of moment equations for the kinetic equation (1.1) truncated at order N then is

$$\partial_t \mathbf{p}(t,\mathbf{x}) + \sum_{i=1}^{d} \partial_{x_i} \langle v_i \mathbf{m} p \rangle = \langle \mathbf{m} \mathcal{L} p \rangle \tag{2.7}$$

To close the system, additional assumptions (usually on the density p) have to be made.

2.3.1 The P_N closure

We start with one of the simplest closures, the polynomial P_N *closure*. Like in Example 2.3, the P_N closure closes the truncated system of moment equations by assuming that p is polynomial of order at most N. More precisely, the P_N closure does a Galerkin semi-discretization in $\mathbf{v}$. We thus choose a basis $\mathbf{m} = (\phi_0(\mathbf{v}), \dots, \phi_{m-1}(\mathbf{v}))^T$ of the space $\mathbb{P}_N(V)$. We then make an ansatz

$$p_{P_N}(t,\mathbf{x},\mathbf{v}) = \sum_{j=0}^{m-1} p^{(j)}(t,\mathbf{x}) \phi_j(\mathbf{v}), \tag{2.8}$$

where $p^{(l)}$ is the moment of p with respect to ϕ_l. The Galerkin projection onto $\mathbb{P}_N(V)$ is done by multiplying equation (1.1) by the respective basis function and integrating over V:

$$\partial_t \langle \phi_i p \rangle + \langle \phi_i \mathbf{v} \cdot \nabla p \rangle = \langle \phi_i \mathcal{L} p \rangle .$$

Using the ansatz (2.8) gives

$$\sum_{j=0}^{m-1} \langle \phi_i \phi_j \rangle \partial_t p^{(j)} + \sum_{k=1}^{d} \sum_{j=0}^{m-1} \langle v_k \phi_i \phi_j \rangle \partial_{x_k} p^{(j)} = \left\langle \phi_i \mathcal{L} p_{P_N} \right\rangle .$$

Defining the matrices $\mathbf{M}$ with $M_{ij} = \langle \phi_i \phi_j \rangle$ and $\mathbf{D}_k$ with $(D_k)_{ij} = \langle v_k \phi_i \phi_j \rangle$ and the vector $\mathbf{r}$ with $r_i = \left\langle \phi_i \mathcal{L} p_{P_N} \right\rangle$, we get the system of equations

$$\partial_t \mathbf{p}(t,\mathbf{x}) + \sum_{k=1}^{d} \mathbf{M}^{-1} \mathbf{D}_k \partial_{x_k} \mathbf{p}(t,\mathbf{x}) = \mathbf{M}^{-1} \mathbf{r}(t,\mathbf{x}). \tag{2.9}$$

$\mathbf{M}$ is invertible because the ϕ_i are linearly independent by the definition of a basis.

The P_N closure is simple and easy to implement and still delivers a sufficient approximation of the underlying distribution in many cases. However, it suffers from the same problem we saw before: the assumed (polynomial) density can become negative and consequently the resulting moments are not always realizable. Furthermore, there can be strong artificial oscillations (see e.g. [25] for illustration) and N may have to be chosen quite large to get accurate results.

2.3.2 The M_N closure

The M_N closure (after G.N. Minerbo [63]) tries to overcome some of the limitations of the P_N closure. To close the system (2.7), the ansatz $p_{M_N}(\mathbf{p}, \mathbf{m})$ is chosen to be the minimizer of a convex optimization problem under the constraint that it should reproduce the moments:

$$p_{M_N}(\mathbf{p}(t,\mathbf{x}),\mathbf{m}(\mathbf{v})) := \underset{g\in L^1_+(V)}{\operatorname{argmin}} \langle \eta \circ g\rangle \text{ subject to } \langle \mathbf{m} g\rangle = \mathbf{p}. \tag{2.10}$$

Here, the entropy $\eta : \mathbb{R} \to \mathbb{R}$ can be any strictly convex function. In practice, it should be chosen such that it is likely to be minimized by the correct solution. In physical applications, natural choices occur from the physical concept of entropy. In the context of gas dynamics, the Maxwell-Boltzmann entropy

$$\eta(z) = z\log(z) - z \tag{2.11}$$

may be selected [57].

Inserting the ansatz (2.10) in (2.7) gives the closed system of equations

$$\partial_t \mathbf{p}(t,\mathbf{x}) + \sum_{i=1}^{d} \partial_{x_i} \mathbf{f}_i(\mathbf{p}(t,\mathbf{x})) = \mathbf{h}(t,\mathbf{x},\mathbf{p}(t,\mathbf{x})) \tag{2.12}$$

with

$$\mathbf{f}_i(\mathbf{p}) := \left\langle v_i \mathbf{m} p_{M_N}\right\rangle, \quad \mathbf{h}(t,\mathbf{x},\mathbf{p}) := \left\langle \mathbf{m}\mathcal{L} p_{M_N}\right\rangle.$$

Instead of directly solving the convex optimization problem (2.10) in $L^1_+(V)$, the optimization is usually done by regarding the dual problem on $\mathbb{R}^m$. It can be shown that if a solution to (2.10) exists, it takes the form (see [57])

$$p_{M_N}(\mathbf{p}(t,\mathbf{x}),\mathbf{m}(v)) = G_{\tilde{\boldsymbol{\alpha}}(\mathbf{p})}, \quad G_{\boldsymbol{\alpha}} := \eta'_*(\boldsymbol{\alpha}^T\mathbf{m})$$

where $\eta'_* : \mathbb{R} \to \mathbb{R}$ is the derivative of the Legendre transform of η and the Lagrange multipliers $\tilde{\boldsymbol{\alpha}} : \mathcal{R}_{\mathbf{m}} \to \mathbb{R}^m$ are given as the solution to the dual problem

$$\underset{\boldsymbol{\alpha}\in\mathbb{R}^{m+1}}{\operatorname{minimize}} \left\{\left\langle \eta'_*(\boldsymbol{\alpha}^T\mathbf{m})\right\rangle - \boldsymbol{\alpha}^T\mathbf{p}\right\}. \tag{2.13}$$

Vice versa, if a solution $\tilde{\boldsymbol{\alpha}}$ to the dual problem (2.13) exists, $G_{\tilde{\boldsymbol{\alpha}}}$ is a solution to the original problem (2.10).

If we choose the Maxwell-Boltzmann entropy (2.11), we get

$$\eta_*(y) = \eta'_*(y) = \exp(y)$$

and thus

$$G_{\boldsymbol{\alpha}}(\mathbf{v}) = \exp(\boldsymbol{\alpha}^T\mathbf{m}(\mathbf{v})).$$

Of course, the question occurs whether the M_N closure is well-defined, i.e. whether a minimizer to (2.10) (or equivalently to the dual problem (2.13)) exists for every realizable moment vector $\mathbf{p} \in \mathcal{R}_{\mathbf{m}}$. In general, this is not the case [36]. However, for bounded domains V and a polynomial basis $\mathbf{m}$ as regarded here, the constraints in (2.10) are continuous in the L^1-norm which ensures the existence of a minimizer [1, 44].

At least formally, the M_N closure guarantees realizable moments by always using a positive ansatz for the density. Whether or not the set $\mathcal{R}_{\mathbf{m}}$ is invariant under the dynamics of (2.12), i.e. whether the moments in the solution stay realizable for all times if we start with realizable moments, seems to be an open question, though. However, as we will see in Section 4.3, it is possible to enforce this invariance on the numerical level and thus ensure that the M_N closure always gives realizable moments. This is a major advantage over the P_N closure. The downside of the M_N closure, however, is its complexity. The dual optimization problem (2.13) has to be solved at every point (of a computational grid) in time and space which causes a considerable amount of computational cost. Furthermore, near to the boundary of the realizability domain the problem can become very ill-conditioned posing additional numerical challenges. The optimization problems are independent (in the spatial domain at one point in time) though and can thus be solved in parallel.

2.3.3 The K_N closure

While the M_N closure has considerable advantages over the P_N closure in some cases, its computational cost is quite high. In [45], Kershaw proposed a closure that produces realizable moments without having the high computational cost of solving the dual problem. The idea is that, on the boundary of the realizability domain, the representing density is uniquely determined and can be calculated (see Theorem 2.11). By linear interpolation between these boundary distributions, a closure can be achieved.

For this procedure, knowledge about the realizability domain $\mathcal{R}_{\mathbf{m}}$ is needed. Thus, the Kershaw closure has been mainly used for problems in one dimension where $\mathcal{R}_{\mathbf{m}}$ is well-characterized. In [64], Kershaw closures for three dimensions have been derived but only for orders $N \leq 2$.

We thus restrict ourselves to the one-dimensional case here and regard a specific example, the case $N = 1$, to illustrate the idea of the Kershaw closure. For simplicity, we further assume $V = [-1, 1]$. Although we regard the K_1 closure, we need the realizability conditions for $N = 2$ which are (see Theorem 2.5)

$$-1 < \hat{p}^{(1)} < 1, \quad (\hat{p}^{(1)})^2 < \hat{p}^{(2)} < 1,$$

where

$$\hat{p}^{(j)} = \frac{p^{(j)}}{p^{(0)}}$$

are the normalized moments. On the boundary of the realizability domain, i.e. for $(\hat{p}^{(1)})^2 = \hat{p}^{(2)}$ or $\hat{p}^{(2)} = 1$, we can determine the atomic distributions $p_{\text{low}}(\hat{p}^{(1)})$ and $p_{\text{up}}(\hat{p}^{(1)})$. By linearity of calculating moments, every convex combination

$$p_{K_N} = a p_{\text{up}} + (1 - a) p_{\text{low}}$$

reproduces the normalized moment $\hat{p}^{(1)}$. If we scale the boundary distributions properly by

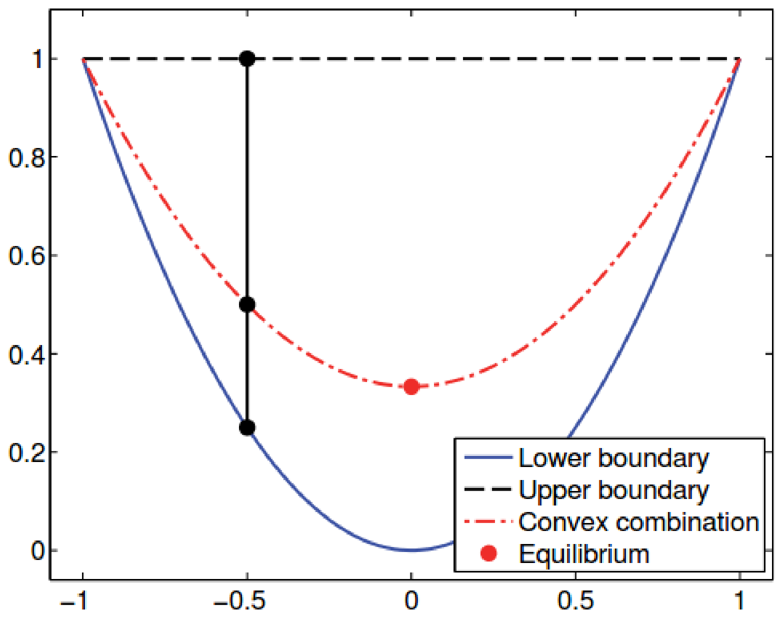


Figure 3: Illustration of the K_1 closure. The normalized moments $\hat{p}^{(1)}$ and $\hat{p}^{(2)}$ are plotted on the x-axis and y-axis, respectively. To close the moment equations, a distribution that reproduces the first normalized moment $\hat{p}^{(1)}$ is calculated as a convex combination of upper and lower boundary distributions. A specific convex combination is chosen by demanding that the moments of the constant (equilibrium) distribution are reproduced exactly. From [72].

multiplying with a constant factor, the moments $p^{(0)}$, $p^{(1)}$ will be reproduced by every such convex combination.

To choose a specific convex combination, we need an additional condition. For that purpose, we demand that the closure is exact for the moments up to order 2 of the constant distribution

$$p_{\text{const}}(t,x,v) = \frac{p^{(0)}(t,x)}{|V|} = \frac{p^{(0)}(t,x)}{2}.$$

Here and in the remainder of the thesis $|V|$ denotes the d-dimensional volume of V. Calculating the normalized moments of p_{const} results in

$$(\hat{p}^{(1)}_{\text{const}}, \hat{p}^{(2)}_{\text{const}}) = (0, \frac{1}{3}) \overset{!}{=} (\hat{p}^{(1)}_{K_N}(\hat{p}^{(1)}_{\text{const}}), \hat{p}^{(2)}_{K_N}(\hat{p}^{(1)}_{\text{const}})).$$

Thus, our condition on a is

$$\frac{1}{3} \overset{!}{=} \hat{p}^{(2)}_{K_N}(\hat{p}^{(1)}_{\text{const}}) = a\hat{p}^{(2)}_{\text{up}}(\hat{p}^{(1)}_{\text{const}}) + (1-a)\hat{p}^{(2)}_{\text{low}}(\hat{p}^{(1)}_{\text{const}}) = a + (1-a)(\hat{p}^{(1)}_{\text{const}})^2 = a.$$

Thus, we get $p_{K_N} = \frac{1}{3}p_{\text{up}} + \frac{2}{3}p_{\text{low}}$. Calculating the second moment gives

$$\hat{p}^{(2)}(\hat{p}^{(1)}) = \frac{1}{3}(2(\hat{p}^{(1)})^2 + 1),$$

closing the moment equations. An illustration of the closure can be seen in Figure 3.

2.3.4 Half and mixed moment closures

M_N and K_N closures may show unphysical shocks. This is due to a "zero netflux problem": when integrating over the velocity component, velocities that point in opposite directions annihilate each other such that the resulting flux becomes zero. To overcome these problems, in one dimension half moment models can be used, where instead of integrating over the whole interval $V = [-1, 1]$, two separate integrations over the half intervals $[-1, 0]$ and $[0, 1]$ are done to build half moments:

$$\mathbf{p}_+ := \int_0^1 \mathbf{m} p \mathrm{d}v =: \langle \mathbf{m} p \rangle_+$$
$$\mathbf{p}_- := \int_{-1}^0 \mathbf{m} p \mathrm{d}v =: \langle \mathbf{m} p \rangle_-$$

Accordingly, the kinetic equation (1.1) is integrated over the half intervals instead of the full interval after multiplying by $\mathbf{m}$:

$$\partial_t \mathbf{p}_+ = -\partial_x \langle v \mathbf{m} p \rangle_+ + \langle \mathbf{m} \mathcal{L} p \rangle_+ \tag{2.14a}$$
$$\partial_t \mathbf{p}_- = -\partial_x \langle v \mathbf{m} p \rangle_- + \langle \mathbf{m} \mathcal{L} p \rangle_- \tag{2.14b}$$

The system of equations (2.14) can be closed similar to the full moments with any of the closures outlined above where only slight adaptions are needed (see [72, 84]).

An advancement of the half moment models are the mixed moment models where full moments are used for the lower order moments and half moments for the higher order ones. Here, the realizability theory of the full moments does not apply anymore and an adapted theory is needed. Once we have the realizability conditions for mixed moments, the closures outlined above (P_N, M_N, K_N) can be done in a similar way (see [72]).

3 Hyperbolic systems of first-order equations

After transforming a given kinetic equation to a system of moment equations and applying a suitable moment closure, we obtain a system of partial differential equations. In order to solve these systems, we investigate their properties in the following. This chapter is largely based on the descriptions in [48, 56].

Definition 3.1. Let $\mathbf{f}^1, \dots, \mathbf{f}^d \in \mathcal{C}^1(\mathbb{R}^m, \mathbb{R}^m)$ and $\mathbf{h} \in \mathcal{C}^0(\mathbb{R}_+ \times \mathbb{R}^d \times \mathbb{R}^m, \mathbb{R}^m)$. A system of first-order equations of the form

$$\partial_t \mathbf{p}(t, \mathbf{x}) + \sum_{i=1}^{d} \partial_{x_i} \mathbf{f}^i(\mathbf{p}(t, \mathbf{x})) = \mathbf{h}(t, \mathbf{x}, \mathbf{p}(t, \mathbf{x})) \tag{3.1}$$

is called a *system of balance laws* with flux functions $\mathbf{f}^1, \dots, \mathbf{f}^d$ and source term $\mathbf{h}$.

Defining $\mathbf{F} := (\mathbf{f}^1, \dots, \mathbf{f}^d)^T$, the system (3.1) can be rewritten as

$$\partial_t \mathbf{p} + \nabla \cdot \mathbf{F}(\mathbf{p}) = \mathbf{h}(t, \mathbf{x}, \mathbf{p}), \tag{3.2}$$

where $\nabla \cdot \mathbf{F} = \sum\limits_{i=1}^{d} \partial_{x_i} \mathbf{f}^i(\mathbf{p})$ is the divergence of $\mathbf{F}$.

These equations are called “balance laws” because they record the changes in $\mathbf{p}$ in a given part of the spatial domain. To illustrate this, let $K \subset \mathbb{R}^d$ with non-empty interior be regular enough that the divergence theorem is valid. If we integrate (3.2) over K and apply the divergence theorem, we get

$$\partial_t \int_K \mathbf{p} = - \int_{\partial K} \mathbf{F} \cdot \mathbf{n} + \int_K \mathbf{h}, \tag{3.3}$$

where $\mathbf{n} = (n_1, \dots, n_d)^T$ is the outer unit normal to ∂K and $\mathbf{F} \cdot \mathbf{n} := \sum\limits_{k=1}^{d} \mathbf{f}^k(\mathbf{p}) n_k$. If we think of $\mathbf{p}$ as a density of particles, the integral on the left-hand side is just the number of particles in the volume K. Assume $\mathbf{h} = \mathbf{0}$ for now. Then the second term on the right-hand side vanishes and (3.3) states that the number of particles is only changed by the flow of particles over the boundary of the domain. No particles are created or destroyed inside the domain. The number of particles is thus a conserved quantity. For this reason, balance laws with $\mathbf{h} = \mathbf{0}$ are also called *conservation laws*. Such conservation laws arise naturally in physics. An example are the Euler equations of gas dynamics, where the physical conservation laws for mass, momentum and energy lead to a conservation law for $\mathbf{p} = (\rho, \rho v, E)$ consisting of the particle density ρ, the mass flux ρv and the total energy E (see [56] for details and Section 6.1.2 for a numerical example). If $\mathbf{h} \neq \mathbf{0}$, the second term on the right-hand side of (3.3) accounts for particles that are created or destroyed within the domain. The number of particles is no longer conserved but we know by (3.3) how it changes.

The balance laws we regard have an additional important property:

Definition 3.2. The system of balance laws (3.1) is called *hyperbolic* in U if the symmetric matrix

$$\mathbf{A}(\mathbf{p};\boldsymbol{\xi}) := \sum_{i=1}^{d} D\mathbf{f}^i(\mathbf{p})\xi_i$$

is diagonalizable with real eigenvalues and a full set of right eigenvectors for all $\mathbf{p} \in U$, $\boldsymbol{\xi} = (\xi_1, \dots, \xi_d)^T \in \mathbb{R}^d$. The system is called *strictly hyperbolic* if $\mathbf{A}$ has m distinct eigenvalues $\lambda_1(\mathbf{p};\boldsymbol{\xi}) < \ldots < \lambda_m(\mathbf{p};\boldsymbol{\xi})$ for all $\mathbf{p}$, $\boldsymbol{\xi}$.

Note that the balance laws coming from the moment models for kinetic equations are not necessarily hyperbolic for arbitrary closures. This problem became already apparent in the first widely known moment model for the Boltzmann equation, Grad's 13-moment system [31], which is only hyperbolic next to the thermodynamic equilibrium [15]. However, most of the closures introduced in Section 2.3 always lead to hyperbolic equations. This can most easily be seen for the P_N equations. All the matrices $\mathbf{D}_k$ (see (2.9)) are symmetric, so any linear combination of these matrices is also symmetric and thus diagonalizable. A little more work has to be done for the minimum entropy closure but still minimum entropy moment equations are strictly hyperbolic [57] (under some restrictions on the space $\mathbb{M}(V)$ in Definition 2.2). For Kershaw closures, there is no general result on hyperbolicity [72], but the lower order Kershaw moment models used in [64, 72] were shown to be hyperbolic, so we can hope that this is also true for higher order Kershaw closures.

3.1 Classical solutions

For the further investigation of hyperbolic systems of equations, we drop the source term and only regard hyperbolic conservation laws (i.e. $\mathbf{h} = \mathbf{0}$). The source term can be handled separately in numerical schemes (see Section 4.1.3). We further assume that $d = 1$ as the one-dimensional case is best understood and the obtained numerical methods can relatively easy be generalized to higher dimensions. Finally, we assume that the flux $\mathbf{f}$ is convex. In addition to the hyperbolic conservation law, we need an initial condition. Thus, the problem we regard in the following is

$$\partial_t \mathbf{p}(t,x) + \partial_x \mathbf{f}(\mathbf{p}(t,x)) = \mathbf{0} \quad \forall\, t \in \mathbb{R}_+,\ x \in \mathbb{R}, \tag{3.4a}$$

$$\mathbf{p}(0,x) = \mathbf{p}_0(x) \quad \forall\, x \in \mathbb{R}, \tag{3.4b}$$

where we assume that $\mathbf{p}_0$ has bounded support in $\mathbb{R}$ and bounded total variation (see Section 4.1.5 for the definition of total variation). If $\mathbf{p}$ fulfills (3.4), we say $\mathbf{p}$ is a *classical solution* for the initial value problem.

Classical solutions can be constructed by the method of characteristics. We will do this here for the scalar case. A characteristic curve $X(t)$ for the hyperbolic conservation law is a curve in the (t,x)-plane fulfilling

$$X'(t) = f'(p(t,X(t))). \tag{3.5}$$

Along characteristic curves, the solution $p(t,x)$ of the conservation law is constant:

$$\begin{aligned} \mathrm{d}_t\, p(t,X(t)) &= (\partial_x p)(t,X(t))X'(t) + (\partial_t p)(t,X(t)) \\ &\overset{(3.5)}{=} (\partial_x p)(t,X(t))f'(p(t,X(t))) + (\partial_t p)(t,X(t)) \overset{(3.4)}{=} 0. \end{aligned}$$

Here, d_t denotes differentiation with respect to t. As p is constant along the characteristic curve, $X'(t) = \text{const}$ by (3.5), so the characteristic curves are straight lines in the (t,x)-plane. Thus, a classical solution to (3.4) can always be found by calculating the characteristic curves and tracing them back to the initial condition at $t = 0$. However, the characteristics may intersect after a (possibly very short) finite time such that the solution may not be well-defined any more. An important exception is the case that f is linear, i.e. if we regard the linear advection equation

$$\partial_t p(t,x) + \lambda p(t,x) = 0 \tag{3.6}$$

with $\lambda \in \mathbb{R}$. Here, $f' = \lambda$ is constant and thus all characteristic curves are parallel and never intersect. The advection equation thus has a global classical solution $p(t,x) = p_0(x - \lambda t)$.

We can use this knowledge about the scalar advection equation to get solutions for linear hyperbolic systems of equations

$$\partial_t \mathbf{p} + \mathbf{A}\partial_x \mathbf{p} = \mathbf{0} \tag{3.7}$$

with a matrix $\mathbf{A} \in \mathbb{R}^{m\times m}$. Due to the hyperbolicity condition (3.2), $\mathbf{A}$ is diagonalizable, so we can write

$$\mathbf{A} = \mathbf{R}\boldsymbol{\Lambda}\mathbf{R}^{-1}, \tag{3.8}$$

where $\mathbf{R}$ is the matrix of right eigenvectors and $\boldsymbol{\Lambda}$ is the diagonal matrix containing the eigenvalues $\lambda_1, \dots, \lambda_m$. We can now introduce new variables

$$\mathbf{w} = \mathbf{R}^{-1}\mathbf{p} \tag{3.9}$$

to reduce the system (3.7) to a system of decoupled advection equations

$$\partial_t \mathbf{w} + \boldsymbol{\Lambda}\partial_x \mathbf{w} = \mathbf{0}. \tag{3.10}$$

The s-th equation in (3.10) is the advection equation

$$\partial_t w_s + \lambda_s \partial_x w_s = 0 \tag{3.11}$$

with solution $w_s(t,x) = (w_0)_s(x - \lambda_s t)$, where the initial values are given as $\mathbf{w}_0 = R^{-1}\mathbf{p}_0$. Once we have the solution w, we can express it in terms of our original variables by (3.9).

3.2 Weak solutions and vanishing viscosity

We have seen that we can find global classical solutions if the flux is linear, but for general fluxes there is no solution to (3.4) for arbitrary times. In addition, the physical relevant solution to a conservation law may contain discontinuities. For example, if the initial condition for the linear advection equation is discontinuous, the solution $p_0(x-\lambda t)$ is still unambiguously defined but cannot be a classical solution as it is discontinuous. For various physical problems that are modelled by hyperbolic conservation laws, discontinuous solutions may arise even from smooth initial data [49]. For that reason, we relax the conditions on $\mathbf{p}$ and allow for solutions that fulfil (3.1) in the distributional sense.

Definition 3.3. Let $L^1_{\text{loc}}(\mathbb{R}^d, \mathbb{R}^m)$ be the space of locally integrable functions $\mathbf{u} : \mathbb{R}^d \to \mathbb{R}^m$ and let $C_0^\infty(\mathbb{R}^d, \mathbb{R}^m)$ be the space of smooth functions with compact support. Let $\mathbf{p}_0 \in L^1_{\text{loc}}(\mathbb{R}^d, \mathbb{R}^m)$. Then $\mathbf{p} \in L^1_{\text{loc}}(\mathbb{R}^d \times \mathbb{R}_+, \mathbb{R}^m)$ is called a *weak solution* of (3.1) together with an initial condition $\mathbf{p}(0, \mathbf{x}) = \mathbf{p}_0(\mathbf{x})$ if

$$\int\limits_{\mathbb{R}^d\times\mathbb{R}_+} (\mathbf{p}\partial_t\boldsymbol{\varphi} + \sum_{j=0}^{d} \mathbf{f}^j(\mathbf{p})\partial_{x_j}\boldsymbol{\varphi} + \mathbf{h}(\mathbf{p})\boldsymbol{\varphi}) = -\int\limits_{\mathbb{R}^d} \mathbf{p}_0\boldsymbol{\varphi}(\cdot, 0)$$

for all $\boldsymbol{\varphi} \in C_0^\infty(\mathbb{R}^d \times \mathbb{R}_+, \mathbb{R}^m)$.

Note that classical solutions are weak solutions, and a weak solution that is smooth enough is a classical solution.

An obvious choice for weak solutions are piecewise classical solutions with a separating discontinuity. It turns out that we cannot simply combine any two classical solutions to get a weak solution. Instead, a jump condition has to be fulfilled along the discontinuity.

Theorem 3.4 (Rankine-Hugoniot). *Let* $\sigma(t) \in C^1(\mathbb{R}_+)$, $\Omega_l = \{(t,x) \in \mathbb{R}_+ \times \mathbb{R} \,|\, x < \sigma(t)\}$, $\Omega_r = \{(t,x) \in \mathbb{R}_+ \times \mathbb{R} \,|\, x > \sigma(t)\}$, $S := \{(t, \sigma(t))\}$ *and*

$$\mathbf{p}(x,t) := \begin{cases} \mathbf{p}_l(x,t) & \textit{if } (x,t) \in \Omega_l, \\ \mathbf{p}_r(x,t) & \textit{if } (x,t) \in \Omega_r, \end{cases}$$

where $\mathbf{p}_l \in C^1(\Omega_l)$ *and* $\mathbf{p}_r \in C^1(\Omega_r)$. *Then* $\mathbf{p}$ *is a weak solution to* (3.4) *if and only if* $\mathbf{p}_l$, $\mathbf{p}_r$ *are classical solutions to* (3.4) *in* Ω_l, Ω_r, *respectively, and*

$$(\mathbf{p}_l(\sigma(t),t) - \mathbf{p}_r(\sigma(t),t))\,\sigma'(t) = \mathbf{f}(\mathbf{p}_l(\sigma(t),t)) - \mathbf{f}(\mathbf{p}_r(\sigma(t),t)) \quad \forall\, t > 0. \tag{3.12}$$

Proof. See e.g. [48]. □

For scalar conservation laws in one dimension, we can calculate the speed of the propagating discontinuity by (3.12) as

$$\sigma'(t) = \frac{f(p_l(t,\sigma(t))) - f(p_r(t,\sigma(t)))}{p_l(t,\sigma(t)) - p_r(t,\sigma(t))}.$$

For linear systems of conservation laws (3.7), we see that $\mathbf{p}_l - \mathbf{p}_r$ always has to be an eigenvector of the flux matrix $\mathbf{A}$ at the discontinuity.

Weak solutions overcome the problem that smooth solutions may not be well-defined after arbitrarily short times but they introduce a new problem: They may not be unique.

Example 3.5 ([55])**.** The two functions

$$p(t,x) = \begin{cases} 0 & \text{for } 2x < t \\ 1 & \text{for } 2x > t \end{cases} \quad \text{and} \quad p(t,x) = \begin{cases} 0 & \text{for } x < 0 \\ x/t & \text{for } 0 < x < t \\ 1 & \text{for } t < x \end{cases}$$

are both weak solutions to the initial value problem for Burgers' equation

$$\partial_t p + \partial_x \frac{p^2}{2} = 0, \qquad p(0,x) = \begin{cases} 0 & \text{if } x < 0, \\ 1 & \text{if } x > 0. \end{cases}$$

In physical problems, however, we expect that there is a unique solution, namely the one which occurs in nature. Thus, an additional criterion is needed to decide which solution is the "right" one. A criterion that may be used is the *vanishing viscosity* approach. This approach is motivated by the fact that gas dynamics may be modelled either with or without viscosity. For the viscous flow, an additional $\epsilon\partial^2_{xx}\mathbf{p}$ term occurs instead of $\mathbf{0}$ on the right-hand side of the conservation law, where ϵ is the viscosity coefficient. This additional viscosity ensures the uniqueness of the solution $\mathbf{p}_\epsilon$. For $\epsilon \to 0$, we would expect that $\mathbf{p}_\epsilon \to \mathbf{p}$, where $\mathbf{p}$ is a solution of the inviscous flow model. Transferring this approach to the hyperbolic balance law leads to the following theorem for the scalar case.

Theorem 3.6 ([48])**.** *Let $p_0 \in L^1(\mathbb{R}^d) \cap L^\infty(\mathbb{R}^d)$. Let Δ denote the Laplace operator and let f^i, h be such that all (partial) derivatives up to second order exist and are bounded. Then, for any $\epsilon > 0$, there exists a uniquely defined classical solution p_ϵ of*

$$\partial_t p + \sum_{i=1}^{d} \partial_{x_i} f^i(p) = h(t,x,p) + \epsilon\Delta p \quad \textit{in } \mathbb{R}^d \times \mathbb{R}_+, \tag{3.13a}$$

$$p(0,x) = p_0(x) \quad \textit{in } \mathbb{R}^d \tag{3.13b}$$

such that p_ϵ converges almost everywhere in $\mathbb{R}^d \times \mathbb{R}_+$ as $\epsilon \to 0$ to a weak solution p of

$$\partial_t p + \sum_{i=1}^{d} \partial_{x_i} f^i(p) = h(t,x,p) \quad \textit{in } \mathbb{R}^d \times \mathbb{R}_+,$$
$$p(0,x) = p_0(x) \quad \textit{in } \mathbb{R}^d.$$

For *systems* of conservation laws, even in one dimension, the above theorem does not hold. For general data, there may even be no weak solution to (3.4) at all (see [48, Section 4.1]). However, in [7] the authors showed that a unique vanishing viscosity solution to (3.4) exists as long as the total variation (see Section 4.1.5) of the initial values is sufficiently small.

Usually we want to choose a unique weak solution as the vanishing viscosity limit of the viscous problem (3.13). However, this condition is hard to work with in practice. Therefore, a variety of conditions has been developed that can be directly applied to weak solutions of the conservation law. These conditions are called *admissibility conditions* or *entropy conditions.* The underlying idea is to make use of an entropy function $\eta(\mathbf{p})$. To choose a weak solution of the conservation law, we require that this entropy, similar to the entropy in physics, should only be allowed to either decrease or increase. While the physical entropy is defined to be non-decreasing, the mathematical entropy functions are usually chosen to be non-increasing.

More precisely, we need an entropy function $\eta(\mathbf{p})$ together with an entropy flux $\psi(\mathbf{p})$ such that an integral conservation law

$$\int_{x_1}^{x_2} \eta(\mathbf{p}(t_2, x))\mathrm{d}x = \int_{x_1}^{x_2} \eta(\mathbf{p}(t_1, x))\mathrm{d}x + \int_{t_1}^{t_2} \psi(\mathbf{p}(t, x_1))\mathrm{d}t - \int_{t_1}^{t_2} \psi(\mathbf{p}(t, x_2))\mathrm{d}t \tag{3.14}$$

is fulfilled whenever $\mathbf{p}$ is smooth and that is not fulfilled at discontinuities. The condition that the entropy must be non-increasing over time is then given by (3.14) where we replace the "=" by "$\leq$".

If $\mathbf{p}$ is smooth enough, we can derive the differential form of (3.14)

$$\partial_t \eta(\mathbf{p}) + \partial_x \psi(\mathbf{p}) = 0. \tag{3.15}$$

If η, ψ are smooth enough we can rewrite (3.15) as

$$\mathbf{D}\eta(\mathbf{p})\partial_t \mathbf{p} + \mathbf{D}\psi(\mathbf{p})\partial_x \mathbf{p} = 0. \tag{3.16}$$

where $\mathbf{D}\eta(\mathbf{p}), \mathbf{D}\psi(\mathbf{p}) \in \mathbb{R}^{1\times m}$ are the Jacobians of η, ψ, respectively. Applying $\mathbf{D}\eta$ to the conservation law (3.4a) gives

$$\mathbf{D}\eta(\mathbf{p})\partial_t \mathbf{p} + \mathbf{D}\eta(\mathbf{p})\mathbf{D}\mathbf{f}(\mathbf{p})\partial_x \mathbf{p} = 0. \tag{3.17}$$

Comparing (3.16) and (3.17) we see that the entropy flux should satisfy

$$\mathbf{D}\psi(\mathbf{p}) = \mathbf{D}\eta(\mathbf{p})\mathbf{D}\mathbf{f}(\mathbf{p}). \tag{3.18}$$

To see whether we can single out the vanishing viscosity solution by the entropy approach, we regard the viscous equation corresponding to the conservation law (3.4a)

$$\partial_t \mathbf{p}_\epsilon + \partial_x \mathbf{f}(\mathbf{p}_\epsilon) = \epsilon \partial_{xx}^2 \mathbf{p}_\epsilon. \tag{3.19}$$

Multiplying by $\mathbf{D}\eta(\mathbf{p}_\epsilon)$ from the left and using (3.18) and the fact that solutions to the parabolic viscous equation 3.19 are always smooth gives

$$\partial_t \eta(\mathbf{p}_\epsilon) + \partial_x \psi(\mathbf{p}_\epsilon) = \epsilon \mathbf{D}\eta(\mathbf{p}_\epsilon)\partial_{xx}^2 \mathbf{p}_\epsilon. \tag{3.20}$$

We rewrite the right-hand side to get

$$\partial_t \eta(\mathbf{p}_\epsilon) + \partial_x \psi(\mathbf{p}_\epsilon) = \epsilon \partial_x (\mathbf{D}\eta(\mathbf{p}_\epsilon)\partial_x \mathbf{p}_\epsilon) - \epsilon(\partial_x \mathbf{p}_\epsilon)^T \mathbf{H}(\eta)(\mathbf{p}_\epsilon)\partial_x \mathbf{p}_\epsilon \tag{3.21}$$

where $\mathbf{H}(\eta)$ is the Hessian matrix of η. Integrating (3.21) over the rectangle $[x_1, x_2] \times [t_1, t_2]$

results in

$$\begin{aligned}\int_{x_1}^{x_2}\eta(\mathbf{p}(t_2,x))\mathrm{d}x &= \int_{x_1}^{x_2}\eta(\mathbf{p}(t_1,x))\mathrm{d}x + \int_{t_1}^{t_2}\psi(\mathbf{p}(t,x_1))\mathrm{d}t - \int_{t_1}^{t_2}\psi(\mathbf{p}(t,x_2))\mathrm{d}t\\ &+ \epsilon\int_{t_1}^{t_2}\Big(\mathbf{D}\eta(\mathbf{p}_\epsilon(t,x_2))\partial_x\mathbf{p}_\epsilon(t,x_2) - \mathbf{D}\eta(\mathbf{p}_\epsilon(t,x_1))\partial_x\mathbf{p}_\epsilon(t,x_1)\Big)\mathrm{d}t\\ &- \epsilon\int_{t_1}^{t_2}\int_{x_1}^{x_2}(\partial_x\mathbf{p}_\epsilon)^T\mathbf{H}(\eta)(\mathbf{p}_\epsilon)\partial_x\mathbf{p}_\epsilon\mathrm{d}x\mathrm{d}t.\end{aligned}$$

This looks exactly like the entropy conservation law (3.14) except for the two additional terms on the right-hand side. The first of these terms vanishes as $\epsilon \to 0$. This is clearly true if the limit function $\mathbf{p}$ is smooth at x_1 and x_2 but can be proven more generally, see [56]. The second term may not vanish in general. However, if we demand that η is convex, then $\mathbf{H}(\eta)$ is positive definite and the second term will always be positive. We thus see that the vanishing viscosity solution satisfies the entropy condition ((3.14) with "$\leq$" instead of "$=$") for convex entropies η. Instead of working with this integral entropy condition, it is often easier to use the weak formulation

$$\int_{\mathbb{R}_+}\int_{\mathbb{R}}\left(\eta(\mathbf{p})\partial_t\phi + \psi(\mathbf{p})\partial_x\phi\right)\mathrm{d}x\mathrm{d}t + \int_{\mathbb{R}}\phi(0,x)\eta(\mathbf{p}(0,x))\mathrm{d}x \geq 0 \tag{3.22}$$

for all test functions $\phi \in \mathcal{C}_0^1(\mathbb{R}\times\mathbb{R})$ now required to be positive ($\phi(t,x) > 0$ for all x, t).

We summarize our results in the following definition.

Definition 3.7. Let $\eta, \psi : \mathbb{R}^m \to \mathbb{R}$. The pair (η, ψ) is called an *entropy pair* for (3.4) if η is convex (i.e. $\mathbf{H}(\eta)$ is positive definite) and

$$\mathbf{D}\psi(\mathbf{p}) = \mathbf{D}\eta(\mathbf{p})\mathbf{D}\mathbf{f}(\mathbf{p}).$$

Let $\mathbf{p}$ be a weak solution of the hyperbolic system (3.4). We say that $\mathbf{p}$ is an *entropy solution* if there exists an entropy pair (η, ψ) such that (3.22) holds.

It can be shown that entropy weak solutions are indeed unique and can be considered a vanishing viscosity limit (see [48] for the scalar case in one or several dimensions and [7] for systems of equations in one dimension). For systems of equations in several dimensions, there is no general existence result for weak solutions.

For scalar conservation laws in one dimension, a simpler condition can be derived to decide whether a weak solution containing discontinuities is the entropy weak solution. An intuitive requirement is that information, i.e. the characteristics, should flow into the discontinuity, not come out of it. It turns out that this is enough to decide whether the discontinuity is admissible in the entropy solution as long as the flux is strictly convex or concave.

Definition 3.8. For a strictly convex (or strictly concave) scalar conservation law, a discontinuity propagating with speed s satisfies the *Lax entropy condition* if

$$f'(p_l) > s > f'(p_r) \tag{3.23}$$

where $p_l(t), p_r(t)$ are the (limit) values of p at the discontinuity from the left and right, respectively. A discontinuity that satisfies the jump condition (3.4) and the Lax entropy condition is called a *shock*.

Note that f' is the characteristic speed. Thus, the Lax entropy condition ensures that the characteristics impinge on the shock.

3.3 The Riemann problem

We now analyse solutions to a special class of initial value problems for the conservation law (3.4) where the initial values are piecewise constant with a single discontinuity at $x = 0$, i.e.

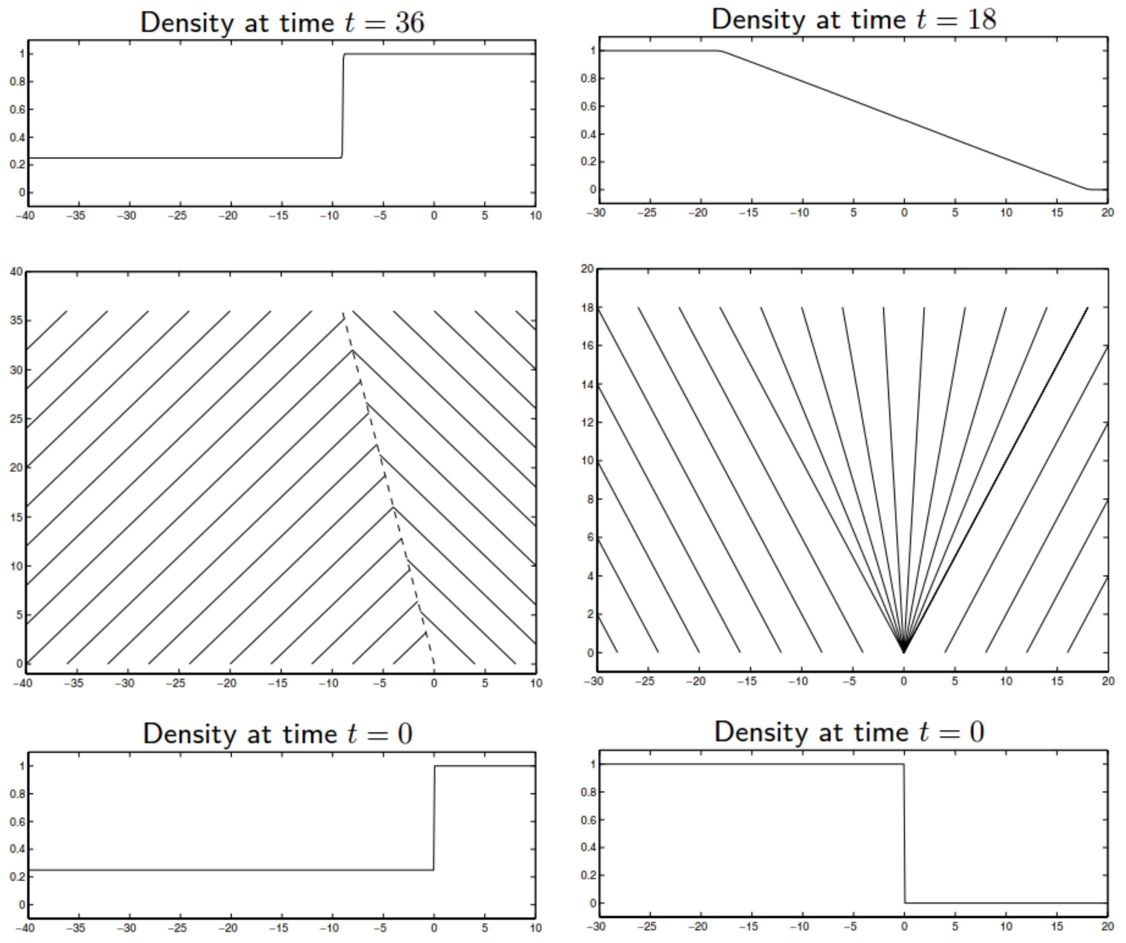
$$\mathbf{p}_0(x) = \begin{cases} \mathbf{p}_l \text{ if } x < 0 \\ \mathbf{p}_r \text{ if } x > 0, \end{cases} \qquad \mathbf{p}_l,\, \mathbf{p}_r \in \mathbb{R}^m. \tag{3.24}$$

These special problems are called Riemann problems and are an important building block for the numerical schemes for hyperbolic conservation laws, as we will see in Section 4.1. We restrict ourselves to the scalar case here. Further, we assume that the flux f is either convex or concave. If we look at the quasilinear form of the scalar conservation law

$$\partial_t p + f'(p)\partial_x p = 0$$

we see that for constant p this is an advection equation with advection speed $f'(p)$. Thus, distant from the discontinuity at $x = 0$ the initial values in the Riemann problem will simply be transported at speed $f'(p_l)$ or $f'(p_r)$. Around $x = 0$, there are essentially two possibilities, depending on the data of the Riemann problem. In Figure 4, this is depicted for a traffic flow model with concave flux $f(p) = p(1 - p)$. In Figure 4a, the initial values are chosen such that $f'(p_l) > 0 > f'(p_r)$. The characteristics meet and a shock with speed given by (3.12) forms which satisfies the Lax entropy condition (3.23). As the speed is negative, the shock moves to the left. In Figure 4b, the initial values imply $f'(p_l) < 0 < f'(p_r)$ and the Lax entropy condition can no longer be satisfied. Instead of moving to the right or left, the discontinuity is spreading out. This is most easily understood if we think of the discontinuity as slightly smoothed out such that p takes all values between 0 and 1 in a very small neighbourhood of $x = 0$. Consequently, $f'(p)$ takes values between -1 and 1 and the density is spreading out in both directions. This is called a *centred rarefaction wave.* For $0 < f'(p_l) < f'(p_r)$ (not depicted in the figure), the discontinuity would also spread out but would be moving to the right which is called a right-going rarefaction wave. Of course, also right-going shocks or left-going rarefaction waves are possible.

It is important to note that the value of the solution at the point $x = 0$ is constant over the time interval $(0, \infty)$. This will be exploited in the numerical schemes in Section 4.1. For left-going shocks or rarefaction waves, the value is p_r, for right-going shocks or rarefaction waves it is p_l. In the case of a centred rarefaction wave, the value is neither p_l nor p_r but the unique (as f is strictly convex or concave) value p_s such that $f'(p_s) = 0$. In the case of a stationary shock with speed 0, the value is not well-defined. Note however that, by the Rankine-Hugoniot condition (3.4), $f(p_l) = f(p_r)$ for a stationary shock. Hence, the value of the flux can still be defined unambiguously which is all that is needed for the numerical scheme.



(a) left-going shock **(b)** centered rarefaction wave

Figure 4: Solutions and characteristics of Riemann problems for a traffic flow model with concave flux $f(p) = p(1-p)$**.** The initial values and the solution at a later time are plotted at the bottom and at the top, respectively. In the middle, the characteristics are depicted. **(a)** Here, $f'(p_l) > 0 > f'(p_r)$. The characteristics meet which leads to a shock that moves with speed given by (3.12) (drawn as a dashed line in the characteristics plot). **(b)** Here, $f'(p_l) < 0 < f'(p_r)$. The characteristics are spreading out (for the characteristics plot it is assumed that the discontinuity in the initial values is slightly smoothed out such that p takes all values between 0 and 1 in a very small neighborhood of $x = 0$). As a result, a rarefaction wave forms. Modified from [56, Section 11.1].

For systems of equations, the situation is more complicated, as there are in general m families of characteristics, resulting in m shocks or rarefaction waves. As long as the system is strictly hyperbolic and the genuine nonlinearity condition, replacing the convexity requirement to the flux in the scalar case, is fulfilled, the solution to the Riemann problem consists of $m+1$ constant states connected by rarefaction waves or shocks. Analogous to the scalar case, conditions for when a shock or rarefaction wave is admissible and which constant states can be connected to each other by either a shock or a rarefaction wave can be derived. Applying these conditions, the Riemann problem can in principle be solved. As in the scalar case, the value of the solution at $x = 0$ is constant over the time $(0, \infty)$. This constant value is one of the $m + 1$ constant states, except if one of the rarefaction waves is centred.

In the case of a linear system of equations, the value at $x = 0$ can be determined particularly easily. In that case, we can solve the system by reduction to a system of decoupled advection equations, where the velocity in each advection equation is given by one of the eigenvalues λ_s of the flux matrix (see Section 3.1). Note that we can write the solution by (3.9) as

$$\mathbf{p}(t,x) = \sum_{s=1}^{m} w_s(t,x)\mathbf{r}_s \tag{3.25}$$

where $\mathbf{r}_s$ is the eigenvector of the flux matrix to the eigenvalue λ_s and w_s is the solution of the s-th advection equation. If we have Riemann initial values for p, we obtain a Riemann problem for each of the decoupled advection equations with left initial value $(w_s)_l$ and right initial value $(w_s)_r$. The advection equations shift the initial values with speed λ_s. Thus, by (3.25), for $t > 0$ we get

$$\mathbf{p}(t,0) = \sum_{s=1}^{m} w_s(t,0)\mathbf{r}_s = \sum_{s\colon \lambda_s<0} (w_s)_r\mathbf{r}_s + \sum_{s\colon \lambda_s>0} (w_s)_l\mathbf{r}_s. \tag{3.26}$$

If one of the eigenvalues is zero, the value at $x = 0$ is not well-defined. However, as in the scalar case, the value of the flux is still unambiguously defined.

4 Numerical methods

4.1 Finite volume methods for hyperbolic balance laws

In the last section, we have seen that solutions to hyperbolic systems of equations may contain discontinuities. This poses difficulties for classical finite difference methods as the differential equation does not hold at the discontinuities. Finite volume methods overcome this problem by being based on the integral form of the conservation law that is also valid for discontinuous solutions. Furthermore, the conservation property (3.3) can be preserved in the discrete solution using finite volume methods.

4.1.1 Computational grids

In numerical computations, we usually cannot simulate the whole $\mathbb{R}^d$ but only a bounded domain $\Omega \subset \mathbb{R}^d$. Finite volume methods work with averages of the solution over small volumes of this domain instead of approximating the solution pointwise. We therefore have to partition the domain Ω into small volumes. For that purpose, we first need some geometrical definitions.

Definition 4.1. A subset $A \subset \mathbb{R}^d$ is called *affine* if for all $\mathbf{x}_1, \mathbf{x}_2 \in A$ the line through $\mathbf{x}_1$ and $\mathbf{x}_2$ also lies in A. For A affine, the set

$$W = \{\mathbf{x} \,|\, \mathbf{x}_0 + \mathbf{x} \in A\}$$

is a linear subspace of $\mathbb{R}^d$ for arbitrary $\mathbf{x}_0 \in A$. If $\dim(W) = r$ then A has *affine dimension* r, denoted $\operatorname{affdim}(A) = r$. For an arbitrary set $B \subset \mathbb{R}^d$, the *affine hull* $\operatorname{aff}(B)$ is defined as the smallest affine set containing B. The affine dimension of B is then defined as

$$\operatorname{affdim}(B) := \operatorname{affdim}(\operatorname{aff}(B)).$$

Definition 4.2. A *convex polytope* $P \subset \mathbb{R}^d$ is the convex hull of a finite set of points $\mathbf{x}_1, \dots, \mathbf{x}_k \in \mathbb{R}^d$. For a convex polytope P, a non-empty subset $F \subset P$ is called an r-face if $\operatorname{affdim}(F) = r$ and for all $\mathbf{x}_1, \mathbf{x}_2 \in P$

$$(\mathbf{x}_1, \mathbf{x}_2) \cap F \neq \emptyset \quad \Rightarrow \quad [\mathbf{x}_1, \mathbf{x}_2] \subset F$$

where $(\mathbf{x}_1, \mathbf{x}_2) = \{a\mathbf{x}_1 + (1-a)\mathbf{x}_2 \,|\, a \in (0,1)\}$ and $[\mathbf{x}_1, \mathbf{x}_2] = (\mathbf{x}_1, \mathbf{x}_2) \cup \{\mathbf{x}_1, \mathbf{x}_2\}$ are the open and closed line segment between $\mathbf{x}_1$ and $\mathbf{x}_2$, respectively.

It can be shown that this definition of a face is equivalent to defining a face as the intersection of P with a supporting hyperplane [37]. Faces of a convex polytope are again convex polytopes.

Example 4.3. A cube is a polytope in $\mathbb{R}^3$. The 0-, 1- and 2-faces of the cube are the vertices, edges and faces, respectively. The cube itself is a 3-face.

Let finally $\operatorname{Poly}(k)$ be the set of convex polytopes in $\mathbb{R}^d$ with at most k vertices (0-faces) and non-empty interior. We can now define a grid on Ω.

Definition 4.4. Let $I := \{0, 1, \dots, N_x - 1\} \subset \mathbb{N}_0$ be a finite index set. The set

$$\tau_h := \{T_i \mid T_i \in \mathrm{Poly}(k), i \in I\}$$

is called *unstructured conformal grid* for Ω if

- $\Omega = \bigcup\limits_{i \in I} T_i$,
- $\mathring{T}_i \cap \mathring{T}_j = \emptyset \ \forall i \neq j \in I$ and
- either $T_i \cap T_j = \emptyset$ or $T_i \cap T_j$ is an r-face of both T_i and T_j for an $r \in \{0, \dots, d-1\}$.

Here, $\mathring{T}_i$ denotes the interior of T_i. Note that this definition of a grid requires that the boundary $\partial\Omega$ consists of $(d-1)$-faces of the grid cells T_i. For arbitrary domains with a smooth boundary, we can never fulfil the requirement that Ω should be the union of a finite set of polytopes. We index these boundary $(d-1)$-faces by an index set $J \subset \mathbb{N}$ with $I \cap J = \emptyset$ and denote $\tilde{I} = I \cup J$.

Let

$$\mathcal{E} := \{(i,j) \in I \times I \mid i \neq j \text{ and } T_i \cap T_j \text{ is a } (d-1)\text{-face of } T_i \text{ and } T_j\}$$

and define $S_{ij} = T_i \cap T_j$ for $(i,j) \in \mathcal{E}$. Let further S_{ij} also denote the boundary $(d-1)$-face with index $j \in J$ where $i \in I$ is chosen such that $S_{ij} \subset T_i$. We define the set of interfaces of T_i as

$$S(i) := \{j \in \tilde{I} \mid S_{ij} \text{ exists}\}. \tag{4.1}$$

We always assume that the grid fulfils the regularity property

$$\alpha h_i^d < |T_i|, \tag{4.2a}$$

$$\alpha |\partial T_i| < h_i^{d-1} \tag{4.2b}$$

for an $\alpha > 0$ and all $i \in I$. Here, $|T_i|$, $|\partial T_i|$ denote the volume and area of $T_i, \partial T_i$, respectively, and the *local grid width* $h_i := \mathrm{diam}(T_i)$ is the length of the longest straight line contained in T_i. The *global grid width* is defined as $h := \max\limits_{i \in I} h_i$. This regularity property essentially ensures that the angles in all polytopes T_i stay bounded away from zero.

Similarly to the spatial discretization of the domain, we only regard a finite time interval $[0, t_{\text{end}}]$ instead of $\mathbb{R}_+$. We subdivide $[0, t_{\text{end}}]$ into N_t intervals $[t^n, t^{n+1}]$ with $t^0 = 0$ and $t^{N_t} = t_{\text{end}}$. We define the n-th *time step* as $\Delta t_n = t^{n+1} - t^n$ for $n = 0, \dots, N_t - 1$.

4.1.2 Finite volume schemes and numerical fluxes

Integrating (3.1) over a single grid cell T_i and the time interval $[t^n, t^{n+1}]$, we get

$$\begin{aligned}
\int_{t^n}^{t^{n+1}} \int_{T_i} \mathbf{h}(t, \mathbf{x}, \mathbf{p}) &= \int_{t^n}^{t^{n+1}} \int_{T_i} (\partial_t \mathbf{p} + \sum_{j=0}^{d} \partial_{x_j} \mathbf{f}^j(p)) \\
&= \int_{T_i} \mathbf{p}(\cdot, t^{n+1}) - \int_{T_i} \mathbf{p}(\cdot, t^n) + \int_{t^n}^{t^{n+1}} \int_{T_i} (\operatorname{div} \mathbf{f}_1, \dots, \operatorname{div} \mathbf{f}_d)^T \\
&= \int_{T_i} \mathbf{p}(\cdot, t^{n+1}) - \int_{T_i} \mathbf{p}(\cdot, t^n) + \int_{t^n}^{t^{n+1}} \int_{\partial T_i} (\mathbf{f}_1 \cdot \mathbf{n}, \dots, \mathbf{f}_d \cdot \mathbf{n})^T \\
&= \int_{T_i} \mathbf{p}(\cdot, t^{n+1}) - \int_{T_i} \mathbf{p}(\cdot, t^n) + \int_{t^n}^{t^{n+1}} \int_{\partial T_i} \sum_{k=1}^{d} \mathbf{f}^k(\mathbf{p}) n_k.
\end{aligned} \tag{4.3}$$

Here, $\mathbf{n}$ is the unit outer normal to ∂T_i and $\mathbf{f}_i = (f_i^1, \dots, f_i^d)^T$ (not to be confused with $\mathbf{f}^i = (f_1^i, \dots, f_d^i)^T$).

If we approximate $\mathbf{p}$ on a grid cell T_i and a point in time t^n by its average on the cell, i.e.

$$\mathbf{p}_i^n := \frac{1}{|T_i|} \int_{T_i} \mathbf{p}(t^n, \mathbf{x}) \mathrm{d}\mathbf{x}, \tag{4.4}$$

we can write (4.3) after dividing by $1/|T_i|$ as

$$\mathbf{p}_i^{n+1} = \mathbf{p}_i^n - \frac{\Delta t_n}{|T_i|} \frac{1}{\Delta t_n} \int_{t^n}^{t^{n+1}} \int_{\partial T_i} \sum_{k=1}^{d} \mathbf{f}^k(\mathbf{p}) n_k + \frac{1}{|T_i|} \int_{t^n}^{t^{n+1}} \int_{T_i} \mathbf{h}(t, \mathbf{x}, \mathbf{p}). \tag{4.5}$$

Thus, to get a discrete method we would like to approximate the flux term

$$\frac{1}{\Delta t_n} \int_{t^n}^{t^{n+1}} \int_{\partial T_i} \sum_{k=1}^{d} \mathbf{f}^k(\mathbf{p}) n_k \tag{4.6}$$

and the source term

$$\frac{1}{|T_i|} \int_{t^n}^{t^{n+1}} \int_{T_i} \mathbf{h}(t, \mathbf{x}, \mathbf{p}) \tag{4.7}$$

by terms that depend only on the discrete values $\mathbf{p}_j^n$, $j \in I$. For that purpose, we need to discretize the time integral by an appropriate quadrature and then approximate the spatial integral at each quadrature time point. For presentation purposes, we use Euler time stepping here (see Section 4.2) and thus approximate the time integral by evaluating at time t^n and multiplying by Δt_n. Higher order methods for the approximation of the integral are regarded in Section 4.2.

For the spatial discretization, we start with the flux term. After the Euler discretization of the time integral, we are left with the approximation

$$\int_{\partial T_i} \sum_{k=1}^{d} \mathbf{f}^k(\mathbf{p}(t^n,\cdot))n_k$$

for the flux term. Information in a hyperbolic system of equations propagates with finite speed (see Section 4.1.4). Hence, if we choose the time step Δt_n small enough, it is safe to assume that we can approximate the flux term by only using the values $\mathbf{p}_j^n$ for cells T_j in a sufficiently small neighbourhood of T_i. Note that ∂T_i consists of the interfaces S_{ij}, $j \in S(i)$. We thus make the ansatz

$$\int_{\partial T_i} \sum_{k=1}^{d} \mathbf{f}^k(\mathbf{p}(t^n,\cdot))n_k = \sum_{j\in S(i)} \int_{S_{ij}} \sum_{k=1}^{d} \mathbf{f}^k(\mathbf{p}(t^n,\cdot))n_k \approx \sum_{j\in S(i)} \mathbf{g}_{ij}(\mathbf{p}_i^n, \mathbf{p}_j^n).$$

The approximation $\mathbf{g}_{ij}$ is called a numerical flux. If $j \in J$, i.e. if the interface S_{ij} is on the boundary, the value $\mathbf{p}_j^n$ does not exist. We thus have to specify boundary values or define $\mathbf{g}_{ij}$ using only $\mathbf{p}_i^n$ (see Section 4.1.6). In the following, we will always assume that $\mathbf{p}_j^n$ is suitably defined.

The main challenge in developing finite volume methods is to find appropriate numerical flux functions $\mathbf{g}_{ij}$. We summarize some requirements on the numerical flux in the following definition.

Definition 4.5. For an interface S_{ij}, $\mathbf{g}_{ij} : (\mathbb{R}^m)^2 \to \mathbb{R}^m$ is called a *numerical flux* with respect to $\mathbf{f}^1, \dots, \mathbf{f}^d$ if it holds

- Lipschitz continuity: $\forall\ \mathbf{p}, \mathbf{q}, \mathbf{p}', \mathbf{q}' \in B_R(\mathbf{0})$:

$$|\mathbf{g}_{ij}(\mathbf{p},\mathbf{q}) - \mathbf{g}_{ij}(\mathbf{p}',\mathbf{q}')| \le L_{\mathbf{g}}|S_{ij}|(|\mathbf{p}-\mathbf{p}'| + |\mathbf{q}-\mathbf{q}'|),$$

- Conservation: $\mathbf{g}_{ij}(\mathbf{p},\mathbf{q}) = -\mathbf{g}_{ji}(\mathbf{q},\mathbf{p})$,
- Consistency: $\mathbf{g}_{ij}(\mathbf{p},\mathbf{p}) = \int_{S_{ij}} \mathbf{F}(\mathbf{p}) \cdot \mathbf{n}_{ij} := \int_{S_{ij}} \sum_{k=1}^{d} \mathbf{f}^k(\mathbf{p})n_{ij,k}$.

Here, $\mathbf{n}_{ij}$ is the unit outer normal to the interface S_{ij} and $B_R(\mathbf{0}) \subset \mathbb{R}^m$ is the ball with radius R centred at $\mathbf{0}$.

If there is no source term, i.e. if $\mathbf{h} = \mathbf{0}$, we are ready to define a numerical scheme to solve (3.1):

Definition 4.6. For each interface S_{ij} of a grid, let $\mathbf{g}_{ij}$ be a numerical flux with respect to $\mathbf{f}^1, \dots, \mathbf{f}^d$. The scheme

$$\mathbf{p}_i^0 := \frac{1}{|T_i|} \int_{T_i} \mathbf{p}_0(\mathbf{x})\mathrm{d}\mathbf{x}, \tag{4.8}$$

$$\mathbf{p}_i^{n+1} := \mathbf{p}_i^n - \frac{\Delta t_n}{|T_i|} \sum_{j\in S(i)} \mathbf{g}_{ij}(\mathbf{p}_i^n, \mathbf{p}_j^n) \tag{4.9}$$

is called *finite volume scheme in conservation form.* The function

$$\mathbf{p}_h(t,\mathbf{x}) := \mathbf{p}_i^n \quad \forall \mathbf{x} \in T_i,\, t \in [t^n, t^{n+1})$$

is called *numerical solution* to the finite volume scheme.

Even if the requirements from Definition 4.5 are fulfilled, the numerical scheme will not necessarily converge to the correct entropy weak solution. Some additional restrictions have to be made to the flux to ensure this convergence. Classes of fluxes that have this property are monotone fluxes and E-fluxes (see e.g. [4, 56]). We do not treat these general classes here but rather regard some specific fluxes.

Example 4.7. One of the simplest numerical fluxes is the *Lax-Friedrichs flux* [55]

$$\mathbf{g}_{ij}^{LF}(\mathbf{p},\mathbf{q}) := \left(\frac{1}{2}(F(\mathbf{p}) + F(\mathbf{q})) \cdot \mathbf{n}_{ij} - \frac{1}{2\lambda_{ij}}(\mathbf{q} - \mathbf{p})\right) |S_{ij}|, \tag{4.10}$$

where λ_{ij} is chosen such that

$$\lambda_{ij} = \lambda_{ji} \geq c > 0, \tag{4.11}$$

$$\lambda_{ij} \sup_{\mathbf{p}} (\mathbf{F}(\mathbf{p}) \cdot \mathbf{n}_{ij})' \leq 1 \tag{4.12}$$

for a constant $c \in \mathbb{R}$.

It can be easily verified that the Lax-Friedrichs flux is indeed a numerical flux in the sense of Definition 4.5. Furthermore, it contains numerical viscosity that ensures convergence to the entropy weak solution [48]. This can most easily be seen in the one-dimensional scalar case on an equidistant grid with $h_i = \Delta x$ for all i. Here, traditionally $\lambda_{ij} = \Delta t_n / \Delta x$ is used which fulfils (4.12) as long as the CFL condition (see Section 4.1.4) is satisfied. Using the Lax-Friedrichs flux, the update formula for p_i^n is

$$p_i^{n+1} = p_i^n + \Delta t_n \frac{f(p_{i+1}^n) - f(p_{i-1}^n)}{2\Delta x} - \frac{(\Delta x)^2}{2} \frac{p_{i+1}^n - 2p_i^n + p_{i-1}^n}{(\Delta x)^2}$$

which looks like we are modelling the viscous equation

$$\partial_t p + \partial_x f(p) = \beta \partial_{xx}^2 p$$

for $\beta = \frac{(\Delta x)^2}{2\Delta t_n}$ with a finite difference method. This ensures the convergence to the entropy solution but often leads to strong smearing of shocks [56].

Instead of using $\lambda_{ij} = \Delta t_n / \Delta x$, a locally determined value

$$\lambda_{ij} = \max |f'(p)| \text{ over all } p \text{ between } p_i^n \text{ and } p_j^n$$

can be used at each interface. This method is called the *local Lax-Friedrichs* method. If the CFL condition is satisfied, then $|f'(p)| \leq \Delta t_n / \Delta x$ everywhere. Hence, the traditional Lax-Friedrichs scheme amounts to using a uniform viscosity that is sufficient everywhere. The local

Lax-Friedrichs methods shows less smearing and still contains enough viscosity to guarantee convergence to the entropy solution [56].

Example 4.8. For the scalar advection equation, information propagates only in one direction given by the velocity vector. For example, in one dimension, a positive velocity λ suggests using only information from the left one of the two cells in the numerical flux

$$g_{ij}(p_i^n, p_j^n) = \begin{cases} \lambda p_i^n & \text{if } T_i \text{ is to the left of } T_j \\ -\lambda p_j^n & \text{else.} \end{cases}$$

If the velocity was negative, information from the right cell would have to be used. Such a method looking at the direction from which information should be coming is called an *upwind method.* For a linear system of equations in one dimension, the above flux can be easily adapted by using the upwind flux on each of the decoupled advection equations.

The *Godunov flux* [28] is one of the most used flux functions and generalizes the upwind flux idea to non-linear equations. Upwind information is obtained by solving one-dimensional Riemann problems with initial values $\mathbf{p}_i$ and $\mathbf{p}_j$ at each interface S_{ij}. To be more precise, we solve the problem

$$\partial_t \mathbf{u}(t, y) + \partial_y(\mathbf{F} \cdot \mathbf{n}_{ij})(\mathbf{u}(t, y)) = \mathbf{0},$$
$$\mathbf{u}_0(y) = \begin{cases} \mathbf{p}_i^n \text{ if } y < 0 \\ \mathbf{p}_j^n \text{ if } y > 0 \end{cases}$$

exactly where we can think of y as being the coordinate normal to the interface S_{ij}. The value of the solution $\mathbf{u}$ at $y = 0$ (or at least the flux using the value of $\mathbf{u}$ at $y = 0$, see Section 3.3) is constant over the time interval (t^n, t^{n+1}). Let $\mathbf{F}(\mathbf{u}_G(\mathbf{p}_i^n, \mathbf{p}_j^n)) = \mathbf{F}(\mathbf{u}(t^n + \frac{1}{2}\Delta t_n, 0))$ be this constant flux value. If we replace $\mathbf{F}(\mathbf{p})$ by $\mathbf{F}(\mathbf{u}_G)$ on S_{ij}, we can determine the flux term (4.6) as

$$\frac{1}{\Delta t_n} \int_{t^n}^{t^{n+1}} \int_{S_{ij}} \mathbf{F}(\mathbf{u}_G) \cdot \mathbf{n}_{ij} = |S_{ij}| \mathbf{F}(\mathbf{u}_G) \cdot \mathbf{n}_{ij}.$$

Thus, the Godunov flux is defined as

$$\mathbf{g}_{ij}^G(\mathbf{p}, \mathbf{q}) := |S_{ij}| \mathbf{F}(\mathbf{u}_G(\mathbf{p}, \mathbf{q})) \cdot \mathbf{n}_{ij}. \tag{4.13}$$

For the scalar advection equation or a linear system of equations in one space dimension, this is just the upwind flux from above.

It can be shown that the finite volume scheme using Godunov's flux converges to the entropy solution as long as the Riemann problem at each interface is solved with the correct entropy solution.

Solving the Riemann problem at each interface in every time step can be computationally demanding. Moreover, the full solution $\mathbf{u}$ is not needed for the Godunov flux, only the value

$\mathbf{u}_G$. Thus, a variety of approximate Riemann solvers has been developed. We will use a simple linearized Riemann solver (see [56, Section 15.3]) where we solve the linearized equation

$$\partial_t \mathbf{u} + \mathbf{A}\partial_y \mathbf{u} = \mathbf{0} \tag{4.14}$$

locally at each cell interface with $\mathbf{A} = \mathbf{D}(\mathbf{F}\cdot\mathbf{n}_{ij})(\frac{1}{2}(\mathbf{p}_i^n + \mathbf{p}_j^n))$ being an approximation of the Jacobian of $\mathbf{F}\cdot\mathbf{n}_{ij}$ at the interface S_{ij}.

4.1.3 Source terms

If $\mathbf{h} \neq \mathbf{0}$, we need to include the source term in our calculation. If we use Euler time stepping, the source term (4.7) becomes

$$\frac{\Delta t_n}{|T_i|}\int\limits_{T_i} \mathbf{h}(t, \mathbf{x}, \mathbf{p}).$$

A simple approach to discretize the source term is to replace $\mathbf{p}$ by $\mathbf{p}_i^n$ in each time step. The remaining integral can then be computed analytically, if possible, or by quadrature. Using a simple midpoint quadrature, for example, the update formula including flux and source term becomes

$$\mathbf{p}_i^{n+1} = \mathbf{p}_i^n - \frac{\Delta t_n}{|T_i|}\sum_{j\in S(i)} \mathbf{g}_{ij}(\mathbf{p}_i^n, \mathbf{p}_j^n) + \Delta t_n \mathbf{h}(t^n, \mathbf{x}_i, \mathbf{p}_i^n), \tag{4.15}$$

where $\mathbf{x}_i$ is the midpoint of the cell T_i.

Another approach introduced by Godunov in [28] is to split the problem (3.1) in two problems that are solved successively in each time step. More precisely, instead of (3.1) we regard the two equations

$$\partial_t \mathbf{p}(t, \mathbf{x}) + \sum_{i=1}^{d} \partial_{x_i} \mathbf{f}^i(\mathbf{p}) = \mathbf{0}, \tag{4.16}$$

$$\partial_t \mathbf{p}(t, \mathbf{x}) = \mathbf{h}(t, \mathbf{x}, \mathbf{p}) \tag{4.17}$$

and solve both equations separately in each time step, using the result of the first equation as data for the second one. The first equation is just the conservation law without source term. Averaging the second equation (4.17) over a grid cell gives the ordinary differential equation (ODE)

$$\mathrm{d}_t \frac{1}{|T_i|}\int\limits_{T_i} \mathbf{p}(t, \mathbf{x})\mathrm{d}\mathbf{x} = \frac{1}{|T_i|}\int\limits_{T_i} \mathbf{h}(t, \mathbf{x}, \mathbf{p})\mathrm{d}\mathbf{x}. \tag{4.18}$$

We can use the finite volume scheme (4.8) for (4.16) and a standard ODE method for (4.17) (or rather the integral form (4.18)). If we use the forward Euler method for both equations, our scheme becomes

$$\mathbf{p}_i^* = \mathbf{p}_i^n - \frac{\Delta t_n}{|T_i|} \sum_{j \in S(i)} \mathbf{g}_{ij}(\mathbf{p}_i^n, \mathbf{p}_j^n), \tag{4.19a}$$

$$\mathbf{p}_i^{n+1} = \mathbf{p}_i^* + \frac{\Delta t_n}{|T_i|} \int\limits_{T_i} \mathbf{h}(t^n, \mathbf{x}, \mathbf{p}_i^*) \mathrm{d}\mathbf{x}. \tag{4.19b}$$

Note that we simply replaced $\mathbf{p}(t^n, \mathbf{x})$ by $\mathbf{p}_i^*$ in the source term $\mathbf{h}$ to get a fully discrete method. It may look like we evolve our equation by $2\Delta t_n$ here, but in each step we use only one of the terms of our original equation, so this gives an approximation to (3.1). This is called a *fractional step method.* The method (4.15) introduced before is called an *unsplit method.* The fractional step method is formally only first order accurate even if the two subproblems are solved exactly because a splitting error of $O(\Delta t)$ is introduced. The advantage of the fractional step method over the unsplit method is, however, that numerical methods for the two equations can be applied separately. In particular, the higher order methods for the conservation law we present in Section 4.1.5 can be applied directly, whereas deriving an unsplit method based on these methods may be more complicated. Using the higher order methods in the fractional step approach can give considerably better results particularly in discontinuous parts of the solution.

Changing the evaluation order of the equations (4.19) in each time step, called Strang splitting [80], can improve the formal accuracy. Despite its low-order formal accuracy, the fractional step approach can give reasonably good results in practice (see [56, Section 17]).

4.1.4 Time step restrictions

In the definition of the finite volume scheme, we assumed that we can approximate the flux using only information from the adjacent cells. This is justified by the observation that information in a hyperbolic system of equations propagates with finite speed. The solution at $(\tilde{t}, \tilde{\mathbf{x}})$ thus does not depend on the initial values for all $\mathbf{x} \in \mathbb{R}^d$ but only on the initial values in a smaller domain. This domain is called the *domain of dependence* of the point $(\tilde{t}, \tilde{\mathbf{x}})$. For the linear advection equation in one dimension with speed λ, the domain of dependence consists only of the single point $\tilde{x} - \lambda\tilde{t}$, whereas for a linear system of equations, the m points $\tilde{x} - \lambda_s\tilde{t}$ build the domain of dependence. For a nonlinear equation in one dimension, the domain of dependence is more complex but is contained in the interval $[\tilde{x} - u\tilde{t}, \tilde{x} + u\tilde{t}]$, where $u = \max_p |f'(p)|$. For a nonlinear system, the domain of dependence can be bound depending on the maximum eigenvalues of $\mathbf{Df}$. In several dimensions, the domain of dependence can be bound in a similar way using the appropriate flux $\mathbf{f}^i$ for each coordinate direction.

A numerical method can only be stable if the numerical domain of dependence contains the analytical domain of dependence. This was first stated by Courant, Friedrichs and Lewy [18] and is thus often called the *CFL condition.* If the numerical domain of dependence does not contain the analytical domain of dependence, a change in the initial values might affect the true solution at the point $(\tilde{t}, \tilde{\mathbf{x}})$ but not the numerical approximation. This shows that the scheme cannot converge for arbitrary initial values.

If we use an equidistant grid with cell width Δx and denote the i-th grid cell by $T_i = [x_{i-\frac{1}{2}}, x_{i+\frac{1}{2}}]$, the numerical domain of dependence of $\mathbf{p}_i^n$ for the finite volume scheme (4.8) in one dimension is

$$[x_{i-\frac{1}{2}} - n\Delta x, x_{i+\frac{1}{2}} + n\Delta x],$$

as in each time step the value p_i^{n+1} depends on the values in the cells T_{i-1}, T_i and T_{i+1} in the previous time step. If we use a constant time step Δt, then $t^n = n\Delta t$. We thus have to require

$$\begin{aligned} x_{i-\frac{1}{2}} - n\Delta x &\leq x_{i-\frac{1}{2}} - un\Delta t, \\ x_{i+\frac{1}{2}} + un\Delta t &\leq x_{i+\frac{1}{2}} + n\Delta x \end{aligned}$$

where u is an upper bound on the information propagation speed, e.g. $u = \max_p |f'(p)|$ for a nonlinear scalar equation. We thus have to restrict the time step such that

$$\Delta t \leq \frac{1}{u}\Delta x.$$

The factor $\frac{\Delta t}{\Delta x}$ is called *CFL number* or *Courant number*. We will use the term Courant number throughout this thesis to avoid ambiguity with the CFL coefficient in Section 4.2. For a scheme with varying cell diameter h_i and time steps Δt_n the Courant number describes the ratio $\frac{\Delta t_n}{h_i}$. In order to get a stable scheme, the Courant number has to be chosen small enough.

The CFL condition is only a necessary condition. In practice, the Courant number may have to be chosen much smaller than the CFL condition requires to get a stable scheme.

4.1.5 Higher order methods

The Lax-Friedrichs and Godunov schemes introduced in Examples 4.7 and 4.8 use the average values $\mathbf{p}_i^n$ and $\mathbf{p}_j^n$ to approximate the flux at the interface S_{ij}. It can be shown that these fluxes (under the assumptions made, i.e. regularity of the grid, bounded total variation of the initial values, suitable time step restriction) give a finite volume scheme that converges to the correct entropy weak solution of the conservation law (see [4, 48, 56]). However, the convergence rate is quite low.

Theorem 4.9 (Convergence rate of finite volume schemes, [4]). *Let p be the entropy weak solution to a scalar hyperbolic conservation law where p_0 has bounded total variation. Let p_h be an approximate solution obtained by the finite volume scheme* (4.8) *(where the regularity condition on the grid* (4.2) *and a CFL-like condition on the time step is satisfied) using one of the fluxes from Examples 4.7 and 4.8. Let $K \subset\subset \mathbb{R}_+ \times \mathbb{R}^d$ be relatively compact. Then there exists a constant $C \geq 0$ such that*

$$\int_K |p - p_h| \leq Ch^{1/4}.$$

Moreover, in the one-dimensional case, the optimal convergence rate of 1/2 is obtained.

It is known that the L^1-norm convergence rate bound of 1/2 is sharp for general conservation laws [4].

The accuracy can be improved by using information from a wider stencil of cells to reconstruct the value of $\mathbf{p}$ at the interface. We do this here for the scalar conservation law in one space dimension ($m = d = 1$, $\mathbf{h} = 0$). Furthermore, for simplicity, we assume an equidistant grid with cell width Δx. Equation 4.3 for the average value over the grid cell $[x_{i-\frac{1}{2}}, x_{i+\frac{1}{2}}]$ then becomes

$$p_i^{n+1} = p_i^n - \frac{\Delta t_n}{\Delta x}\left(\frac{1}{\Delta t_n}\int_{t^n}^{t^{n+1}} f(p(t^n, x_{i+\frac{1}{2}})) - \frac{1}{\Delta t_n}\int_{t^n}^{t^{n+1}} f(p(t^n, x_{i-\frac{1}{2}}))\right).$$

The finite volume scheme (4.8) thus is

$$p_i^{n+1} = p_i^n - \frac{\Delta t_n}{\Delta x}\left(g_{i,i+1}(p_i^n, p_{i+1}^n) + g_{i,i-1}(p_i^n, p_{i-1}^n)\right). \tag{4.20}$$

Here, the numerical flux $g_{i,i+1}$ approximates the flux at the point $x_{i+\frac{1}{2}}$ using the average cell values p_i^n and p_{i+1}^n. The method can be improved by making a better guess of the value $p(t^n, x_{i+\frac{1}{2}})$ from both sides of the interface $x_{i+\frac{1}{2}}$ and using these values instead of the average cell values in the numerical flux $g_{i,i+1}$.

One way to improve the approximation of p at the interface $x_{i+\frac{1}{2}}$ is to do a linear reconstruction. We first calculate a slope σ_i^n on each cell using the values p_j^n of the adjacent cells. We can, for example, use one of the following slopes

$$(\sigma_i^n)_{\text{centred}} = \frac{p_{i+1}^n - p_{i-1}^n}{2\Delta x}, \quad (\sigma_i^n)_{\text{left}} = \frac{p_i^n - p_{i-1}^n}{\Delta x}, \quad (\sigma_i^n)_{\text{right}} = \frac{p_{i+1}^n - p_i^n}{\Delta x}.$$

We then reconstruct the value of p at the interface $x_{i+\frac{1}{2}}$ in both adjacent cells as

$$\begin{aligned}(p_i^n)_+ &= p_i^n + \frac{\Delta x}{2}\sigma_i^n,\\ (p_{i+1}^n)_- &= p_{i+1}^n - \frac{\Delta x}{2}\sigma_{i+1}^n.\end{aligned}$$

Using the reconstructed values, the finite volume scheme (4.20) becomes

$$p_i^{n+1} = p_i^n - \frac{\Delta t_n}{\Delta x}\Big(g_{i,i+1}((p_i^n)_+, (p_{i+1}^n)_-) + g_{i,i-1}((p_i^n)_-, (p_{i-1}^n)_+)\Big). \tag{4.21}$$

This approach gives better accuracy in smooth parts of the solution. Near discontinuities, however, artificial oscillations are introduced by this method as the slopes may become very large and cause overshoots (see e.g. [56, Section 6.6] for an illustration of the problem). Hence, we would like to use a smaller slope such that artificial oscillations do not occur. If we limit the slope too much, however, we do not obtain higher accuracy at all. To figure out how to choose the slope, we first need a measure of oscillations in the solution.

Definition 4.10. The *total variation* of a function p is defined as

$$TV(p) = \sup_{s \in \mathbb{N}} \sup_{y_0 < \ldots < y_s} \sum_{k=1}^{s} |p(y_s) - p(y_{s-1})|.$$

For a grid function $p^n = (p_i^n)_{i=-\infty}^{\infty}$, we define the total variation as

$$TV(p^n) = \sum_{k=-\infty}^{\infty} |p_k^n - p_{k-1}^n|.$$

For smooth functions, an equivalent definition is

$$TV(p) = \int_{-\infty}^{\infty} |p'(y)| \mathrm{d}y.$$

The total variation will be infinite if p does not tend towards a constant value for $x \to \pm\infty$. However, if we start with initial values that have bounded total variation, the solution of a hyperbolic conservation law is *total variation non-increasing* (see [34]), i.e.

$$TV(p(t_1, \cdot)) \leq TV(p(t_2, \cdot)) \quad \text{for all} \quad t_1 \leq t_2. \tag{4.22}$$

Somewhat inaccurately, this property is usually called *total variation diminishing* (TVD) instead of total variation non-increasing (TVNI).

It is reasonable to demand that a numerical scheme for the hyperbolic conservation law should also be TVD, i.e. the approximate solution at time t^{n+1} should always have lower or equal total variation compared to the approximate solution at time t^n. This prevents artificial oscillations as they would increase the total variation. The method (4.21) introduced above using linear reconstruction thus is not TVD. It can be shown that the finite volume scheme (4.20) using the Lax-Friedrichs flux or the Godunov flux is TVD [56]. We can get these methods from the linear reconstruction method (4.21) by setting the slope to zero. Thus, we might expect that the scheme (4.21) is TVD if we limit the slope sufficiently. It turns out that this is indeed the case. Choices of limited slopes that give a TVD method include (see [56, Section 6.9])

$$(\sigma_i^n)_{\text{minmod}} = \operatorname{minmod}\left((\sigma_i^n)_{\text{right}}, (\sigma_i^n)_{\text{left}}\right), \tag{4.23}$$

$$(\sigma_i^n)_{\text{superbee}} = \operatorname{maxmod}\left(\operatorname{minmod}\left((\sigma_i^n)_{\text{right}}, 2(\sigma_i^n)_{\text{left}}\right), \operatorname{minmod}\left(2(\sigma_i^n)_{\text{right}}, (\sigma_i^n)_{\text{left}}\right)\right), \tag{4.24}$$

$$(\sigma_i^n)_{\text{MC}} = \operatorname{minmod}\left((\sigma_i^n)_{\text{centred}}, 2(\sigma_i^n)_{\text{left}}, 2(\sigma_i^n)_{\text{right}}\right), \tag{4.25}$$

where the minmod function chooses the argument with smallest absolute value if all of its arguments have the same sign and is zero else. The maxmod function is defined analogously but chooses the argument with greatest absolute value.

These methods can be generalized to linear systems of equations by carrying out the reconstruction process for each of the decoupled advection equations after the variable transformation. The generalization to nonlinear systems is more complicated. Simply doing the reconstruction for each component of the solution separately is in general not sufficient. Instead, a wave decomposition has to be done and each of the waves has to be regarded separately similar to the linear case (see [56, Section 15.4]).

Note that while the flux limiter methods described here give faster convergence in practice, there is no theoretical a priori result similar to Theorem 4.9 for these methods with a convergence rate higher than 1/2 [4, 48].

4.1.6 Boundary conditions

Boundary conditions for hyperbolic systems of equations must be chosen with caution to get a well-posed problem and are a difficult topic on their own. We do not go into any details here but only regard some basic concepts. See, for example, [27, 81] for an introduction to the topic.

Consider the scalar advection equation (3.6) on the domain $\Omega = [x_L, x_R]$ with initial values $p_0(x)$. If $\lambda > 0$, the characteristics leave from the left boundary $x = x_L$. We hence need to prescribe the value of p on that boundary, $p(t, x_L) = p_L(t)$. In contrast, the characteristics impinge on the right boundary $x = x_R$ and thus the value of p on that boundary is fully determined by the initial values p_0. If we prescribed boundary values in this case, the problem would be ill-posed in general.

Consider a finite volume scheme using the Lax-Friedrichs flux. If we use an equidistant grid with grid width $\Delta x = (x_R - x_L)/N_x$, we have N_x cells $T_i = [x_L + i\Delta x, x_L + (i+1)\Delta x]$, $i = 0, \ldots, N_x - 1$. On the boundary cells T_0 and T_{N_x-1}, we cannot simply use the Lax-Friedrichs flux as this would require information from neighbouring cells to both sides. One possibility to solve this problem is to use the Lax-Friedrichs flux on the inner cells $T_1, \ldots, T_{N_x-2}$ and use a different method using only information from one side (e.g. an upwind flux) on the boundary cells (see [56, Section 7]). However, often it is easier to use the same method for all cells and add *ghost cells* at the boundary, i.e. additional cells that are not in the physical domain. Here, we use one extra cell on both sides, i.e. a cell $T_{-1} = [x_L - \Delta x, x_L]$ and a cell $T_{N_x} = [x_R, x_R + \Delta x]$. On the left boundary, we can then set the value of p^n_{-1} in the cell T_{-1} according to the boundary conditions, for example by averaging $p_L(t)$ over the time interval in each time step. On the right boundary, on the other hand, we cannot impose boundary conditions but still need a value in the cell T_{N_x} for the Lax-Friedrichs flux. This can be solved by calculating the value $p^n_{N_x}$ from the values in the cells T_{N_x-2} and T_{N_x-1}, for example by simply setting $p^n_{N_x} = p^n_{N_x-1}$ (zero-order interpolation) or getting $p^n_{N_x}$ by linear interpolation from the two values $p^n_{N_x-2}$ and $p^n_{N_x-1}$. This way, the information still comes only from the cells in the physical domain and no extra boundary conditions need to be imposed. Thus, this approach is similar to using some kind of upwind method in the right-most cell.

If we use one of the higher order methods from Section 4.1.5, two ghost cells on both sides and suitable values in these cells are needed.

For a linear hyperbolic system of equations, there may be both incoming and outgoing characteristics on each boundary. Here, a characteristic decomposition in decoupled scalar advection equations can be done and the values in the ghost cells can be calculated as above for each advection equation. For nonlinear equations, requirements to the boundary conditions can be derived for example by a linearization of the problem at the boundary (see [27]).

Often, we actually regard a problem in an unbounded domain (with initial values that have bounded support) but need to use a bounded domain for the computations. In this case, we do not want the boundary to have any influence on the solution. This can be done by choosing the domain large enough that the support of the solution never reaches the boundary over the time regarded, but of course this causes a lot of extra computational cost. Instead, we can use *absorbing* (also called *non-reflecting*) boundary conditions that are supposed to completely absorb any wave that reaches the boundary. For Godunov-type methods, this can be done by simply using zero-order extrapolation [56]. Another possible choice of boundary conditions in this case, especially if information mostly flows in one direction, are *periodic* boundary conditions. Here, the ghost cells are filled with the values of p in the cells next to the boundary on the opposite side of the domain.

If we regard hyperbolic systems of equations that come from moment models, additional problems occur. Usually, boundary values for the kinetic equation are given only for velocities that correspond to incoming data (see (1.3)) but the definition of moments includes integration over the whole velocity space. In practice, this can sometimes be solved by extending the definition of the kinetic boundary values to all velocities or choosing the domain large enough that the boundary conditions have negligible effect on the solution, but in general this is an open question [1, 35].

4.2 Runge-Kutta methods

In Section 4.1 we derived an equation for the cell averages $\mathbf{p}_i^n$ at time t^n in the balance law (3.1) by integrating both over a grid cell T_i and a time interval (t^n, t^{n+1}). Recall that we approximated the time integral by assuming that the integrand is constant over the time interval. Hence, the numerical flux functions we derived in Section 4.1 essentially are an approximation to the flux at time t^n. We can make use of this fact to get better temporal accuracy in the finite volume scheme. We only regard hyperbolic conservation laws without source terms in this section as we incorporate the source terms in the scheme by the fractional step approach (see Section 4.1.3).

If we integrate (3.1) (with $\mathbf{h} = \mathbf{0}$) over the cell T_i only and divide by $1/|T_i|$, we get

$$\mathrm{d}_t\, \mathbf{p}_i(t) = -\frac{1}{|T_i|} \int\limits_{\partial T_i} \sum_{k=1}^{d} \mathbf{f}^k(\mathbf{p}(t,\mathbf{x})) n_k \mathrm{d}\mathbf{x}$$

with

$$\mathbf{p}_i(t) := \frac{1}{|T_i|} \int\limits_{T_i} \mathbf{p}(t,\mathbf{x}) \mathrm{d}\mathbf{x}.$$

If we use the interpretation of the numerical fluxes as approximation to the flux at time t, we get the semidiscrete scheme

$$\mathrm{d}_t\,\mathbf{p}_i(t) = -\frac{1}{|T_i|}\sum_{j\in S(i)} \mathbf{g}_{ij}(t,\mathbf{p}(t)), \tag{4.26}$$

where we collected the cell averages $\mathbf{p}_i(t)$ in the array $\mathbf{p}(t) = (\mathbf{p}_0(t),\dots,\mathbf{p}_{N_x-1}(t))$. We can write this in the abstract form

$$\mathrm{d}_t\,\mathbf{p}(t) = L(t,\mathbf{p}(t)),\ t\in[0,T], \tag{4.27a}$$

$$\mathbf{p}(0) = \tilde{\mathbf{p}}_0, \tag{4.27b}$$

where $\mathbf{p} : [0,t_{\text{end}}] \to (\mathbb{R}^m)^{N_x}$, $L : [0,t_{\text{end}}]\times(\mathbb{R}^m)^{N_x} \to (\mathbb{R}^m)^{N_x}$, $\tilde{\mathbf{p}}_0 = (\frac{1}{|T_i|}\int\limits_{T_i}\mathbf{p}_0(\mathbf{x}))_{i\in I}$.

This is a system of ordinary differential equations (ODEs). Note that while we assumed that the flux function in the conservation law is not explicitly time-dependent, L can well be time-dependent, e.g. if there are time-dependent boundary values.

We can now use standard ODE solvers to discretize (4.27) in time, but we need to take care that the time discretization does not break the desired TVD property of the numerical schemes from Section 4.1. We thus regard Runge-Kutta methods [50, 68], some of which can conserve the TVD property, as we will see in the following.

If we have the value of $\mathbf{p}$ at time t^n, we can calculate the value of $\mathbf{p}$ at the next time step t^{n+1} by

$$\mathbf{p}(t^{n+1}) = \mathbf{p}(t^n) + \int\limits_{t^n}^{t^{n+1}} \mathrm{d}_t\mathbf{p}(\xi)d\xi \overset{(4.27)}{=} \mathbf{p}(t^n) + \int\limits_{t^n}^{t^{n+1}} L(\xi,\mathbf{p}(\xi))d\xi. \tag{4.28}$$

Given an approximation $\mathbf{p}^n$ of $\mathbf{p}(t^n)$, Runge-Kutta methods replace the integral in (4.28) by a quadrature to get an approximation $\mathbf{p}^{n+1}$ of $\mathbf{p}(t^{n+1})$ as

$$\mathbf{p}^{n+1} = \mathbf{p}^n + \Delta t_n \sum_{k=0}^{s-1} b_k\mathbf{k}_k, \tag{4.29a}$$

where

$$\mathbf{k}_k = L(t^n + c_k\Delta t_n, \mathbf{p}^n + \Delta t_n \sum_{l=0}^{s-1} a_{kl}\mathbf{k}_l),\ k = 0,\dots,s-1. \tag{4.29b}$$

Thus, starting with $\mathbf{p}^0 = \tilde{\mathbf{p}}_0$, we obtain a series of approximations $\mathbf{p}^n$ to $\mathbf{p}$ at the time points t^n. The particular Runge-Kutta method is given by the number of stages $s\in\mathbb{N}$, the weights $b_k\in\mathbb{R}$, the nodes $c_k\in\mathbb{R}$ and the coefficients $a_{kl}\in\mathbb{R}$ $(0\le k,l\le s-1)$. This data is often arranged in a table called *Butcher array* or *Butcher tableau*:

$$\begin{array}{c|c} \mathbf{c} & \mathbf{A} \\ \hline & \mathbf{b}^T \end{array} = \begin{array}{c|ccc} c_0 & a_{00} & \dots & a_{0,s-1} \\ \vdots & \vdots & & \vdots \\ c_{s-1} & a_{s-1,0} & \dots & a_{s-1,s-1} \\ \hline & b_0 & \dots & b_{s-1} \end{array} . \tag{4.30}$$

The error introduced by the scheme (4.29) in a single time step is called the local truncation error, whereas the error at the endpoint $t^{N_t} = t_{\text{end}}$ is called global truncation error.

Definition 4.11. For all $n = 0, \dots, N_t - 1$ the *local truncation error* is defined as

$$\text{le}_n = |\mathbf{p}(t^{n+1}) - \mathbf{p}(t^n) - \Delta t_n \sum_{k=0}^{s-1} b_k \mathbf{k}_k|.$$

The *global truncation error* is defined as

$$E = |\mathbf{p}^{N_t} - \mathbf{p}(t_{\text{end}})|.$$

Definition 4.12. The scheme (4.29) has order r if r is the largest integer such that

$$\text{le}_n = O((\Delta t_n)^{r+1}),\ n = 0, \dots N_t - 1$$

for sufficiently smooth problems (4.27), i.e. if the Taylor series for the exact solution and the approximation agree up to the $(\Delta t_n)^r$-term for smooth problems.

If the Runge-Kutta scheme has order r, the global truncation error is $O((\Delta t_{\max})^r)$ (see [33]) where

$$\Delta t_{\max} = \max_{n=0,\dots,N_t-1} \Delta t_n.$$

In general, (4.29b) is a nonlinear system of $m \cdot s$ equations that has to be solved at every time step, which can cause a considerable amount of computational cost. However, there is a subclass of Runge-Kutta methods that avoid this problem:

Definition 4.13. A Runge-Kutta method is called *explicit* if

$$a_{kl} = 0 \text{ for } l \geq k.$$

The coefficient matrix $\mathbf{A}$ for an explicit Runge-Kutta method is strictly lower triangular. Thus, $\mathbf{k}_k$ depends only on $\mathbf{k}_l$ with $l < k$, so the $\mathbf{k}_k$ can be calculated successively. We use a simplified Butcher tableau for explicit methods omitting the zeros on and above the diagonal of $\mathbf{A}$.

Example 4.14. The simplest explicit Runge-Kutta method is the forward Euler method

$$\mathbf{p}^{n+1} = \mathbf{p}^n + \Delta t_n L(t^n, \mathbf{p}^n) \tag{4.31}$$

with the following Butcher array.

$$\begin{array}{c|c} 0 & \\ \hline & 1 \end{array}$$

Note that this method approximates the temporal integral by evaluating at t^n and multiplying by Δt. This is exactly what we did to approximate the integral in the flux and source term in Section 4.1. The methods regarded there are thus TVD if the Euler method is used for time stepping.

Example 4.15. Heun's method [39]

$$\begin{array}{c|cc} 0 & & \\ 1 & 1 & \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array}$$

is a two-stage Runge-Kutta method.

Example 4.16. The classical Runge-Kutta method [50, 68]

$$\begin{array}{c|cccc} 0 & & & & \\ \frac{1}{2} & \frac{1}{2} & & & \\ \frac{1}{2} & 0 & \frac{1}{2} & & \\ 1 & 0 & 0 & 1 & \\ \hline & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6} \end{array}$$

has four stages.

In the following, we set some additional restrictions to the coefficients of the Runge-Kutta scheme.

Lemma 4.17. *An explicit Runge-Kutta scheme is consistent, i.e.*

$$\lim_{\Delta t_{\max}\to 0} \max_n \frac{\mathrm{le}_n}{\Delta t_n} = 0,$$

if and only if

$$\sum_{k=0}^{s-1} b_k = 1. \tag{4.32}$$

Proof. By the continuity of L we have $\mathbf{k}_k \to L(t^n, \mathbf{p}(t^n))$ as $\Delta t_n \to 0$. Furthermore,

$$\mathbf{p}(t^{n+1}) - \mathbf{p}(t^n) = \Delta t_n \mathrm{d}_t \mathbf{p}(t^n) + o(\Delta t_n) = \Delta t_n L(t^n, \mathbf{p}(t^n)) + o(\Delta t_n)$$

by Taylor expansion and (4.27). Thus

$$\begin{aligned} \mathrm{le}_n &= |\mathbf{p}(t^{n+1}) - \mathbf{p}(t^n) - \Delta t_n \sum_{k=0}^{s-1} b_k \mathbf{k}_k| \\ &\overset{\Delta t_n \to 0}{\longrightarrow} |\Delta t_n \left(1 - \sum_{k=0}^{s-1} b_k\right) L(t^n, \mathbf{p}(t^n)) + o(\Delta t_n)|. \end{aligned}$$

□

Consistency implies convergence for Runge-Kutta methods (see e.g. [51]). Thus, we can be sure that the scheme converges as long as we choose $\mathbf{b}$ according to (4.32).

We only consider explicit methods from now on. We further demand that

$$c_k = \sum_{l=0}^{s-1} a_{kl}. \tag{4.33}$$

Note that this implies $c_0 = 0$ and thus $\mathbf{k}_0 = L(t^n, \mathbf{p}^n)$ for explicit methods. This condition is not necessary for lower order methods but simplifies the derivation of higher order methods (see e.g. [33, 51]).

Under these conditions, we can write (4.29) in a slightly different form

$$\mathbf{p}^{(k)} = \mathbf{p}^n + \Delta t_n \sum_{l=0}^{k-1} d_{kl} L(t^n + c_l \Delta t_n, \mathbf{p}^{(l)}),\ k = 1, \ldots, s, \tag{4.34a}$$

$$\mathbf{p}^{(0)} = \mathbf{p}^n,\ \mathbf{p}^{(s)} = \mathbf{p}^{n+1} \tag{4.34b}$$

with $d_{kl} = a_{kl}$ for $k = 1, \ldots, s-1$ and $d_{sl} = b_l$. The $\mathbf{p}^{(k)}$ are called the stages of the Runge-Kutta method and the $\mathbf{k}_k = L(t^n + c_k \Delta t_n, \mathbf{p}^{(k)})$ are sometimes called the stage derivatives.

If we choose arbitrary $\alpha_{kl} \geq 0$, $0 \leq l < k \leq s$ with $\sum\limits_{l=0}^{k-1} \alpha_{kl} = 1$, we get from (4.34a) (see [76])

$$\begin{aligned}
\mathbf{p}^{(k)} &= \sum_{l=0}^{k-1} \alpha_{kl} \mathbf{p}^{(0)} + \Delta t_n \sum_{l=0}^{k-1} d_{kl} L(t^n + c_l \Delta t_n, \mathbf{p}^{(l)}) \\
&= \alpha_{k0} \mathbf{p}^{(0)} + \sum_{l=1}^{k-1} \alpha_{kl} \left(\mathbf{p}^{(l)} - \Delta t_n \sum_{q=0}^{l-1} d_{lq} L(t^n + c_q \Delta t_n, \mathbf{p}^{(q)}) \right) \\
&\qquad\qquad + \Delta t_n \sum_{l=0}^{k-1} d_{kl} L(t^n + c_l \Delta t_n, \mathbf{p}^{(l)}) \\
&= \sum_{l=0}^{k-1} \left(\alpha_{kl} \mathbf{p}^{(l)} + \Delta t_n \left(d_{kl} - \sum_{z=l+1}^{k-1} d_{zl} \alpha_{kz} \right) L(t^n + c_l \Delta t_n, \mathbf{p}^{(l)}) \right).
\end{aligned}$$

If we define

$$\beta_{kl} = d_{kl} - \sum_{z=l+1}^{k-1} d_{zl} \alpha_{kz}, \tag{4.35}$$

we can rewrite (4.34a) equivalently as

$$\mathbf{p}^{(k)} = \sum_{l=0}^{k-1} \left(\alpha_{kl} \mathbf{p}^{(l)} + \Delta t_n \beta_{kl} L(t^n + c_l \Delta t_n, \mathbf{p}^{(l)}) \right). \tag{4.36}$$

The transformation of (4.34a) to (4.36) is not unique and depends on the choice of the α_{kl}. However, a method of the form (4.36) defines a unique Runge-Kutta method of the form (4.34a) by

$$d_{kl} = \beta_{kl} + \sum_{z=l+1}^{k-1} d_{zl}\alpha_{kz}. \tag{4.37}$$

This is just a rearrangement of (4.35). Note that the formula for d_{kl} depends only on d_{zl} with $z < k$. By (4.33), we get from (4.37)

$$c_k = \sum_{l=0}^{k-1} \left(\alpha_{kl}c_l + \beta_{kl}\right). \tag{4.38}$$

In the following we only regard methods of the form (4.36) knowing that we can always obtain a unique Runge-Kutta method of the form (4.34a). We assume that the numerical scheme using the explicit Euler forward method (4.31) is TVD under the time step restriction

$$\Delta t \le \Delta t_{\text{Euler}}. \tag{4.39}$$

We would now like to know whether the scheme is still TVD if higher order Runge-Kutta methods are used, possibly under a different time-step restriction. It turns out that this is not true for general Runge-Kutta methods. In the classical form of the Runge-Kutta methods, the coefficients d_{kl} (and thus the c_k) are positive. We also required the α_{kl} to be positive but the β_{kl} can be negative. If $\beta_{kl} \ge 0$ and $\alpha_{kl} > 0$ for all $0 \le l < k \le s$, the scheme is indeed TVD:

Lemma 4.18 ([76])**.** *If the forward Euler method is TVD under the time step restriction* (4.39) *and* $\beta_{kl} \ge 0$*, then the method* (4.36) *is TVD under the time step restriction*

$$\Delta t \le C\Delta t_{Euler} \quad \textit{with} \quad C = \min_{k,l} \frac{\alpha_{kl}}{\beta_{kl}}. \tag{4.40}$$

Here, we define $\frac{\alpha_{kl}}{\beta_{kl}} = \infty$ *if* $\beta_{kl} = 0$*. The constant* C *is called the* CFL coefficient*.*

Proof. ([75]) If (4.40) is satisfied, the Euler method is TVD with time step size $\frac{\beta_{kl}}{\alpha_{kl}}\Delta t$ for all k, l. Thus,

$$TV(\mathbf{p}^{(l)} + \frac{\beta_{kl}}{\alpha_{kl}}\Delta t L(t + c_l\Delta t, \mathbf{p}^{(l)})) \le TV(\mathbf{p}^{(l)}). \tag{4.41}$$

We now use induction to prove

$$TV(\mathbf{p}^{(l)}) \le TV(\mathbf{p}^n) \tag{4.42}$$

for $l = 0, \dots, s$. Obviously, (4.42) is true for $l = 0$. If (4.42) is true for $l \le k-1$, we get

$$\begin{aligned} TV(\mathbf{p}^{(k)}) &= TV\left(\sum_{l=0}^{k-1}\left(\alpha_{kl}\mathbf{p}^{(l)} + \beta_{kl}\Delta t L(t + c_l\Delta t, \mathbf{p}^{(l)})\right)\right) \\ &\le \sum_{l=0}^{k-1}\alpha_{kl}TV\left(\mathbf{p}^{(l)} + \frac{\beta_{kl}}{\alpha_{kl}}\Delta t L(t + c_l\Delta t, \mathbf{p}^{(l)})\right) \qquad \le \sum_{l=0}^{k-1}\alpha_{kl}TV(\mathbf{p}^{(l)}) \\ &\le \sum_{l=0}^{k-1}\alpha_{kl}TV(\mathbf{p}^n) = TV(\mathbf{p}^n). \qquad \square \end{aligned}$$

If negative β_{kl} occur, some more work has to be done. We consider an adjoint problem to (4.27)

$$\mathrm{d}_t \mathbf{p} = \tilde{L}(t, \mathbf{p}) \tag{4.43}$$

where we assume that the Euler backward time discretization

$$\mathbf{p}^{n+1} = \mathbf{p}^n - \Delta t_n \tilde{L}(t^n, \mathbf{p}^n) \tag{4.44}$$

is TVD. In the context of hyperbolic conservation laws, where (4.27) comes from a spatial discretization of a PDE

$$\partial_t \mathbf{p} + \partial_x \mathbf{f}(\mathbf{p}) = \mathbf{0}, \tag{4.45}$$

the adjoint problem corresponds to a spatial discretization of the backward-in-time version of the conservation law

$$\partial_t \mathbf{p} - \partial_x \mathbf{f}(\mathbf{p}) = \mathbf{0}. \tag{4.46}$$

The spatial discretization for (4.46) can be done in the same way as for the original conservation law except that the upwind direction changes.

Once we have the adjoint problem, we can use it in the Runge Kutta scheme whenever β_{kl} is negative.

Lemma 4.19 ([76])**.** *If the forward Euler method* (4.31) *and the backward Euler method* (4.44) *are both TVD, then the Runge-Kutta method* (4.36)*, with $\beta_{kl}L$ replaced by $\beta_{kl}\tilde{L}$ whenever β_{kl} is negative, is TVD with CFL coefficient*

$$C = \min_{k,l} \frac{\alpha_{kl}}{|\beta_{kl}|}. \tag{4.47}$$

Methods with negative values of β are less common in applications because of the need to deal with $\tilde{L}$, which is extra work and causes extra computational and storage cost [75]. Unfortunately, methods with order greater than 4 must have negative β (see [46, 47, 69]). Thus, multistep multistage Runge-Kutta methods or other modifications are investigated (see e.g. [11, 17, 46]). However, within this thesis, we stick to the singlestep Runge-Kutta methods regarded above with positive coefficients. The optimal (with respect to the CFL coefficient and the positivity of the β_{kl}) second and third order schemes with two and three stages, respectively, are given in [30]:

Lemma 4.20 ([30])**.** *The optimal second order two-stage TVD-Runge-Kutta method is Heun's method* (4.15) *with a CFL coefficient C of* 1*. The optimal third order three-stage TVD-Runge-Kutta method has the Butcher array*

$$\begin{array}{c|ccc} 0 & & & \\ 1 & 1 & & \\ \frac{1}{2} & \frac{1}{4} & \frac{1}{4} & \\ \hline & \frac{1}{6} & \frac{1}{6} & \frac{2}{3} \end{array} \tag{4.48}$$

and also a CFL coefficient of 1*.*

Note that while the time steps do not have to be shortened with these methods, the computational cost is still doubled and tripled for the second order and third order TVD method, respectively, in comparison to the Euler method.

An order four four-stage method has to have at least one negative β_{kl}. If we allow more stages, fourth-order methods with positive β_{kl} can be found, e.g. a five-stage method with $C = 1.508$ (see [29, 79]).

4.3 Additional considerations for entropy closures

If the M_N entropy closure (see Section 2.3.2) is used to close the moment system for a kinetic equation, the flux in the hyperbolic balance law is defined via the solution of the dual problem (2.13). This has the advantage that we can numerically enforce the invariance of the set $\mathcal{R}_{\mathbf{m}}$ and ensure that the solution will always be realizable. On the other hand, a convex optimization problem has to be solved in every grid cell and time step, which is not only computationally intensive but can be challenging even if we leave the performance aspect apart.

To illustrate some of the additional issues that may occur when using an entropy closure, we regard the kinetic scheme introduced in [1] for the solution of the M_N model to the one-dimensional kinetic equation

$$\partial_t p + v\partial_x p + \sigma_t(x)p = \frac{\sigma_s(x)}{2}\langle p\rangle \tag{4.49}$$

on the bounded interval $\Omega = [x_L, x_R]$ with non-negative coefficient functions $\sigma_s(x)$, $\sigma_t(x)$. Again, angle brackets denote integration over $V = [-1, 1]$.

The M_N model using the Maxwell-Boltzmann entropy (2.11) and the first $N + 1$ Legendre polynomials then is

$$\partial_t \mathbf{p} + \partial_x \mathbf{f}(\mathbf{p}) + \sigma_t \mathbf{p} = \sigma_s \mathbf{Q}\mathbf{p} \tag{4.50}$$

with $\mathbf{Q} \in \mathbb{R}^{(N+1)\times(N+1)}$ given by $Q_{00} = 1$, $Q_{ij} = 0$ else and flux $\mathbf{f} = \langle v\mathbf{m}G_{\hat{\boldsymbol{\alpha}}}\rangle$ (see Section 2.3.2).

To discretize (4.50) we choose an equidistant grid with N_x cells and width $\Delta x = (x_R - x_L)/N_x$. We further choose a uniform time step Δt. Let $t^n = n\Delta t$ and x_i be the center of the i-th cell $T_i = [x_L + i\Delta x, x_L + (i + 1)\Delta x]$. Note that the spatial cells with indices $i \in \{-2, -1, N_x, N_x + 1\}$ are ghost cells (see Section 4.1.6). We use a semidiscrete scheme (see (4.26))

$$\mathrm{d}_t \mathbf{p}_i + \frac{\mathbf{g}_{i+1/2} - \mathbf{g}_{i-1/2}}{\Delta x} + \sigma_t \mathbf{p}_i = \sigma_s \mathbf{Q}\mathbf{p}_i, \quad i \in \{0, \dots, N_x - 1\}, \tag{4.51}$$

with numerical flux $\mathbf{g}_{i+1/2}$ given by

$$\mathbf{g}_{i+1/2} = \left\langle v\mathbf{m}\hat{G}_{i+1/2}\right\rangle, \tag{4.52}$$

where $\hat{G}_{i+1/2}$ is an approximation of the entropy ansatz at the cell boundary, defined by upwinding:

$$\hat{G}_{i+1/2}(t,v) := \begin{cases} \hat{G}_i(t,v) + \frac{\Delta x}{2}\hat{s}_i(t,v), & v > 0, \\ \hat{G}_{i+1}(t,v) - \frac{\Delta x}{2}\hat{s}_{i+1}(t,v), & v < 0. \end{cases}$$

Here, $\hat{G}_i$ is the entropy minimizer from (2.10) associated to $\mathbf{p}_i$:

$$\hat{G}_i(t,v) = G_{\hat{\boldsymbol{\alpha}}(\mathbf{p}_i(t))}(v) \tag{4.53}$$

and $\hat{s}_i$ is an approximation of the spatial derivative of $\hat{G}$ in the cell T_i:

$$\hat{s}_i = \operatorname{minmod}\left\{\theta\frac{\hat{G}_i - \hat{G}_{i-1}}{\Delta x}, \frac{\hat{G}_{i+1} - \hat{G}_{i-1}}{2\Delta x}, \theta\frac{\hat{G}_{i+1} - \hat{G}_i}{\Delta x}\right\} \tag{4.54}$$

where $1 \le \theta \le 2$ is a parameter that controls numerical diffusion (see [65], [85]). See Section 4.1.5 for the definition of the minmod function.

For the temporal discretization, we use Heun's method (4.15). As it is easier to handle here we number the stages slightly different than in (4.34a):

$$\begin{aligned} \mathbf{p}^{(0)} &= \mathbf{p}^n, \\ \mathbf{p}^{(k)} &= \mathbf{p}^{(k-1)} + \Delta t L(\mathbf{p}^{(k-1)}),\ k \in \{1,2\}, \\ \mathbf{p}^{n+1} &= \frac{1}{2}(\mathbf{p}^{(0)} + \mathbf{p}^{(2)}). \end{aligned} \tag{4.55}$$

4.3.1 Preserving Realizability

As mentioned before, we can enforce on the numerical level that the moments in the solution are always realizable. This is done by ensuring that, in each stage of the Runge-Kutta method, the moments are realized by a positive density function if the moments in the prior stage were realizable.

In the numerical scheme, we cannot use the exact solution to the dual problem (2.13) in (4.53) but need to use a numerical approximation. Let $\bar{\boldsymbol{\alpha}}(\mathbf{p})$ be the approximate solution and $\bar{G}_i^{(k)} := G_{\bar{\boldsymbol{\alpha}}(\mathbf{p}_i^{(k)})}$. With the approximate solution, the numerical flux (4.52) becomes

$$\mathbf{g}_{i+1/2} = \left\langle v\mathbf{m}\bar{G}_{i+1/2}\right\rangle, \tag{4.56}$$

where $\bar{G}_{i+1/2}$ is calculated analogously to $\hat{G}_{i+1/2}$, using $\bar{G}_i$ instead of $\hat{G}_i$. As a measure of the relative error introduced by solving the dual problem inexactly, we define

$$\gamma_i^{(k)} := \frac{\bar{G}_i^{(k)}}{\hat{G}_i^{(k)}},\ k \in \{0,1\}, \quad \gamma_{\max} := \max_{\substack{k\in\{0,1\} \\ i\in\{-2,\dots,N_x+1\} \\ v\in[-1,1]}} \gamma_i^{(k)}(v), \tag{4.57}$$

where k represents the stage in Heun's method.

We can now show that the scheme (4.51) always gives realizable moments under a suitable time step restriction depending on the error introduced in solving the dual problem.

Theorem 4.21 ([1]). *Let $\mathbf{p}_i^n \in \mathcal{R}_\mathbf{m}$ for all $i \in \{-2, \dots, N_x + 1\}$. Let the moments in the ghost cells be realizable at each stage of Heun's method and*

$$\gamma_{\max} \frac{\Delta t}{\Delta x} \frac{2+\theta}{2} + \sigma_t \Delta t < 1. \tag{4.58}$$

Then $\mathbf{p}_i^{n+1} \in \mathcal{R}_\mathbf{m}$ for all $i \in \{0, \dots, N_x - 1\}$.

Proof. We only give a sketch of the proof here. If we can show for $k \in \{1, 2\}$:

$$\mathbf{p}_i^{(k-1)} \in \mathcal{R}_\mathbf{m} \text{ for } i \in \{-2, \dots, N_x + 1\} \quad \Rightarrow \quad \mathbf{p}_i^{(k)} \in \mathcal{R}_\mathbf{m} \text{ for } i \in \{0, \dots, N_x - 1\},$$

then the realizability of $\mathbf{p}_i^{n+1}$ follows from (4.55) and the convexity of $\mathcal{R}_\mathbf{m}$. The key point of the proof is to observe that

$$\mathbf{p}_i^{(k)} = \left\langle \mathbf{m} \Phi_i^{(k)} \right\rangle, \, k \in \{1, 2\}, \tag{4.59}$$

with

$$\Phi_i^{(k)} := \hat{G}_i^{(k-1)} - v \frac{\Delta t}{\Delta x} \left(\bar{G}_{i+1/2}^{(k-1)} - \bar{G}_{i-1/2}^{(k-1)} \right) + \Delta t \left(-\sigma_t \hat{G}_i^{(k-1)} + \frac{\sigma_s}{2} \left\langle \hat{G}_i^{(k-1)} \right\rangle \right). \tag{4.60}$$

Therefore, it is left to show that $\Phi_i^{(k)} \geq 0$. It turns out that this is indeed the case under the time step restriction (4.58). See [1] for the details of the proof. $\square$

4.3.2 Solving the dual problem

The function that is supposed to be minimized in the dual problem (2.13) is

$$f(\boldsymbol{\alpha}) := \langle G_{\boldsymbol{\alpha}} \rangle - \boldsymbol{\alpha}^T \mathbf{p}.$$

We further term the gradient of f as

$$\mathbf{g}(\boldsymbol{\alpha}) := \nabla f(\boldsymbol{\alpha}) = \langle \mathbf{m} G_{\boldsymbol{\alpha}} \rangle - \mathbf{p}$$

and its Hessian as

$$\mathbf{H}(\boldsymbol{\alpha}) := \mathbf{H}(f(\boldsymbol{\alpha})) = \left\langle \mathbf{m}\mathbf{m}^T G_{\boldsymbol{\alpha}} \right\rangle.$$

Note that f is strictly convex and $\mathbf{H}$ is symmetric and positive definite. The convexity of f ensures that there is at most one minimum and no inner maximum, so it is sufficient to find a root of $\mathbf{g}$. This is done by Newton's method with an Armijo backtracking line search. We start with an initial estimate $\boldsymbol{\alpha}_0$ of the root. For $t = 0$, $\boldsymbol{\alpha}_0$ is chosen such that $G_{\boldsymbol{\alpha}_0}$ is the isotropic distribution with moment $p^{(0)}$. If the moments are normalized such that $\boldsymbol{\alpha}_0 = 1$, this means $\boldsymbol{\alpha}_0 = (-\log(2), 0, \dots, 0)$. For $t > 0$, the root $\bar{\boldsymbol{\alpha}}$ from the last time step is chosen as initial estimate. Once we have an estimate, the next estimate is calculated as

$$\boldsymbol{\alpha}_{k+1} = \boldsymbol{\alpha}_k + \beta^a \mathbf{d}(\boldsymbol{\alpha}_k) \tag{4.61}$$

with $\mathbf{d}(\boldsymbol{\alpha}_k) = -\mathbf{H}^{-1}(\boldsymbol{\alpha}_k)\mathbf{g}(\boldsymbol{\alpha}_k)$, $\beta \in (0,1)$ and $a \in \mathbb{N}_0$ minimal such that

$$f(\boldsymbol{\alpha}_k + \beta^a \mathbf{d}(\boldsymbol{\alpha}_k)) \leq f(\boldsymbol{\alpha}_k) + \beta^a \xi \mathbf{g}(\boldsymbol{\alpha}_k)^T \mathbf{d}(\boldsymbol{\alpha}_k) \tag{4.62}$$

for $\xi \in (0, 1/2)$. The backtracking condition ensures a sufficient decrease of f in each step (where "sufficient" is defined by the selection of $\xi \in (0, 1/2)$). For a stopping condition, we choose parameters τ, $\epsilon_\gamma > 0$. We stop at $\boldsymbol{\alpha}_k$ if

$$||\mathbf{g}(\boldsymbol{\alpha}_k)|| \leq \tau \quad \text{and} \quad \exp(5\zeta||\mathbf{d}(\boldsymbol{\alpha}_k)||) \leq 1 + \epsilon_\gamma$$

with $\zeta = \max_v ||\mathbf{m}(v)||$ and $|| \cdot ||$ being the Euclidean norm on $\mathbb{R}^{N+1}$. The first condition is natural as we want to find a root of $\mathbf{g}$. The second condition is important for realizability. In order to choose the time step length such that realizability is maintained during the whole scheme, an upper estimate of $\gamma_{\max}$ is needed. If we define $\gamma_k := G_{\boldsymbol{\alpha}_k}/G_{\hat{\boldsymbol{\alpha}}}$, we obtain (using the definition of G and the Cauchy-Schwarz inequality)

$$\max_{v \in [-1,1]} \gamma_k(v) = \max_{v \in [-1,1]} \exp((\boldsymbol{\alpha}_k - \hat{\boldsymbol{\alpha}})^T \mathbf{m}(v)) \leq \exp(\zeta||\boldsymbol{\alpha}_k - \hat{\boldsymbol{\alpha}}||).$$

As the exact minimizer $\hat{\boldsymbol{\alpha}}$ is unknown, we further approximate $||\boldsymbol{\alpha}_k - \hat{\boldsymbol{\alpha}}||$ by $||\mathbf{d}(\boldsymbol{\alpha}_k)||$. Asymptotically, this is a good estimate as Newton's method converges quadratically. As it is not a rigorous estimate though and γ_k is not an upper bound on $\gamma_{\max}$, an additional factor of 5 is inserted into the stopping condition to increase the probability that $\gamma_{\max}$ stays below this bound. In [1], the authors reported that with this approach they were able to use a time step length of about 90 % of the optimal time step length, i.e. the time step length which corresponds to solving the dual problem exactly and is calculated by setting $\gamma_{\max} = 1$ in (4.57), in typical applications.

4.3.3 Difficulties near the realizable boundary

The definition of the objective function, its gradient and its Hessian include integration over V. In general, these integrals cannot be evaluated exactly but a quadrature has to be used. This causes several problems that we will investigate in the following.

Let $\mathcal{Q}$ denote a quadrature on V with a number of $n_{\mathcal{Q}}$ nodes v_i and weights w_i. We define the $\mathcal{Q}$-realizable set as

$$\mathcal{R}_{\mathbf{m}}^{\mathcal{Q}} = \left\{ \mathbf{p} \;\middle|\; \mathbf{p} = \sum_{i=1}^{n_{\mathcal{Q}}} w_i \mathbf{m}(v_i) f_i,\ f_i > 0 \right\}. \tag{4.63}$$

$\mathcal{R}_{\mathbf{m}}^{\mathcal{Q}}$ is a strict polytopic subset of $\mathcal{R}_{\mathbf{m}}$ and like $\mathcal{R}_{\mathbf{m}}$ it is a convex open cone. Furthermore, the set $\mathcal{R}_{\mathbf{m}}^{\mathcal{Q}}|_{p^{(0)}=1}$ is the interior of the convex hull of the points $\mathbf{m}(v_i)$, $i = 1, \ldots, n_{\mathcal{Q}}$ [2].

Although the dual problem is solvable for each $\mathbf{p} \in \mathcal{R}_{\mathbf{m}}$, the use of a quadrature means that the dual problem will be solvable only if $\mathbf{p} \in \mathcal{R}_{\mathbf{m}}^{\mathcal{Q}}$. Furthermore, in the Newton scheme the matrix $\mathbf{H}$ needs to be inverted. This is always possible for the exact matrix $\mathbf{H}$, as it is positive definite, but with the quadrature an approximation $\mathbf{H}_{\mathcal{Q}}$ is used. This is especially a problem near the boundary of the realizability domain. On the boundary $\partial\mathcal{R}_{\mathbf{m}}$, the moments are uniquely represented by atomic densities, i.e. linear combinations of delta distributions (see Section 2.2). Near $\partial\mathcal{R}_{\mathbf{m}}$, the representing distributions are often "almost" linear combinations of delta distributions, i.e. they have a small support and vary over several orders of magnitude. Thus, near $\partial\mathcal{R}_{\mathbf{m}}$, many quadrature points may be needed to resolve the structure of $G_{\hat{\boldsymbol{\alpha}}(\mathbf{p})}$ and get a sufficient approximation $\mathbf{H}_{\mathcal{Q}}(\hat{\boldsymbol{\alpha}}(\mathbf{p}))$. With too few quadrature points, $\mathbf{H}_{\mathcal{Q}}(\hat{\boldsymbol{\alpha}}(\mathbf{p}))$ may be singular. Even if $\mathbf{H}_{\mathcal{Q}}(\hat{\boldsymbol{\alpha}}(\mathbf{p}))$ is not singular, Newton's method may not converge. To ensure global convergence of the numerical scheme, a quadrature with a great number of quadrature points has to be used although for most $\mathbf{p}$ a quadrature with relatively few points would be sufficient.

To avoid excessive computational cost due to an overprecise quadrature, the authors in [1] propose an adaptive quadrature approach where the number of quadrature points is adapted to the particular $G_{\boldsymbol{\alpha}}$. This can be implemented as follows: We start with a Gauss-Legendre quadrature with $n_{\mathcal{Q}} = N + 5$ quadrature points. To test the accuracy of the quadrature, the $(2n_{\mathcal{Q}} + 1)$-Gauss-Kronrod quadrature is determined and the results of the two quadratures are compared. If the difference is below a tolerance parameter $\tau_{\mathcal{Q}}$, the quadrature is accepted. Otherwise, $n_{\mathcal{Q}}$ is incremented by one and the test is repeated until the difference is below the tolerance or until $n_{\mathcal{Q}}$ reaches a specified upper bound $n_{\mathcal{Q}}^{\mathrm{MAX}}$.

In some cases, often near the boundary $\partial\mathcal{R}_{\mathbf{m}}$, $\mathbf{H}$ is that ill-conditioned that the Newton scheme does not converge for any reasonably fine quadrature. In these cases, a regularization procedure can be used to improve the condition while altering the moments $\mathbf{p}$ as few as possible [1]. For the regularization, we choose $r \in (0, 1)$ as small as possible and define the regularized moment

$$\mathbf{q}(r) = (1 - r)\mathbf{p} + r\mathbf{Q}\mathbf{p}. \tag{4.64}$$

The regularization exploits the convexity of $\mathcal{R}_{\mathbf{m}}$ to move the moment vector slightly away from $\partial\mathcal{R}_{\mathbf{m}}$ and towards the moment vector associated with the isotropic distribution $p(t, x, v) = p^{(0)}(t, x)/2$. Note that $p^{(0)}$ is not changed by this regularization.

To implement the regularization, we set a soft upper bound $n_{\mathcal{Q}}^{\max}$ to the number of quadrature points. We further choose a small $r_0 > 0$ and create a sequence $r_{l+1} = \min(2r_l, r_{\max})$ where $r_{\max}$ is a prescribed upper bound for r. If the soft upper bound $n_{\mathcal{Q}}^{\max}$ is encountered during the adaptive quadrature, the problem is considered too difficult to solve. The optimization procedure is then restarted with $\mathbf{q}(r_0)$ instead of $\mathbf{p}$. If the number of quadrature points exceeds $n_{\mathcal{Q}}^{\max}$ again, the optimization is tried with $\mathbf{q}(r_1)$ and so forth, until the optimization finishes using less than $n_{\mathcal{Q}}^{\max}$ quadrature points or until $r_{\max}$ is reached. If $r_{\max}$ is reached, the optimization is carried out with $\mathbf{q}(r_{\max})$, no matter how many quadrature points it takes (up to the hard upper bound $n_{\mathcal{Q}}^{\mathrm{MAX}}$).

While the adaptive quadrature takes care of a sufficient approximation and avoids excessive computational cost, it can cause some other numerical problems. For example, the $\mathcal{Q}$-realizable set changes with every change of the quadrature. Thus, an iterate in the Newton scheme that is realizable in the old quadrature may become unrealizable in the new quadrature, forcing the use of techniques like regularization that affect accuracy. To avoid these problems, an alternative approach was proposed in [2] that uses a fixed quadrature but an adaptive change of basis. Here, in each iteration of the Newton scheme, the basis $\mathbf{m}$ is changed such that the Hessian at the current iterate becomes the identity matrix. This keeps the condition of the Hessian under control and avoids some complexity and computational cost. It is important to choose a quadrature such that $\mathcal{R}_{\mathbf{m}} \backslash \mathcal{R}_{\mathbf{m}}^{Q}$ is small, as only moments from $\mathcal{R}_{\mathbf{m}}^{Q}$ will be realizable with the fixed quadrature. For this reason, a Curtis-Clenshaw quadrature is used in [2] instead of a Gauss-Legendre quadrature. The authors in [2] note that this approach reduces regularization considerably and thus increases accuracy and convergence rate compared to the adaptive quadrature approach.

5 Implementation

The major goal of this thesis was to create an efficient solver for P_N moment models. For that purpose, finite volume schemes for hyperbolic systems of PDEs were implemented in the software framework DUNE. Analytical flux as well as boundary and initial conditions for the P_N models were then calculated manually from the kinetic equation and given to the solver for hyperbolic systems.

In this section, we will regard some aspects of the implementation. A full working example can be found in Appendix A.

5.1 DUNE

DUNE (Distributed and Unified Numerics Environment, [5, 6, 24]) is “a modular toolbox for solving partial differential equations (PDEs) with grid-based methods” [24]. The modules are written in C++ and can be downloaded separately. However, there are a number of core modules that most of the modules depend on. These core modules (dune-common, dune-geometry, dune-grid, dune-istl and dune-localfunctions) provide the basic functionality for solving PDEs numerically. This includes infrastructure for debugging and exception handling, classes for linear algebra (dense and sparse container classes and corresponding solvers) and the interface for grids together with some grid implementations.

A `Grid` (here and in the following we write C++ quantities in typewriter font) in DUNE is rather a collection of several grids in the sense of Section 4.1.1. For example, we could do a series of grid refinements and store the unrefined grid and all subsequent refined grids together with their hierarchy in a `Grid`. What we usually refer to as a grid in this thesis is a `GridView` in DUNE. `Grid`s and `GridView`s in DUNE consist of a set of *entities* which are abstractions for convex polytopes (see Section 4.1.1). The entities can be divided in equivalence classes according to their shape and are represented by one representative of this class, the *reference element* [6]. For instance, the reference element for the equivalence class of r-dimensional cube entities in DUNE is the unit cube in r dimensions.

The entities are positioned in the grid domain by their `Geometry` which essentially represents an injective map $\Phi : \mathbb{R}^r \to \mathbb{R}^d$ from the reference element in $\mathbb{R}^r$ to the grid domain in $\mathbb{R}^d$. Thus, there are two different sets of coordinates for each `Entity`: *local coordinates*, i.e. the coordinates on the reference element, can be mapped to *global coordinates* which are the coordinates in the grid domain. The map Φ is called the *local-to-global map.*

Entities in the grid can also be classified by their *codimension*, which is the difference between grid dimension and dimension of the reference element, i.e. $d - r$ in the notation above. Entities of codimension 0 are what we are usually referring to as grid cells in this thesis, whereas the interfaces S_{ij} of the grid (see Section 4.1.1) are entities of codimension 1.

We can access entities in a `GridView` by `Iterator`s which are essentially pointers to an `Entity` that can be incremented to point to the next `Entity`. There are several iterators, for example allowing for iteration over all entities of a given codimension or a given shape. Each `Entity` of codimension 0 (representing a grid cell T_i) provides iterators to walk over all of its `Intersection`s S_{ij}. On the other hand, each `Intersection` S_{ij} provides the `inside` method to get the `Entity` corresponding to T_i and the `outside` method to get the `Entity` corresponding to T_j. If S_{ij} is on the boundary of the domain, T_j may not exists (if there are no ghost cells) and the `outside` method will throw an error. To check whether an `Intersection` is on the boundary and whether it has an outside `Entity`, the `boundary` and `neighbor` bool method can be used, respectively.

In the following, the term entity will always describe an entity of codimension 0 if not noted otherwise.

5.2 Implementation in dune-stuff

The DUNE module dune-stuff [62] provides extensions to the core modules that are mostly aimed at enhancing usability. For example, it improves handling of configuration files and contains the `fromString` and `toString` methods to convert vector and matrix classes to a string representation and vice versa.

Often it is much easier to implement a method, e.g. a numerical quadrature, on the reference element where we exactly know the shape than for an arbitrary entity in the grid domain. To take advantage of this, dune-stuff has the concept of a localizable function. Consider a function f on the grid domain. Let Φ^{T_i} denote the local-to-global map on the grid cell T_i. If we want to evaluate the function at a point $\mathbf{x} \in T_i$, we can instead also evaluate $f \circ \Phi^{T_i}$ at the local point $\mathbf{y} = (\Phi^{T_i})^{-1}\mathbf{x}$ in the reference element. To calculate the integral of f over T_i we can use the transformation theorem for integrals (the `Geometry` class provides the necessary factor in the integral) and use a quadrature on the reference element. The function $f \circ \Phi^{T_i}$ is called the *local function* of f with respect to T_i. Functions that can be evaluated locally on an entity and provide a local function are represented in dune-stuff by the `LocalizableFunctionInterface`.

To be able to calculate several quantities in a single walk over the grid, e.g. calculate the integral over the domain for several functions at once instead of iterating over the grid for every single function, dune-stuff provides the `Walker` class. Via the `add` method, several functors can be added to the walker before using the `walk` method to apply all functors in a single walk over the grid. The `add` method takes an additional argument that specifies on which entities the functor should be applied. Thus, functors can also be classified by codimension. A functor that is only applied on entities of codimension 0 is a functor of codimension 0, whereas a functor that is only applied on intersections is a functor of codimension 1. As an example, consider a finite volume scheme where we want to evaluate the operator L (see Section 4.2). For that purpose, we need to evaluate the numerical flux $\mathbf{g}_{ij}$ at each intersection S_{ij}. As we have seen in Section 4.1.6, we may have to treat the boundary intersections differently as there is no neighbour cell on the outside of the intersection. In addition, there may be source terms that have to be evaluated on each entity. In dune-stuff, this could be realized as

```
1 using namespace Dune::Stuff::Grid;
2 Walker< GV > walker(grid_view);
3 walker.add(flux_operator, new ApplyOn::InnerIntersections< GV >());
4 walker.add(boundary_flux_operator, new ApplyOn::BoundaryIntersections< GV >());
5 walker.add(source_operator, new ApplyOn::AllEntities< GV >());
6 walker.walk();
```

In line 6 the walker walks over all entities of the grid and apply the source operator (represented by a functor of codimension 0). On each entity, the flux operator and the boundary flux operator (functors of codimension 1) are applied on all inner intersections and boundary intersections, respectively.

5.2.1 Periodic boundary conditions

If periodic boundary conditions are used, we would expect the `outside` method to return the periodic neighbor entity on the opposite side of the grid for boundary `Intersections`. To our knowledge, this is not the case for any of the standard grids in DUNE. We thus implemented the class `PeriodicGridView` in `dune/stuff/grid/periodicview.hh`. It takes any `GridView` and replaces the `IntersectionIterator` and `Intersection` classes of the Grid by a `PeriodicIntersectionIterator` and `PeriodicIntersection`. These periodic classes are derived from the respective classes of the original `GridView`. The only difference is that a `PeriodicIntersection` on a periodic boundary will return `true` in both the `boundary` and `neighbor` method and the `outside` method will return the periodically adjacent `Entity`.

```
typedef typename Dune::Stuff::Grid::PeriodicGridView< GridViewType > PeriodicGridViewType;
std::bitset< 3 > periodic_directions(std::string("100"));
const PeriodicGridViewType periodic_grid_view(grid_view, periodic_directions);
/* use periodic_grid_view exactly like any other grid_view */
```

Currently, the `PeriodicGridView` works only for `GridViews` on an axis-parallel cubic domain. The optional `bitset` argument can be used to specify periodic coordinate directions. In the example above, the `periodic_grid_view` is only periodic in the first coordinate direction. Note that this periodicity holds only with respect to intersections by now. Corresponding grid nodes on the periodic boundary do not share the same global index, for example.

5.2.2 Time-dependent functions

On non-periodic boundaries we have to specify boundary conditions that may be time-dependent. There were no time-dependent functions available in dune-stuff so we implemented the `Time-DependentExpression` function in `dune/stuff/functions/expression.hh` built on the already existing `Expression` function, which is a localizable function that can be created from a string.

```
Dune::Stuff::Functions::TimeDependentExpression</*...*/> time_dep_function("x","sin(t*x[0])");
auto function_ptr = time_dep_function.evaluate_at_time(3.0); // sin(3*x[0])
```

The `evaluate_at_time(value)` method replaces all occurrences of t in the string by `value` and returns a pointer to an `Expression` function using the resulting string. Note that currently all occurrences of t are replaced so the variable name ("x" in the example above) must not contain the letter t.

5.3 Implementation in dune-gdt

The DUNE module dune-gdt [61, 70] is a generic discretization toolbox which contains building blocks for discretization methods. Initially, it was mainly focused on methods for linear elliptic problems. In this thesis, support for hyperbolic problems should be added.

5.3.1 Discrete function spaces

For finite element methods the discrete solution is expressed in terms of a set of basis functions associated with the *degrees of freedom* (DoFs) of the ansatz space. The support of these basis functions often includes only a few grid cells. Thus, to evaluate or integrate an element of the ansatz space, on each grid cell we only need to take the basis functions into account that are non-zero on this grid cell. In this context, dune-gdt provides discrete function spaces derived from the `SpaceInterface` that defines the methods

```
BaseFunctionSetType base_function_set(const EntityType& entity) const;
const MapperType& mapper() const;
```

providing a set of local functions (as defined for dune-stuff, see above) corresponding to the non-zero basis functions on each entity and a mapper mapping from the indices of the local basis functions to the global DoFs. To represent an element of such a discrete function space, dune-gdt provides the `DiscreteFunction` class.

In the finite volume setting for a hyperbolic system of m equations, the discrete solution is an element of the space of functions that are constant on each entity, called *finite volume space* in the following. Thus, we can choose $\{\mathbf{e}_1, \dots, \mathbf{e}_m\}$ as the set of local basis functions on each entity, where $\mathbf{e}_j$ is the vector with $e_j = 1$, $e_i = 0$ for $i \neq j$. The value of a `DiscreteFunction` on an entity is given by a vector $\mathbf{c}$ in local coordinates, i.e. to evaluate a `DiscreteFunction` on an entity we have to calculate $c_1\mathbf{e}_1 + \dots + c_m\mathbf{e}_m$. This is rather inefficient for finite volume spaces as it involves many multiplications and additions with ones and zeros. Instead of regarding the finite volume space as a space of vector-valued piecewise constant functions, we can also see it as the product of m spaces of scalar piecewise constant functions. We thus implemented a `ProductSpaceInterface` (in `dune/gdt/spaces/productinterface.hh`) and derived the `FV::DefaultProduct` space from it (see `dune/gdt/spaces/fv/defaultproduct.hh`), implementing the methods

```
template< size_t ii > const FactorSpaceType factor() const;
const FactorMapperType& factor_mapper() const;
```

that give the *ii*-th factor space and a mapper that allows to map from local indices in the factor space to global indices in the product spaces. This way, we only have to calculate $c_j \cdot 1$ for all factor spaces j to evaluate a discrete function. Because this is still less efficient than simply returning the local vector directly, we added a specialization to the `DiscreteFunction` for finite volume spaces. However, the product finite volume space is still useful as it makes accessing a component of the vector-valued function easier. For example, we can use the existing infrastructure for scalar-valued functions to visualize a component of the vector-valued discrete function.

5.3.2 Operators on discrete function spaces

Given a finite volume discrete function space, we need to implement operators on these spaces. We denote the domain of an operator as *source space* and the image as *range space*. The term source here is not to be confused with the source term in the hyperbolic equation. Both source and range space of the operator $L(t, \cdot)$ in a finite volume scheme at a fixed time t are the finite volume space.

We implemented the operator L for different finite volume schemes. The operator is derived from the already existing `OperatorInterface` in dune-gdt and thus has a method

```
void apply(const SourceType& source, RangeType& range) const
```

that takes a function `source` representing $\mathbf{p}^n$ and stores the update $L(\mathbf{p}^n)$ in the function `range`. Usually, both `source` and `range` will be `DiscreteFunctions` on the finite volume space. As an example, in the `AdvectionLaxFriedrichs` operator for the Lax-Friedrichs scheme the apply method takes the following form:

```
template< class SourceType, class RangeType >
void apply(const SourceType& source, RangeType& range, const double time = 0.0) const
{
  auto current_boundary_values = boundary_values_.evaluate_at_time(time);
  AdvectionLaxFriedrichsLocalizable</*...*/> localizable_operator(analytical_flux_,
                                                                  dx_,
                                                                  dt_,
                                                                  source,
                                                                  *current_boundary_values,
                                                                  range,
                                                                  /*...*/);
  localizable_operator.apply();
}
```

Note that we had to add the optional `time` argument to allow for time-dependent operators which is not covered by the `OperatorInterface` yet. The `apply` method creates a localizable operator that takes (in addition to boundary values, `source` and `range`) a representation of the analytical flux $\mathbf{f}$, the current time step length Δt (`dt_`) and a `LocalizableFunction dx_` that gives the width h_i on each grid cell T_i. Variables with an underscore at the end of their name are members of `AdvectionLaxFriedrichs` and created on construction of the operator. The localizable operator is derived from the `SystemAssembler` class, which itself is derived from the `Walker` class in dune-stuff. The `apply` method of the localizable operator is thus implemented as:

```
void apply()
{
  this->add(inner_assembler_, source_, range_,
            new DSG::ApplyOn::InnerIntersections< GridViewType >());
  this->add(inner_assembler_, source_, range_,
            new DSG::ApplyOn::PeriodicIntersections< GridViewType >());
  this->add(boundary_assembler_, source_, range_,
            new DSG::ApplyOn::NonPeriodicBoundaryIntersections< GridViewType >());
  this->assemble();
}
```

Local assemblers that are applied to inner and periodic boundary intersections or to non-periodic boundary intersections are added to the localizable operator. The grid walk is started using the `assemble` method. The local assemblers are created in the constructor of `AdvectionLaxFriedrichsLocalizable` using the analytical flux and time step and grid width information (see `dune/gdt/operators/advection.hh`). Together with the `source_` function representing $\mathbf{p}^n$, they thus have all data to calculate the numerical flux using the formulas from Section 4.1. The `assemble` method redirects to the dune-stuff `Walker` class and ends up calling the following method:

```
template< class EntityRange >
void walk_range(const EntityRange& entity_range)
{
  for (const EntityType& entity : entity_range) {
    // apply codim0 functors
    apply_local(entity);
    // walk the intersections using the entity's intersection iterator
    const auto intersection_it_end = grid_view_.iend(entity);
    for (auto intersection_it = grid_view_.ibegin(entity);
         intersection_it != intersection_it_end;
         ++intersection_it) {
      const auto& intersection = *intersection_it;
      // apply codim1 functors
      if (intersection.neighbor()) {
        const auto neighbor_ptr = intersection.outside();
        apply_local(intersection, entity, *neighbor_ptr);
      } else
        apply_local(intersection, entity, entity);
19      } // walk the intersections
20    } // walk the entities
21  }
```

Usually, the `entity_range` contains all entities in the grid view (except if we use multi-threading, see Section 5.3.4). Because we did not add any functors of codimension 0, line 6 does not have any effect here, but will be used for the source terms (see below). For the finite volume scheme, only the walk over all intersections of the entity (line 9-19) is important. From each intersection, the outside entity, if there is one, is extracted and the `apply_local` method called. This method then calculates the numerical flux $\frac{1}{|T_i|}\mathbf{g}_{ij}$ at that intersection, using the inner and boundary assembler we added to the localizable operator in its `apply` method. The result is added to the value of the discrete function `range_` on that entity, which, if it was zero on each entity before, thus contains $-L(t^n, \mathbf{p}^n)$ after the grid walk.

In addition to the `AdvectionLaxFriedrichs` operator for the Lax-Friedrichs scheme, we implemented `AdvectionGodunov` and `AdvectionGodunovWithReconstruction` operators for the Godunov scheme and Godunov scheme with linear reconstruction and slope limiting in `dune/gdt/operators/advection.hh`. The `AdvectionLaxFriedrichs` operator can also be configured in the constructor to use the local Lax-Friedrichs scheme. The linear reconstruction is done in the constructor of the `AdvectionGodunovWithReconstructionLocalizable` operator in an additional grid walk. The slope limiter (minmod, mc or superbee) can be chosen using the `Dune::GDT::Operators::SlopeLimiters` enum.

For the source terms $\mathbf{h}(t, \mathbf{x}, \mathbf{p})$ in the hyperbolic balance law, we implemented a similar operator `AdvectionSource` that essentially evaluates the source terms on each entity. Here, only a functor of codimension 0 is applied on each entity and there is no walk over the intersections.

5.3.3 Runge-Kutta time stepping

The `RungeKutta` class was implemented in `dune/gdt/timestepper/rungekutta.hh` to do the time stepping. The actual Runge-Kutta scheme is implemented in the member method `apply_RK_scheme`:

```
template< class OperatorType >
void apply_RK_scheme(const OperatorType& op, const double dt, const double factor)
{
  for (size_t k = 0; k < num_stages_; ++k) {
    p_stages_[k].vector() *= 0;
    p_tmp_.vector() = p_n_.vector();
    for (size_t l = 0; l < k; ++l)
      p_tmp_.vector() += p_stages_[l].vector()*(dt*factor*(A_[k][l]));
    op.apply(p_tmp_, p_stages_[k], t_ + dt*c_[k]);
  }
  for (size_t l = 0; l < num_stages_; ++l)
    p_n_.vector() += p_stages_[l].vector()*(factor*dt*b_[l]);
13 }
```

This corresponds to the Runge-Kutta scheme (4.34). The operator `op` is any of the operators mentioned above, e.g. the `AdvectionLaxFriedrichs` operator. The argument `dt` is the time step length and the third argument `factor` can be ignored for now. The discrete function `p_n_` stores the approximative solution $\mathbf{p}^n$ at the current time step t^n (member variable `t_`). The first `for` loop (lines 4-10) calculates the stage derivatives $\mathbf{k}_k$, $k = 0, \dots, s-1$, and stores them in the vector of discrete functions `p_stages_`. In each iteration k of the loop, the stage $\mathbf{p}^{(k)}$ is assembled in `p_tmp_` (lines 6-8) and the operator `op` is applied is to obtain $\mathbf{k}_k$. Note that we do not need to loop over all stages but only the stages $l < k$ in the inner `for` loop (lines 7-8) as we only allow explicit schemes. The second `for` loop (lines 11-12) then gives $\mathbf{p}^{n+1}$.

A time step of the fractional step scheme (see Section 4.1.3) is then easily implemented:

```
double step(const double dt)
{
  apply_RK_scheme(flux_operator_, dt, -1.0);  // delta_t p = L(t,p)
  apply_RK_scheme(source_operator_, dt, 1.0); // delta_t p = h(t,x,p)
  t_ += dt; // increment time
  /* calculate new time step dt_new */
  return dt_new;
}
```

Here, we see why we needed the additional `factor` argument in the method above: The `flux_operator_` we implemented actually is $-L(t^n, \mathbf{p}^n)$ so we need to add the factor -1.0 to get the correct sign. After the fractional steps, a different step length `dt_new` for the next time step could be calculated depending on the result. We always use a fixed time step length in this thesis and thus choose `dt_new` $=$ `dt`.

Based on these methods, the `RungeKutta` time stepper provides the method

```
void solve(const double t_end, const double first_dt, const double save_step_length,
           const bool save_solution,
           const bool visualize_solution, const std::string filename_prefix,
           std::vector< std::pair< double, DiscreteFunctionType > >& solution)
```

that solves a problem (specified in the constructor of `RungeKutta`) up to time `t_end` and saves or visualizes (if `save_solution` or `visualize_solution` is true, respectively) the solution at an interval of `save_step_length`.

5.3.4 Parallelization

To take advantage of multi-core processors, we use the shared memory multi-threading capacity that is already implemented in dune-stuff based on Intel Threading Building Blocks (TBB). It works by splitting the set of entities of the `GridView` in several partitions. Instead of using one iterator (or `EntityRange`) for the whole `GridView`, an `EntityRange` for each partition is created. The `walk_range` method in the `Walker` class (see above) can then be applied separately to each of these `EntityRange`s. This can easily be done in parallel: The input data is not modified during the grid walk and local functions and operators that act on one entity do not write to variables that are associated with another entity. We used the existing implementation and copied the approach to also parallelize the grid walk for the linear reconstruction in the `AdvectionGodunovWithReconstructionLocalizable` operator.

DUNE also supports parallelism using the Message Passing Interface (MPI) for distributed parallel computations, which we did not utilize in this work.

6 Numerical Results

6.1 Convergence tests

To check the implementation of the finite volume schemes, we experimentally determine convergence rates in some test problems. Recall that Theorem 4.9 states that the L^1-error between the approximate solution $\mathbf{p}_h$ from the finite volume scheme and the exact solution $\mathbf{p}$ is $O(h^\kappa)$, where $\kappa = \frac{1}{4}$ in general and $\kappa = \frac{1}{2}$ in one dimension. If Theorem 4.9 held with equality, i.e.

$$||\mathbf{p} - \mathbf{p}_h||_{L^1(\mathbb{R}^d \times \mathbb{R}_+)} = Ch^\kappa, \tag{6.1}$$

we could divide (6.1) for two grid widths $h \neq h'$ to get

$$\frac{||\mathbf{p} - \mathbf{p}_h||_{L^1}}{||\mathbf{p} - \mathbf{p}_{h'}||_{L^1}} = \left(\frac{h}{h'}\right)^\kappa$$

which can be solved for κ. This motivates the following procedure to determine the exponent κ experimentally. We take a series of equidistant grids $(\tau_{h_1}, \tau_{h_2}, \ldots, \tau_{h_k})$ where the grid widths fulfil $h_1 > h_2 > \ldots > h_k$ and solve the same test problem on each of these grids (with a fixed Courant number to determine the time steps t_h^n) to obtain the approximate solutions $(\mathbf{p}_{h_1}, ..., \mathbf{p}_{h_k})$. For each $i = 2, \ldots, k-1$, an estimate for κ, the *experimental order of convergence* (EOC), is then calculated as

$$EOC(h_i, h_{i-1}) := \frac{\log \frac{||\tilde{\mathbf{p}} - \mathbf{p}_{h_i}||_{L^1}}{||\tilde{\mathbf{p}} - \mathbf{p}_{h_{i-1}}||_{L^1}}}{\frac{h_i}{h_{i-1}}}. \tag{6.2}$$

Here, $\tilde{\mathbf{p}}$ is the exact solution discretized in space and time by averaging on the finest grid τ_{h_k} (see (4.4)) at every time point $t_{h_k}^n$. If there is no exact solution available, the solution on the finest grid can be used as reference solution, $\tilde{\mathbf{p}} = \mathbf{p}_{h_k}$.

6.1.1 Scalar advection equation

We start with a simple test case for the scalar advection equation

$$\partial_t p + \mathbf{v} \cdot \nabla p = 0 \quad \text{for } \mathbf{x} \in \mathbb{R}^d, t \in [0, 1], \tag{6.3a}$$

$$p(\mathbf{x}, 0) = p_{0,d}(\mathbf{x}) \quad \text{for } \mathbf{x} \in \mathbb{R}^d \tag{6.3b}$$

with initial values

$$p_{0,1}(x) = \begin{cases} 10^4 \cdot (x-0.2)^2 \cdot (x-0.4)^2 \cdot \mathrm{e}^{0.02-(x-0.2)^2-(x-0.4)^2} & \text{if } 0.2 \leq x < 0.4 \\ 1 & \text{if } 0.6 \leq x < 0.8 \\ 0 & \text{else} \end{cases} \tag{6.4a}$$

for $d = 1$ and

$$p_{0,d}(\mathbf{x}) = \prod_{i=1}^{d} p_{0,1}(x_i) \tag{6.4b}$$

for $d > 1$. The initial values can be seen plotted in Figure 5 ("Exact solution"). They were chosen to contain both a smooth bump and a discontinuous one to make sure that the numerical schemes can handle either. The velocity was chosen as $v = 1$ for $d = 1$ and $\mathbf{v} = (1, 2)^T$ for $d = 2$. The calculations were done on the domain $[0, 1]^d$ with periodic boundary conditions to simulate $\mathbb{R}^d$.

As can be seen in Table 1, the Godunov flux delivers the expected convergence rate in one dimension. On the coarsest grids, the EOC is lower than expected, which may be due to a bad resolution of the initial values and a relative error of about 100 % (the L^1-norm of the exact solution is about 0.3). The Lax-Friedrichs method shows similar behaviour, except that a finer grid is needed to obtain an EOC of 1/2. The local Lax-Friedrichs flux is not shown here as it agrees with the Godunov flux for the scalar advection equation in one dimension.

The higher order slope limiter methods clearly show better convergence in this case. The EOC is around 1, which results in a much lower error on the finest grid. This can also be seen graphically in Figure 5. The Lax-Friedrichs solution shows strong smearing both at the smooth bump and at the shock on coarse grids. Moreover, it introduces some oscillations at the shock, which is known to be a problem with the Lax-Friedrichs flux despite it being TVD (see [12], [82]). The Godunov flux performs better but still smears out the bumps. The slope limiter methods overcome these problems and sharply resolve the shock, but struggle with the smooth bump. Because of the slope limiting, the maximum is flattened out and appears more like a shock with these methods. On the finer grid, the smearing can still be seen for the Lax-Friedrichs and Godunov fluxes but to a much lesser extent. The flattening of the smooth maximum is only visible for the minmod limiter, whereas the superbee and MC limiter are close to the real solution.

We used Euler time stepping here even for the slope limiter methods as the use of higher order TVD time stepping methods (see Section 4.2) did not improve the results noticeably.

The convergence results for the advection equation in two dimensions can be seen in Table 2. Both the Lax-Friedrichs scheme and the Godunov scheme deliver a convergence rate above the expected 1/4 for the finer grids. On the coarse grids, the EOC is again quite low and even becomes negative in the first step, which indicates that these grids are too coarse to obtain a reasonable approximation to the true solution. Because higher order methods were only implemented for linear hyperbolic equations in one spatial dimension, we could not use them here, although it would be desirable as can be seen in Figure 6. Even on the finest grid, both the Lax-Friedrichs scheme and the Godunov scheme show a considerable amount of smearing. The height of the smooth bump is decreased and the sharp discontinuity is smoothed out. The Lax-Friedrichs scheme shows some distortion whereas the solution obtained by the Godunov scheme is more symmetric. However, even in this simple test case, finer grids are clearly needed to obtain an accurate approximation with these methods which is computationally expensive especially in several dimensions.

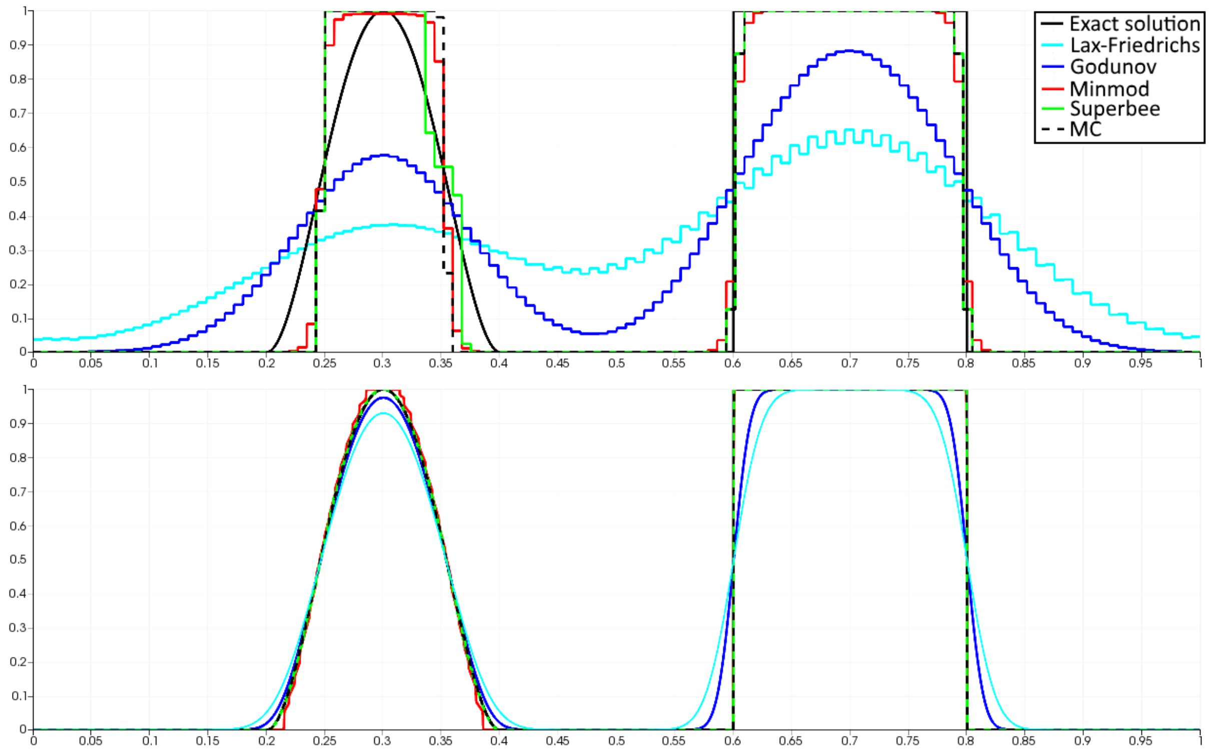
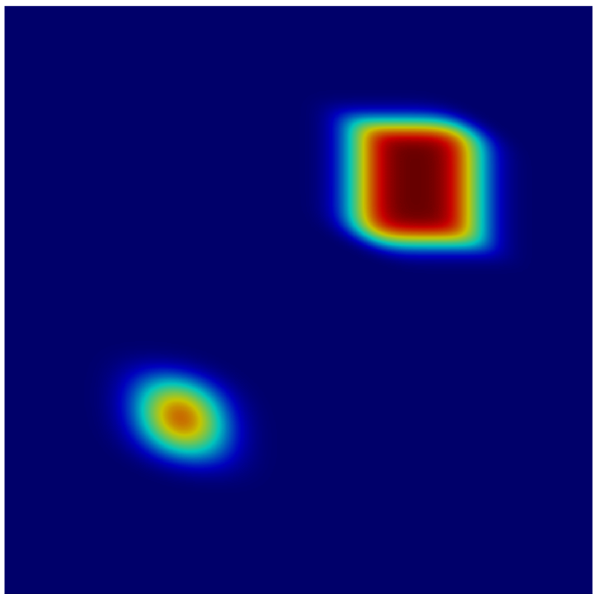


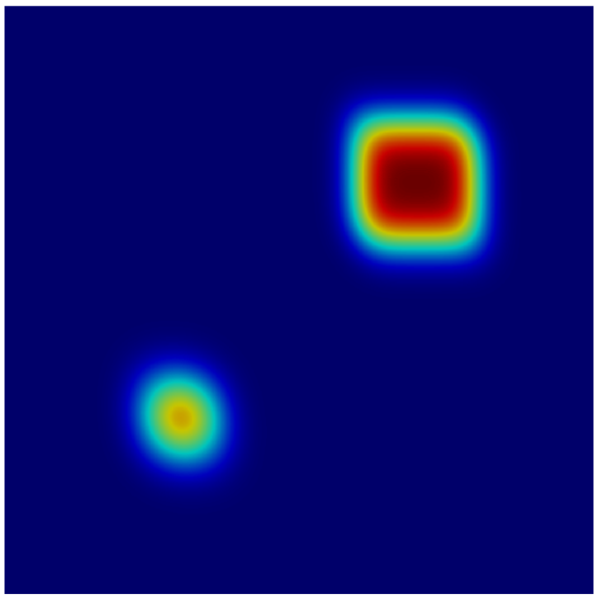
Figure 5: Solutions to the scalar advection equation test case (6.3) **in one dimension at time $t = 1$ on a grid with 128 (top) and 4,096 grid cells (bottom)**. Because of the periodic boundary conditions, the exact solution at time $t = 1$ equals the initial values (6.4) (as the velocity was chosen as 1 and the domain has length 1). On the coarse grid, the Lax-Friedrichs flux and the Godunov flux both show strong smearing. The slope limiter methods do not show this problem but struggle with the smooth maximum instead. On the finer grid, the smearing produced by the Lax-Friedrichs and Godunov flux can still be seen but to a much lesser extent. The slope limiter methods are close to the analytic solution here.

N_x	h	Lax-Friedrichs		Godunov		Minmod		Superbee	
		L^1-error	EOC	L^1-error	EOC	L^1-error	EOC	L^1-error	EOC
8	1.25e-1	3.64e-1	–	3.25e-1	–	3.02e-1	–	2.77e-1	–
16	6.25e-2	3.56e-1	0.03	2.87e-1	0.18	1.82e-1	0.73	1.16e-1	1.26
32	3.13e-2	3.11e-1	0.20	2.18e-1	0.40	7.42e-2	1.29	6.12e-2	0.92
64	1.56e-2	2.48e-1	0.32	1.54e-1	0.50	3.74e-2	0.99	4.17e-2	0.55
128	7.81e-3	1.86e-1	0.42	1.06e-1	0.54	3.50e-2	0.10	2.77e-2	0.59
256	3.91e-3	1.31e-1	0.50	7.06e-2	0.59	2.96e-2	0.24	1.75e-2	0.66
512	1.95e-3	8.91e-2	0.56	4.65e-2	0.60	1.77e-2	0.74	8.03e-3	1.12
1,024	9.77e-4	5.92e-2	0.59	3.06e-2	0.60	9.23e-3	0.94	3.74e-3	1.10
2,048	4.88e-4	3.90e-2	0.60	2.03e-2	0.59	4.53e-3	1.03	1.86e-3	1.01
4,096	2.44e-4	2.57e-2	0.60	1.36e-2	0.58	2.24e-3	1.02	9.85e-4	0.92

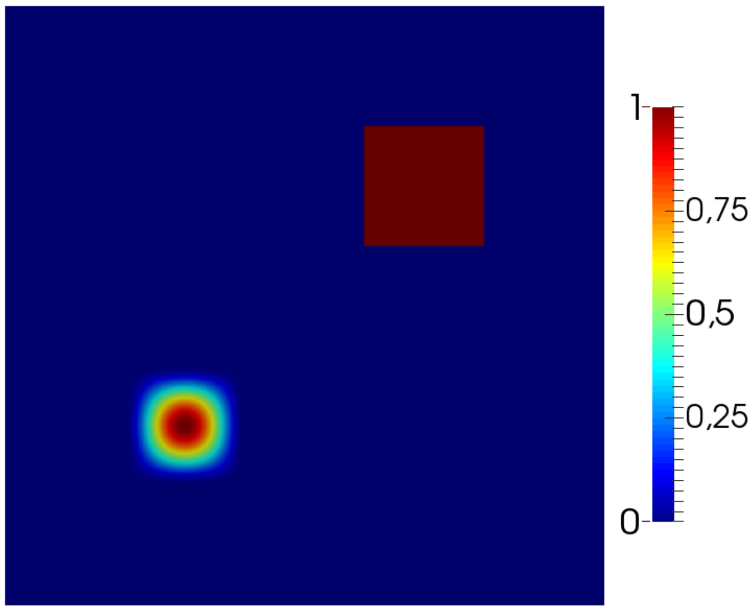
Table 1: Experimental order of convergence for the scalar advection equation in one dimension. The Lax-Friedrichs and Godunov fluxes deliver a convergence order of about 1/2, whereas the higher order methods show a convergence order of about 1 in this test case. Data for the MC-Limiter is not shown as it is similar to the Superbee data. N_x: number of grid cells, h: grid width.



(a) Lax-Friedrichs scheme



(b) Godunov scheme



(c) Exact solution

Figure 6: Solutions to the scalar advection equation test case (6.3) **in two spatial dimensions at time** $t = 1$ **on a grid with 262,144 cells.** Because of the periodic boundary conditions the exact solution at time $t = 1$ equals the initial values (6.4). Both the Lax-Friedrichs scheme and the Godunov scheme show some smearing. The solution obtained by the Godunov scheme is quite symmetric whereas the Lax-Friedrichs solution is a little deformed.

N_x	h	Lax-Friedrichs		Godunov	
		L^1-error	EOC	L^1-error	EOC
64	1.77e-1	6.87e-2	–	6.86e-2	–
256	8.84e-2	7.52e-2	-0.13	7.79e-2	-0.18
1,024	4.42e-2	6.77e-2	0.15	7.44e-2	0.07
4,096	2.21e-2	5.09e-2	0.41	5.79e-2	0.36
16,384	1.10e-2	3.79e-2	0.42	4.36e-2	0.41
65,536	5.52e-3	2.80e-2	0.44	3.22e-2	0.44
262,144	2.76e-3	2.00e-2	0.49	2.29e-2	0.49

Table 2: Experimental order of convergence for the scalar advection equation in two dimensions. Both the Lax-Friedrichs flux and the Godunov flux deliver a convergence order slightly below 1/2. N_x: number of grid cells, h: grid width.

6.1.2 Sod's shock tube

As a second test case, we choose the shock tube test case for the nonlinear Euler equations of gas dynamics introduced by Gary A. Sod in [78]. It models a one-dimensional tube with two chambers that are separated by a membrane. The gas in both chambers is initially at rest and the pressure and density are discontinuous across the membrane. At time $t = 0$, the membrane is removed. For an ideal gas this model results in the Riemann problem

$$\partial_t \mathbf{p} + \partial_x \mathbf{f}(\mathbf{p}) = 0 \qquad \text{for } x \in [0,1], t \in [0, 0.25], \tag{6.5a}$$

$$\mathbf{p}(x,0) = \begin{cases} \mathbf{p}_L & \text{for } x < 0.5, \\ \mathbf{p}_R & \text{for } x > 0.5, \end{cases} \tag{6.5b}$$

$$\mathbf{p}(t,0) = \mathbf{p}_L, \quad \mathbf{p}(t,1) = \mathbf{p}_R \qquad \text{for } t \in [0, 0.25], \tag{6.5c}$$

where the conserved variables are the density ρ, the mass flux ρu and the energy E:

$$\mathbf{p} = \begin{pmatrix} p_1 \\ p_2 \\ p_3 \end{pmatrix} = \begin{pmatrix} \rho \\ \rho u \\ E \end{pmatrix}, \quad \mathbf{f}(\mathbf{p}) = \begin{pmatrix} p_2 \\ \frac{4}{5}\frac{p_2^2}{p_1} + \frac{2}{5}p_3 \\ \frac{7}{5}\frac{p_2 p_3}{p_1} - \frac{1}{5}\frac{p_2^3}{p_1^2} \end{pmatrix}. \tag{6.5d}$$

The initial values were chosen as

$$\mathbf{p}_L = \begin{pmatrix} 1 \\ 0 \\ 2.5 \end{pmatrix}, \quad \mathbf{p}_R = \begin{pmatrix} 0.125 \\ 0 \\ 0.25 \end{pmatrix}. \tag{6.5e}$$

The Riemann problem can be solved semi-analytically, which has been done in [59]. To determine the solution, the root of a transcendental function has to be found which is done numerically. The solution consists of four constant states connected by a rarefaction wave and two shocks (see Figure 7 for a plot of the density ρ at time $t = 0.25$). We use this solution (with Γ in [59] chosen as 7/5) as a reference solution. Note that none of the waves in the solution have reached

N_x	h	Lax-Friedrichs		Local L.-F.		Godunov	
		L^1-error	EOC	L^1-error	EOC	L^1-error	EOC
8	1.25e-1	3.47e-2	–	2.43e-2	–	1.86e-2	–
16	6.25e-2	2.07e-2	0.74	1.42e-2	0.78	1.03e-2	0.86
32	3.13e-2	1.27e-2	0.71	8.39e-3	0.76	5.86e-3	0.81
64	1.56e-2	7.80e-3	0.70	5.04e-3	0.74	3.43e-3	0.77
128	7.81e-3	4.79e-3	0.70	3.03e-3	0.73	2.03e-3	0.76
256	3.91e-3	2.93e-3	0.71	1.81e-3	0.74	1.21e-3	0.75
512	1.95e-3	1.77e-3	0.73	1.08e-3	0.75	7.14e-4	0.76
1,024	9.77e-4	1.05e-3	0.75	6.36e-4	0.76	4.18e-4	0.77
2,048	4.88e-4	6.22e-4	0.76	3.79e-4	0.75	2.44e-4	0.78
4,096	2.44e-4	3.64e-4	0.77	2.30e-4	0.72	1.41e-4	0.79

Table 3: Experimental order of convergence for the shock tube test case. All fluxes deliver a similar rate of convergence of about 3/4. Still, as can be seen by the error on the finest grid, the Godunov scheme performs best followed by the local Lax-Friedrichs scheme. N_x: number of grid cells, h: grid width, L.-F.: Lax-Friedrichs.

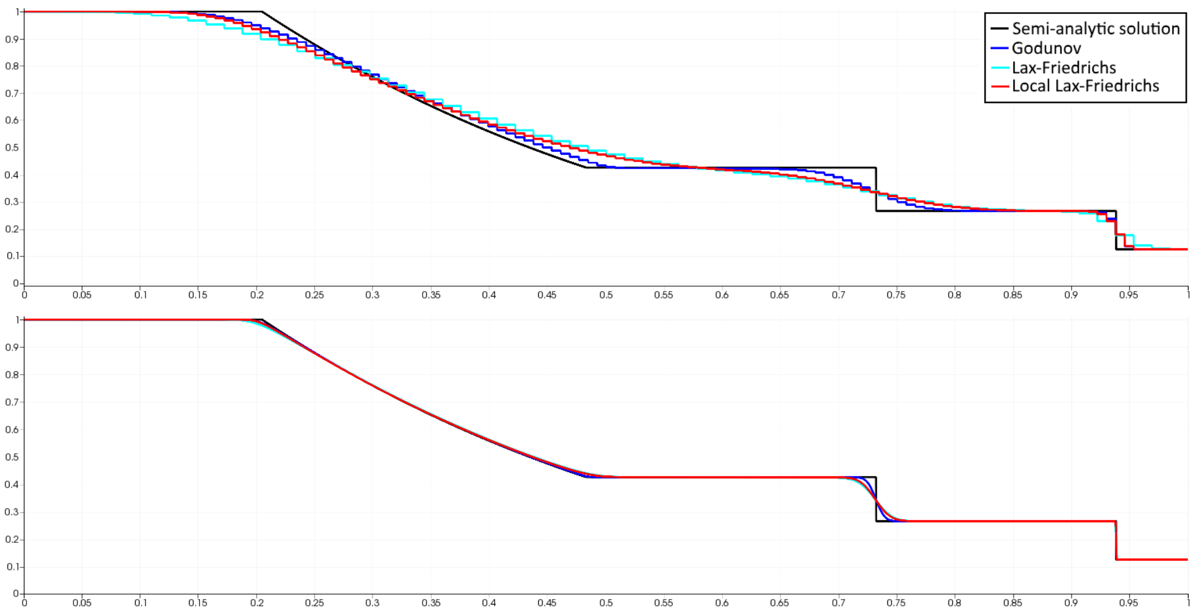


Figure 7: Solutions to the shocktube test case at time $t = 0.25$ on a grid with 128 (top) and 4,096 cells (bottom). Plotted is only the first component of the solution, the density ρ. On the coarse grid, all schemes show some smearing. This is strongest for the Lax-Friedrichs and local Lax-Friedrichs schemes but can also clearly be seen with the Godunov scheme. On the fine grid, all schemes are close to the semi-analytic solution except at one of the shocks that is still smoothed out.

the boundary at time $t = 0.25$. Therefore, choosing the boundary values according to the initial constant states is reasonable and will not affect well-posedness.

We tested the convergence using the Lax-Friedrichs flux, the local Lax-Friedrichs flux and the Godunov flux (with a linearized Riemann solver, see (4.14)) in the finite volume scheme with Euler time stepping. The results can be found in Table 3. Surprisingly, the (local) Lax-Friedrichs and Godunov schemes perform better in this case than for the linear advection equation. All methods obtain a convergence rate of about 3/4 which results in a relatively small error on the finest grid (see Figure 7). On a coarse grid with 128 cells, all methods show smearing at the discontinuities, which is strongest for the global Lax-Friedrichs flux and weakest for the Godunov flux. On a fine grid with 4,096 cells, all schemes are close to the semi-analytic solution, except at one of the two shocks that is still smoothed noticeably. Again, the Godunov flux performs best.

Naively using the slope reconstruction in the linearized Riemann solver for the Godunov flux together with Heun's method resulted in solutions that are almost identical to what was obtained with the unmodified Godunov flux. To actually get more accurate results, higher order methods designed for nonlinear equations would have to be implemented (see [56]).

6.2 Comparison with existing implementations

To test the implementation further, we compare the results obtained by our implementation with results obtained by existing implementations for P_N moment models.

6.2.1 Fokker-Planck equation in one spatial dimension

As a test case in one dimension, we regard the Fokker-Planck equation which describes particle transport through a background medium where the particle scattering is very forward-peaked, i.e. the majority of the scattering events include very little energy transfer and almost no change of direction of the particle. We get a mathematically one-dimensional problem if we regard a slab-like geometry with infinite extension in two spatial directions and assume symmetry in these directions such that the solution depends only on the third spatial direction (see [38]). This gives the kinetic equation

$$\partial_t \mathbf{p}(t,x,\mu) + \mu \partial_x p(t,x,\mu) = \mathbf{Q}(x) + \frac{T(x)}{2}\Delta_\mu \mathbf{p}(t,x,\mu) - \sigma_a(x) p(t,x,\mu), \tag{6.6}$$

where $\mu \in V = [-1,1]$ is the cosine of the angle between the x-axis and the direction of particle travel and Δ_μ is the Laplace-Beltrami operator on the unit sphere (see [1]).

The corresponding P_N model using Legendre polynomials as a basis for $\mathbb{P}(V)$ then is

$$\partial_t \mathbf{p}(t,x) + \mathbf{A}\partial_x \mathbf{p}(t,x) = \left(\mathbf{Q}(x) - \left(\sigma_a(x) + \frac{T(x)}{2}\right)\mathbf{I}\right)\mathbf{p}(t,x) \tag{6.7}$$

with $\mathbf{I}$ being the unit matrix, $\mathbf{Q}(x)\mathbf{p}(t,x) = (Q(x)p^{(0)}(t,x), 0, \dots, 0)$ and $\mathbf{A}$ such that

$$A_{ij} = \begin{cases} \frac{i+1}{2i+1} & \text{if } j = i+1, \\ \frac{i}{2i+1} & \text{if } j = i-1, \\ 0 & \text{else.} \end{cases}$$

We will focus on the two beams test case from [1] that models particles entering an absorbing medium in the spatial domain $\Omega = [-0.5, 0.5]$ from both sides. Here, we have $T(x) = Q(x) = 0$ and $\sigma_a(x) = 4$. Further, we have initial conditions

$$p(0, x, \mu) = 10^{-4} \quad \text{for} \quad x \in (-0.5, 0.5),\ \mu \in [-1, 1],$$

and boundary conditions

$$p(t, -0.5, \mu > 0) = 100\delta(\mu - 1), \quad p(t, -0.5, \mu < 0) = 100\delta(\mu + 1)$$

where δ is the Dirac delta distribution. Initial and boundary conditions for the moments are calculated from these conditions by assuming the boundary conditions are given for all μ.

6.2.2 Validity of results

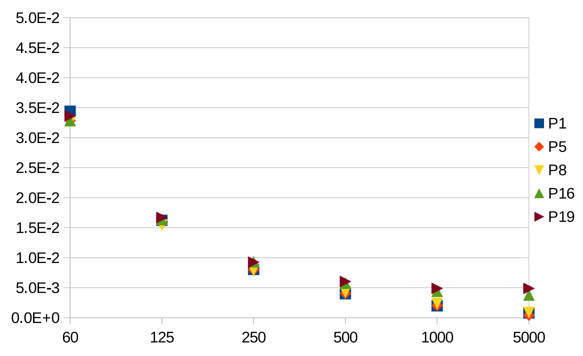
There are two existing solvers for the Fokker-Planck P_N equations that we will take as a reference. The first one is a MATLAB implementation by Florian Schneider based on the original code of Martin Frank that was used in [1]. The second one is a Python implementation by Julia Brunken [13] in the pyMOR framework [61].

We solved the two beams test case for the Fokker-Planck P_N equations up to $t_{\text{end}} = 2$ with $N = 1, 5, 8, 16, 19$ on different grids and calculated the relative L^1-error between the solutions by

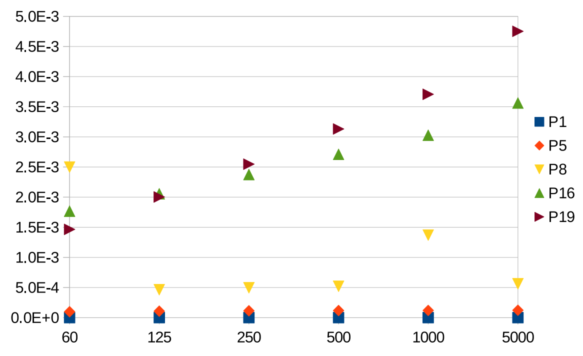
$$\frac{||p^{(0)}_{\text{DUNE}} - p^{(0)}_{\text{Other}}||_{L^1([0,2]\times\Omega)}}{||p^{(0)}_{\text{DUNE}}||_{L^1([0,2]\times\Omega)}}$$

where $p^{(0)}_{\text{DUNE}}$ is the first component of the discrete solution calculated by the DUNE implementation.

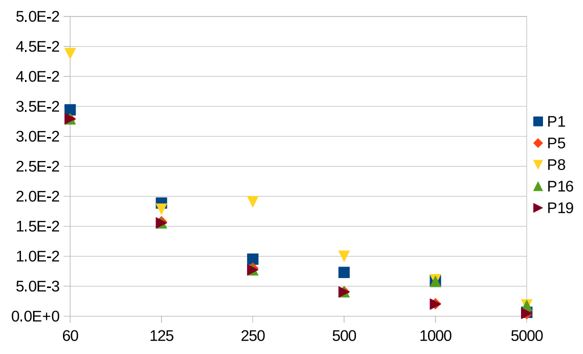
The results for the comparison with the pyMOR implementation can be found in Figure 8a. On the coarsest grid, the relative L^1-error of the pyMOR solution with respect to the DUNE solution is about 3 %. The error is decreasing as the grid becomes finer. This can be explained by the fact that the pyMOR implementation uses an unsplit method to incorporate the source terms while the DUNE implementation uses a fractional step approach (see Section 4.1.3) which may result in different solutions. As $h \to 0$, both methods converge to the true solution and thus approach each other. To eliminate this error source in our comparison, we temporarily made DUNE use the same unsplit method and tested again (see Figure 8b). The errors are considerably lower now and are almost completely gone for the P_1 equations (around 10^{-7}).



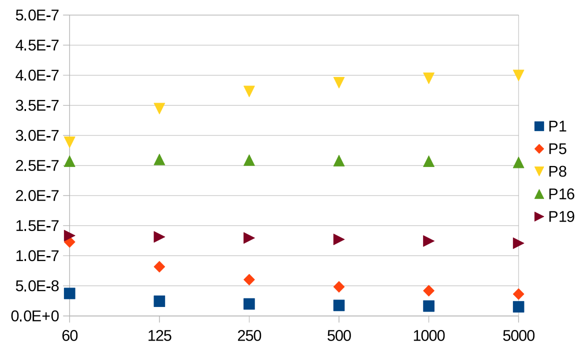
(a) DUNE vs. pyMOR



(b) DUNE unsplit vs. pyMOR



(c) DUNE vs. pyMOR exact data



(d) DUNE unsplit vs. pyMOR exact data

Figure 8: Comparison of solutions to the two beams test case for the Fokker-Planck P_N equations. The horizontal axis represents the number of grid cells. The relative L^1-error of the pyMOR solution with respect to the DUNE solution is displayed on the vertical axis. **(a)** The difference between the solutions becomes smaller with increasing grid size but does not vanish completely. **(b)** If DUNE uses an unsplit method for the source terms, the difference is considerably smaller but still quite high for the higher order equations and increases with grid size. **(c)** If pyMOR uses the exact data, the difference vanishes with increasing grid size. **(d)** If DUNE uses the unsplit method and pyMOR uses the exact data, there is essentially no difference between the solutions.

However, for the higher order equations the error is still quite high considering that the same finite volume method on the same grid is used. Additionally, it seems to increase with the grid size. We found that this is due to the pyMOR implementation of the Legendre polynomials.

Instead of always using Legendre polynomials as a basis for the velocity function space, the pyMOR implementation is designed to find a better (i.e. more efficient) choice for the basis functions using reduced basis methods. A grid for the angular domain V is chosen (here, we choose a grid with 1,000 cells for the domain $[-1, 1]$). The basis functions are then chosen from the space of continuous functions that are linear (first order polynomials) on each grid cell. For order greater than 1, the Legendre polynomials are not contained in this space. Thus, if the pyMOR implementation is told not to search for an ideal basis and just uses Legendre polynomials, a projection is done by evaluating the Legendre polynomials at each intersection of the grid and interpolating linearly in between. The integrals over the velocity component in the definition of the P_N equations are then calculated by a quadrature which is first order on each cell of the velocity grid and thus does not introduce additional errors. Because of the projection, pyMOR does not solve the P_N equations for the Legendre polynomials but the P_N equations for a slightly different basis. Thus, even with an exact solver the results will differ. For testing purposes, we bypassed the system assembly from the basis and provided the pyMOR solver with the exact flux and source terms and initial and boundary conditions. If DUNE uses the fractional step method for the source terms and pyMOR uses the exact data, the error decreases with increasing grid size (see Figure 8c), also for the higher order equations. If in addition DUNE uses the unsplit method, there is essentially no difference between the solutions (relative L^1-error of about 10^{-7}, see Figure 8d).

The MATLAB implementation uses the exact Legendre polynomials as a basis and an unsplit method for the source terms. Thus, the comparison between the DUNE and the MATLAB implementation looks like Figures 8c and 8d. The error decreases with increasing grid size and if DUNE uses the unsplit method, there is essentially no difference in the solutions.

We conclude that the observed differences are fully explained by the above considerations and there does not seem to be a systematic error in the DUNE implementation.

6.2.3 Performance comparison

We also used the two beams test case to measure the relative performance of the implementations. As the pyMOR implementation is not optimized for performance yet and is thus slower than the MATLAB implementation, we did not include it in our comparison.

We measured execution times for the P_5 and P_{50} two beams test case up to time $t_{\text{end}} = 2$ for the smaller grids and up to $t_{\text{end}} = 0.01$ for the largest grid to keep computing time reasonably short. The automatic parallelization that MATLAB uses for some built-in functions and linear algebra operations did not kick in during the P_5 tests as it is only applied if the data size (for example vectors and matrices in a linear algebra operation) exceeds some threshold [60]. During the P_{50} test, however, MATLAB made heavy use of multiple cores. Surprisingly, this did not improve performance. Execution times were up to three times longer in comparison to disabled parallelization. Explicitly setting processor affinity did not help, nor did testing on another system or manually measuring time to exclude an error in the timing. We thus decided to test MATLAB with multithreading disabled (option "-singleCompThread") to get the best performance.

As can be seen in Table 4, MATLAB is not much slower than DUNE in the P_5 test case. If no parallelization is used, it is sometimes even faster. This may be due to the fact that iterating over the grid, as the DUNE implementation does, comes at some cost in comparison to simply using vectorized computations. The advantage of the DUNE approach is, however, that non-equidistant grids and grids with different cell shapes in several dimensions can be used much easier. Using

t_{end}	N_x	MATLAB [s]	DUNE 1 thread [s]	relative performance	DUNE 16 threads [s]	relative performance
2.00	100	5.2	0.5	10.3	0.6	9.5
2.00	1,000	52.2	21.1	2.5	11.8	4.4
2.00	5,000	336.4	485.6	0.7	208.1	1.6
0.01	50,000	173.0	222.1	0.8	71.3	2.4

(a) P_5

t_{end}	N_x	MATLAB [s]	DUNE 1 thread [s]	relative performance	DUNE 16 threads [s]	relative performance
2.00	100	15.0	3.2	4.6	2.9	5.1
2.00	1,000	508.0	97.0	5.2	44.5	11.4
2.00	5,000	10,239.5	2,823.0	3.6	463.2	22.1
0.01	50,000	6,833.7	890.3	7.7	139.0	49.2

(b) P_{50}

Table 4: Execution times for the two beams test case. Average of three runs. The relative performance is calculated by dividing the execution time for DUNE 1 thread/DUNE 16 threads by the MATLAB execution time. **(a)** In the P_5 test case, the DUNE implementation is not much faster than the MATLAB implementation. If no multithreading is used, it is sometimes even slightly slower. **(b)** In the P_{50} test case, DUNE is considerably faster even with a single thread. N_x: number of grid cells, t_{end}: end time.

16 threads, DUNE is about twice as fast as MATLAB on the larger grids.

In the P_{50} test case, the vectors involved in the computations on each grid cell are considerably larger than in the P_5 test case which increases computational cost and the amount of memory allocated. Here, the picture is completely different. DUNE is faster across the board even when using only a single thread. With 16 threads DUNE is almost 50 times as fast on the largest grid.

We also took a look at the scaling with additional threads for the DUNE implementation. To investigate strong scaling, we measured the execution time for the P_{50} two beams test case on a grid with $N_x = 10^5$ cells for an increasing number of threads. For the weak scaling test, we started with a grid with $N_x = 3{,}000$ cells and doubled N_x with every doubling of the thread count. In both cases we calculated up to $t_{\text{end}} = 0.02$ and used a fixed time step of $\Delta t = 5 \cdot 10^{-6}$ for all grids, which gives a Courant number of 0.5 on the finest grid. We did not alter the time step length in the weak scaling test with the grid size because it is clear that our implementation can only scale with the number of grid cells and cannot parallelize the time steps. If the execution time with k threads is t_k, we calculate the scaling efficiency as

$$\frac{t_1}{k \cdot t_k} \quad \text{and} \quad \frac{t_1}{t_k}$$

for strong and weak scaling, respectively.

The results can be found in Table 5. In the strong scaling test, the execution time shrinks by about 1/3 for each doubling of the number of threads, which gives a scaling efficiency of about

No. threads	N_x	Wall time [s]	Scaling efficiency
1	10^5	7,253	–
2	10^5	4,848	0.75
4	10^5	3,018	0.60
8	10^5	2,042	0.44
16	10^5	1,380	0.33
32	10^5	1,139	0.20

(a) Strong scaling

No. threads	N_x	Wall time [s]	Scaling efficiency
1	3,000	259	–
2	6,000	315	0.82
4	12,000	383	0.68
8	24,000	477	0.54
16	48,000	618	0.42
32	96,000	1,071	0.24

(b) Weak scaling

Table 5: Scaling of the DUNE implementation in the two beams test case.
(a) In the strong scaling test, the execution time shrinks by about 1/3 for each doubling of the number of threads. **(b)** In the weak scaling test, execution takes about 25 % longer with each concurrent doubling of problem size and number of threads.

3/4 in the first step and an accordingly lower scaling efficiency with the subsequent doublings. An exception is the last step from 16 to 32 threads which gives a much lower speed-up. This is probably because the calculations were done on a system with two CPUs with 16 cores each such that the first tests can be run on a single CPU whereas in the last step both CPUs have to be used which comes with some additional overhead.

The weak scaling test shows a similar outcome. With each concurrent doubling of the problem size and number of threads, the execution time extends by about 20 - 30 % which gives a scaling efficiency of about 4/5 in the first step and a correspondingly worse scaling efficiency in the subsequent steps. Again, the scaling is worse in the last step where we go from 16 to 32 threads.

In conclusion, the use of multithreading gives a considerable speed-up, even though the scaling is not optimal. However, optimal scaling could not be expected since multithreading always comes with some overhead. Furthermore, the problem is not completely parallelizable as the time stepping has to be done serially. This includes addition and multiplication of some large vectors which is not parallelized in the implementation yet.

6.2.4 A two dimensional test for the Boltzmann equation

As a two-dimensional test, we use the Checkerboard test case from [14]. Applying the P_N approach using spherical harmonics to the Boltzmann equation in two dimensions gives the model

$$\partial_t \mathbf{p}(t,\mathbf{x}) + \mathbf{X}\partial_x \mathbf{p}(t,\mathbf{x}) + \mathbf{Z}\partial_z \mathbf{p}(t,\mathbf{x}) = \mathbf{s}(t,\mathbf{x}) + \left(\sigma_s(\mathbf{x})\mathbf{Q} - \sigma_t(\mathbf{x})\mathbf{I}\right)\mathbf{p}(t,\mathbf{x}) \tag{6.8}$$

where $\mathbf{x} = (x, z)$, $\mathbf{I}$ is the unit matrix and $Q_{00} = 1$, $Q_{ij} = 0$ else. The positive coefficients σ_s and σ_t describe scattering and total cross section, respectively, and $\mathbf{s}(t,\mathbf{x})$ is a particle source. See [14] for the definitions of the matrices $\mathbf{X}$ and $\mathbf{Z}$ and the derivation of the model.

The Checkerboard test case assumes a spatial domain $[0, 7] \times [0, 7]$ that is divided in 49 axis-

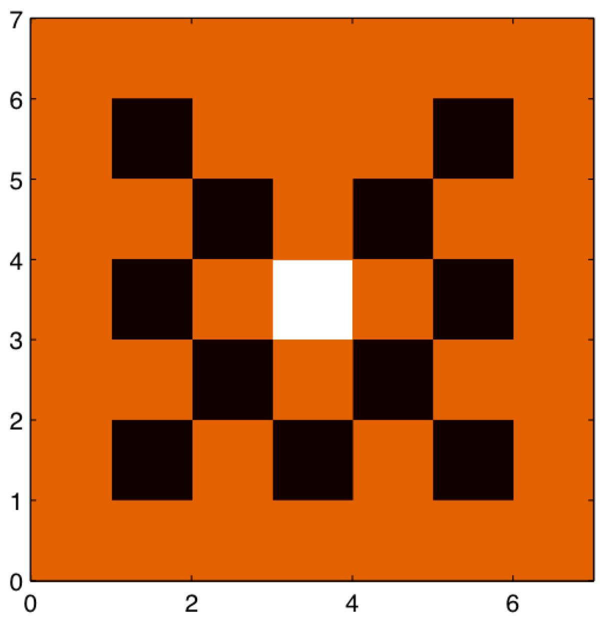
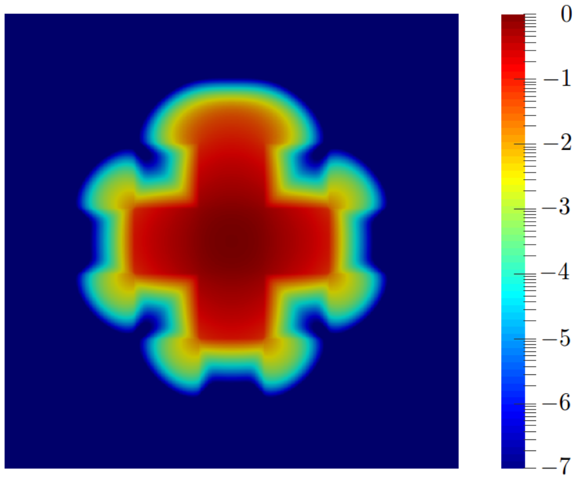


Figure 9: Illustration of the domain used in the Checkerboard test case. The orange and white regions are scattering regions with $\sigma_t = \sigma_s = 1$. The black regions are absorbing regions with $\sigma_t = 10$, $\sigma_s = 0$. At $t = 0$, a source is turned on in the centre region (white). From [73].

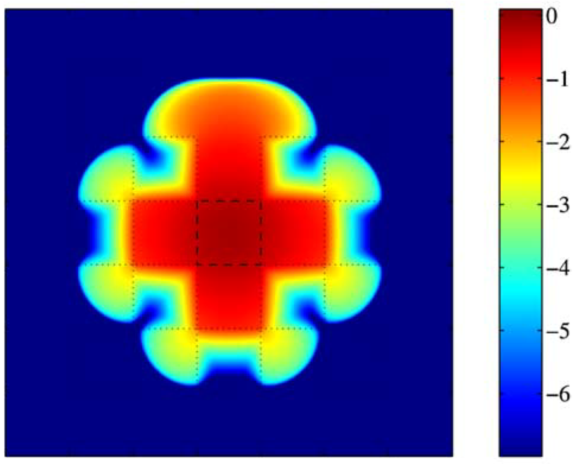
parallel cubes with edge width 1. Most of the domain is composed of a scattering material with $\sigma_t = \sigma_s = 1$ but there are 11 scattering regions with $\sigma_t = 10$, $\sigma_s = 0$ (see Figure 9). At time $t = 0$, a source with strength 1 is turned on in the centre region, i.e. $\mathbf{s}(t, \mathbf{x}) = (1, 0, \dots, 0)^T$ for $\mathbf{x}$ in the centre region and $\mathbf{s}(t, \mathbf{x}) = \mathbf{0}$ else. At the boundary and as an initial condition, the solution is set to zero.

In this setting, the particles coming from the source in the centre region will spread out uniformly in all directions until an absorbing region is met. Considering the distribution of absorbing regions, we expect a cross-shaped region of high particle density in the centre with particles leaking to the top and between the absorbers.

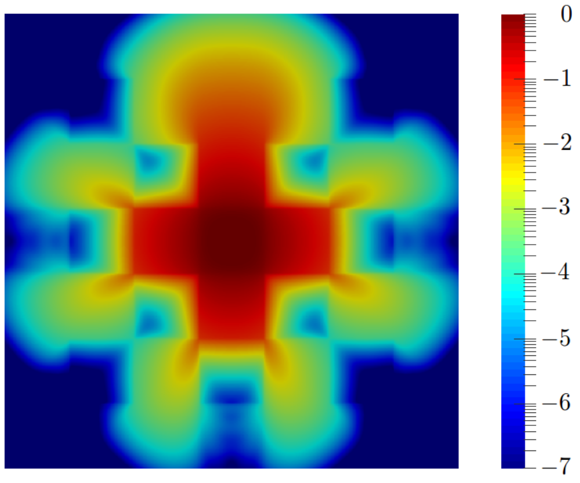
We computed solutions for the P_1 and the P_{15} moment models using the Godunov flux and explicit Euler time stepping on a grid with 700×700 cells. The results can be found in Figure 10 where we included the solutions from [14] calculated by a second order scheme as a reference. Note that the color scales may differ slightly. Still, it can be seen that both the P_1 and the P_{15} solution agree very well with the reference solutions. The P_1 solutions differ slightly in the wave going to the top. The P_{15} reference solution exhibits some oscillations behind the absorbers on the left, right and bottom that are almost absent in the DUNE solution. These are probably artificial oscillations introduced by the P_{15} model that would not appear in the true solution of the kinetic equation but should be present in the exact solution of the P_{15} model. Here, the second order scheme used in the reference solution apparently gives a better resolution of the oscillations than the first order Godunov scheme.



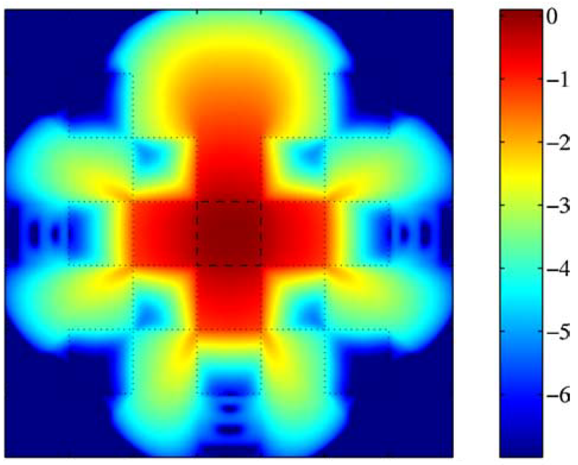
(a) P_1



(b) P_1 Reference



(c) P_{15}



(d) P_{15} Reference

Figure 10: Solutions to the Boltzmann Checkerboard test case in two dimensions. The solutions on the left are computed with DUNE whereas the reference solutions are taken from [14]. Note that the color scales may differ slightly. Both the P_1 and the P_{15} solutions agree very well with the reference solutions. The P_1 solutions differ slightly in the wave going to the top. The reference P_{15} solution shows some oscillations that are almost absent in the DUNE solution.

7 Conclusion and outlook

Kinetic equations, such as the Boltzmann equation of gas dynamics, play an important role in many physical applications. However, due to their high dimensionality, directly solving the kinetic equations with standard numerical methods often causes a prohibitive amount of computational cost. In this thesis, we investigated a family of model reduction techniques for kinetic equations. These models reduce the dimension of the problem by first transferring it to a coupled system of infinitely many partial differential equations for the moments of the kinetic equation and then truncating the system at a finite order. The resulting truncated system is underdetermined and needs to be closed. There are several approaches to achieve that closure which lead to different moment models (see Section 2.3). However, most of them result in hyperbolic systems of PDEs that need to be solved.

An efficient solver for hyperbolic systems of equations was implemented in the C++ software framework DUNE [5, 6]. The implementation uses the finite volume method and provides several numerical flux functions and Runge-Kutta time stepping. Both Dirichlet and periodic boundary conditions are supported. Furthermore, calculations are done in parallel using the built-in shared-memory parallelization functionality from dune-stuff [62] and dune-gdt [70] (see Section 5.3.4).

The implementation was verified using test problems with known analytical solution in one and two dimensions. Furthermore, it was tested against existing solvers for the P_N moment models [13, 14, 72]. The results obtained by the DUNE implementation were shown to agree with the results obtained by the existing solvers. Execution times were shown to be significantly shorter compared to the existing MATLAB and Python implementations.

While the DUNE implementation proved to be able to efficiently solve test problems in one and two dimensions, there are several improvements that can be made. More accurate schemes using slope reconstruction were only implemented for linear problems in one dimension. Similar schemes should also be implemented for several dimensions to be able to accurately solve complex problems without the need of a very fine grid. Especially in three dimensions, the computational cost increases rapidly with each refinement of the grid. For a cube grid in three dimensions the number of grid cells increases by a factor of 8 if the grid width is halved in each coordinate direction. Using a fixed Courant number this amounts to an increase of the computational cost by a factor of 16.

Even if higher-order schemes are used, the computational cost will be quite high in several dimensions for complex problems, probably too high to solve on a single processor. To be able to efficiently compute on clusters with distributed memory, the MPI functionality in DUNE should be used.

Another approach to minimize computational cost is, instead of using a fixed polynomial basis for the velocity space, to use reduced basis methods in order to find an optimal basis. This was investigated in [13] and implemented in Python in the pyMOR framework [61]. That implementation is not optimized for performance yet. To improve performance, we plan to couple the basis generation process in pyMOR with the efficient solver for hyperbolic systems in DUNE.

The current DUNE implementation is mainly centred at P_N models. For M_N entropy moment models, tools for the solution of the dual problem such as a root finder, adaptive quadrature or adaptive change of basis and regularization techniques have to be implemented. In addition, higher-order schemes for nonlinear hyperbolic systems of equations would be beneficial.

In summary, the DUNE implementation proved to be able to efficiently solve hyperbolic problems arising from P_N moment models in one or two dimensions. Furthermore, it can be seen as a starting point for the development of a fast solver for general moment models. Due to the modular nature of DUNE, the implementation can easily be extended to allow for MPI parallelism, higher order schemes in several dimensions, coupling with model reduction techniques for basis generation and different closure approaches.

A Example implementation

In this appendix, we investigate a `main` function for the solution of the two beams test case using our finite volume implementation. The example was implemented in the DUNE module dune-hdd [71] and can be found in `dune/hdd/examples/hyperbolic/twobeams.cc`

```
#include "config.h"

#include <string>
#include <vector>

#include <dune/stuff/common/string.hh>
#include <dune/stuff/grid/provider/cube.hh>
#include <dune/stuff/grid/information.hh>
#include <dune/stuff/la/container/common.hh>

#include <dune/gdt/discretefunction/default.hh>
#include <dune/gdt/operators/advection.hh>
#include <dune/gdt/operators/projections.hh>
#include <dune/gdt/spaces/fv/defaultproduct.hh>
#include <dune/gdt/timestepper/rungekutta.hh>

#include <dune/hdd/hyperbolic/problems/twobeams.hh>

using namespace Dune::GDT;

int main(int argc, char* argv[])
{
    DS::threadManager().set_max_threads(8);                // number of threads used
    DSC_CONFIG.set("threading.partition_factor", 1, true); // one partition per thread

    static const size_t dimDomain = 1;
    static const size_t momentOrder = 5;

    //choose GridType
    typedef Dune::YaspGrid< dimDomain >                                     GridType;
    typedef typename GridType::Codim< 0 >::Entity                           EntityType;

    //choose problem (P_5 two beams test case)
    typedef Dune::HDD::Hyperbolic::Problems::TwoBeams< EntityType, double,
                                                       dimDomain, double,
                                                       momentOrder >        ProblemType;
    // create problem
    const auto problem_ptr = ProblemType::create();
    const auto& problem = *problem_ptr;

    // get data from problem
    typedef typename ProblemType::FunctionType          FunctionType;
    typedef typename FunctionType::DomainFieldType      DomainFieldType;
    typedef typename ProblemType::RangeFieldType        RangeFieldType;
    const auto analytical_flux_ptr = problem.flux();
    const auto initial_values_ptr = problem.initial_values();
    const auto boundary_values_ptr = problem.boundary_values();
    const auto source_ptr = problem.source();
    static const size_t dimRange = ProblemType::dimRange; // number of moments

    // get grid configuration from problem and create Grid and GridView
    auto grid_config = problem.grid_config();
    typedef Dune::Stuff::Grid::Providers::Cube< GridType >  GridProviderType;
    GridProviderType grid_provider = *(GridProviderType::create(grid_config));
    auto grid_ptr = grid_provider.grid_ptr();
    typedef typename GridType::LeafGridView                 GridViewType;
    const GridViewType grid_view = grid_ptr->leafGridView();

    // make a product finite volume space on the grid_view
    typedef Spaces::FV::DefaultProduct< GridViewType,
                                    RangeFieldType, dimRange, 1 >   FVSpaceType;
    const FVSpaceType fv_space(grid_view);

    // create a discrete function
    typedef typename Dune::Stuff::LA::CommonDenseVector< RangeFieldType > VectorType;
    typedef DiscreteFunction< FVSpaceType, VectorType >  DiscreteFunctionType;
    DiscreteFunctionType p(fv_space, "solution");

    // project initial values to discrete function
    project(*initial_values, p);

    // choose Courant (CFL) number and t_end and calculate dx and dt
    Dune::Stuff::Grid::Dimensions< GridViewType > dimensions(fv_space.grid_view());
    const double dx = dimensions.entity_width.max();
    const double CFL = 0.5;
    double dt = CFL*dx;
    const double t_end = 2;

    // define operator types
    typedef typename Dune::Stuff::Functions::Constant
                    < EntityType, DomainFieldType, dimDomain,
                      RangeFieldType, dimRange, 1 >          ConstantFunctionType;
    typedef typename Dune::GDT::Operators::AdvectionGodunovWithReconstruction
            < AnalyticalFluxType, ConstantFunctionType,
              BoundaryValueType, FVSpaceType,
              Operators::SlopeLimiters::minmod >                OperatorType;
    typedef typename Dune::GDT::Operators::AdvectionSource
                               < SourceType, FVSpaceType >     SourceOperatorType;
    typedef typename Dune::GDT::TimeStepper::RungeKutta
      < OperatorType, SourceOperatorType, FVFunctionType, double > TimeStepperType;

```

```cpp
    // create butcher_array for Heun's method
    typedef typename Dune::DynamicMatrix< RangeFieldType >        DynamicMatrixType;
    typedef typename Dune::DynamicVector< RangeFieldType >        DynamicVectorType;
    DynamicMatrixType A(DSC::fromString< DynamicMatrixType >("[0 0; 1 0]"));
    DynamicVectorType b(DSC::fromString< DynamicVectorType >("[0.5 0.5]"));
    DynamicVectorType c(DSC::fromString< DynamicVectorType >("[0 1]"));

    //create Operators
    ConstantFunctionType dx_function(dx); // we use an equidistant grid, so dx is constant
    OperatorType advection_operator(*analytical_flux, dx_function, dt,
                                    *boundary_values, fv_space, true);
    SourceOperatorType source_operator(*source, fv_space);
    TimeStepperType timestepper(advection_operator, source_operator, p, A, b, c);

    // solve and visualize solution
    const double saveInterval = t_end/100.0;
    timestepper.solve(t_end, dt, saveInterval, true, false, "twobeams");

    return 0;
} // ... main(...)
```

In the beginning (line 23), the number of threads that will be used has to be specified. Here, we set a maximum of 8 threads directly in the code. We could also read a command line option to allow changing the number of threads after compilation. In the next line, we choose how many partitions of the grid should be created per thread.

We then specify the grid dimension and the moment order N for the P_N model and choose to use the built-in `YaspGrid` which also gives the type for entities with codimension 0 (lines 26-31).

In line 34, we choose the `TwoBeams` class with the specified moment order and grid dimension as `ProblemType`. The `TwoBeams` class contains all information about the analytical flux, boundary and initial values and source terms for the two beams test case and is implemented in `dune/hdd/hyperbolic/problems/twobeams.hh`. In addition to the `TwoBeams` class, we also implemented similar problem classes for the other test cases in [72], the test cases for the Boltzmann equation from [14] and some classical hyperbolic problems such as the one-dimensional shallow water equations. Furthermore, a problem interface and a default problem implementation that aim to make the addition of new problems as simple as possible can be found in `dune/hdd/hyperbolic/problems/`.

In the next two lines (38-39), an object `problem` is instantiated, which includes creation of functions for the problem data such as analytical flux and initial values. The data functions are obtained from the `problem` in lines 42-49. The `problem` also stores data about the grid that should be used such as the grid domain and the number of grid cells in each coordinate direction. This information is used to create a `Grid` and subsequently a `GridView` in lines 52-57. If we would like to use periodic boundary conditions, we could create a `PeriodicGridView` from the `GridView` but for this test case we use Dirichlet boundary conditions.

In lines 60-67, a finite volume space (seen as a product of `dimRange` scalar finite volume spaces) and a discrete function on this space are instantiated. The initial values are then projected to the discrete function.

In lines 73-77, the grid width is determined and a Courant number of 0.5 is chosen to calculate the time step length.

We choose a Godunov flux with linear reconstruction and the minmod slope limiter. Further we choose `AdvectionSource` as an operator for the source terms and the `RungeKutta` time stepper (lines 80-90). To use Heun's method, we create the appropriate matrix and vectors forming the butcher array (lines 93-97). We instantiate the operators for source terms and numerical flux using the problem data and then create the Runge-Kutta time stepper using these operators and the butcher array (lines 100-104).

In the end, we specify in which interval the solution should be written and start the solving process using the `solve` method of the time stepper. With the arguments used here, the solution will be visualized at 100 equidistant time points in the interval $[0, 2]$.

References

[1] G. W. Alldredge, C. D. Hauck and A. L. Tits. 'High-Order Entropy-Based Closures for Linear Transport in Slab Geometry II: A Computational Study of the Optimization Problem'. In: *SIAM Journal on Scientific Computing* 34.4 (2012), B361–B391.

[2] G. W. Alldredge et al. 'Adaptive change of basis in entropy-based moment closures for linear kinetic equations'. In: *Journal of Computational Physics* 258 (2014), pp. 489–508.

[3] C.-G. Ambrozie. 'Multivariate truncated moments problems and maximum entropy'. In: *Analysis and Mathematical Physics* 3.2 (2013), pp. 145–161.

[4] T. Barth and M. Ohlberger. 'Finite Volume Methods: Foundation and Analysis'. In: *Encyclopedia of Computational Mechanics.* Ed. by E. Stein, R. d. Borst and T. J. R. Hughes. Chichester, UK: John Wiley & Sons, Ltd, 2004.

[5] P. Bastian et al. 'A generic grid interface for parallel and adaptive scientific computing. Part I: Abstract framework'. In: *Computing* 82.2-3 (2008), pp. 103–119.

[6] P. Bastian et al. 'A generic grid interface for parallel and adaptive scientific computing. Part II: Implementation and tests in DUNE'. In: *Computing* 82.2-3 (2008), pp. 121–138.

[7] S. Bianchini and A. Bressan. 'Vanishing Viscosity Solutions of Nonlinear Hyperbolic Systems'. In: *Annals of Mathematics* 161.1 (2005), pp. 223–342.

[8] G. A. Bird. 'Direct Simulation and the Boltzmann Equation'. In: *Physics of Fluids* 13.11 (1970), p. 2676.

[9] L. Boltzmann. 'Weitere Studien über das Wärmegleichgewicht unter Gasmolekülen'. In: *Sitzungsberichte der Akademie der Wissenschaften, Mathematische-Naturwissenschaftliche Klasse* 66 (1872), pp. 275–370.

[10] A. Bressan. *Notes on the Boltzmann equation.* `https://www.math.psu.edu/bressan/PSPDF/boltz.pdf`. 2005.

[11] C. Bresten et al. 'Strong Stability Preserving Multistep Runge-Kutta Methods'. In: *ArXiv e-prints* (2013). eprint: `1307.8058`.

[12] M. Breuß. 'The correct use of the Lax-Friedrichs method'. In: *ESAIM: Mathematical Modelling and Numerical Analysis* 38.3 (Mar. 2010), pp. 519–540.

[13] J. Brunken. 'Model Reduction for Kinetic Equations'. MA thesis. Westfälische Wilhelms-Universität Münster, 2015.

[14] T. A. Brunner and J. P. Holloway. 'Two-dimensional time dependent Riemann solvers for neutron transport'. In: *Journal of Computational Physics* 210.1 (2005), pp. 386–399.

[15] Z. Cai, Y. Fan and R. Li. 'On hyperbolicity of 13-moment system'. In: *Kinetic and Related Models* 7.3 (2014), pp. 415–432.

[16] C. Cercignani. *The Boltzmann Equation and Its Applications.* Vol. 67. Applied Mathematical Sciences. New York, NY: Springer New York, 1988.

[17] E. M. Constantinescu and A. Sandu. 'Optimal Explicit Strong-Stability-Preserving General Linear Methods'. In: *SIAM Journal on Scientific Computing* 32.5 (2010), pp. 3130–3150.

[18] R. Courant, K. Friedrichs and H. Lewy. 'Über die partiellen Differenzengleichungen der mathematischen Physik'. In: *Mathematische Annalen* 100.1 (1928), pp. 32–74.

[19] N. Crouseilles, H. Hivert and M. Lemou. 'Multiscale numerical schemes for kinetic equations in the anomalous diffusion limit'. In: *Comptes Rendus Mathematique* 353.8 (2015), pp. 755–760.

[20] R. E. Curto and L. A. Fialkow. 'Truncated K-moment problems in several variables'. In: *J. Operator Theory* 54 (2005), pp. 189–226.

[21] R. E. Curto and L. A. Fialkow. 'The Truncated Complex K-Moment Problem'. In: *Transactions of the American Mathematical Society* 352.6 (2000), pp. 2825–2855.

[22] R. Curto and L. Fialkow. 'Recursiveness, positivity and truncated moment problems'. In: *Houston J. Math* 17.4 (1991), pp. 603–635.

[23] J. J. Duderstadt and W. R. Martin. *Transport theory.* New York: Wiley, 1979.

[24] *DUNE (Distributed and Unified Numerics Environment).* `http://www.dune-project.org/`.

[25] C. K. Garrett and C. D. Hauck. 'A Comparison of Moment Closures for Linear Kinetic Transport Equations: The Line Source Benchmark'. In: *Transport Theory and Statistical Physics* 42.6-7 (2014), pp. 203–235.

[26] G. T. Gilbert. 'Positive Definite Matrices and Sylvester's Criterion'. In: *The American Mathematical Monthly* 98.1 (1991), p. 44.

[27] E. Godlewski and P.-A. Raviart. *Numerical approximation of hyperbolic systems of conservation laws.* Vol. 118. Applied Mathematical Sciences. New York: Springer, 1996.

[28] S. K. Godunov. 'A finite difference method for the numerical computation of the equations of fluid dynamics'. In: *Mat. Sb.* 47 (1959), pp. 271–290.

[29] S. Gottlieb, Z. J. Grant and D. Higgs. 'Optimal Explicit Strong Stability Preserving Runge-Kutta Methods with High Linear Order and optimal Nonlinear Order'. In: *ArXiv e-prints* (2014). eprint: `1403.6519`.

[30] S. Gottlieb and C.-W. Shu. 'Total variation diminishing Runge-Kutta schemes'. In: *Mathematics of Computation of the American Mathematical Society* 67.221 (1998), pp. 73–85.

[31] H. Grad. 'On the kinetic theory of rarefied gases'. In: *Communications on Pure and Applied Mathematics* 2.4 (1949), pp. 331–407.

[32] K. P. Hadeler. ‘Reaction transport systems in biological modelling’. In: *Mathematics Inspired by Biology*. Ed. by V. Capasso. Lecture Notes in Mathematics. Springer Berlin Heidelberg, 1999.

[33] E. Hairer, G. Wanner and S. P. Nørsett. *Solving Ordinary Differential Equations I: Nonstiff Problems*. Second Revised Edition. Vol. 8. Springer Series in Computational Mathematics. Springer-Verlag Berlin Heidelberg, 1993.

[34] A. Harten. ‘High resolution schemes for hyperbolic conservation laws’. In: *Journal of Computational Physics* 49.3 (1983), pp. 357–393.

[35] C. D. Hauck. ‘High-order entropy-based closures for linear transport in slab geometry’. In: *Communications in Mathematical Sciences* 9.1 (2011), pp. 187–205.

[36] C. D. Hauck, C. D. Levermore and A. L. Tits. ‘Convex Duality and Entropy-Based Moment Closures: Characterizing Degenerate Densities’. In: *SIAM Journal on Control and Optimization* 47.4 (2008), pp. 1977–2015.

[37] M. Henk, J. Richter-Gebert and G. M. Ziegler. ‘Basic Properties Of Convex Polytopes’. In: *Handbook of discrete and computational geometry*. Ed. by J. E. Goodman and J. O’Rourke. Boca Raton, USA: CRC Press, 1997, pp. 243–270.

[38] H. Hensel, R. Iza-Teran and N. Siedow. ‘Deterministic model for dose calculation in photon radiotherapy’. In: *Physics in Medicine and Biology* 51.3 (2006), p. 675.

[39] K. Heun. ‘Neue Methoden zur approximativen Integration der Differentialgleichungen einer unabhängigen Veränderlichen’. In: *Zeitschrift für Mathematik und Physik* 5 (1900), pp. 23–38.

[40] T. Hillen. ‘M5 mesoscopic and macroscopic models for mesenchymal motion’. In: *Journal of mathematical biology* 53.4 (2006), pp. 585–616.

[41] T. Hillen and K. J. Painter. ‘Transport and Anisotropic Diffusion Models for Movement in Oriented Habitats’. In: *Dispersal, Individual Movement and Spatial Ecology*. Ed. by M. A. Lewis, P. K. Maini and S. V. Petrovskii. Vol. 2071. Lecture Notes in Mathematics. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 177–222.

[42] S. Jin. ‘Asymptotic preserving (AP) schemes for multiscale kinetic and hyperbolic equations: a review’. In: *Rivista di Matematica della Università di Parma. New Series* 2.2 (2010).

[43] S. Jin. ‘Efficient Asymptotic-Preserving (AP) Schemes For Some Multiscale Kinetic Equations’. In: *SIAM Journal on Scientific Computing* 21.2 (1999), pp. 441–454.

[44] M. Junk. ‘Maximum entropy for reduced moment problems’. In: *Mathematical Models and Methods in Applied Sciences* 10.07 (2000), pp. 1001–1025.

[45] D. S. Kershaw. *Flux limiting nature‘s own way – A new method for numerical solution of the transport equation*. Lawrence Livermore National Lab., 1976.

[46] D. I. Ketcheson, S. Gottlieb and C. B. Macdonald. ‘Strong Stability Preserving Two-step Runge-Kutta Methods’. In: *SIAM Journal on Numerical Analysis* 49.6 (2011), pp. 2618–2639.

[47] Kraaijevanger, J. F. B. M. ‘Contractivity of Runge-Kutta methods’. In: *BIT* 31.3 (1991), pp. 482–528.

[48] D. Kröner. *Numerical schemes for conservation laws.* Wiley-Teubner series, advances in numerical mathematics. Chichester, New York and Stuttgart: Wiley and Teubner, 1997.

[49] A. G. Kulikovskii, N. V. Pogorelov and A. Y. Semenov. *Mathematical aspects of numerical solution of hyperbolic systems.* Vol. 118. Chapman & Hall/CRC monographs and surveys in pure and applied mathematics. Boca Raton: Chapman & Hall/CRC, 2001.

[50] W. Kutta. ‘Beitrag zur näherungsweisen Integration totaler Differentialgleichungen’. In: *Zeitschrift für Mathematik und Physik* 46 (1901), pp. 435–453.

[51] J. D. Lambert. *Numerical methods for ordinary differential systems: The initial value problem.* Chichester and New York: Wiley, 1991.

[52] K. Lanckau. ‘Cercignani, C., The Boltzmann Equation and Its Applications.’ In: *ZAMM - Journal of Applied Mathematics and Mechanics / Zeitschrift für Angewandte Mathematik und Mechanik* 69.11 (1989), p. 423.

[53] J. B. Lasserre. ‘Bounds on measures satisfying moment conditions’. In: *The Annals of Applied Probability* 12.3 (2002), pp. 1114–1137.

[54] J. B. Lasserre. ‘Global Optimization with Polynomials and the Problem of Moments’. In: *SIAM Journal on Optimization* 11.3 (2001), pp. 796–817.

[55] P. D. Lax. ‘Weak solutions of nonlinear hyperbolic equations and their numerical computation’. In: *Communications on Pure and Applied Mathematics* 7.1 (1954), pp. 159–193.

[56] R. J. LeVeque. *Finite volume methods for hyperbolic problems.* Vol. 31. Cambridge university press, 2002.

[57] C. D. Levermore. ‘Moment closure hierarchies for kinetic theories’. In: *Journal of Statistical Physics* 83.5-6 (1996), pp. 1021–1065.

[58] Z. Li et al. ‘Convergence proof of the DSMC method and the Gas-Kinetic Unified Algorithm for the Boltzmann equation’. In: *Science China Physics, Mechanics and Astronomy* 56.2 (2013), pp. 404–417.

[59] F. D. Lora-Clavijo et al. ‘Exact solution of the 1D Riemann problem in Newtonian and relativistic hydrodynamics’. In: *Rev. Mex. Fis. E 59* (2013).

[60] MathWorks. *Answers.* `https://www.mathworks.com/matlabcentral/answers/95958-which-matlab-functions-benefit-from-multithreaded-computation`.

[61] R. Milk, S. Rave and F. Schindler. ‘pyMOR - Generic Algorithms and Interfaces for Model Order Reduction’. In: *ArXiv e-prints* (June 2015). eprint: `1506.07094`.

[62] R. Milk and F. Schindler. *dune-stuff*. `https://github.com/wwu-numerik/dune-stuff`.

[63] G. N. Minerbo. ‘Maximum entropy Eddington factors’. In: *Journal of Quantitative Spectroscopy and Radiative Transfer* 20.6 (1978), pp. 541–545.

[64] P. Monreal. ‘Moment Realizability and Kershaw Closures in Radiative Transfer’. PhD thesis. Aachen: RWTH Aachen, 2012.

[65] H. Nessyahu and E. Tadmor. ‘Non-oscillatory central differencing for hyperbolic conservation laws’. In: *Journal of Computational Physics* 87.2 (1990), pp. 408–463.

[66] S. P. Parker. *McGraw-Hill Encyclopedia of Physics*. 2nd ed. New York: McGraw-Hill, 1993.

[67] W. Ren, H. Liu and S. Jin. ‘An asymptotic-preserving Monte Carlo method for the Boltzmann equation’. In: *Journal of Computational Physics* 276 (2014), pp. 380–404.

[68] C. Runge. ‘Über die numerische Auflösung von Differentialgleichungen’. In: *Mathematische Annalen* 46 (1895), pp. 167–178.

[69] S. J. Ruuth and R. J. Spiteri. ‘Two Barriers on Strong-Stability-Preserving Time Discretization Methods’. In: *Journal of Scientific Computing* 17.1/4 (2002), pp. 211–220.

[70] F. Schindler. *dune-gdt*. `https://github.com/pymor/dune-gdt`.

[71] F. Schindler. *dune-hdd*. `https://github.com/pymor/dune-hdd`.

[72] F. Schneider et al. ‘Higher order mixed moment approximations for the Fokker-Planck equation in one space dimension’. In: *SIAM Journal on Applied Mathematics* 74.4 (2014), pp. 1087–1114.

[73] B. Seibold and M. Frank. ‘StaRMAP - A second order staggered grid method for spherical harmonics moment equations of radiative transfer’. In: *ArXiv e-prints* (Nov. 2012). eprint: `1211.2205`.

[74] J. Shohat and J. D. Tamarkin. *The problem of moments*. [Rev. ed.] Vol. no. 1. Mathematical surveys. Providence, R.I.: American Mathematical Society, 1963.

[75] C.-W. Shu. *A Survey of Strong Stability Preserving High Order Time Discretizations*. `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.22.4711`. 2001.

[76] C.-W. Shu and S. Osher. ‘Efficient implementation of essentially non-oscillatory shock-capturing schemes’. In: *J. Comput. Phys.* 77.2 (Aug. 1988), pp. 439–471.

[77] J. Singh. *Modern Physics for Engineers*. Weinheim, Germany: Wiley-VCH Verlag GmbH, 1999.

[78] G. A. Sod. ‘A survey of several finite difference methods for systems of nonlinear hyperbolic conservation laws’. In: *Journal of Computational Physics* 27.1 (1978), pp. 1–31.

[79] R. J. Spiteri and S. J. Ruuth. ‘A New Class of Optimal High-Order Strong-Stability-Preserving Time Discretization Methods’. In: *SIAM Journal on Numerical Analysis* 40.2 (2002), pp. 469–491.

[80] G. Strang. ‘On the Construction and Comparison of Difference Schemes’. In: *SIAM Journal on Numerical Analysis* 5.3 (1968), pp. 506–517.

[81] J. C. Strikwerda. *Finite difference schemes and partial differential equations.* 2nd ed. Philadelphia, PA: Society for Industrial and Applied Mathematics, 2004.

[82] H. Tang and G. Warnecke. ‘A note on (2K+1)-point conservative monotone schemes’. In: *ESAIM: Mathematical Modelling and Numerical Analysis* 38.2 (Mar. 2010), pp. 345–357.

[83] Thomas Hillen. `http://www.math.ualberta.ca/~thillen/research.html`.

[84] R. Turpault et al. ‘Multigroup half space moment approximations to the radiative heat transfer equations’. In: *Journal of Computational Physics* 198.1 (2004), pp. 363–371.

[85] B. van Leer. ‘Towards the ultimate conservative difference scheme. IV. A new approach to numerical convection’. In: *Journal of Computational Physics* 23.3 (1977), pp. 276–299.

[86] Z.-A. Wang, T. Hillen and M. Li. ‘Mesenchymal Motion Models in One Dimension’. In: *SIAM Journal on Applied Mathematics* 69.2 (2008), pp. 375–397.

Acknowledgements

First of all, I would like to thank my supervisor Prof. Dr. Mario Ohlberger for giving me the chance to work on this interesting and challenging project and for guidance throughout the whole thesis.

Moreover, I thank all members of the workgroup for their support and the nice working atmosphere.

Special thanks to Felix Schindler who introduced me to the world of DUNE and always helped me when I had questions, even if that meant writing plenty of long emails from Switzerland. Thanks for constant motivation and also for helpful comments on this thesis.

Thanks to René Milk for assistance on profiling and parallelization and solving several problems with DUNE and C++ .

Thanks to Julia Brunken and Barbara Verfürth for good company in the office, helpful discussions and proofreading of this thesis.

Finally, I want to thank my family and friends for always supporting and motivating me throughout my whole studies.

Plagiatserklärung

Hiermit versichere ich, dass die vorliegende Arbeit mit dem Titel *Numerical methods for kinetic equations* selbstständig verfasst worden ist, dass keine anderen Quellen und Hilfsmittel als die angegebenen benutzt worden sind und dass die Stellen der Arbeit, die anderen Werken – auch elektronischen Medien – dem Wortlaut oder Sinn nach entnommenen wurden, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht worden sind.

(Datum, Unterschrift)

Ich erkläre mich mit einem Abgleich der Arbeit mit anderen Texten zwecks Auffindung von Übereinstimmungen sowie mit einer zu diesem Zweck vorzunehmenden Speicherung der Arbeit in eine Datenbank einverstanden.

(Datum, Unterschrift)