Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Term Project

Formulation and Implementation of a Finite Volume Discontinuous Galerkin Scheme for Nonlinear Convection-Diffusion Problems

Adrian Burri and Christoph Winkelmann

February 27, 2003

advised by

Prof. Christoph Schwab
Seminar for Applied Mathematics
Department of Mathematics
Swiss Federal Institute of Technology
Zürich

Abstract

A finite volume discontinuous Galerkin scheme for nonlinear scalar convection-diffusion problems in three dimensions with Dirichlet boundary conditions is formulated. The scheme is implemented using *Concepts* with piecewise linear orthogonal basis functions on unstructured tetrahedral meshes; with linear advection flux and Burger's flux; with upwind, Lax-Friedrichs, Godunov and Osher numerical fluxes; with an adaptive limiting strategy and with an Euler and a total variation diminishing Runge-Kutta timestepping scheme of second order.

Contents

1 Formulation of the FV DG Method

This chapter presents the Finite Volume Discontinuous Galerkin (FV DG) method as considered in [3]. Furthermore, it shows the formulation of the fully discrete scheme.

As we will test our implementation without diffusion first, special attention is drawn to the case of a pure convection problem.

We first present the continuous problem. Then, we discretize in space. The obtained semidiscrete problem is equivalent to a nonlinear system of ordinary differential equations. The fully discrete scheme is finally obtained applying a suitable time discretization to this system. This approach to the numerical solution of initial-boundary value problems via the space semidiscretization is called the method of lines.

1.1 Continuous Problem

Section 1.1.1 covers the full convection-diffusion problem. It is essentially a summary of chapter 1 of [3]. Section 1.1.2 describes the necessary modifications when a problem without diffusion is considered.

1.1.1 Full Convection-Diffusion Problem

Let $\Omega \subset \mathbb{R}^3$ be a bounded polyhedral domain and $T > 0$. We set $Q_T = \Omega \times (0,T)$. By $\overline{\Omega}$ and $\partial\Omega$ we denote the closure and boundary of Ω, respectively. Let us consider the following *initial-boundary value problem*: Find $u : Q_T \to \mathbb{R}$ such that

$$\partial_t u + \boldsymbol{\nabla} \cdot \boldsymbol{f}(u) = \varepsilon \, \Delta u + g \quad \text{in } Q_T \tag{1.1}$$

$$u\,\big|_{\partial\Omega\times(0,T)} = u_D \tag{1.2}$$

$$u(\boldsymbol{x},0) = u^0(\boldsymbol{x}), \quad \boldsymbol{x} \in \Omega \tag{1.3}$$

We assume that the data satisfy the following conditions:

$$\boldsymbol{f} = \{f_s\}_{s=1}^3 : \ f_s \in C^1(\mathbb{R}) \text{ are Lipschitz-continuous and } f_s(0) = 0 \tag{1.4}$$

$$\varepsilon > 0 \tag{1.5}$$

$$g \in C([0,T]; L^2(\Omega)) \tag{1.6}$$

$$u_D = \text{trace of some } u^* \in C([0,T]; H^1(\Omega)) \cap L^\infty(Q_T) \text{ on } \partial\Omega \times (0,T) \tag{1.7}$$

$$u^0 \in L^2(\Omega) \tag{1.8}$$

We use the standard notation for function spaces: $L^p(\Omega)$, $L^p(Q_T)$ denote the Lebesgue spaces, $W^{k,p}(\Omega)$, $H^k(\Omega) = W^{k,2}(\Omega)$ are the Sobolev spaces, $L^p(0,T;X)$ is the Bochner space of functions p-integrable over the interval $(0,T)$ with values in a Banach space X, $C([0,T];X)$ ($C^1([0,T];X)$) is the space of continuous (continuously differential) mappings of the interval $[0,T]$ into X. By $H_0^1(\Omega)$ we denote the subspace of all functions from $H^1(\Omega)$ with zero trace on $\partial\Omega$.

We consider our problem for 3-dimensional domains Ω. The assumption that $f_s(0) = 0$, $s \in \{1,2,3\}$, does not cause any loss of generality, as can be seen from equation (1.1). The functions f_s represent convective terms, $\varepsilon > 0$ is the diffusion coefficient. The diffusion term can be, in general, more complicated (in some cases even nonlinear). It is also possible to consider mixed Dirichlet-Neumann boundary conditions. For simplicity we prescribe Dirichlet conditions on the whole boundary.

A sufficiently regular function satisfying (1.1) – (1.3) pointwise is called a *classical solution*. It is suitable to introduce the concept of a weak solution. To this end we introduce the following *notation*:

$$(u,v) = \int_\Omega u\, v \,\mathrm{d}\boldsymbol{x}, \quad u,\ v \in L^2(\Omega): \quad \text{scalar product in } L^2(\Omega) \tag{1.9}$$

$$a(u,v) = \varepsilon \int_\Omega \boldsymbol{\nabla} u \cdot \boldsymbol{\nabla} v \,\mathrm{d}\boldsymbol{x}, \quad u,\ v \in H^1(\Omega) \tag{1.10}$$

$$b(u,v) = \int_\Omega \boldsymbol{\nabla} \cdot \boldsymbol{f}(u)\, v \,\mathrm{d}\boldsymbol{x}, \quad u \in H^1(\Omega) \cap L^\infty(\Omega),\ v \in L^2(\Omega) \tag{1.11}$$

$$\|u\|_{H^1(\Omega)} = \Big(\int_\Omega \big(|u|^2 + |\boldsymbol{\nabla} u|^2 \big) \,\mathrm{d}\boldsymbol{x} \Big)^{1/2}, \quad u \in H^1(\Omega): \quad \text{norm in } H^1(\Omega) \tag{1.12}$$

$$|u|_{H^1(\Omega)} = \Big(\int_\Omega |\boldsymbol{\nabla} u|^2 \,\mathrm{d}\boldsymbol{x} \Big)^{1/2}, \quad u \in H^1(\Omega): \quad \text{seminorm in } H^1(\Omega) \tag{1.13}$$

It is known that $|\cdot|_{H^1(\Omega)}$ is a norm on $H_0^1(\Omega)$ equivalent to $\|\cdot\|_{H^1(\Omega)}$.

The *weak solution* of (1.1) – (1.3) is defined as a function u satisfying the conditions

$$u - u^* \in L^2(0,T;H_0^1(\Omega)), \quad u \in L^\infty(Q_T) \tag{1.14}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}(u(t),v) + b(u(t),v) + a(u(t),v) = (g(t),v) \tag{1.15}$$
$$\forall\, v \in H_0^1(\Omega) \text{ in the sense of distributions on } (0,T)$$

$$u(0) = u_0 \quad \text{in } \Omega \tag{1.16}$$

By $u(t)$ we denote the function on Ω such that $u(t)\,(\boldsymbol{x}) = u(\boldsymbol{x},t)$, $\boldsymbol{x} \in \Omega$.

It is possible to prove that there exists a unique weak solution. Moreover, it satisfies the condition $\partial_t u \in L^2(Q_T)$. Then (1.15) can be rewritten as

$$(\partial_t u(t),v) + b(u(t),v) + a(u(t),v) = (g(t),v) \tag{1.17}$$
$$\forall\, v \in H_0^1(\Omega) \text{ and almost every } t \in (0,T)$$

We say that u satisfying (1.14) – (1.16) is a *strong solution*, if

$$u \in L^2(0,T;H^2(\Omega)), \quad \partial_t u \in L^2(0,T;H^1(\Omega)) \tag{1.18}$$

It is possible to show that the strong solution u satisfies equation (1.1) pointwise (almost everywhere) and $u \in C([0,T],H^1(\Omega))$.

1.1.2 Pure Convection Problem

In the special case of a pure convection problem, i. e. $\varepsilon = 0$, equation (1.1) reads:

$$\partial_t u + \boldsymbol{\nabla} \cdot \boldsymbol{f}(u) = g \quad \text{in } Q_T \tag{1.19}$$

This equation is hyperbolic, and therefore it is not possible to prescribe Dirichlet boundary conditions on the whole boundary $\partial\Omega$ as in (1.2).

We consider two possible replacements for (1.2). The first possibility is to prescribe periodic boundary conditions. In this case, the domain Ω needs to be the result of a periodic partition of $\mathbb{R}^3$. We choose:

$$\Omega = (0,1)^3 \tag{1.20}$$

The second possibility is to prescribe Dirichlet boundary conditions only on the inflow boundary Γ_{in}:

$$\Gamma_{in}(t) = \partial\Omega \cap \{\boldsymbol{x}; \boldsymbol{n} \cdot \partial_u \boldsymbol{f}(u(\boldsymbol{x},t)) < 0\} \tag{1.21}$$
$$\Gamma_{out}(t) = \partial\Omega \setminus \Gamma_{in}(t) \tag{1.22}$$
$$u(\boldsymbol{x},t) = u_D(\boldsymbol{x},t) \;\forall\; t \in (0,T),\; \boldsymbol{x} \in \Gamma_{in}(t) \tag{1.23}$$

$\boldsymbol{n}$ denotes the outer unit normal of $\partial\Omega$. Note that the boundary condition is now (implicitly) time dependent, i. e. Γ_{in} may grow or shrink over time. Moreover, Γ_{in} depends on u, which is not a-priori known.

1.2 Space Discretization

Essentially, this section is a summary of chapter 2 of [3]. Section 1.2.1 introduces the used triangulations and in section 1.2.2, the broken Sobolev spaces are presented. We then derive the discontinuous variational form in section 1.2.3 and the discrete variational form in section 1.2.4.

1.2.1 Triangulation

Let $\mathcal{T}_h$ $(h > 0)$ denote a partition of the closure $\overline{\Omega}$ of the domain Ω into a finite number of closed 3-dimensional convex polyhedra K with mutually disjoint interiors. We call $\mathcal{T}_h$ a triangulation of $\overline{\Omega}$. Although the usual conforming properties from the finite element method are *not* required , we choose $\mathcal{T}_h$ to be a mesh of tetrahedrons K without hanging nodes.

We shall use the following *notation*. We set $h_K = \mathrm{diam}(K), h = \max_{K\in\mathcal{T}_h} h_K$. By $|K|$ we denote the 3-dimensional Lebesgue measure of K, i. e. the volume. All elements of $\mathcal{T}_h$ will be numbered so that $\mathcal{T}_h = \{K_i\}_{i\in I}$, where $I \subset Z^+ = \{0, 1, 2, \ldots\}$ is a suitable index set. If two elements $K_i,\ K_j \in \mathcal{T}_h$ contain a nonempty open face which is a part of a plane, we call them *neighbours*. We set in this case

$$\Gamma_{ij} = \partial K_i \cap \partial K_j \tag{1.24}$$

and assume that the whole set Γ_{ij} is a part of a plane. (See Figure 1.1, showing a possible situation.) We set

$$F \;=\; \{(i,j);\ i,\ j \in I,\ K_j \text{ is a neighbour of } K_i,\ i < j\} \tag{1.25}$$

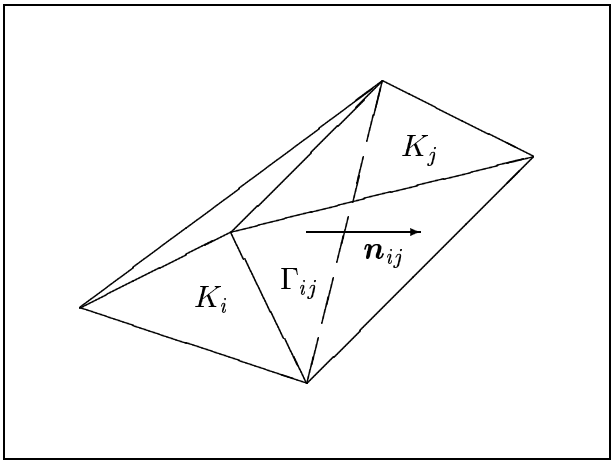

Figure 1.1: Neighbouring elements $K_i,\ K_j$

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

$$G \;=\; \{(i,j);\ i \in I,\ j \in I_b,\ S_j \text{ is a face of } K_i\} \tag{1.26}$$

$$\Gamma_{ij} \;=\; S_j \text{ for } K_i \in \mathcal{T}_h \text{ such that } S_j \subset \partial K_i,\ j \in I_b.$$

$$\overline{F} \;=\; F \cup G \tag{1.27}$$

Furthermore, we use the following notation: $\boldsymbol{n}_{ij}$ = unit outer normal to ∂K_i on the face Γ_{ij} (see Figure 1.1), $|\Gamma_{ij}|$ = 2-dimensional measure of Γ_{ij}, i. e. the area of Γ_{ij}, and $d(\Gamma_{ij}) = \mathrm{diam}(\Gamma_{ij})$.

1.2.2 Broken Sobolev Spaces

Over the triangulation $\mathcal{T}_h$ we define the *broken Sobolev space*

$$H^k(\Omega, \mathcal{T}_h) \;=\; \{v; v|_K \in H^k(K) \;\forall\; K \in \mathcal{T}_h\} \tag{1.28}$$

and equip it with the norm

$$\|v\|_{H^k(\Omega,\mathcal{T}_h)} \;=\; \Big(\sum_{K\in\mathcal{T}_h} \|v\|^2_{H^k(K)} \Big)^{1/2} \tag{1.29}$$

and the seminorm

$$|v|_{H^k(\Omega,\mathcal{T}_h)} \;=\; \Big(\sum_{K\in\mathcal{T}_h} |v|^2_{H^k(K)} \Big)^{1/2} \tag{1.30}$$

For $v \in H^1(\Omega, \mathcal{T}_h)$ we introduce the following notation

$$v|_{\Gamma_{ij}} \;=\; \text{the trace of } v|_{K_i} \quad (\in H^1(K_i)) \text{ on } \Gamma_{ij} \tag{1.31}$$

$$v|_{\Gamma_{ji}} \;=\; \text{the trace of } v|_{K_j} \quad (\in H^1(K_j)) \text{ on } \Gamma_{ji} \tag{1.32}$$

$$\langle v \rangle_{\Gamma_{ij}} \;=\; \frac{1}{2}\left(v|_{\Gamma_{ij}} + v|_{\Gamma_{ji}}\right), \quad [v]_{\Gamma_{ij}} = v|_{\Gamma_{ij}} - v|_{\Gamma_{ji}} \tag{1.33}$$

1.2.3 Discontinous Variational Form

In order to derive the discrete problem, we start from the strong solution u, multiply equation (1.1) by an arbitrary $v \in H^2(\Omega, \mathcal{T}_h)$, integrate over each $K_i \in \mathcal{T}_h$ and apply Green's theorem. We obtain the identity

$$\begin{aligned} &\int_{K_i} \partial_t u \; v \; \mathrm{d}\boldsymbol{x} + \int_{\partial K_i} \boldsymbol{f}(u(t)) \cdot \boldsymbol{n} \; v \; \mathrm{d}\boldsymbol{S} - \int_{K_i} \boldsymbol{f}(u(t)) \cdot \boldsymbol{\nabla} v \; \mathrm{d}\boldsymbol{x} \\ &+\varepsilon \int_{K_i} \boldsymbol{\nabla} u(t) \cdot \boldsymbol{\nabla} v \; \mathrm{d}\boldsymbol{x} - \varepsilon \int_{\partial K_i} (\boldsymbol{\nabla} u(t) \cdot \boldsymbol{n}) \; v \; \mathrm{d}\boldsymbol{S} = \int_{K_i} g(t) \; v \; \mathrm{d}\boldsymbol{x} \end{aligned} \tag{1.34}$$

Here $\boldsymbol{n}$ denotes the unit outer normal to ∂K_i.

Summing (1.34), over all $K_i \in \mathcal{T}_h$ and using the relations

$$\int_{\Gamma_{ij}} \langle \boldsymbol{\nabla} v \rangle \cdot \boldsymbol{n}_{ij} \; [u] \; \mathrm{d}\boldsymbol{S} = 0 \tag{1.35}$$

$$\langle \boldsymbol{\nabla} u \rangle_{\Gamma_{ij}} = \boldsymbol{\nabla} u|_{\Gamma_{ij}} = \boldsymbol{\nabla} u|_{\Gamma_{ji}} \tag{1.36}$$

$$\int_{S_j} (\boldsymbol{\nabla} v \cdot \boldsymbol{n}) \; u(t) \; \mathrm{d}\boldsymbol{S} = \int_{S_j} (\boldsymbol{\nabla} v \cdot \boldsymbol{n}) \; u_D(t) \; \mathrm{d}\boldsymbol{S} \quad \forall \; j \in I_b \tag{1.37}$$

valid for the strong solution u, test functions $v \in H^2(\Omega, \mathcal{T}_h)$ and the boundary condition (1.2), we obtain the identity

$$
\begin{aligned}
&(\partial_t u(t), v) + \sum_{(i,j)\in F} \int_{\Gamma_{ij}} \boldsymbol{f}(u(t)) \cdot \boldsymbol{n}_{ij}\, [v]\, \mathrm{d}\boldsymbol{S} \qquad (1.38)\\
&+ \sum_{(i,j)\in G} \int_{\Gamma_{ij}} \boldsymbol{f}(u(t)) \cdot \boldsymbol{n}_{ij}\, v\, \mathrm{d}\boldsymbol{S} - \sum_{i\in I} \int_{K_i} \boldsymbol{f}(u(t)) \cdot \boldsymbol{\nabla} v\, \mathrm{d}\boldsymbol{x}\\
&+\varepsilon \sum_{i\in I} \int_{K_i} \boldsymbol{\nabla} u(t) \cdot \boldsymbol{\nabla} v\, \mathrm{d}\boldsymbol{x}\\
&-\varepsilon \sum_{(i,j)\in F} \int_{\Gamma_{ij}} \langle \boldsymbol{\nabla} u(t) \rangle \cdot \boldsymbol{n}_{ij} [v]\, \mathrm{d}\boldsymbol{S} + s\, \varepsilon \sum_{(i,j)\in F} \int_{\Gamma_{ij}} \langle \boldsymbol{\nabla} v \rangle \cdot \boldsymbol{n}_{ij} [u(t)]\, \mathrm{d}\boldsymbol{S}\\
&-\varepsilon \sum_{j\in I_b} \int_{S_j} (\boldsymbol{\nabla} u(t) \cdot \boldsymbol{n})\, v\, \mathrm{d}\boldsymbol{S} + s\, \varepsilon \sum_{j\in I_b} \int_{S_j} (\boldsymbol{\nabla} v \cdot \boldsymbol{n})\, u(t)\, \mathrm{d}\boldsymbol{S}\\
&= \int_\Omega g(t)\, v\, \mathrm{d}\boldsymbol{x} + s\, \varepsilon \sum_{j\in I_b} \int_{S_j} (\boldsymbol{\nabla} v \cdot \boldsymbol{n})\, u_D(t)\, \mathrm{d}\boldsymbol{S}
\end{aligned}
$$

We have added some terms consistently with a free constant parameter s. Usual choices are $s = -1$ and $s = 1$. See section 3.2.2 for the consequences on the numerics.

1.2.4 Discrete Variational Form

Now we shall introduce the discrete problem approximating identity (1.38) with the aid of the DG FE method combined with a FV approach. We define the space of discontinuous piecewise polynomial functions:

$$S^{p,-1}(\Omega, \mathcal{T}_h) = \{v; v|_K \in \mathcal{P}^p \ \forall\, K \in \mathcal{T}_h\} \qquad (1.39)$$

$$\mathcal{P}^p = \mathrm{span}\{x_1^{\alpha_1} x_2^{\alpha_2} x_3^{\alpha_3} : \alpha_i \in \mathbb{N}_0,\ \alpha_1 + \alpha_2 + \alpha_3 \le p\} \qquad (1.40)$$

Further, we set

$$S_h = S^{1,-1}(\Omega, \mathcal{T}_h) \qquad (1.41)$$

Now, we derive the discrete problem in such a way that in (1.38) $u(t)$ is approximated by $u_h(t) \in S_h$, v is replaced by $v_h \in S_h$. In the expressions corresponding to the convection terms containing the flux function $\boldsymbol{f}(u)$ we use the approximations $u(t) \approx \pi_u u_h(t)$, $v \approx \pi_v v_h$. π_u and π_v are the limiters used for ansatz- and testfunctions respectively. Limiters are used to avoid spurious oscillations near discontinuities, which in the worst case can lead to numerical instability of the method. We are considering three possible choices of π_u and π_v:

$$\pi_u = \pi_v = \mathbf{1} \qquad (1.42)$$

$$\pi_u = \pi_{ad},\ \pi_v = \mathbf{1} \qquad (1.43)$$

$$\pi_u = \pi_v = \pi_0 \qquad (1.44)$$

π_{ad} denotes the adaptive limiter, π_0 the non-adaptive limiter, and **1** the absence of a limiter. See section 1.4 for the definition of the limiters π_0 and π_{ad}. Similarly as in the FV method the fluxes $\int_{\Gamma_{ij}} \boldsymbol{f}(u)\cdot\boldsymbol{n}_{ij}\,[v]\,\mathrm{d}\boldsymbol{S}$ will be approximated with the aid of the so-called numerical flux $H(u,u',\boldsymbol{n})$ by the expression $\int_{\Gamma_{ij}} H(\pi_u u|_{\Gamma_{ij}}, \pi_u u|_{\Gamma_{ji}}, \boldsymbol{n}_{ij})\,[\pi_v v]\,\mathrm{d}\boldsymbol{S}$ for $(i,j)\in\overline{F}$. Of course, for $(i,j)\in G$ and $\Gamma_{ij}\subset\partial\Omega$, it is necessary to specify the meaning of K_j and $\pi_u u|_{\Gamma_{ji}}$. In the case of prescribed Dirichlet boundary condition on Γ_{ij}, we put

$$K_j = K_i, \quad \pi_u u|_{\Gamma_{ji}} = u_D|_{\Gamma_{ij}}, \quad \text{for } (i,j)\in G \tag{1.45}$$

If no boundary conditions are prescribed on Γ_{ij}, we shall use the simple extrapolation

$$K_j = K_i, \quad \pi_u u|_{\Gamma_{ji}} = \pi_u u|_{\Gamma_{ij}}, \quad \text{for } (i,j)\in G \tag{1.46}$$

In this way we obtain the approximation $b_h(u,v)$ of the convection form $b(u,v)$:

$$\begin{aligned} b_h(u,v) &= \sum_{(i,j)\in F}\int_{\Gamma_{ij}} H(\pi_u u|_{\Gamma_{ij}}, \pi_u u|_{\Gamma_{ji}}, \boldsymbol{n}_{ij})\,[\pi_v v]\,\mathrm{d}\boldsymbol{S} \\ &+ \sum_{(i,j)\in G}\int_{\Gamma_{ij}} H(\pi_u u|_{\Gamma_{ij}}, \pi_u u|_{\Gamma_{ji}}, \boldsymbol{n}_{ij})\,\pi_v v\,\mathrm{d}\boldsymbol{S} - \sum_{i\in I}\int_{K_i}\boldsymbol{f}(u)\cdot\boldsymbol{\nabla} v\,\mathrm{d}\boldsymbol{x} \end{aligned} \tag{1.47}$$

We shall assume that the numerical flux has the following properties:

Assumptions (H):

1. $H(u,v,\boldsymbol{n})$ is defined on $\mathbb{R}^2\times\mathcal{S}_1$, where $\mathcal{S}_1=\{\boldsymbol{n}\in\mathbb{R}^3;\,|\boldsymbol{n}|=1\}$, and is *Lipschitz-continuous* with respect to u, v: for $u,\ v,\ u^*,\ v^*\in\mathbb{R},\ \boldsymbol{n}\in\mathcal{S}_1$ holds

$$|H(u,v,\boldsymbol{n}) - H(u^*,v^*,\boldsymbol{n})| \le C_1(|u-u^*| + |v-v^*|) \tag{1.48}$$

2. $H(u,v,\boldsymbol{n})$ is *consistent*:

$$H(u,u,\boldsymbol{n}) = \boldsymbol{f}(u)\cdot\boldsymbol{n}, \quad u\in\mathbb{R},\ \boldsymbol{n}\in\mathcal{S}_1 \tag{1.49}$$

3. $H(u,v,\boldsymbol{n})$ is *conservative*:

$$H(u,v,\boldsymbol{n}) = -H(v,u,-\boldsymbol{n}), \quad u,\,v\in\mathbb{R},\ \boldsymbol{n}\in\mathcal{S}_1 \tag{1.50}$$

Now we introduce for $u, v \in H^2(\Omega, \mathcal{T}_h)$ the forms

$$\begin{aligned} a_h(u,v) &= \varepsilon \sum_{i\in I} \int_{K_i} \boldsymbol{\nabla} u \cdot \boldsymbol{\nabla} v \ \mathrm{d}x \\ &- \varepsilon \sum_{(i,j)\in F} \int_{\Gamma_{ij}} \langle \boldsymbol{\nabla} u \rangle \cdot \boldsymbol{n}_{ij} [v] \ \mathrm{d}\boldsymbol{S} \ + \ s\,\varepsilon \sum_{(i,j)\in F} \int_{\Gamma_{ij}} \langle \boldsymbol{\nabla} v \rangle \cdot \boldsymbol{n}_{ij} [u] \ \mathrm{d}\boldsymbol{S} \\ &- \varepsilon \sum_{(i,j)\in G} \int_{\Gamma_{ij}} (\boldsymbol{\nabla} u \cdot \boldsymbol{n}_{ij})\ v \ \mathrm{d}\boldsymbol{S} \ + \ s\,\varepsilon \sum_{(i,j)\in G} \int_{\Gamma_{ij}} (\boldsymbol{\nabla} v \cdot \boldsymbol{n}_{ij})\ u \ \mathrm{d}\boldsymbol{S} \end{aligned} \tag{1.51}$$

$$\begin{aligned} \ell_h(v)\ (t) &= (g(t), v) \ + \ s\,\varepsilon \sum_{(i,j)\in G} \int_{\Gamma_{ij}} u_D(t)\ (\boldsymbol{\nabla} v \cdot \boldsymbol{n}_{ij}) \ \mathrm{d}\boldsymbol{S} \\ &+ \ \varepsilon \sum_{(i,j)\in G} \frac{1}{d(\Gamma_{ij})} \int_{\Gamma_{ij}} u_D(t)\ v \ \mathrm{d}\boldsymbol{S} \end{aligned} \tag{1.52}$$

$$J_h(u,v) = \sum_{(i,j)\in F} \frac{1}{d(\Gamma_{ij})} \int_{\Gamma_{ij}} [u]\ [v] \ \mathrm{d}\boldsymbol{S} \ + \sum_{(i,j)\in G} \frac{1}{d(\Gamma_{ij})} \int_{\Gamma_{ij}} u\ v \ \mathrm{d}\boldsymbol{S} \tag{1.53}$$

Since $S_h \subset H^2(\Omega, \mathcal{T}_h)$, the above forms have sense for $u = u_h,\ v = v_h \in S_h$. In the discrete problem the form J_h represents a *stabilization term* and replaces the continuity requirement of the approximate solution in conforming FE methods.

Now we define the *semidiscrete approximate solution* as a function $u_h : Q_T \to I\!R$ satisfying the conditions

$$u_h \in C^1([0,T]; S_h) \tag{1.54}$$

$$\begin{aligned} &(\partial_t u_h(t), v_h) + a_h(u_h(t), v_h) + b_h(u_h(t), v_h) \\ &+\varepsilon J_h(u_h(t), v_h) = \ell_h(v_h)\ (t) \quad \forall\ v_h \in S_h,\ t \in [0,T] \end{aligned} \tag{1.55}$$

$$(u_h(0), v_h) = (u^0, v_h) \quad \forall\ v_h \in S_h \tag{1.56}$$

Taking into account that the exact strong solution u satisfies (1.38), $[u]|_{\Gamma_{ij}} = 0$ and $u|_{\partial\Omega\times(0,T)} = u_D$, we can see that for $v_h \in S_h, t \in (0,T)$ it satisfies (1.55).

1.3 Nonlinear System of ODEs

In order to transform (1.55) into a system of ordinary differential equations (ODEs), we choose a suitable basis φ_m of S_h:

$$\dim(S_h) = N, \quad S_h = \mathrm{span}\{\varphi_m(\boldsymbol{x})\}_{m=1}^N \tag{1.57}$$

We define the column vector $\boldsymbol{\varphi}(\boldsymbol{x})$:

$$\boldsymbol{\varphi}(\boldsymbol{x}) = \{\varphi_m(\boldsymbol{x})\}_{m=1}^N \tag{1.58}$$

Using this definition, u_h and v_h can now be written in this basis:

$$\begin{aligned} u_h(t) &= \sum_{m=1}^N u_m(t)\ \varphi_m(\boldsymbol{x}) = \boldsymbol{u}(t)\cdot\boldsymbol{\varphi}(\boldsymbol{x}) \\ v_h &= \sum_{n=1}^N v_n\ \varphi_n(\boldsymbol{x}) = \boldsymbol{\varphi}(\boldsymbol{x})\cdot\boldsymbol{v} \end{aligned} \tag{1.59}$$

We now plug (1.59) into (1.55) and use the linearity of $b_h(\cdot,\cdot)$ in the second argument, the bilinearity of $(\cdot,\cdot)$, $a_h(\cdot,\cdot)$ and $J_h(\cdot,\cdot)$ as well as the linearity of $l_h(\cdot)$. This yields:

$$\dot{\boldsymbol{u}}^T\boldsymbol{M}\boldsymbol{v} + \boldsymbol{u}^T\boldsymbol{A}\boldsymbol{v} + \boldsymbol{b}(\boldsymbol{u})^T\boldsymbol{v} + \varepsilon\boldsymbol{u}^T\boldsymbol{J}\boldsymbol{v} = \boldsymbol{l}(t)^T\boldsymbol{v} \quad \forall\ \boldsymbol{v}\in I\!R^N \tag{1.60}$$

where

$$\begin{aligned} (\boldsymbol{M})_{mn} &= (\varphi_m,\varphi_n) && (1.61)\\ (\boldsymbol{A})_{mn} &= a_h(\varphi_m,\varphi_n) && (1.62)\\ (\boldsymbol{b}(\boldsymbol{u}))_n &= b_h(\boldsymbol{u}\cdot\boldsymbol{\varphi}(\boldsymbol{x}),\varphi_n) && (1.63)\\ (\boldsymbol{J})_{mn} &= J_h(\varphi_m,\varphi_n) && (1.64)\\ (\boldsymbol{l}(t))_n &= \ell_h(\varphi_n)(t) && (1.65) \end{aligned}$$

As (1.60) has to hold $\forall\ \boldsymbol{v}\in I\!R^N$, it can be rewritten after transposition as:

$$\boldsymbol{M}^T\dot{\boldsymbol{u}} + \boldsymbol{A}^T\boldsymbol{u} + \boldsymbol{b}(\boldsymbol{u}) + \varepsilon\boldsymbol{J}^T\boldsymbol{u} = \boldsymbol{l}(t) \tag{1.66}$$

Rearranging (1.66) yields:

$$\boldsymbol{M}^T\dot{\boldsymbol{u}} = -(\boldsymbol{A}^T\boldsymbol{u} + \boldsymbol{b}(\boldsymbol{u}) + \varepsilon\boldsymbol{J}^T\boldsymbol{u}) + \boldsymbol{l}(t) \tag{1.67}$$

This nonlinear system of ODEs has now the form which allows to plug it into a suitable timestepping scheme. The schemes we implemented are presented in section 2.7.

Note that all terms are fully defined above, but the nonlinear advection term $\boldsymbol{b}(\boldsymbol{u})$ and the right hand side $\boldsymbol{l}(t)$ require integrations which in general cannot be done in closed form. We will therefore use the quadrature formulae presented in section 2.3.2.

1.4 Limiter

In this section, the different limiters implemented in this work are presented.

1.4.1 Non-adaptive Limiter π_0

When piecewise linear shapefunctions are used, i.e. for $v \in S^{1,-1}(\Omega, \mathcal{T}_h)$, the numerical solution near discontinuities produces unphysical oscillations. These oscillations can be damped in the whole computational domain Ω by using a projection of the shapefunctions on the space of piecewise constant function $S^{0,-1}(\Omega, \mathcal{T}_h)$ for the advection terms:

$$v \to \pi_0 v: \quad \pi_0 v|_K = \frac{1}{|K|} \int_K v \, \mathrm{d}\boldsymbol{x} \;\forall\; K \in \mathcal{T}_h \tag{1.68}$$

However, the order of the method is reduced to one when using a π_0-limiting globally. To prevent this degradation in the accuracy, an adaptive limiter can be used, which limits selectively the elements near discontinuities. An adaptive limiter with this ability is presented in the following section.

1.4.2 Adaptive Limiter π_{ad}

The limiting procedure consists of two steps:

1. Choosing the elements which need to be limited.
2. Applying the limiter to those elements.

For the first step, a suitable criterion is needed. The criterion used here was presented in [4]. It is based on the following observation

$$\left[u_h^k\right] = \mathcal{O}(h^2) \text{ for smooth regions} \tag{1.69}$$
$$\left[u_h^k\right] = \mathcal{O}(1) \text{ near discontinuities} \tag{1.70}$$

where $\left[u_h^k\right]$ is the inter-element jump of the numerical solution. Integrating over the edges of an element K yields

$$g_K^1 = \frac{1}{h^6} \int_{\partial K} \left[u_h^k\right]^2 \mathrm{d}\boldsymbol{S} = \mathcal{O}(1) \text{ for smooth regions} \tag{1.71}$$
$$g_K^2 = \frac{1}{h^2} \int_{\partial K} \left[u_h^k\right]^2 \mathrm{d}\boldsymbol{S} = \mathcal{O}(1) \text{ near discontinuities} \tag{1.72}$$

We define the adaptive limiter π_{ad} as

$$u \to \pi_{ad} u: \quad \pi_{ad} u|_K = \begin{cases} \pi_0 u & \text{for} \quad g_K > 1 \\ u & \text{for} \quad g_K \leq 1 \end{cases} \quad \forall\; K \in \mathcal{T}_h \tag{1.73}$$

where g_K is the indicator defined below. Hence the adaptive limiter projects the function $u|_K$ onto the space of constant functions if the indicator criterion $g_K > 1$ is complied.

We use the indicator g_K

$$g_K = \frac{1}{|K_i|^{\frac{\alpha-2}{3}}} \sum_{j,(i,j)\in\bar{F}} \frac{1}{|\Gamma_{ij}|} \int_{\Gamma_{ij}} \left[u_h^k\right]^2 \mathrm{d}\boldsymbol{S}, \;\; \alpha \in [2,6] \tag{1.74}$$

where α is a tuning parameter in the range $[2, 6]$. The influence of α on the limiting behavior can be deduced from equation (1.71) and (1.72): The limiter acts more restrictive as the value of α increases because even smaller deviations from optimal smooth conditions entail a limiting of the element. The element's volume $|K|$ is used to weight the mean value of the jump $\left[u_h^k\right]$ in order to guarantee that the allowed magnitude of the jump decreases in refined meshes. $|\Gamma_{ij}|$ is a measure for the element's face. In [4], $|\Gamma_{ij}|$ was replaced by $\mathrm{diam}(K)$ (in two dimensions).

2 Implementation in Concepts

In this chapter, some of the implementation details are presented. The code written for this work is part of the *Concepts* library, which is presented in [5].

2.1 Class Design

To solve equation (1.1) using the FV DG method, additional structures needed to be implemented in *Concepts*. Some of them are just refinements of available structures, which will be addressed in section 2.1.1. In addition, some new concepts were introduced, namely numerical fluxes and element pairs which reflect the finite volume perspective of the method. These topics will be covered in section 2.1.2.

2.1.1 Extensions

As presented in chapter 1.2.2, the spaces of the FV DG method are special in the sense that they have discontinuous shape functions. This is reflected by the introduction of the abstract class `FvdgSpace` which reflects a general FV DG space. Spaces with different polynomial degree are implemented in separate classes (`FvdgSpaceP0, FvdgSpaceP1`) as can be seen in figure 2.1. These spaces correspond to the spaces $S^{0,-1}(\Omega, \mathcal{T})$ and $S^{1,-1}(\Omega, \mathcal{T})$ introduced in equation (1.39). Similarly, to every concrete FV DG space belongs a class for an FV DG element (`FvdgP0TetElem, FvdgP1TetElem`) which is derived from the abstract base class `FvdgElement`. Note that `FvdgElement` models only tetrahedral elements and that the rest of the FV DG classes presented here rely on this by hard-wiring the quadrature rules. The element classes for the FV DG methods offer additional functionality that facilitates the access to information related to the element's representation in physical space, i.e. physical coordinates of its vertices and edge centers and the like. This is necessary since in the FV DG method - opposite to classical FE methods - expressions are evaluated in the physical space and not on the reference elements. See section 2.5 for further details.

The application of the FV DG method to hyperbolic and parabolic equations requires the use of the timestepping classes provided by *Concepts*. Especially, the classes for the time dependent part of the diffusion $l_h(v)$ in equation (1.52) (see section 2.6) inherit from `TimeLinearForm`. Several new timestepping strategies were added, which are presented

in section 2.7. Some additional timestep strategies allow the use of a limiter. The concept of limiters is introduced in section 1.4. To optimize the performance of the timestepping by exploiting the diagonal structure of the mass matrix (see section 2.3 for details), a class for diagonal matrices (`DiagonalMatrix`) along with facilities to easily invert them (`DiagonalSolver, DiagonalSolverFabric`) was implemented and added to the *Concepts* namespace.

2.1.2 New Concepts

As presented in section 1.2.3, terms in the FV DG method often have face integral contributions:

$$\sum_{(i,j)\in F} \int_{\Gamma_{ij}} f(u_h, v_h) \mathrm{d}\boldsymbol{S} \tag{2.1}$$

Since *Concepts* provides no means to store the adjacency lists F, G in (1.25) and (1.26), the concept of element pairs and element pair lists (which allow to iterate over a list of element pairs in the same way this is done for element lists) was introduced. The class `ElementPairList` acts as a container class for objects of `ElementPair`. `ElementPairList` and `ElementPair` belong to the *Concepts* namespace. The class `FvdgElementPair` represents a specialization of `ElementPair`, as can be seen in 2.1. Additional information on the common face of the pair's elements are stored for performance reasons. The element pairs are used by the class `Advection` for the face integral terms in the convection terms and by the class `Assembly` for the assembly of the matrices belonging to the face integral terms in the diffusion term. For `Assembly`, a special form of the class `BilinearForm`, the class `FvdgBilinearForm` representing must be provided to be able to exploit the additional information in `FvdgElementPair`. The bilinear terms $a_h(u, v)$, $J_h(u, v)$ from equations (1.51) and (1.53) are modelled by the classes `LaplaceVolBf, LaplaceInnerBf, LaplaceBoundaryBf` and `PenaltyBf`. The linear term $\ell_h(v)(t)$ is collapsed into a single linear form `FvdgTimeRhsDiff`, which inherits from `TimeLinearForm` so that it can be used as a time dependent right hand side in the timestepping algorithms.

The class `Advection` is the representation of the non-linear convection term $b_h(u, v)$ in equation (1.47). It is dependent on the concepts of flux functions and numerical fluxes, as can be seen in figure 2.2. The flux functions available are a linear advection flux (class `LinearAdvectionFlux`) and Burger's flux (class `BurgerFlux`). These flux functions are described in section 2.4.1. The numerical flux classes are `Upwind, LaxFriedrichs, Godunov, Osher`, the form of which is described in section 2.4.2. The limiter classes implemented are `FvdgPi0Limiter, FvdgAdaptiveLimiter`. See section 1.4 for more details on the use and form of them.

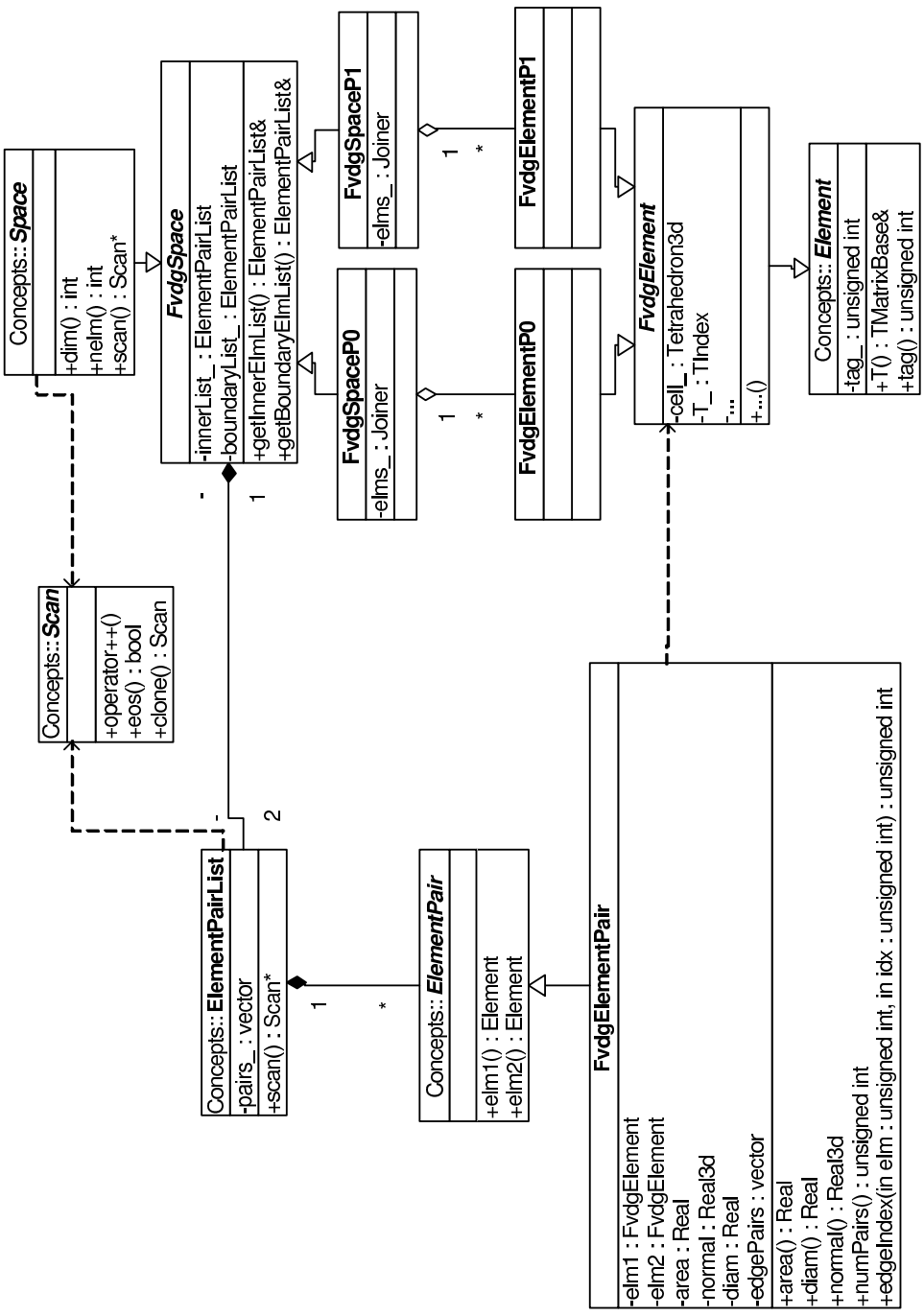

Figure 2.1: Element and Space Classes. This UML (Unified Modelling Language) diagram shows the relationsships between the core classes of the FV DG method implemented. Especially, the role of the element pair `ElementPair` and element pair list `ElementPairList` as an extension of the scanner concept, which allows to scan not only over the element, but also to scan over the faces of element pairs.

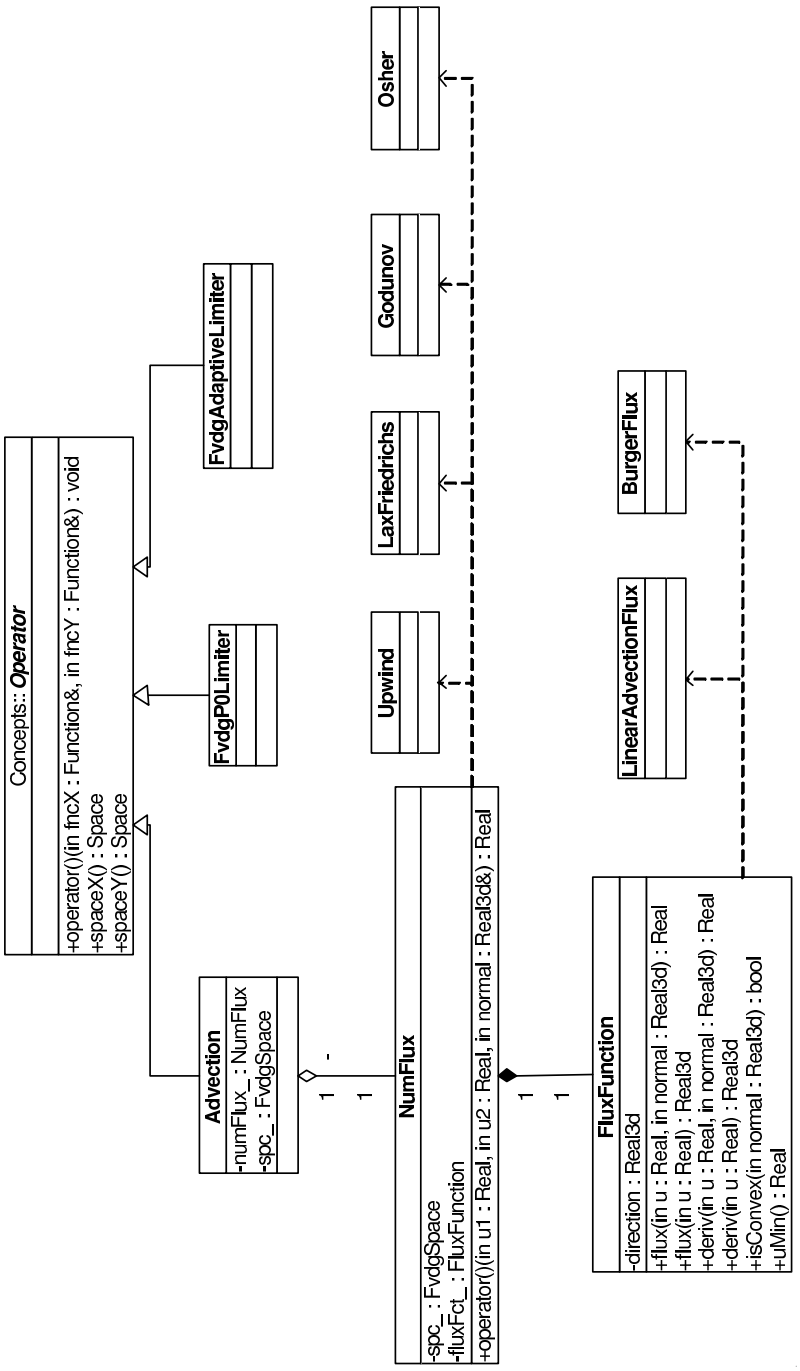

Figure 2.2: Advection Classes. This UML diagram shows the relationship between the core classes solving the advection parts in equation (1.1). In order to optimize the performance, the numerical flux `NumFlux` and the flux function `FluxFunction` can be thought of as a concept, the implementation of which is realized in the associated classes `Upwind`, `LaxFriedrichs`, `Godunov`, `Osher`, resp. `LinearAdvectionFlux`, `BurgerFlux`.

2.1.3 Utility Classes

The utility classes don't directly model any of the terms in the equations (1.51)–(1.53) but are used to investigate the methods properties.

The class `Norm` collects methods to calculate the following norms:

- Discrete Maximum Norm:
  calculates the maximum norm of a function $u_h(\boldsymbol{x}) = \boldsymbol{u} \cdot \boldsymbol{\varphi}(\boldsymbol{x}) \in S_h$

$$\text{discrete } L^\infty \text{ norm}: \quad \boldsymbol{u} \to \max_{\boldsymbol{x}\in\Omega} \boldsymbol{u} \cdot \boldsymbol{\varphi}(\boldsymbol{x}) = \|u_h\|_{L^\infty(\Omega,\mathcal{T}_h)} \tag{2.2}$$

- Discrete Matrix Norm:
  calculates the norm of a function $u_h(\boldsymbol{x}) = \boldsymbol{u} \cdot \boldsymbol{\varphi}(\boldsymbol{x}) \in S_h$, induced by the matrix $\boldsymbol{B}$

$$\begin{aligned} \text{discrete matrix norm}: \quad & \boldsymbol{u}, \boldsymbol{B} \to \sqrt{\boldsymbol{u}^T \boldsymbol{B} \boldsymbol{u}} \\ \text{note:} \quad & \sqrt{\boldsymbol{u}^T \boldsymbol{M} \boldsymbol{u}} = \|u_h\|_{L^2(\Omega,\mathcal{T}_h)} \end{aligned} \tag{2.3}$$

- Broken H^1 Seminorm:
  calculates the broken H^1 seminorm (1.30) of the difference between a function $u_h(\boldsymbol{x}) = \boldsymbol{u} \cdot \boldsymbol{\varphi}(\boldsymbol{x}) \in S_h$ and another function $u^* \in C^1(\Omega)$. The seminorm is calculated using the quadrature rule presented in section 2.3.2. It is therefore only exact for functions $u^* \in \mathcal{P}^2(\Omega)$.

$$\begin{aligned} \text{broken } H^1 \text{ seminorm}: \boldsymbol{u}, \boldsymbol{\nabla} u^* \quad &\to \quad \|\boldsymbol{\nabla}(\boldsymbol{u} \cdot \boldsymbol{\varphi}(\cdot)) - \boldsymbol{\nabla} u^*\|_{L^2(\Omega,\mathcal{T}_h)} \\ &= \quad |\boldsymbol{u} \cdot \boldsymbol{\varphi}(\cdot) - u^*|_{H^1(\Omega,\mathcal{T}_h)} \end{aligned} \tag{2.4}$$

- Broken Maximum Norm:
  calculates the broken L^∞ norm of the difference between a function $u_h(\boldsymbol{x}) = \boldsymbol{u} \cdot \boldsymbol{\varphi}(\boldsymbol{x}) \in S_h$ and another function $u^* \in C^1(\Omega)$.

$$\text{broken } L^\infty \text{ norm}: \boldsymbol{u}, u^* \quad \to \quad \|\boldsymbol{u} \cdot \boldsymbol{\varphi}(\cdot) - u^*\|_{L^\infty(\Omega,\mathcal{T}_h)} \tag{2.5}$$

- Broken L^1 Norm:
  calculates the broken L^1 norm of the difference between a function $u_h(\boldsymbol{x}) = \boldsymbol{u} \cdot \boldsymbol{\varphi}(\boldsymbol{x}) \in S_h$ and another function $u^* \in C^1(\Omega)$. The norm is calculated using the quadrature rule presented in section 2.3.2.

$$\text{broken } L^1 \text{ norm}: \boldsymbol{u}, u^* \quad \to \quad \|\boldsymbol{u} \cdot \boldsymbol{\varphi}(\cdot) - u^*\|_{L^1(\Omega,\mathcal{T}_h)} \tag{2.6}$$

The class `NodalBurgerError` calculates an error measure for Burger's flux, which is described in section 2.4.1.

The class `TimeStepEstimator` calculates an estimate for the timestep size based on an initial condition and the CFL criterion

$$\Delta t \quad \leq \quad c \, \frac{h}{C_1} \tag{2.7}$$

where C_1 is defined in (1.48). As we do not know C_1 explicitly, we estimate the timestep as follows:

$$\Delta t_{est} \quad = \quad c \, \min_{K \in \mathcal{T}_h} \, \min_{\boldsymbol{x} \in V(K)} \frac{h_K}{|\boldsymbol{f}'(u_h^0(\boldsymbol{x}))|} \tag{2.8}$$

where u_h^0 is the projected initial condition and h_K is an appropriate measure for the size of $K \in \mathcal{T}_h$. We get the best results putting $h_K = |K|^{1/3}$. The constant c needs to be chosen by the user to suit the stability criterion of the timestepping scheme used.

2.2 Mesh generation

2.2.1 General Nonperiodic Domains

For the generation of meshes of general domains, i. e. $\Omega \to \mathcal{T}_h$, we do not implement our own code. Instead we use *Netgen* [8]. In order to get the mesh from *Netgen* into *Concepts*, we want to use the existing mesh import classes from *Concepts*. We therefore wrote an add-in to *Netgen* which allows to export the mesh in the suitable file format.

The mesh import interface of *Concepts* requires three files. The first file contains a numbered list of nodes with their coordinates. The second file contains a list of tetrahedrons, defined by four node numbers. The order of the nodes must be strictly right- or left-handed throughout the whole file. The third file contains a list of boundary triangles, defined by three node numbers and its attribute, i. e. the boundary condition. The mesh import interface of *Concepts* is able to reconstruct all additional information like shared edges and inner faces from the given data in time $\mathcal{O}(N)$ where N denotes the number of tetrahedrons. As reading the files also takes time $\mathcal{O}(N)$, it is not worth passing more information from *Netgen* to *Concepts*.

The information needed to write the three required files is directly accessible in the programming interface of *Netgen*. All our add-in does therefore is to query this information and to write it into the three files.

2.2.2 Periodic Unit Cube

The most natural representation of periodic domains is topologic periodicity: A node that is - geometrically spoken - a periodic image of another node is represented topologically as the same node. The same holds for edges and faces. As the interface for

the mesh import into *Concepts* does not support this topologic periodicity but *Concepts* itself does, we wrote a separate class representing a periodic unit cube.

The unit cube $\Omega = (0,1)^3$ is split into n^3 cubes first. These cubes are then split into six tetrahedrons each. The value of $n \in \mathbb{N}$ can be specified by the user.

This procedure leads to a structured mesh. The structure in the mesh may cause undesired effects in numerical tests. We therefore add the possibility to move each inner point of the cube randomly. The displacements are chosen uniformly in $(-j/n, j/n)^3$. The jitter value j can be chosen by the user. The effect of different values of j is tested in section 3.1.3.

2.3 Shape Functions

As mentioned in section 1.2.2, the basis functions φ in the FV DG method are chosen to be piecewise polynomial, i.e.

$$\varphi \in S^{p,-1}(\Omega, \mathcal{T}_h) = \{\varphi; \varphi|_K \in \mathcal{P}^p(K) \ \forall \ K \in \mathcal{T}_h\} \tag{2.9}$$

The support of a basis function is exactly one single element K_i. Thus, each global basis function φ_k can be identified with an element's local shape function N_i^l.

In the present work, only piecewise constant basis functions ($p = 0$) and piecewise linear basis functions ($p = 1$) for tetrahedral meshes are implemented. In the following, the formulation and the properties of the corresponding linear shape functions are described.

2.3.1 Orthogonal Basis

Like in conforming FE methods, the shape functions are defined on a reference element $\hat{K}$ depicted in figure 2.4. Accordingly, one defines the reference element shape functions $\hat{N}_i$ and gets the related shape functions N_i by an affine transformation of $\hat{K}$ onto the element in the physical space, K.

Since u_h, v_h are discontinous at element interfaces, one is not restricted to use nodal shape functions. Instead, the reference element shape functions $\hat{N}_i$ are chosen to be orthogonal on the reference element:

$$\int_{\hat{K}} \hat{N}_i \hat{N}_j \, \mathrm{d}\boldsymbol{x} = c_i \ \delta_{ij} \tag{2.10}$$

As $(\boldsymbol{M})_{mn} = (\varphi_m, \varphi_n)$, the mass matrix $\boldsymbol{M}$ becomes diagonal, which is crucial for the method's performance, since the mass matrix needs to be inverted in every timestep. Still, there are several degrees of freedom on how to actually choose the form of the shape functions. The ones used here have the following properties in addition to their orthogonality:

- The shape function $\hat{N}_0 = 1$ is constant, $\hat{N}_i, i \in \{1, 2, 3\}$ are non-constant and have vanishing mean value.

- The non-constant shape functions vanish on all but two edge centers of the tetrahedron.

These properties determine except for a constant factor the form of the reference element shape functions:

$$\hat{N}_0 = 1 \tag{2.11}$$
$$\hat{N}_1 = 2\xi_2 + 2\xi_3 - 1 \tag{2.12}$$
$$\hat{N}_2 = 2\xi_1 + 2\xi_3 - 1 \tag{2.13}$$
$$\hat{N}_3 = 2\xi_1 + 2\xi_2 - 1 \tag{2.14}$$

Throughout the implementation, the special form of the shape functions is exploited, especially when quadrature is used.

2.3.2 Quadrature

The FV DG method contains integrals over element faces and volumes. Therefore, appropriate quadrature rules suited to the chosen shape functions are needed. The two first quadrature rules presented below, which are used in most cases, are second order accurate. However, in the convection term in equation (1.1), the accuracy of the quadrature rule needs to be adapted to the form of the numerical flux. In the case of the Burger flux, the quadrature needs to be exact for polynomials of degree 3. The last quadrature rule presented here is exact for polynomials of degree 5, which makes it a reasonable choice for the face integrals of the convection terms.

Face Quadrature Let $\hat{E}$ be the set of edge centers of a face in the reference element $\hat{\Gamma}$ depicted in figure 2.3. Then the quadrature rule for the face integral is

$$\int_{\hat{\Gamma}} \hat{f}(\boldsymbol{\xi}) \mathrm{d}\boldsymbol{\xi} \approx \frac{1}{6} \sum_{\hat{\boldsymbol{e}} \in \hat{E}} \hat{f}(\hat{\boldsymbol{e}}) \tag{2.15}$$

A transformation into physical space yields the following result:

$$\int_{\Gamma} f(\boldsymbol{x}) \mathrm{d}\boldsymbol{x} \approx \frac{|\Gamma|}{3} \sum_{\boldsymbol{e} \in E} f(\boldsymbol{e}) \tag{2.16}$$

where E is the set of edge centers of the face Γ in physical coordinates and $|\Gamma|$ its area.

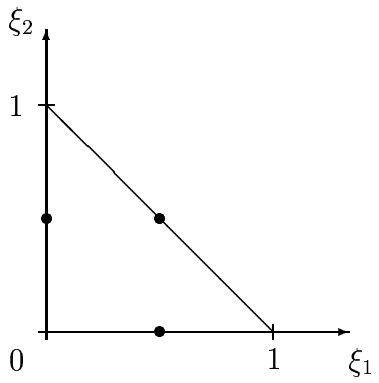

Figure 2.3: Reference element of a face. The abscissae of the quadrature rule are shown with filled circles.

Volume Quadrature Let $\hat{V}$ be the set of the corners and $\hat{E}$ be the set of its edge centers of the reference element $\hat{K}$ depicted in figure 2.4. Then the quadrature rule of the volume integral is

$$\int_{\hat{K}} \hat{f}(\boldsymbol{\xi}) \mathrm{d}\boldsymbol{\xi} \approx \frac{1}{120} \left(\sum_{\hat{\boldsymbol{e}} \in \hat{E}} 4\hat{f}(\hat{\boldsymbol{e}}) - \sum_{\hat{\boldsymbol{v}} \in \hat{V}} \hat{f}(\hat{\boldsymbol{v}}) \right) \tag{2.17}$$

A transformation into physical space yields the following result:

$$\int_{K} f(\boldsymbol{x}) \mathrm{d}\boldsymbol{x} \approx \frac{|K|}{20} \left(\sum_{\boldsymbol{e} \in E} 4f(\boldsymbol{e}) - \sum_{\boldsymbol{v} \in V} f(\boldsymbol{v}) \right) \tag{2.18}$$

where V is the set of corners and E the set of the edge centers of the element K in physical coordinates and $|K|$ is its volume.

Higher Order Face Quadrature Let $\hat{Q}$ be the set of quadrature points of a face in the reference element. Then the quadrature rule for the face integral is

$$\int_{\hat{\Gamma}} \hat{f}(\boldsymbol{\xi}) \mathrm{d}\boldsymbol{\xi} \approx \frac{1}{2} \sum_{\hat{\boldsymbol{q}} \in \hat{Q}} w(\hat{\boldsymbol{q}}) \hat{f}(\hat{\boldsymbol{q}}) \tag{2.19}$$

where $w(\hat{\boldsymbol{q}})$ are the weights belonging to an abscissa $\hat{\boldsymbol{q}}$. The values of the weights and the abscissae in barycentric coordinates are given in the table below:

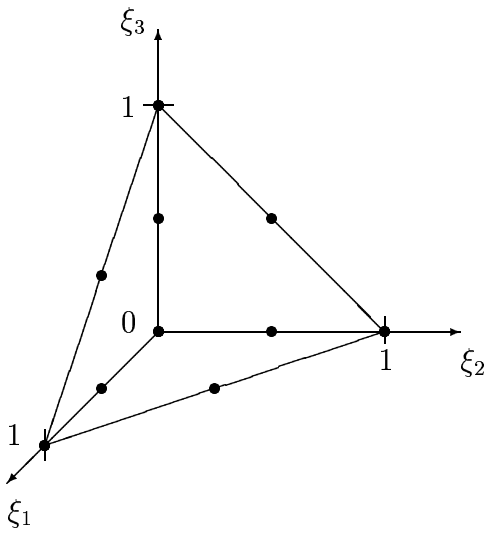

Figure 2.4: Reference tetrahedron. The abscissae of the quadrature rule are shown with filled circles.

Index	Coordinates	Weights
0	$\left(\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right)$	$\frac{9}{40}$
1	$\left(\frac{1}{21}(9+2\sqrt{15}), \frac{1}{21}(6-\sqrt{15}), \frac{1}{21}(6-\sqrt{15})\right)$	$\frac{155-\sqrt{15}}{1200}$
2	$\left(\frac{1}{21}(6-\sqrt{15}), \frac{1}{21}(9+2\sqrt{15}), \frac{1}{21}(6-\sqrt{15})\right)$	$\frac{155-\sqrt{15}}{1200}$
3	$\left(\frac{1}{21}(6-\sqrt{15}), \frac{1}{21}(6-\sqrt{15}), \frac{1}{21}(9+2\sqrt{15})\right)$	$\frac{155-\sqrt{15}}{1200}$
4	$\left(\frac{1}{21}(9-2\sqrt{15}), \frac{1}{21}(6+\sqrt{15}), \frac{1}{21}(6+\sqrt{15})\right)$	$\frac{155+\sqrt{15}}{1200}$
5	$\left(\frac{1}{21}(6+\sqrt{15}), \frac{1}{21}(9-2\sqrt{15}), \frac{1}{21}(6+\sqrt{15})\right)$	$\frac{155+\sqrt{15}}{1200}$
6	$\left(\frac{1}{21}(6+\sqrt{15}), \frac{1}{21}(6+\sqrt{15}), \frac{1}{21}(9-2\sqrt{15})\right)$	$\frac{155+\sqrt{15}}{1200}$

A transformation into physical space yields the following result:

$$\int_{\hat{\Gamma}} \hat{f}(\boldsymbol{\xi}) \mathrm{d}\boldsymbol{\xi} \approx |\Gamma| \sum_{\hat{\boldsymbol{q}} \in \hat{Q}} w(\hat{\boldsymbol{q}}) \hat{f}(\hat{\boldsymbol{q}}) \tag{2.20}$$

where $|\Gamma|$ is the area of the face and where $\boldsymbol{q} \in Q$ represent the images of the elements in the set $\hat{Q}$ in physical space.

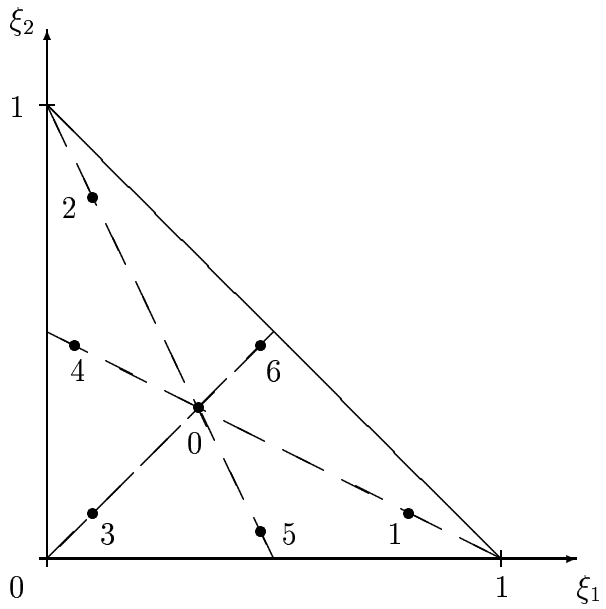

Figure 2.5: Reference element of a face. The abscissae of the quadrature rule are shown with filled circles.

2.4 Advection

The convection term $b_h(u, v)$ in equation (1.47) is defined by the use of a flux vector $\boldsymbol{f}(u)$ and a numerical flux $H(u, v, \boldsymbol{n})$. In the following, the form of the implemented flux vectors and numerical fluxes is presented.

2.4.1 Flux Formulation

The flux vectors $\boldsymbol{f}(u)$ implemented here are generalizations of one-dimensional flux functions to the three dimensional case. This implies that all vectors have the form

$$\boldsymbol{f}(u) = f(u) \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \tag{2.21}$$

where $f(u)$ is the flux function of the one-dimensional case. Note that the value of the flux is only dependent on the variable u, not on the coordinates $\boldsymbol{x}$ or the time t.

Linear Advection The linear advection flux has the following form:

$$\boldsymbol{f}_{LA}(u) = au \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \tag{2.22}$$

It propagates the solution at a constant velocity a in the direction $(1,1,1)^T$.

Burger The Burger flux has the following form:

$$\boldsymbol{f}_{B}(u) = \frac{1}{2}u^2 \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \tag{2.23}$$

Its one dimensional flux function is strictly convex, which will be exploited by the numerical fluxes.

2.4.2 Numerical Fluxes in 3D

Numerical fluxes in one space dimension can be obtained by considering the one-imensional Riemann problem

$$u_0 = \begin{cases} u_L & for \quad x < 0 \\ u_R & for \quad x > 0 \end{cases} \tag{2.24}$$

Similarly, the three-dimensional counterparts to the one-dimensional numerical fluxes can be derived by a generalization of the Riemann problem to three space dimensions, as explained in [6], [7].

In most cases, it is enough to replace the scalar flux function $f(u)$ by a term of the form $\hat{f}(u) = \boldsymbol{f}(u) \cdot \boldsymbol{n}$, where $\boldsymbol{n}$ is the outer normal vector of the face in consideration. However, care must be taken regarding the convexity, which is now dependent on the direction of the outer normal. A one-dimensional flux function $f(u)$ which is convex thus leads to flux functions $\hat{f}(u)$ that are either convex or concave. Some of the schemes described below, namely the Godunov and the Osher flux, are only valid for convex flux functions. To adhere to this condition, one can use the property (1.50) of numerical flux functions. Hence, one needs to evaluate the form of the numerical flux that belongs to the convex form of $\hat{f}(u)$.

Upwind

$$H(u_L, u_R, \boldsymbol{n}) = \begin{cases} \boldsymbol{f}(u_L) \cdot \boldsymbol{n} & for \quad \boldsymbol{f}'(\bar{u}) \cdot \boldsymbol{n} > 0 \\ \boldsymbol{f}(u_R) \cdot \boldsymbol{n} & for \quad \boldsymbol{f}'(\bar{u}) \cdot \boldsymbol{n} < 0 \end{cases} \tag{2.25}$$

$$\bar{u} = \frac{1}{2}(u_L + u_R) \tag{2.26}$$

Local Lax-Friedrichs

$$H(u_L, u_R, \boldsymbol{n}) = \frac{1}{2}\left(\boldsymbol{f}(u_L)\cdot\boldsymbol{n} + \boldsymbol{f}(u_R)\cdot\boldsymbol{n}\right) - \frac{1}{2\lambda}\left(u_R - u_L\right) \tag{2.27}$$

$$\lambda = \frac{1}{\max_u |\boldsymbol{f}'(u)\cdot\boldsymbol{n}|}, \quad u \in [u_L, u_R] \tag{2.28}$$

The equation for λ can be simplified for convex flux functions:

$$\lambda = \frac{1}{\max(|\boldsymbol{f}'(u_L)\cdot\boldsymbol{n}|, |\boldsymbol{f}'(u_R)\cdot\boldsymbol{n}|)} \tag{2.29}$$

Godunov

$$H(u_L, u_R, \boldsymbol{n}) = \begin{cases} \boldsymbol{f}(u_L)\cdot\boldsymbol{n} & for \quad s > 0 \wedge u_L > q_s \\ \boldsymbol{f}(u_R)\cdot\boldsymbol{n} & for \quad s < 0 \wedge u_R < q_s \\ \boldsymbol{f}(q_s)\cdot\boldsymbol{n} & for \quad u_L < q_s < u_R \end{cases} \tag{2.30}$$

$$s = \frac{\boldsymbol{f}(u_L)\cdot\boldsymbol{n} - \boldsymbol{f}(u_R)\cdot\boldsymbol{n}}{u_L - u_R} \tag{2.31}$$

$$q_s = (\boldsymbol{f}')^{-1}(\boldsymbol{0}) \tag{2.32}$$

Here, s is the shock velocity and q_s is the argument at the stationary point of $\boldsymbol{f}(u)$. Note that it is not possible to determine q_s analytically in a general three-dimensional flux function, but the special formulation of the flux functions in this work allows the use of the argument at the stationary point of the related one-dimensional flux function $f(u)$.

Osher

$$H(u_L, u_R, \boldsymbol{n}) = \begin{cases} \boldsymbol{f}(u_L)\cdot\boldsymbol{n} & \text{for} \quad u_L > q_s \wedge u_R > q_s \\ \boldsymbol{f}(u_R)\cdot\boldsymbol{n} & \text{for} \quad u_L < q_s \wedge u_R < q_s \\ (\boldsymbol{f}(u_R) + \boldsymbol{f}(u_R) - \boldsymbol{f}(q_s))\cdot\boldsymbol{n} & \text{for} \quad u_L > q_s \wedge u_R < q_s \\ \boldsymbol{f}(q_s)\cdot\boldsymbol{n} & \text{for} \quad u_L < q_s \wedge u_R > q_s \end{cases} \tag{2.33}$$

$$q_s = (\boldsymbol{f}')^{-1}(\boldsymbol{0}) \tag{2.34}$$

For q_s the same argument as for the Godunov scheme holds.

2.5 Diffusion

2.5.1 Diffusion Matrix

The diffusion matrix is defined by the following equation introduced in section 1.2.4:

$$(\boldsymbol{A})_{mn} = a_h(\varphi_m, \varphi_n) \tag{2.35}$$

The bilinear form a_h is defined by equation (1.51). We split it up into a volume part, a part from inner faces and a part from boundary faces:

$$a_h(\varphi_m, \varphi_n) = a_h^{vol}(\varphi_m, \varphi_n) + a_h^{inn}(\varphi_m, \varphi_n) + a_h^{bnd}(\varphi_m, \varphi_n) \tag{2.36}$$

$$a_h^{vol}(\varphi_m, \varphi_n) = \varepsilon \sum_{i \in I} \int_{K_i} \boldsymbol{\nabla}\varphi_m \cdot \boldsymbol{\nabla}\varphi_n \, \mathrm{d}x \tag{2.37}$$

$$\begin{aligned} a_h^{inn}(\varphi_m, \varphi_n) = & -\varepsilon \sum_{(i,j) \in F} \int_{\Gamma_{ij}} \langle \boldsymbol{\nabla}\varphi_m \rangle \cdot \boldsymbol{n}_{ij}[\varphi_n] \, \mathrm{d}\boldsymbol{S} \\ & + s\,\varepsilon \sum_{(i,j) \in F} \int_{\Gamma_{ij}} \langle \boldsymbol{\nabla}\varphi_n \rangle \cdot \boldsymbol{n}_{ij}[\varphi_m] \, \mathrm{d}\boldsymbol{S} \end{aligned} \tag{2.38}$$

$$\begin{aligned} a_h^{bnd}(\varphi_m, \varphi_n) = & -\varepsilon \sum_{(i,j) \in G} \int_{\Gamma_{ij}} (\boldsymbol{\nabla}\varphi_m \cdot \boldsymbol{n}_{ij}) \, \varphi_n \, \mathrm{d}\boldsymbol{S} \\ & + s\,\varepsilon \sum_{(i,j) \in G} \int_{\Gamma_{ij}} (\boldsymbol{\nabla}\varphi_n \cdot \boldsymbol{n}_{ij}) \, \varphi_m \, \mathrm{d}\boldsymbol{S} \end{aligned} \tag{2.39}$$

Analogously, we define:

$$\boldsymbol{A} = \boldsymbol{A}^{vol} + \boldsymbol{A}^{inn} + \boldsymbol{A}^{bnd} \tag{2.40}$$

$$(\boldsymbol{A}^{vol})_{mn} = a_h^{vol}(\varphi_m, \varphi_n) \tag{2.41}$$

$$(\boldsymbol{A}^{inn})_{mn} = a_h^{inn}(\varphi_m, \varphi_n) \tag{2.42}$$

$$(\boldsymbol{A}^{bnd})_{mn} = a_h^{bnd}(\varphi_m, \varphi_n) \tag{2.43}$$

We introduce the matrices $\boldsymbol{T}_k$ such that the column vector of global basisfunctions $\boldsymbol{\varphi}(\boldsymbol{x})$ can be written in terms of the column vectors of local shape functions $\boldsymbol{N}_k(\boldsymbol{x}) = \{N_k^p\}_{p=0}^3$ as follows:

$$\boldsymbol{\varphi}(\boldsymbol{x}) = \sum_{k \in I} \boldsymbol{T}_k^T \boldsymbol{N}_k(\boldsymbol{x}) \tag{2.44}$$

The volume diffusion matrix $\boldsymbol{A}^{vol}$ can then be assembled from local volume diffusion matrices $\boldsymbol{A}_i^{vol}$:

$$\boldsymbol{A}^{vol} = \sum_{i \in I} \boldsymbol{T}_i^T \boldsymbol{A}_i^{vol} \boldsymbol{T}_i \tag{2.45}$$

$$(\boldsymbol{A}_i^{vol})_{pq} = a_{loc}^{vol}(i,p,q) = \varepsilon \int_{K_i} \boldsymbol{\nabla} N_i^p \cdot \boldsymbol{\nabla} N_i^q \, \mathrm{d}\boldsymbol{x} \tag{2.46}$$

This volume assembly process was already implemented in *Concepts* and could be used. Only the local bilinear form $a_{loc}^{vol}(i,p,q)$ was implemented in `LaplaceVolBf`.

The inner diffusion matrix $\boldsymbol{A}^{inn}$ can be assembled from local inner diffusion matrices $\boldsymbol{A}_{ijkl}^{inn}$:

$$\boldsymbol{A}^{inn} = \sum_{(i,j)\in F} \boldsymbol{T}_i^T \boldsymbol{A}_{ijii}^{inn} \boldsymbol{T}_i + \boldsymbol{T}_i^T \boldsymbol{A}_{ijij}^{inn} \boldsymbol{T}_j + \boldsymbol{T}_j^T \boldsymbol{A}_{ijji}^{inn} \boldsymbol{T}_i + \boldsymbol{T}_j^T \boldsymbol{A}_{ijjj}^{inn} \boldsymbol{T}_j \tag{2.47}$$

$$(\boldsymbol{A}_{ijkl}^{inn})_{pq} = a_{loc}^{inn}(i,j,k,l,p,q) \tag{2.48}$$

This inner assembly process is performed by the new class `Assembly`. It takes the local bilinear form $a_{loc}^{inn}(i,j,k,l,p,q)$ (`LaplaceBoundaryBf`) and builds the matrix $\boldsymbol{A}^{inn}$.

The local bilinear form $a_{loc}^{inn}(i,j,k,l,p,q)$ is defined as:

$$\begin{aligned} a_{loc}^{inn}(i,j,k,l,p,q) &= -(\delta_{il} - 1/2)\ \varepsilon \int_{\Gamma_{ij}} \boldsymbol{\nabla} N_k^p \cdot \boldsymbol{n}_{ij}\ N_l^q \,\mathrm{d}\boldsymbol{S} \\ &\quad +(\delta_{ik} - 1/2)\ s\ \varepsilon \int_{\Gamma_{ij}} \boldsymbol{\nabla} N_l^p \cdot \boldsymbol{n}_{ij}\ N_k^q \,\mathrm{d}\boldsymbol{S}. \end{aligned} \tag{2.49}$$

The indices i and j define the face Γ_{ij} to integrate over, while the indices k and l specify the the supports of the shape functions N_k^p and N_l^q. This distinction is necessary because from the operators $\langle\cdot\rangle$ and $[\cdot]$, the shape functions of both elements K_i and K_j contribute to both ansatzfunction and testfunction. The factors $(\delta_{il} - 1/2)$ and $(\delta_{ik} - 1/2)$ take into account the factor $1/2$ in the average operator $\langle\cdot\rangle$ and the sign introduced by the jump operator $[\cdot]$.

The boundary diffusion matrix $\boldsymbol{A}^{bnd}$ can be assembled from local boundary diffusion matrices $\boldsymbol{A}_{ijkl}^{bnd}$:

$$\boldsymbol{A}^{bnd} = \sum_{(i,j)\in G} \boldsymbol{T}_i^T \boldsymbol{A}_{ijii}^{bnd} \boldsymbol{T}_i \tag{2.50}$$

$$(\boldsymbol{A}_{ijkl}^{bnd})_{pq} = a_{loc}^{bnd}(i,j,k,l,p,q) \tag{2.51}$$

This boundary assembly process is also performed by the new class `Assembly`. It takes the local bilinear form $a_{loc}^{bnd}(i,j,k,l,p,q)$ (`LaplaceInnBf`) and builds the matrix $\boldsymbol{A}^{bnd}$.

The local bilinear form $a_{loc}^{bnd}(i,j,k,l,p,q)$ is defined as:

$$a_{loc}^{bnd}(i,j,k,l,p,q) = -\varepsilon \int_{\Gamma_{ij}} \boldsymbol{\nabla} N_k^p \cdot \boldsymbol{n}_{ij}\ N_l^q \,\mathrm{d}\boldsymbol{S} + s\ \varepsilon \int_{\Gamma_{ij}} \boldsymbol{\nabla} N_l^p \cdot \boldsymbol{n}_{ij}\ N_k^q \,\mathrm{d}\boldsymbol{S} \tag{2.52}$$

The integrals in the bilinear forms are computed using the quadrature rules presented in section 2.3.2. As all occuring integrands are in $\mathcal{P}^2$, all integrals are computed exactly by our quadrature formulae.

In traditional FE discretizations of diffusion terms, one usually transforms the integral to a reference element. The resulting integral can then be calculated once in closed form.

This is advantageous for cases where the additional transformation is cheaper than the economized integration. In our case, especially for the face integrals, the transformations become far more expensive than the quadrature in physical space. We therefore choose the latter option.

Using the implemented local bilinear forms, a matrix free formulation using the same structures could easily be realized. New assembly routines would be needed to take the existing bilinear forms and a state vector $\boldsymbol{u}$, and to calculate and assemble the vector $\boldsymbol{Au}$.

2.5.2 Penalty Matrix

The penalty matrix $\boldsymbol{J}$ can now be treated like the diffusion matrix $\boldsymbol{A}$. The bilinear form (1.53) and the matrix are split into a inner and a boundary term. Note that there is no volume term. With the local bilinear form `PenaltyBf`

$$J_{loc}(i,j,k,l,p,q) \;=\; (2\delta_{kl}-1)\frac{1}{d(\Gamma_{ij})}\int_{\Gamma_{ij}} N_k^p\; N_l^q \;\mathrm{d}\boldsymbol{S} \tag{2.53}$$

both the inner and the boundary matrices can be assembled using the assembly processes (2.47) and (2.50).

The factor $(2\delta_{kl}-1)$ takes into account the sign introduced by the jump operator $[\cdot]$.

2.6 Right Hand Side

The load vector is defined by the following equation, introduced in section 1.2.4:

$$(\boldsymbol{l})_n(t) \;=\; \ell_h(\varphi_n)\;(t) \tag{2.54}$$

The linear form ℓ_h is defined by equation (1.52) introduced in section 1.2.3:

$$\begin{aligned}\ell_h(\varphi_n)\;(t) \;&=\; (g(t),\varphi_n) \;+\; s\,\varepsilon \sum_{(i,j)\in G}\int_{\Gamma_{ij}} u_D(t)\;(\boldsymbol{\nabla}\varphi_n\cdot\boldsymbol{n}_{ij})\;\mathrm{d}\boldsymbol{S} \\ &+\; \varepsilon \sum_{(i,j)\in G}\frac{1}{d(\Gamma_{ij})}\int_{\Gamma_{ij}} u_D(t)\;\varphi_n\;\mathrm{d}\boldsymbol{S}\end{aligned} \tag{2.55}$$

For the linear form and the load vector, we stick to the available assembly process in *Concepts*, which performs a loop over elements $K_i, i \in I$ and not a loop over element

pairs $(i,j) \in G$. So the load vector $\boldsymbol{l}(t)$ is assembled from local load vectors $\boldsymbol{l}_i(t)$:

$$\boldsymbol{l}(t) = \sum_{i \in I} \boldsymbol{T}_i^T \boldsymbol{l}_i(t) \tag{2.56}$$

$$(\boldsymbol{l}_i(t))_p = \ell_{loc}(i,p)\ (t) \tag{2.57}$$

$$\ell_{loc}(i,p) = \int_{K_i} g(t)\ N_i^p\ \mathrm{d}\boldsymbol{x} + \ s\ \varepsilon \sum_{j|(i,j)\in G} \int_{\Gamma_{ij}} u_D(t)\ (\boldsymbol{\nabla} N_i^p \cdot \boldsymbol{n}_{ij})\ \mathrm{d}\boldsymbol{S} \tag{2.58}$$

$$+\ \varepsilon \sum_{j|(i,j)\in G} \frac{1}{d(\Gamma_{ij})} \int_{\Gamma_{ij}} u_D(t)\ N_i^p\ \mathrm{d}\boldsymbol{S}$$

2.7 Timestepping

For the integration in time, the timestepping facilities of *Concepts* are used. In this framework, the equation to be solved is of the form

$$\boldsymbol{D}_1 \dot{\boldsymbol{u}} = -\boldsymbol{D}_0 \boldsymbol{u} + \boldsymbol{g}(t) \tag{2.59}$$

which corresponds to a system of ODEs that can be solved by one of the timestepping schemes presented in section 2.7.1. A comparison with equation (1.67) yields the identities

$$\boldsymbol{D}_1 = \boldsymbol{M}^T = \boldsymbol{M} \tag{2.60}$$

$$\boldsymbol{D}_0 = \boldsymbol{A}^T + \boldsymbol{b}(\cdot) + \varepsilon \boldsymbol{J}^T \tag{2.61}$$

$$\boldsymbol{g}(t) = \boldsymbol{l}(t) \tag{2.62}$$

where $\boldsymbol{M}$, $\boldsymbol{A}$, $\boldsymbol{b}(\boldsymbol{u})$, $\boldsymbol{J}$ and $\boldsymbol{l}(t)$ are defined in section 1.3. In the following, the right hand side terms will be grouped together as

$$\boldsymbol{L}_h(\boldsymbol{u},t) = -\boldsymbol{D}_0 \boldsymbol{u} + \boldsymbol{g}(t) \tag{2.63}$$

The equation to be solved then has the form

$$\boldsymbol{D}_1 \boldsymbol{y} = \boldsymbol{L}_h(\boldsymbol{u},t) \tag{2.64}$$

In every timestep, the matrix $\boldsymbol{D}_1$ needs to be inverted. As we show in section 2.3, the basefunctions φ_m can be chosen to be orthogonal, causing $\boldsymbol{D}_1$ to be diagonal. Thus equation (2.64) can be solved extremely efficiently.

2.7.1 Integration Schemes

To guarantee the stability of the FV DG method with explicit timestepping schemes, total variation diminishing (TVD) Runge-Kutta schemes need to be employed. The general form of such integration schemes was introduced by Shu [9]. These schemes are a specialized form of the general explicit Runge-Kutta schemes and are defined as follows:

1. At $t = t^k$:

$$\boldsymbol{u}^{(0)} = \boldsymbol{u}^k \tag{2.65}$$
$$\tau^{(0)} = 0 \tag{2.66}$$

2. For $i \in 1, \ldots, n$:

$$\boldsymbol{u}^{(i)} = \sum_{l=0}^{i-1} a_{i,l}\boldsymbol{u}^{(l)} + \Delta t^k b_{i,l}\boldsymbol{D}_1^{-1}\boldsymbol{L}_h(\boldsymbol{u}^{(l)}, \tau^{(l)}) \tag{2.67}$$
$$\tau^{(i)} = \sum_{l=0}^{i-1} a_{i,l}\tau^{(l)} + b_{i,l}\Delta t^k \tag{2.68}$$

3. Update solution:

$$\boldsymbol{u}^{k+1} = \boldsymbol{u}^{(n)} \tag{2.69}$$

A tabular representation of $a_{i,l}$ and $b_{i,l}$ can be used to characterize a specific scheme. The schemes which are implemented are shown below.

Euler This is the most simple representative of a TVD-Runge-Kutta scheme.

$a_{i,l}$	$b_{i,l}$
1	1

According to Cockburn [1] [2], this scheme is unconditionally unstable for the FV DG method.

2-Step Runge-Kutta This scheme is presented in [1].

$a_{i,l}$		$b_{i,l}$	
1		1	
1/2	1/2	0	1/2

This is a 2-step TVD-scheme of second order accuracy and the stability criterion is

$$CFL = \max_{i,l} \frac{b_{i,l}}{a_{i,l}} < 1 \tag{2.70}$$

2.7.2 Limiting

When limiters are used, the general scheme has to be modified. On every intermediate step of the scheme, the intermediate solution $\boldsymbol{u}^{(l)}$ is replaced by its limited counterpart $\pi_u \boldsymbol{u}^{(l)}$:

1. At $t = t^k$:

$$\boldsymbol{u}^{(0)} = \boldsymbol{u}^k \tag{2.71}$$
$$\tau^{(0)} = 0 \tag{2.72}$$

2. For $i \in 1, \ldots, n$:

$$\boldsymbol{u}^{(i)} = \sum_{l=0}^{i-1} a_{i,l} \pi_u \boldsymbol{u}^{(l)} + \Delta t^k b_{i,l} \boldsymbol{D}_1^{-1} \boldsymbol{L}_h(\boldsymbol{u}^{(l)}, \tau^{(l)}) \tag{2.73}$$
$$\tau^{(i)} = \sum_{l=0}^{i-1} a_{i,l} \tau^{(l)} + b_{i,l} \Delta t^k \tag{2.74}$$

3. Update solution:

$$\boldsymbol{u}^{k+1} = \boldsymbol{u}^{(n)} \tag{2.75}$$

In the FV DG method, the advection term $b_h(u, v)$, which is part of $\boldsymbol{L}_h(\boldsymbol{u}, t)$ here, is evaluated with the limited solution $\pi_u \boldsymbol{u}^{(l)}$ as well. See section 1.4 for more details on the limiting procedure.

2.8 Further Work

The presented implementation of the method can only be used for scalar problems. A generalization to vector problems would be useful, for instance to solve fluid flow problems. A parallelization of the code would be desirable as well, since the number of the degrees of freedom increase rapidly with the problem size. In this connection, alternative techniques to improve the efficiency of the method should be considered, such as adaptivity in space and time. An increase of the polynomial degree of the shape functions might also be helpful, but it seems inpractical with the current implementation, because all the quadrature formulae are hard-wired to the currently used shape functions.

3 Numerical Results

3.1 Pure Advection Problem

This part is concerned with testing the implementation of the convection part of equation (1.1), which can be written as

$$\partial_t u + \boldsymbol{\nabla} \cdot \boldsymbol{f}(u) = 0 \tag{3.1}$$

The test involves calculations with piecewise constant shapefunctions in section 3.1.2 and piecewise linear shapefunctions in section 3.1.4. An error analysis and stability analysis is presented in both cases. The calculations are done using a unit cube domain $\Omega = (0,1)^3$ with periodic boundary conditions. The initial conditions are chosen as

$$u^0(\boldsymbol{x}) = \frac{1}{4}\sin(\pi(2x_1 + 2x_2 + 2x_3 - 3)) \tag{3.2}$$

In section 3.1.1 the analytical solution of this problem for the case of the linear advection flux and Burger's flux is presented.

3.1.1 Exact Solution and Error Measures

For pure advection problems, exact solutions can be found using characteristics. Equation (3.1) can be written as

$$\partial_t u + \boldsymbol{f}'(u) \cdot \boldsymbol{\nabla} u = 0 \tag{3.3}$$

where

$$\boldsymbol{f}'(u) = \partial_u \boldsymbol{f}(u) \tag{3.4}$$

Using the ansatz for u

$$u(\boldsymbol{x}, t) = u(\boldsymbol{x}(t), t) \tag{3.5}$$

the characteristics $\boldsymbol{x}(t)$ can be identified with the following equation:

$$\frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}t} = \boldsymbol{f}'(u) \tag{3.6}$$

Further inspection of equation (3.1) shows that the value of u doesn't change along the characteristics defined in (3.6):

$$\frac{\mathrm{d}u(\boldsymbol{x}(t))}{\mathrm{d}t} = 0 \tag{3.7}$$

Since $\boldsymbol{f}'$ only depends on u, the characteristics are straight lines and hence a differentiable solution $u(\boldsymbol{x}, t)$ can be retraced to its initial condition $u^0(\boldsymbol{x}_0)$ using the identity which follows from equation (3.6):

$$\boldsymbol{x}_0(\boldsymbol{x}, t) = \boldsymbol{x} - t\boldsymbol{f}'(u(\boldsymbol{x}, t)) \tag{3.8}$$

These results are now applied to the flux functions presented in section 2.4.1 and suitable error measures are introduced. Due to computational difficulties calculating the total error, we only assess the approximation error. By considering only this error, we are neglecting the discretization error. For basis functions of polynomial order p, the discretization error is known to be $\mathcal{O}(h^{p+1})$. Thus, the effect of this neglect can be estimated as follows:

$$\begin{aligned} \|u_h - u_{exact}\| &= \|u_h - \pi_p u_{exact} + \pi_p u_{exact} - u_{exact}\| \\ &\leq \|u_h - \pi_p u_{exact}\| + \|\pi_p u_{exact} - u_{exact}\| \\ &= \mathcal{O}(h^{\alpha}) + \mathcal{O}(h^{p+1}) = \mathcal{O}(h^{\min(\alpha, p+1)}) \end{aligned}$$

where π_p is the L^2 projection into $S^{p,-1}$, defined as:

$$\pi_p u \in S^{p,-1} \tag{3.9}$$

$$(u, v) = (\pi_p u, v)\ \forall v \in S^{p,-1} \tag{3.10}$$

Thus, we know that when observing a convergence order $\alpha \leq p+1$ in the approximation error, we are guaranteed to have a convergence order α in the total error.

Linear Advection For the linear advection flux (2.22), the initial condition is transported at the constant speed a:

$$u_{exact}(\boldsymbol{x}, t) = u^0(\boldsymbol{x} - t\,a\,(1, 1, 1)^T) \tag{3.11}$$

For the error analysis presented in section 3.1.2, the maximum norm

$$e_h = \|u_{exact} - u_h\|_{L^\infty(\Omega, \mathcal{T}_h)} \tag{3.12}$$

is used.

Burger's Flux For Burger's flux (2.23), the exact solution can still be evaluated using characteristics:

$$u_{exact}(\boldsymbol{x}, t) = u^0(\boldsymbol{x} - t\, u_{exact}(\boldsymbol{x}, t)\,(1,1,1)^T) \tag{3.13}$$

However, the state of u_{exact} at a given $\boldsymbol{x}$ and t depends on the state itself, so that an explicit evaluation of equation (3.13) is impossible. Thus, a different approach to measure the error must be taken. For the error measure used here, the numerical solution $u_h(\boldsymbol{x}, t)$ is regarded as an exact solution to a different initial condition $\tilde{u}^0$:

$$\tilde{u}_0(\boldsymbol{x}_0) = u_h(\boldsymbol{x}(t)) - t\, u_h(\boldsymbol{x}, t)\,(1,1,1)^T) \tag{3.14}$$

For the error analysis in section 3.1.4, the maximum norm

$$e_h = \|u^0 - \tilde{u}^0\|_{L^\infty(\Omega, \mathcal{T}_h)} \tag{3.15}$$

is used. Due to the non-linearity of the Burger flux, shocks may form even for smooth initial conditions. In the shock region, the solution in the shock region is not unique and it is impossible to reconstruct the initial conditions using characteristics as mentioned above. It is therefore crucial to measure the error before the shock formation time T_{shock}, which in the one dimensional case is determined by

$$T_{shock} = -\frac{1}{\min_{x\in\Omega}\, f''(u_0)\; \partial_x u_0(x)} \tag{3.16}$$

In the special case of the Burger flux, the relation (3.16) can be generalized using a directional derivative. For the Burger flux (2.23) this generalization yields

$$T_{shock} = -\frac{1}{\min_{x\in\Omega}\, \frac{\sqrt{3}}{3}\; \boldsymbol{\nabla} u_0(\boldsymbol{x}) \cdot (1,1,1)^T} \tag{3.17}$$

For the initial conditions (3.2), a value of $T_{shock} = 0.37$ can be found.

3.1.2 Piecewise Constant Functions

The test cases with piecewise constant shapefunctions are used to test the general framework of the code implementing the convective terms. This includes tests with the different formulations of the numeric fluxes presented in section 2.4.2, as well as their behavior in conjunction with the two different flux functions from section 2.4.1. For time integration, the second order Runge-Kutta scheme of section 2.7 is used. The results are plotted at time $T = 0.25$, which lies well before the shock formation time T_{shock} specified in equation (3.17). The constant for the CFL-criterion is set to 0.3, in order to guarantee stability on all meshes. For a discussion on the problems related to the CFL-criterion, see section 3.1.4.

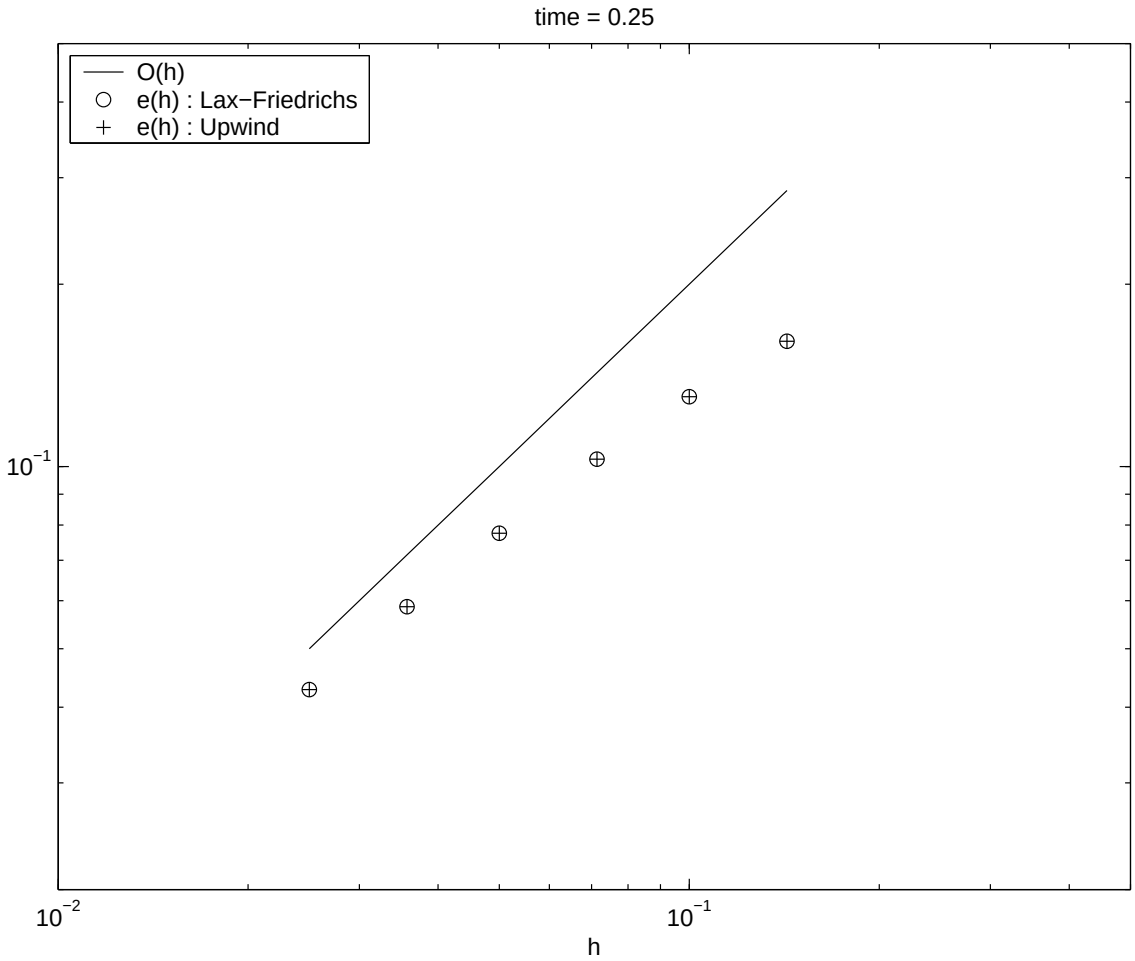

Figure 3.1: Mesh convergence for linear advection flux test cases. The plot shows the behaviour of the error $e_\infty(h)$ as a function of the mesh size h.

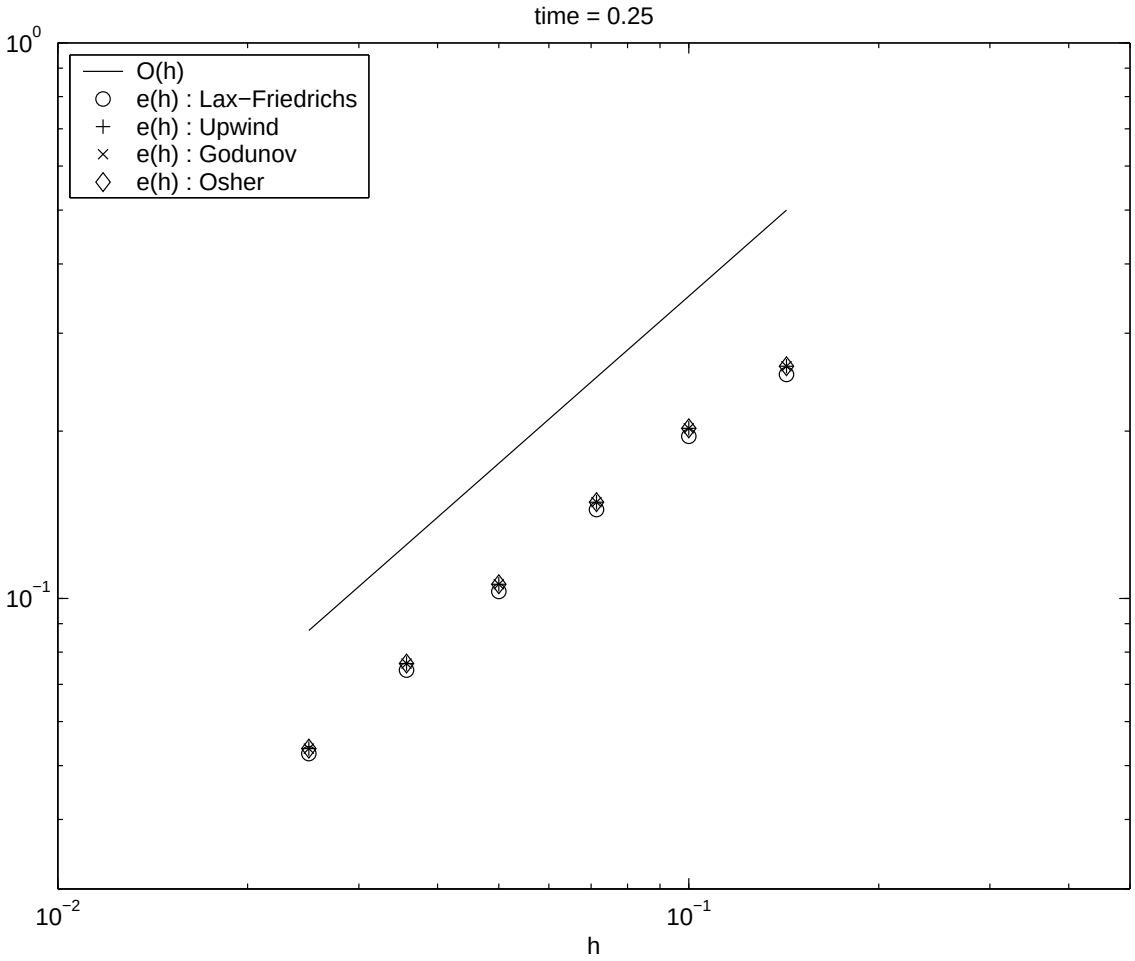

Figure 3.2: Mesh convergence for Burger flux test cases. The plot shows the behaviour of the error $e_\infty(h)$ as a function of the mesh size h.

As can be seen in figures 3.1 and 3.2, the mesh convergence is of order $\mathcal{O}(h)$, as expected from the analysis in [2].

3.1.3 Effect of Mesh Structure

With our implementation of the periodic cube as presented in section 2.2.2, we now can assess the effect of structure in the mesh. To this end, we perform the same tests as described in section 3.1.2, but with a different set of parameters. We vary j between 0 and 0.07, but only for for linear advection flux with upwind numerical flux, with mesh size $h = 0.1$ and a fixed constant for the CFL-criterion $c = 0.4$. We do not increase j beyond 0.07 because for $j > \frac{7-\sqrt{41}}{8} \approx 0.0746$, the resulting tetrahedrons may degenerate i. e. have negative volume.

The degeneracy of tetrahedrons is just the extreme manifestation of another effect occuring also at lower values of j: Not only the structure but also the regularity of the mesh decreases when increasing j. Due to this regularity effect, an increase of the error with increasing j has to be expected.

For the error analysis we use the maximum norm and the L^2 norm:

$$e_\infty(j) = \|u_{exact} - u_h\|_{L^\infty(\Omega,\mathcal{T}_j)} \tag{3.18}$$

$$e_2(j) = \|u_{exact} - u_h\|_{L^2(\Omega,\mathcal{T}_j)} \tag{3.19}$$

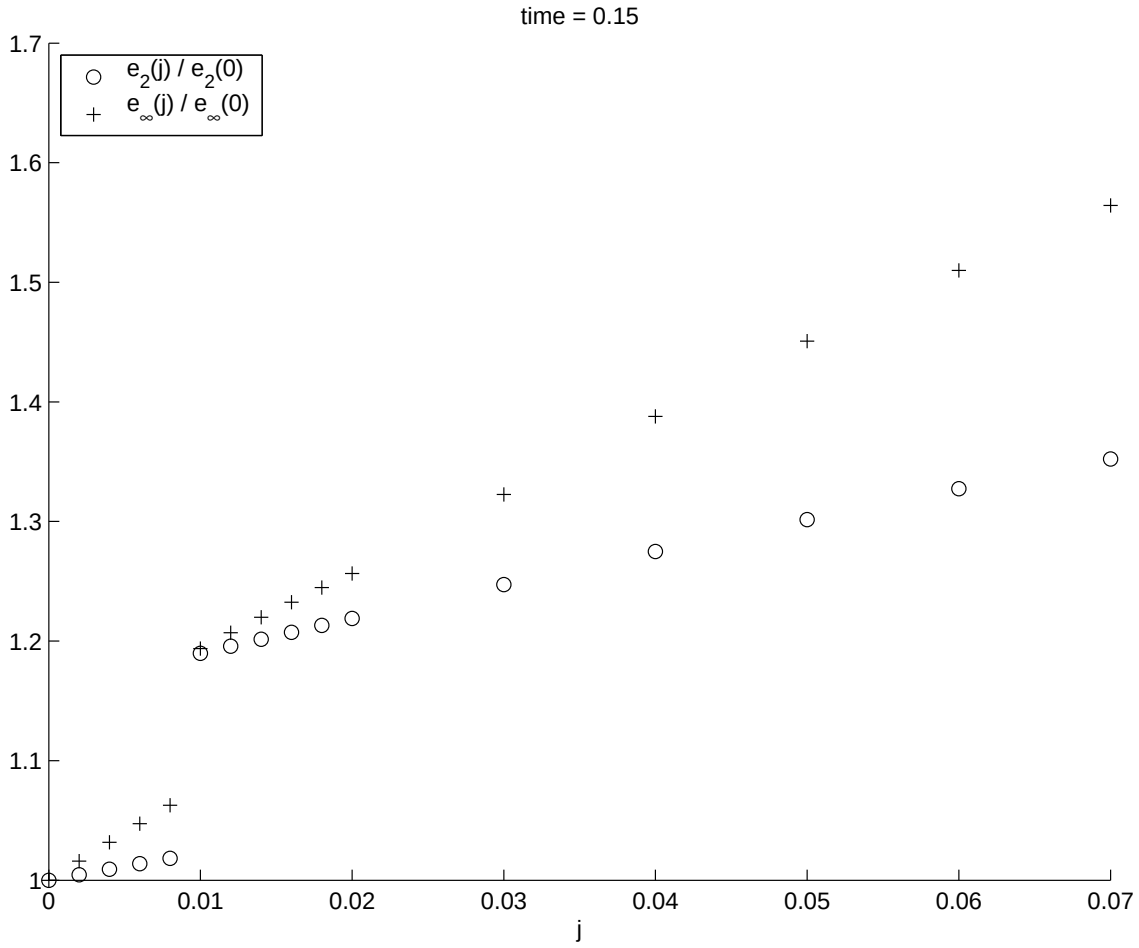

Figure 3.3: Mesh structure influence for linear advection flux cases. The plot shows the behaviour of the error measures $e_2(j)$ and $e_\infty(j)$ as a function of the jitter parameter j.

Figure 3.3 shows the behaviour of the error measures $e_2(j)$ and $e_\infty(j)$ as a function of the jitter parameter j. We observe three phenomena:

- Growth:
  The error measures grow with increasing j, as could be expected from regularity decay.

- Boundedness:
  The growth is moderate, indicating that the numerical results are not influenced much by the structure in the mesh.

- Jump:
  There is a jump in both error measures at $j \approx 0.009$. We do not see any explanation of this phenomenon, but it is certainly interesting!

3.1.4 Piecewise Linear Functions

In this section, the mesh convergence for ansatz and test functions in $\mathcal{P}^1$ is investigated. Again, both flux functions implemented - linear advection flux and Burger's flux - are tested, but only in conjunction with Upwind und Lax-Friedrichs numerical fluxes. Additionally, the impact of limiters on the convergence and on the CFL-criterion is explored.

Mesh Convergence To study the general mesh convergence, the same settings as in section 3.1.2 are used: the constant for the CFL-criterion is 0.3 for the Burger's flux and 0.1 for the linear advection flux and the integration is done with the second order Runge-Kutta scheme. The adaptive limiter from section 1.4 is used to stabilize the method.

For the linear advection flux, the convergence is or order $\mathcal{O}(h)$. This clearly contradicts [4], where a convergence of order $\mathcal{O}(h^2)$ is predicted. The situation gets even worse for the Burger's flux, where no mesh convergence could be shown. The origin of this behaviour is unclear, especially since the full convection-diffusion problem defined by equation (1.1) shows convergence of order $\mathcal{O}(h^2)$ (see section 3.3). In both cases, the choice of the numerical flux has little influence on the result.

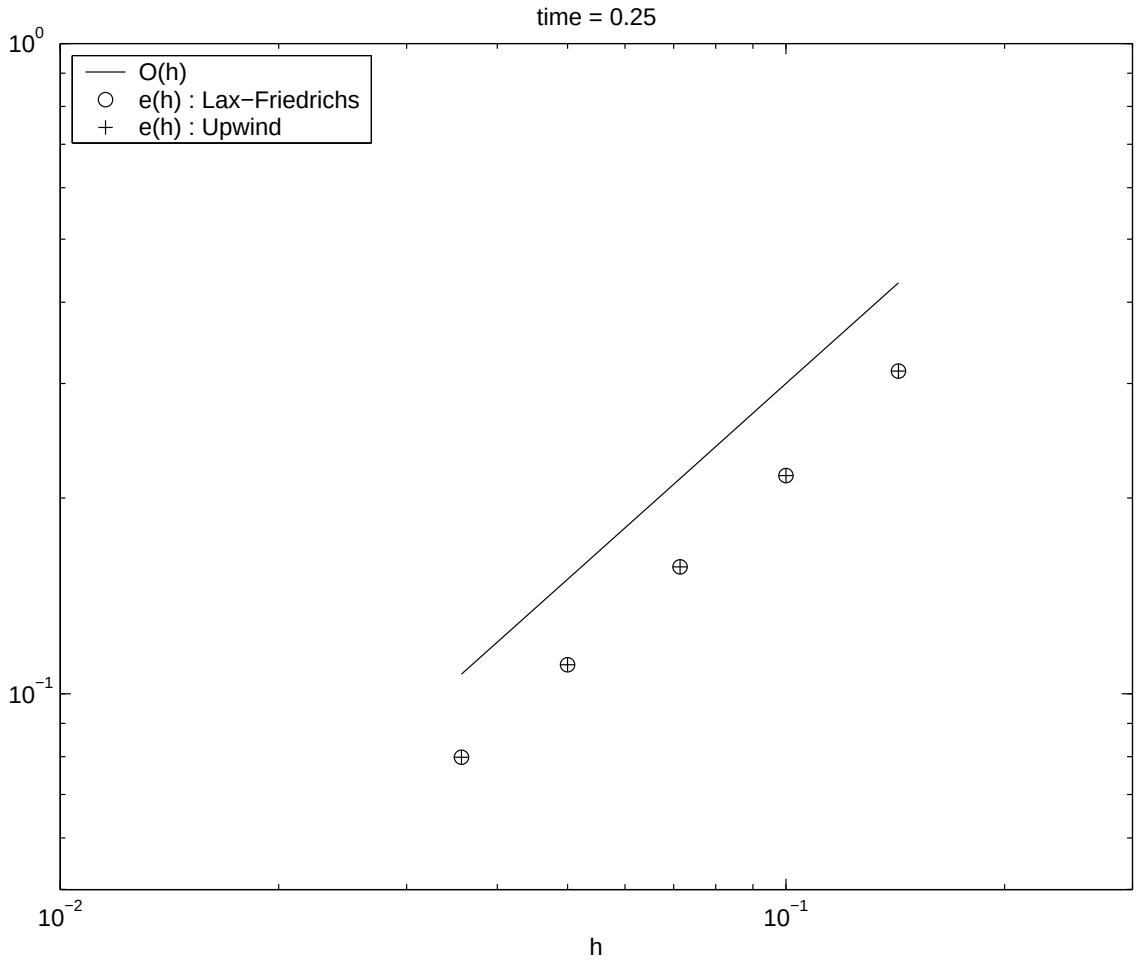

Figure 3.4: Mesh convergence for linear advection flux test cases. The plot shows the behaviour of the error $e(h)$ as a function of the mesh size h.

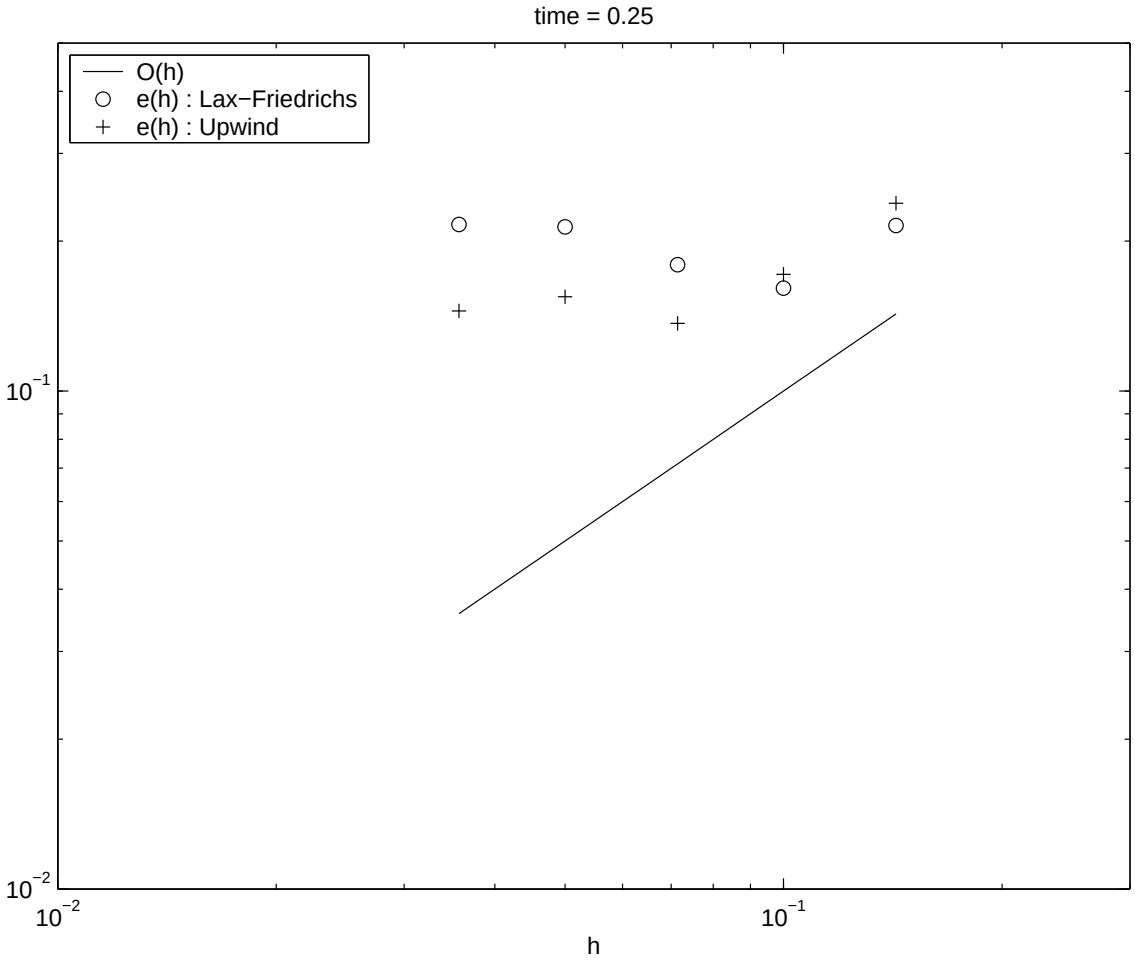

Figure 3.5: Mesh convergence for Burger flux test cases. The plot shows the behaviour of the error $e(h)$ as a function of the mesh size h.

Influence of Limiters As stated in section 2.7.2, different kinds of limiters can be used to avoid oscillations near discontinuities. In this test cases here, two combinations of limiters, the adaptive limiter and the double π_0-limiter are compared to a test case without limiter:

$$\pi_u = \pi_v = \mathbf{1}: \quad \text{no limiter} \tag{3.20}$$

$$\pi_u = \pi_{ad},\ \pi_v = \mathbf{1}: \quad \text{adaptive limiter} \tag{3.21}$$

$$\pi_u = \pi_v = \pi_0: \quad \text{double } \pi_0\text{-limiter} \tag{3.22}$$

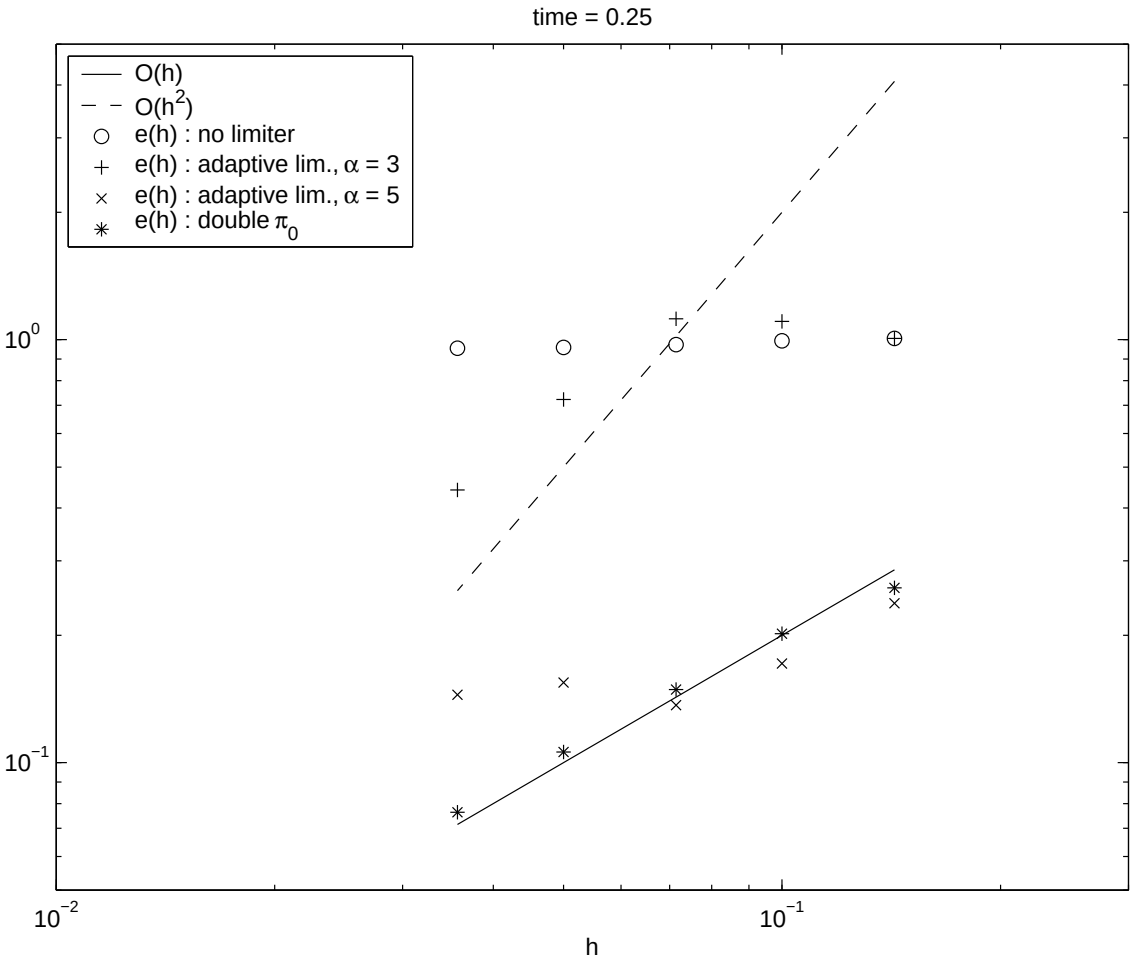

Figure 3.6: Mesh convergence for Burger flux test cases. The plot shows the behaviour of the error $e(h)$ as a function of the mesh size h.

For the adaptive limiter, two parameter values for α are tested. The value $\alpha = 5$ leads to a strong limiting behaviour, whereas $\alpha = 3$ should not influence the solution much,

since the solution for this test case is expected to be smooth at time $T = 0.25$. The settings for the other parameters are the same as for the other test cases presented in this section, except that only Burger's flux with upwinding is tested.

Most strikingly, the method fails to converge when no limiter is applied. On the other hand, the double π_0-limiter reduces the order to $\mathcal{O}(h)$, which is expected from [3]. The behaviour of the adaptive limiter remains obscure. Whereas for $\alpha = 3$ there seems to have a mesh convergence of order $\mathcal{O}(h^2)$ on refined meshes, the method cannot even guarantee a mesh convergence of order $\mathcal{O}(h)$ for $\alpha = 5$. It is unclear whether this is theoretically justified or the cause of the implementation.

CFL-criterion In this paragraph, an upper bound for the CFL constant c is sought. In [2], some information on this issue is presented, but the exact form of the CFL condition is left out. For our purpose, we use the definition presented in section 2.1.3:

$$\Delta t \;=\; c \min_{K \in \mathcal{T}_h} \min_{\boldsymbol{x} \in V(K)} \frac{h_K}{|\boldsymbol{f}'(u_h(\boldsymbol{x}))|} \tag{3.23}$$

where u_h is a projected initial condition and h_K is an appropriate measure for the size of $K \in \mathcal{T}_h$. Here, we use $h_K = |K|^{1/3}$.

In the table below, the numerically found upper bounds for the CFL constant c are presented, depending on the flux function and the limiter used. The results are shown for different h_K to see if the condition is independent from the mesh's size. All results are obtained using the TVD Runge-Kutta method presented in section 2.7

	Linear Advection			**Burger**		
h_K	no limiter	adapt. lim.	π_0-limiter	no limiter	adapt. lim.	π_0-limiter
0.14	0.25	0.56	0.25	0.69	3.06	0.69
0.10	0.25	0.56	0.25	0.56	1.50	0.56
0.07	0.25	0.50	0.25	0.44	1.13	0.44

Table 3.1: Numerically determined upper bounds for CFL constant c in equation 3.23, depending on mesh size h_K, limiter and flux function.

It can be seen that the choice of the flux function has a large impact on the CFL constant. For the Burger's flux, the resulting upper bound is not mesh independent, which is not desirable.

3.2 Pure Diffusion Problems

In order to test the discretization of the diffusion terms separately, we solve two problems without advection terms. In section 3.2.1, we consider the Poisson equation. In section 3.2.2, the heat equation is tested in order to see the consequences of the choice of the parameter s introduced in section 1.2.3.

3.2.1 Elliptic Problem

When we neglect the time derivative and the advection term in (1.1) and set $\varepsilon = 1$, we obtain the following *boundary value problem*: Find $u : \Omega \to \mathbb{R}$ such that

$$-\Delta u = g \quad \text{in } \Omega, \tag{3.24}$$

$$u|_{\partial\Omega} = u_D. \tag{3.25}$$

This is the three-dimensional Poisson equation. We are now looking for an approximate solution $u_h \in S_h$. Using the same discretization techniques as for (1.1) yields

$$(\boldsymbol{A} + \boldsymbol{J})^T \boldsymbol{u} = \boldsymbol{l} \tag{3.26}$$

where $\boldsymbol{A}$, $\boldsymbol{J}$, $\boldsymbol{u}$ and $\boldsymbol{l}$ are defined as in section 1.3, except that neither $\boldsymbol{u}$ nor $\boldsymbol{l}$ depend on time. This linear system can be solved with a suitable solver. As $\boldsymbol{A} + \boldsymbol{J}$ is sparse but not necessarily symmetric, it is favorable to use an iterative solver, but one cannot use the conjugate gradient method. We choose a GMRes algorithm already implemented in *Concepts*.

For the sake of simplicity, we choose the following data:

$$\Omega = \{\boldsymbol{x}; |\boldsymbol{x}| < 1\} \subset \mathbb{R}^3 \tag{3.27}$$

$$g = -6 \tag{3.28}$$

$$u_D = 0 \tag{3.29}$$

The problem (3.24) – (3.25) then has the exact solution:

$$u_{exact}(\boldsymbol{x}) = \boldsymbol{x} \cdot \boldsymbol{x} - 1 \in \mathcal{P}^2(\Omega) \tag{3.30}$$

We assess the performance of the method by calculating the H^1 seminorm of the error:

$$e(h) = |u_h - u_{exact}|_{H^1(\Omega, \mathcal{T}_h)} \tag{3.31}$$

$$u_h(\boldsymbol{x}) = \boldsymbol{u} \cdot \boldsymbol{\varphi}(\boldsymbol{x}) \tag{3.32}$$

Because $u_{exact} \in \mathcal{P}^2(\Omega)$, the seminorm can be calculated exactly with the quadrature rule presented in section 2.3.2.

As we have a piecewise constant ansatz for $\boldsymbol{\nabla} u$, and the seminorm measures only the gradient, we should expect first order convergence, i. e. $e(h) = \mathcal{O}(h)$. Figure 3.7 shows that this convergence rate is in fact observed, indicating a correct implementation of the discretized diffusion terms.

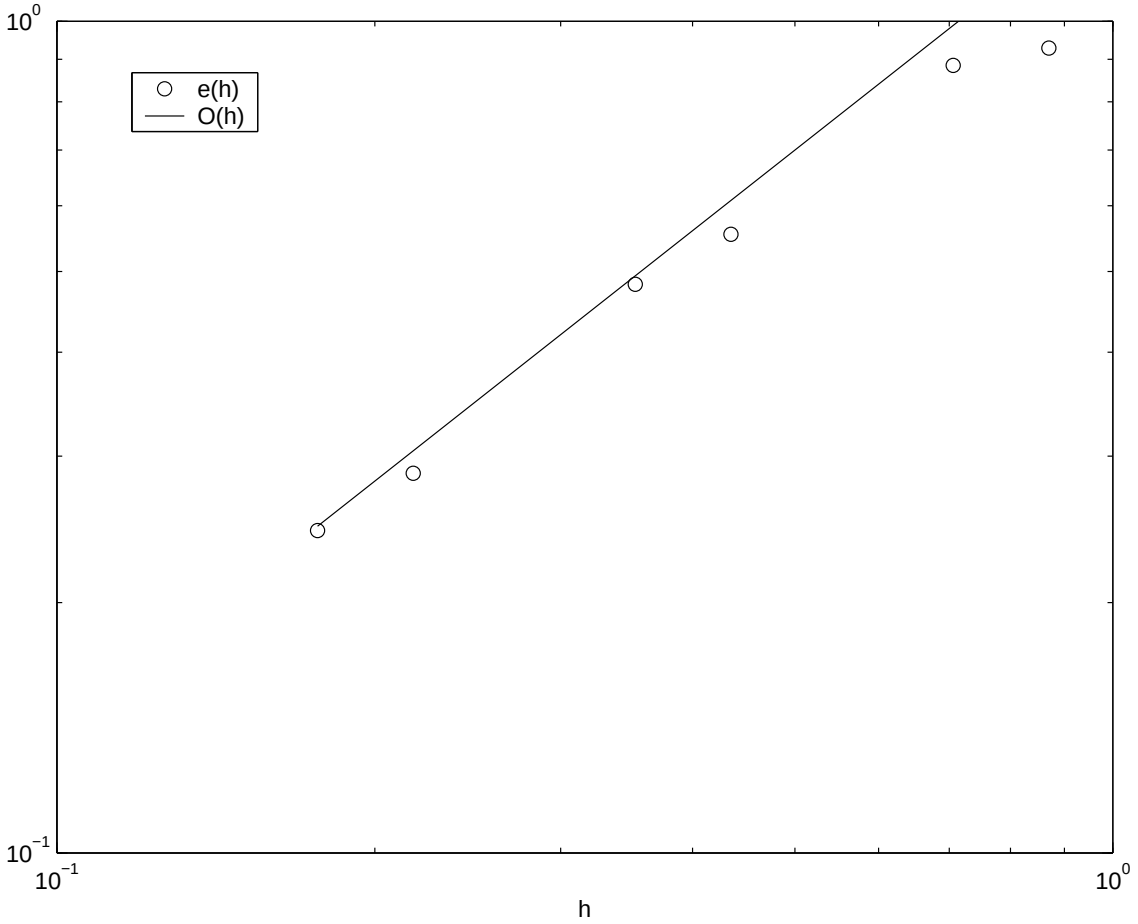

Figure 3.7: Mesh convergence for elliptic problem. The plot shows the behaviour of the error $e(h)$ as a function of the mesh size h.

3.2.2 Parabolic Problem

When we neglect only the advection term in (1.1) and set $\varepsilon = 1$, we obtain the following *initial-boundary value problem*: Find $u : Q_T \to I\!R$ such that

$$\partial_t u - \Delta u = g \quad \text{in } Q_T, \tag{3.33}$$

$$u\,\big|_{\partial\Omega\times(0,T)} = u_D, \tag{3.34}$$

$$u(\boldsymbol{x}, 0) = u^0(\boldsymbol{x}), \quad \boldsymbol{x} \in \Omega. \tag{3.35}$$

This is the three-dimensional heat equation. We are looking for an approximate solution at time t: $u_h(t) \in S_h$. Using the same discretization techniques as for (1.1) yields

$$\boldsymbol{M}\dot{\boldsymbol{u}} + (\boldsymbol{A} + \boldsymbol{J})^T\boldsymbol{u} \;=\; \boldsymbol{l}(t) \tag{3.36}$$

where $\boldsymbol{M}$, $\boldsymbol{A}$, $\boldsymbol{J}$, $\boldsymbol{u}$ and $\boldsymbol{l}(t)$ are defined as in section 1.3. This linear system of ordinary differential equations can be solved with a suitable ODE solver.

As our mass matrix $\boldsymbol{M}$ is not only sparse but even diagonal, a step of an explicit solver is much cheaper than a step of an implicit solver. On the other hand, explicit methods require for the timestep $\Delta t = \mathcal{O}(h^2)$. This has the positive side effect, that when choosing the simple Euler method for timestepping, the error will be $\mathcal{O}(\Delta t) = \mathcal{O}(h^2)$. Because the space discretization error is also $\mathcal{O}(h^2)$, it is not worth choosing a better timestepping scheme. Therefore, we will conduct our tests with the Euler method.

For the stability of the scheme, the choice of the parameter s introduced in section 1.2.3 is crucial:

- Antisymmetric discretization of diffusive fluxes, $s = 1$
  For $s = 1$, the matrix $\boldsymbol{A}$ is not symmetric: The contribution of the volume integral is symmetric, while the contribution of the face integrals is antisymmetric. At first sight, this seems to be unnatural for the self-adjoint Laplacian. But it turns out that in this case all eigenvalues of $\boldsymbol{A}$ have positive real part. This means that, for explicit timestepping, all eigenfunctions are damped, what can be expected from a diffusion operator. The nonvanishing imaginary parts of some eigenvalues don't make sense physically, but this is not a stability issue.

- Symmetric discretization of diffusive fluxes, $s = -1$
  Instead, if we choose $s = -1$, $\boldsymbol{A}$ is symmetric and has only real valued, but positive and negative eigenvalues. This means, that some eigenfunctions are amplified instead of damped, what makes this discretization unphysical and unstable. This instability was observed numerically. In what follows, we will therefore use $s = 1$.

For the sake of simplicity, we choose the following data:

$$\Omega = (-2,2)^3 \tag{3.37}$$

$$g = 0 \tag{3.38}$$

$$u^*(\boldsymbol{x},t) = \frac{1}{(4\pi(t+0.1))^{3/2}} \exp\Big(-\frac{\boldsymbol{x}\cdot\boldsymbol{x}}{4(t+0.1)}\Big) \tag{3.39}$$

$$u_D = u^*|_{\partial\Omega} \tag{3.40}$$

$$u^0 = u^*|_{t=0} \tag{3.41}$$

The problem (3.33) – (3.35) then has the exact solution:

$$u_{exact} = u^* \tag{3.42}$$

We assess the performance of the method by calculating the L^1 norm and the L^∞ norm of the error:

$$e_1(h) = \|u_h - u_{exact}\|_{L^1(\Omega,\mathcal{T}_h)} \tag{3.43}$$

$$e_\infty(h) = \|u_h - u_{exact}\|_{L^\infty(\Omega,\mathcal{T}_h)} \tag{3.44}$$

$$u_h(\boldsymbol{x}) = \boldsymbol{u}\cdot\boldsymbol{\varphi}(\boldsymbol{x}) \tag{3.45}$$

As we have a piecewise linear ansatz for u, we should expect second order convergence, i. e. $e_1(h) = \mathcal{O}(h^2)$ and $e_\infty(h) = \mathcal{O}(h^2)$. Figure 3.8 shows that this convergence rate is in fact observed.

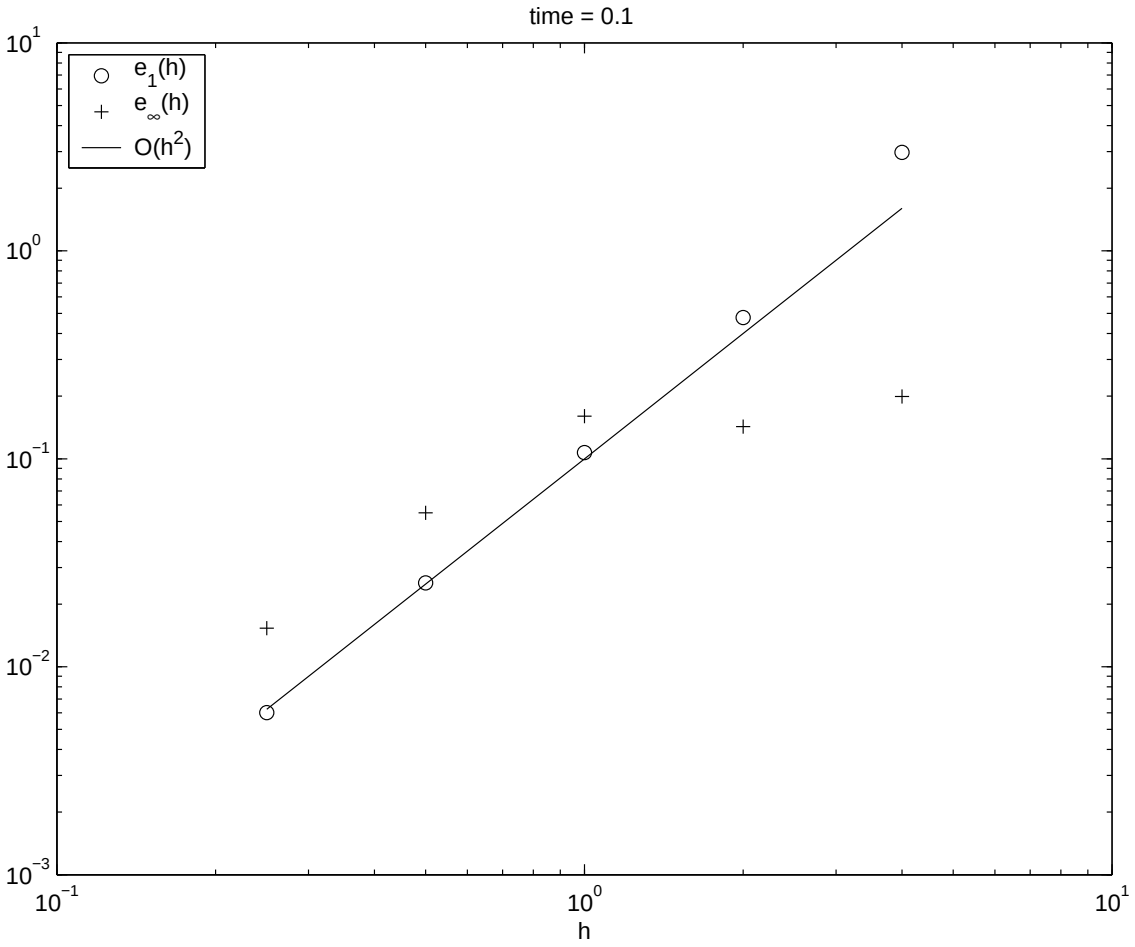

Figure 3.8: Mesh convergence for parabolic problem. The plot shows the behaviour of the error measures $e_1(h)$ and $e_\infty(h)$ as a function of the mesh size h.

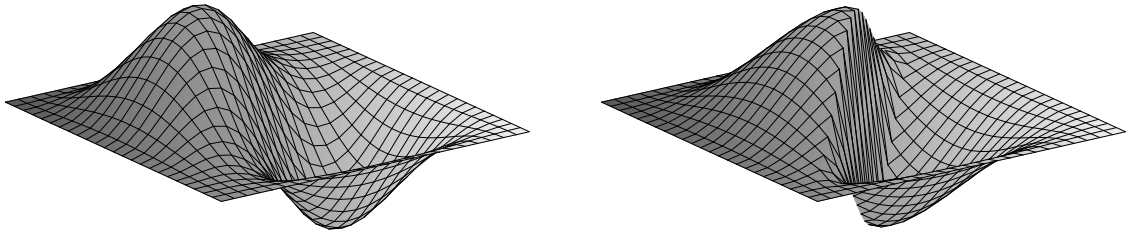

Figure 3.9: The exact solution u^* at $x_3 = 0$ and $t = 1$ for $\nu = 0.5$ (left) and $\nu = 0.01$ (right).

3.3 Full Advection Diffusion Problem

Finally, we solve the full problem 1.1 – 1.3 numerically as presented in chapter 1. We use a similar testcase as in [3], though generalized to three dimensions:

$$\Omega = (-1,1)^3 \tag{3.46}$$

$$\boldsymbol{f}(u) = \frac{1}{2}u^2(1,1,1)^T \tag{3.47}$$

$$u^*(\boldsymbol{x},t) = -(1-x_1^2)^2(1-x_2^2)^2(1-x_3^2)^2 \ \text{atan}\frac{x_1+x_2+x_3+3-3t}{\nu} \tag{3.48}$$

$$u_D = u^*|_{\partial\Omega} = 0 \tag{3.49}$$

$$u^0 = u^*|_{t=0} \tag{3.50}$$

$$g = \partial_t u^* + \boldsymbol{\nabla}\cdot\boldsymbol{f}(u^*) - \varepsilon\Delta u^* \tag{3.51}$$

$$\tag{3.52}$$

The problem (1.1) – (1.3) then has the exact solution:

$$u_{exact} = u^* \tag{3.53}$$

As in [3], we set $\varepsilon = 0.002$ and run the scheme for $\nu = 0.5$ and $\nu = 0.01$. The former gives a very smooth solution, whereas the latter leads to a solution with a steep gradient. Figure 3.9 shows the exact solution u^* in a cutting plane at $x_3 = 0$ for both values of ν at time $t = 1$, when the solution has the steepest gradient, located along the plane $x_1 + x_2 + x_3 = 0$. We therefore integrate until $T = 1$ and measure the error at this time.

For the full problem, the timestep can be dictated either by the advection term or the diffusion term. The CFL-criterion is expected to look like:

$$\Delta t \leq \min\left(c_a\ h\ ,\ c_d\ \frac{h^2}{\varepsilon}\right) \tag{3.54}$$

where c_a denotes the CFL-constant for the advection and c_d the CFL-constant for the diffusion. The chosen value $\varepsilon = 0.002$ makes sure that for the meshes in consideration, the advection is dominant and the timestep is $\Delta t = \mathcal{O}(h)$.

We use the upwind numerical flux presented in section 2.4.2, and test again three combinations of limiters:

$$\pi_u = \pi_v = \mathbf{1}: \qquad \text{no limiter} \tag{3.55}$$

$$\pi_u = \pi_v = \pi_0: \qquad \text{double } \pi_0 \text{ limiter} \tag{3.56}$$

$$\pi_u = \pi_{ad},\ \pi_v = \mathbf{1}: \qquad \text{adaptive limiter} \tag{3.57}$$

For the adaptive limiter, the parameter α is set to 3.5. All tests are conducted with the TVD Runge-Kutta timestepping scheme of second order presented in section 2.7.1. We assert for the timestep $\Delta t = c_a\ h$, where c_a is constant. The error introduced by the timestepping scheme is therefore known to be $\mathcal{O}(h^2)$.

We assess the performance of the method by calculating the L^2 norm of the approximation error at $t = 1$:

$$e(h) = \|u_h - \pi_1 u_{exact}\|_{L^2(\Omega,\mathcal{T}_h)} \tag{3.58}$$

$$u_h(\boldsymbol{x}) = \boldsymbol{u} \cdot \boldsymbol{\varphi}(\boldsymbol{x}) \tag{3.59}$$

Figure 3.10 shows the convergence for $\nu = 0.5$. As the solution has no very steep gradient, the scheme without limiter has convergence order 2. This corresponds to the theoretical result presented in [2]. Applying the π_0-limiter reduces the convergence order to 1, which is expected from [3]. The adaptive limiter is able to restore the convergence order 2 as expected from [4]. It seems that the diffusion, even if very small, has a good influence on the convergence properties of the method.

Figure 3.11 shows the convergence for $\nu = 0.01$. Here, the gradient is very steep, which is numerically equivalent to a discontinuity. This causes all schemes to have convergence order 1. The scheme without limiter diverged at $h = 0.025$, which makes it unusable. The scheme with double π_0-limiter works well in this case, but is unable to achieve second order convergence for smooth solutions (see figure 3.10). Only the adaptive limiter has optimal convergence order in both cases.

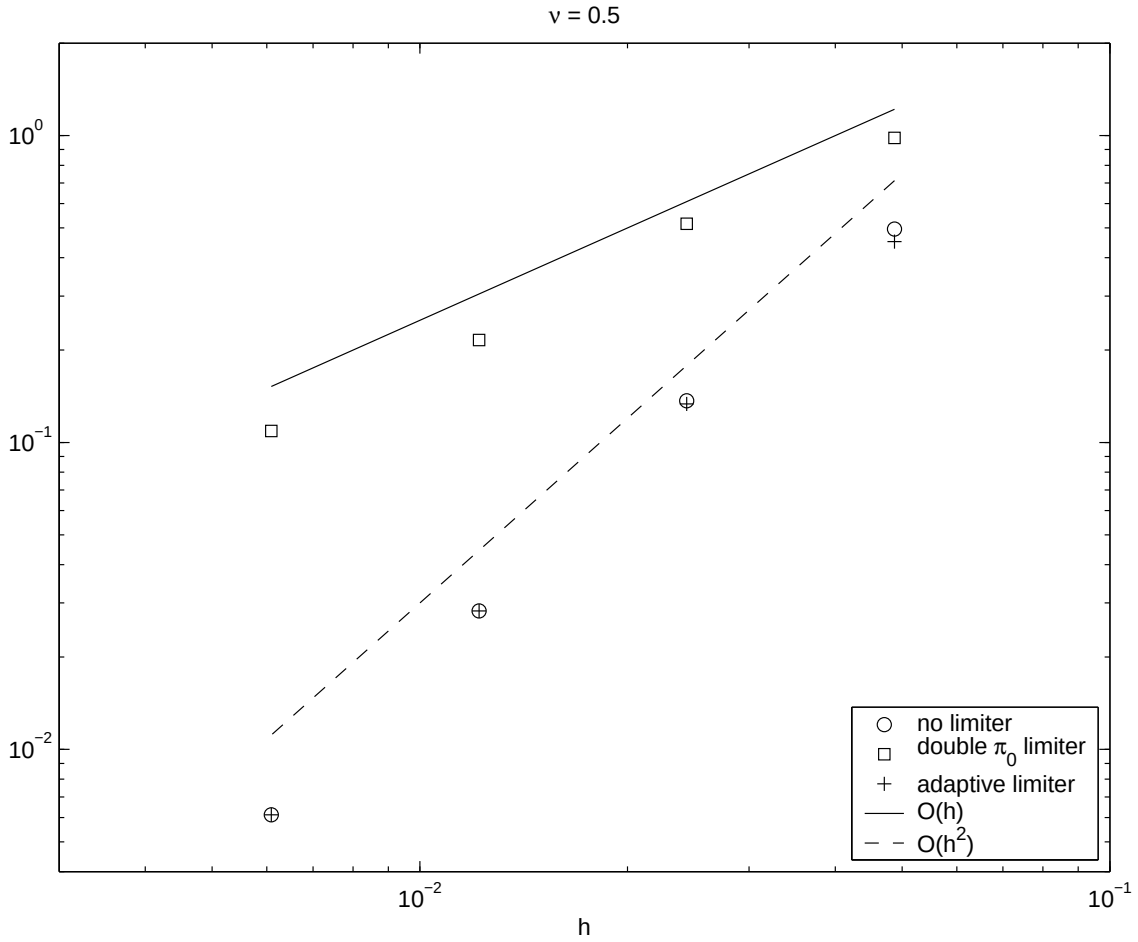

Figure 3.10: Mesh convergence for full problem. The plot shows the behaviour of the error $e(h)$ as a function of the mesh size h for $\nu = 0.5$ and different limiters.

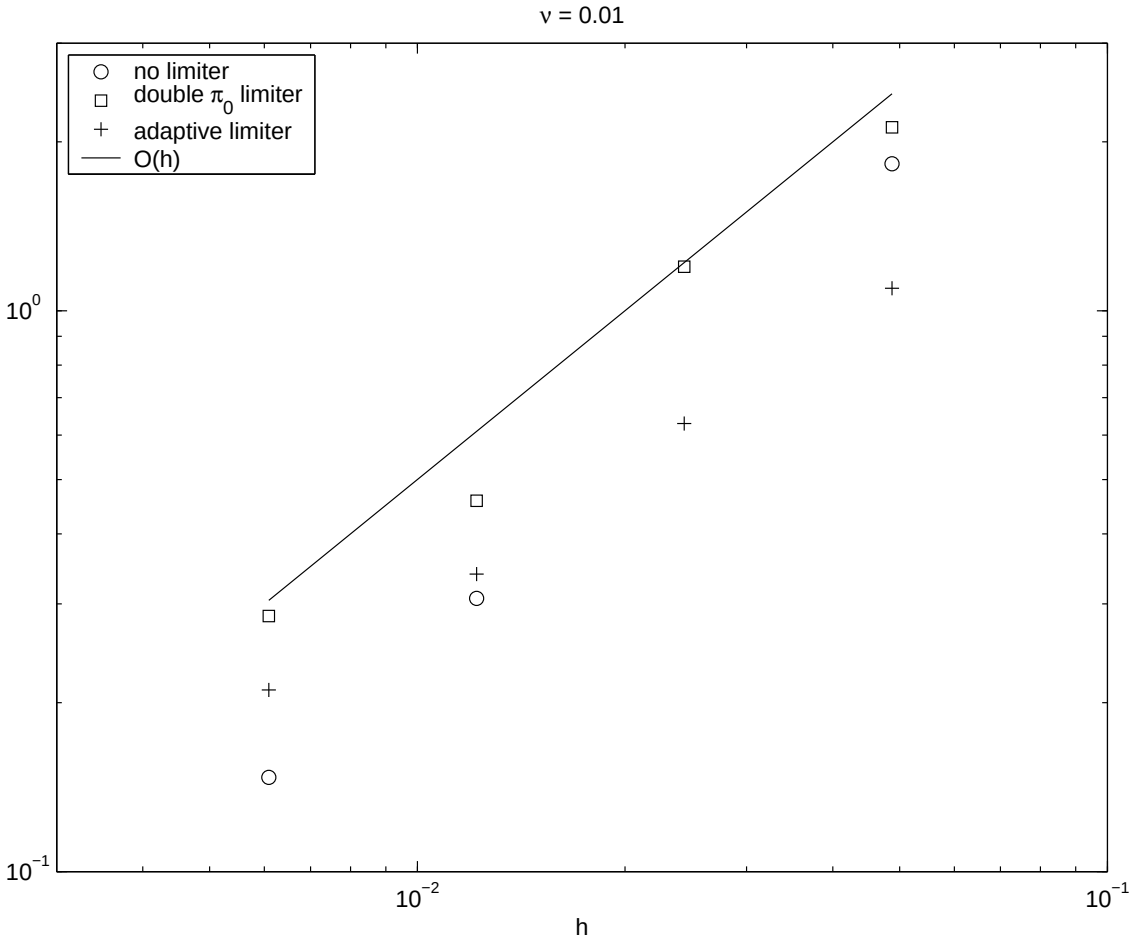

Figure 3.11: Mesh convergence for full problem. The plot shows the behaviour of the error $e(h)$ as a function of the mesh size h for $\nu = 0.01$ and different limiters.

Acknowledgements

We would like to thank: Prof. C. Schwab for his guidance, P. Frauenfelder for his advice and enduring patience, M. Torrilhon for sharing his knowledge on finite volume methods, T. Wihler for his insights in discontinuous Galerkin methods and M. Walser for explaining the timestepping facilities of Concepts.

Bibliography

[1] B. Cockburn: Discontinuous Galerkin Methods for Convection Dominated Problems. In: High–Order Methods for Computational Physics. Lecture Notes in Computational Science and Engineering 9, Springer, Berlin, 1999.

[2] B. Cockburn, G. E. Karniadakis, C.–W. Shu (eds.): Discontinuous Galerkin Methods. Lecture Notes in Computational Science and Engineering 11. Springer, Berlin, 2000.

[3] V. Dolejší, M. Feistauer, C. Schwab: A Finite Volume Discontinous Galerkin Scheme for Nonlinear Convection-Diffusion Problems. Preprint, Forschungsinstitut für Mathematik ETH Zürich, 2001.

[4] V. Dolejší, M. Feistauer, C. Schwab: On some Aspects of the Discontinous Galerkin Finite Element Method for Conservation Laws. Preprint, Mathematics and Computers in Simulation, 2001.

[5] P. Frauenfelder, C. Lage: Concepts – An Object-Oriented Software Package for Partial Differential Equations. Research Report No. 2002-09, Seminar für Angewandte Mathematik ETH Zürich, 2002.

[6] E. Godlewski, P.–A. Raviart: Numerical Approximation of Hyperbolic Systems of Conservation Laws. Applied Mathematical Sciences 118, Springer, New York, 1996.

[7] D. Kröner: Numerical Schemes for Conservation Laws. Teubner, Stuttgart, 1997.

[8] J. Schöberl: NETGEN, `http://www.sfb013.uni-linz.ac.at/~joachim/netgen`

[9] C.–W. Shu, Total-Variation-Diminishing Time Discretizations, SIAM J. Sci. Statist. Comput. 9 (1988), 1073-1084.