# Wissenschaftliches Rechnen

Vorlesung vom WS 2009/2010


Mario Ohlberger

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

Dieses Skript beruht auf einer Vorlesung *Wissenschaftliches Rechnen*, die ich gemeinsam mit Andreas Dedner im SS 2006 an der Universität Freiburg gehalten habe. Das Skript wird im Laufe der Vorlesung im WS 2009 aktualisiert und überarbeitet.

Der Schwerpunkt der Vorlesung ist die Diskretisierung und effiziente Implementierung von sogenannten Discontinuous-Galerkin-Verfahren zur Approximation von elliptischen, parabolischen und hyperbolischen Partiellen Differentialgleichungen. Discontinuous-Galerkin-Verfahren zeichnen sich aus durch eine hohe Flexibilität bei der Behandlung sehr unterschiedlicher Probleme; so können sowohl glatte Lösungen mit hoher Genauigkeit approximiert werden, wie auch unstetige Lösungen effektiv berechnet werden. Im Gegensatz zum Standard Finite-Elemente-Verfahren wird beim Discontinuous-Galerkin-Verfahren keine Stetigkeit der Approximation über Elementgrenzen hinweg gefordert. Dadurch erhält die Approximation einen lokalen Charakter, der sich positiv auswirkt bei Maßnahmen zur Steigerung der Effizienz des Verfahrens, wie etwa bei Parallelisierung und Adaptivität. Des Weiteren lässt sich ein großer Teil der Algorithmen generisch implementieren, d.h. unabhängig vom Typ der partiellen Differentialgleichung und von den konkreten Daten. Um allerdings ein vielseitiges, effizientes und leicht wieder verwertbares Programm zu schreiben, sind einige Gesichtspunkte beim Design zu berücksichtigen. Dazu werden in der Vorlesung und anhand praktischer Übungen fortgeschrittene Methoden der C++-Programmiersprache besprochen.

Mario Ohlberger

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# KAPITEL 1

# Einleitung

**Ziel** (der Vorlesung)**.**
Entwicklung einer generischen Diskretisierung für "beliebige" partielle Differentialgleichungen und deren Implementierung in C++.

**Ansatz:** Local-Discontinuous-Galerkin-Verfahren (LDG). Bei Discontinuous-Galerkin-Verfahren wird der Ansatzraum mit stückweise polynomialen Funktionen gewählt, aber es wird keine Stetigkeit zwischen Gitterzellen gefordert.

**Resultat:** Bezeichnet $V$ den kontinuierlichen Lösungsraum, und $V_h$ den diskreten Lösungsraum, so gilt $V_h \not\subset X$!

$\Rightarrow$ LDG-Verfahren sind nicht konforme Finite-Elemente-Verfahren.

**Implementierung:** Möglichst allgemeiner Schnittstellen-Ansatz.

1) Ermittlung von Eigenschaften eines Gitters, die zur Diskretisierung von Partiellen DGLn benötigt werden.

2) Entwurf eines Interface für diese einzelnen Bausteine mit dem Ziel, keine Einbußen bei der Rechenzeit zu haben.

3) Zentrales Konzept: Template-Klassen

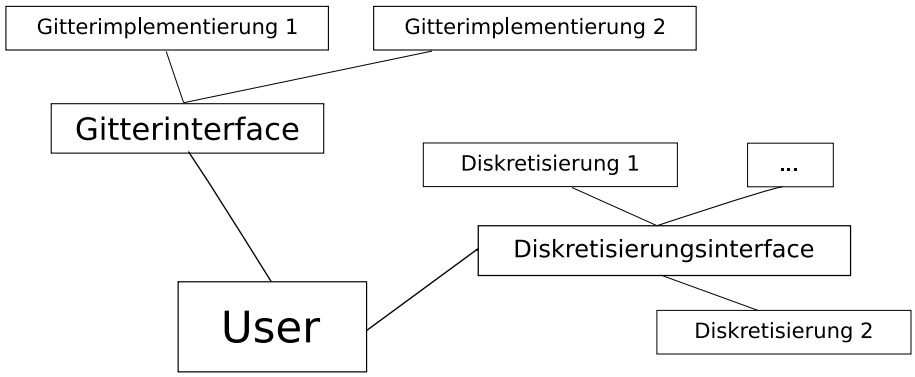



Abbildung 1.1: schematischer Aufbau des Interface

## 1.1 Finite Elemente für elliptische Probleme

**Definition 1.1** (Laplace-Problem)**.**
Seien $\Omega \subset \mathbb{R}^n$ ein Gebiet mit Lipschitz-Rand, $V = H_0^{1,2}(\Omega)$ und $f \in L^2(\Omega)$. Dann heißt $u \in V$ schwache Lösung von

$$\begin{aligned} -\Delta u &= f && \text{in } \Omega, \\ u &= 0 && \text{auf } \partial\Omega, \end{aligned}$$

falls gilt

$$a(u,v) = b(v) \qquad \forall v \in V, \tag{1.1}$$

mit

$$\begin{aligned} a(u,v) &:= \int_\Omega \nabla u \nabla v, \\ b(v) &:= \int_\Omega f v. \end{aligned}$$

**Definition 1.2** (Finite-Elemente-Methode (FEM))**.**

1. **Schritt:** Konstruktion eines Gitters $\mathcal{T}_h := \{T_i\}_{i \in I}$ auf $\Omega$, siehe Abbildung 1.2 (genauere Bedingungen s.u.).
2. **Schritt:** Wahl eines endlichdimensionalen Unterraumes $V_h \subset V$ mit einer Basis $\{v_1, \ldots, v_N\}$, $\dim(V_h) = N$. Dieser Raum wird anhand des Gitters $\mathcal{T}_h$ definiert (s.u.).
3. **Schritt:** Konstruktion von $u_h \in V_h$ mit

$$a(u_h, v_h) = b(v_h) \qquad \forall v_h \in V_h. \tag{1.2}$$

Dazu setzen wir an:

$$u_h(x) = \sum_{j=1}^{N} u_j v_j(x)$$

mit $u_j \in \mathbb{R}$ $(1 \le j \le N)$.
Man nennt $\{u_j\}_{j=1}^N$ die Freiheitsgrade von $u_h$ (degrees of freedom, DOF). Da $\{v_1, \ldots, v_N\}$

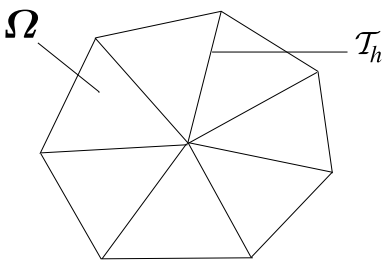



Abbildung 1.2: eine Triangulierung

eine Basis von $V_h$ ist, genügt es, (1.2) mit $v_i$, $i = 1, \dots, N$, zu testen:

$$\begin{aligned} & a(u_h, v_i) = b(v_i) && \forall i \in \{1, \dots, N\} \\ \Longrightarrow \quad & \sum_{j=1}^{N} u_j a(v_j, v_i) = b(v_i) && \forall i \in \{1, \dots, N\}. \end{aligned} \tag{1.3}$$

Mit

$$\begin{aligned} b &:= (b(v_i))_{i=1}^N \in \mathbb{R}^N, \\ A &:= (a(v_j, v_i))_{1 \le i,j \le N} \in \mathbb{R}^{N \times N}, \\ U &:= (u_j)_{j=1}^N \in \mathbb{R}^N, \end{aligned}$$

ist (1.3) äquivalent zu

$$AU = b,$$

d.h. die DOFs erhalten wir als Lösung eines linearen Gleichungssystems. Die Matrix $A$ heißt Steifigkeitsmatrix.

**4. Schritt:** Numerische Lösung von $AU = b$, d.h. Konstruktion einer Approximation $\overline{U} \in \mathbb{R}^N$, so dass $\|U - \overline{U}\|$ klein ist (oder alternativ so dass das Residuum $\|A\overline{U} - b\|$ klein ist).

**Also benötigen wir:**

1) Methoden zur Gitterkonstruktion und Verwaltung,
2) Konstruktion der Basisfunktionen $v_1, \dots, v_N$,
3) a) Berechnung der Einträge $b_i = \int_\Omega f v_i$, d.h. numerische Quadraturen,
   b) Berechnung der Steifigkeitsmatrix $A$,
4) Lösungsverfahren für lineare Gleichungssysteme.

Wir betrachten im folgenden Finite-Elemente-Verfahren basierend auf Referenzelementen.

## 1.2 Das Rechengitter

**Definition 1.3** (Dreiecksgitter in 2D, Referenzsimplex, Subentitäten)**.**
Seien im folgenden $n = 2$ und $\Omega \subset \mathbb{R}^2$ polygonal berandet. Sei $\mathcal{T}_h = \{T_i \mid i \in I := \{1, \dots, M\}\}$, wobei $T_1, \dots, T_M$ Dreiecke sind mit:

1) $\bigcup_{i=1}^N \overline{T_i} = \Omega$,
2) $\mathring{T}_i \cap \mathring{T}_j = \begin{cases} \emptyset, & i \neq j; \\ T_i, & i = j, \end{cases}$
3) $\overline{T_i} \cap \overline{T_j} = \begin{cases} \overline{T_i}, & i = j, \\ \emptyset, e_{ij}, p_k & \text{sonst}, \end{cases}$ mit $e_{ij}$ gemeinsame Kante von $T_i$ und $T_j$ und $p_k$ Knoten in $T_i$ und $T_j$.

**Definition der Referenzelemente:** Sei $\hat{p}_0 = (0,0),\ \hat{p}_1 = (1,0),\ \hat{p}_2 = (0,1)$ und $\hat{T}$ die konvexe Hülle von $\hat{p}_0,\ \hat{p}_1,\ \hat{p}_2$, d.h.

$$\hat{T} := \{(x,y) \mid 0 \le x \le 1,\ 0 \le y \le 1 - x\}\,.$$

$\hat{T}$ wird *Referenzdreieck* oder *Referenzsimplex* in 2D genannt.

$\hat{p}_0,\ \hat{p}_1,\ \hat{p}_2$ sind die Knoten oder *Subentitäten der Kodimension 2* (codim = 2, Codim-2-Entitäten) von $\hat{T}$ (allgemein: $l$ hat Kodimension $m \le n$, falls $l \in \mathbb{R}^{n-m}$).

$$\mathcal{E}^2(\hat{T}) := \{\hat{p}_0,\ \hat{p}_1,\ \hat{p}_2\}\,.$$

Die Kanten $\hat{e}_0,\ \hat{e}_1,\ \hat{e}_2$ (s. Abbildung 1.3) heißen Subentitäten der Kodimension 1,

$$\mathcal{E}^1(\hat{T}) := \{\hat{e}_0,\ \hat{e}_1,\ \hat{e}_2\}\,.$$

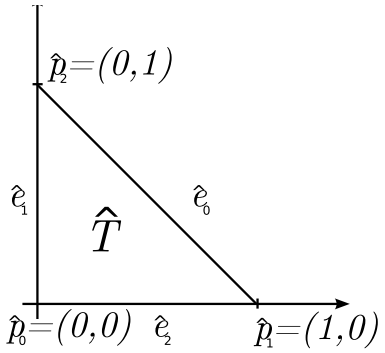



Abbildung 1.3: Referenzsimplex in 2D

**Definition 1.4** (Referenzabbildung)**.**
Die Elemente $T$ eines Dreiecksgitters $\mathcal{T}_h$ ergeben sich aus einer affinen Transformation des Referenzelementes, d.h. $T = F_T(\hat{T})$, wobei die *Referenzabbildung* $F_T$ gegeben ist durch $F_T(\hat{x}) = A_T x + b_T$ mit $A_T \in \mathbb{R}^{2\times 2},\ b_T \in \mathbb{R}^2$.

Die Menge der Subentitäten der Kodimension 1 von $T$ ist $\mathcal{E}^1(T) = \left({e_0}^T,\ {e_1}^T,\ {e_2}^T\right)$ mit ${e_i}^T := F_T(\hat{e}_i)$ und die Menge der Codim-2-Entitäten von $T$ ist $\mathcal{E}^2(T) = \left({p_0}^T,\ {p_1}^T,\ {p_2}^T\right)$ mit ${p_i}^T := F_T(\hat{p}_i)$.

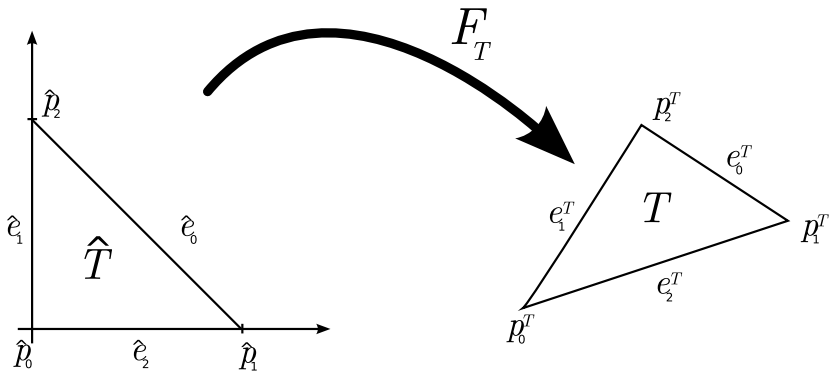



Abbildung 1.4: Referenzabbildung in 2D

**Bemerkung.**
Durch die Festlegung einer lokalen Nummerierung der Subentitäten auf dem Referenzelement wird eine lokale Nummerierung auf $T$ induziert.

**Definition 1.5** (Entitäten)**.**
*Entitäten* der Kodimension $c = 0, \dots, 2$ (bzw. Codim-$c$-Entitäten), sind:

$$\begin{aligned} c = 0: &\quad \mathcal{E}^0 = \{T \mid T \in \mathcal{T}_h\}, \\ c = 1: &\quad \mathcal{E}^1 = \bigcup_T \mathcal{E}^1(T) = \bigcup_T \{F_T(\hat{e}_0),\ F_T(\hat{e}_1),\ F_T(\hat{e}_2)\}, \\ c = 2: &\quad \mathcal{E}^2 = \bigcup_T \mathcal{E}^2(T) = \bigcup_T \{F_T(\hat{p}_0),\ F_T(\hat{p}_1),\ F_T(\hat{p}_2)\}. \end{aligned}$$

**Definition 1.6** (konforme Triangulierung)**.**
$\mathcal{T}_h$ ist eine *konforme Triangulierung*, falls $\bigcup_{T\in\mathcal{T}_h} \overline{T} = \overline{\Omega}$ und falls für $T, T' \in \mathcal{T}_h, T \neq T'$ gilt: $T \cap T' \neq \emptyset$ dann ist $T \cap T' \in \mathcal{E}^c(T) \cap \mathcal{E}^c(T')$ für ein $c \in \{1, 2\}$.

Für die Finite-Elemente-Funktionen müssen die Freiheitsgrade (DOFs) an die Entitäten "angehängt" werden. Etwa für lineare Finite Elemente:

$$v_h(x) = \sum_{p\in\mathcal{E}^2} v_p \varphi_p(x).$$

Wir benötigen zur Implementierung jedoch einen Ausdruck der Form: $\sum_{i=1}^{N} v_i \varphi_i(x)$.

**Definition 1.7** (Indexabbildung)**.**
Für $c = 0, 1, 2$ sei eine *Indexabbildung* $\mu^c : \mathcal{E}^c \to \mathbb{N}$, injektiv, gegeben. Zur Vereinfachung der Notation nehmen wir an, dass $\mu^c : \mathcal{E}^c \to \{1, \dots, N_c\}$ mit $N_c = |\mathcal{E}^c|$ eine bijektive Abbildung ist.

Wir schreiben auch häufig $p_i$ für die Codim-2-Entität $p \in \mathcal{E}^2$ mit $\mu^2(p) = i$.

Für $k = 0, 1, 2$ wird durch

$$\mu^{T,2}(k) := \mu^2({p_k}^T) \text{ bzw. } \mu^{T,1}(k) := \mu^1({e_k}^T)$$

eine Abbildung von lokaler zu globaler Nummerierung induziert (siehe Abb. 1.5).

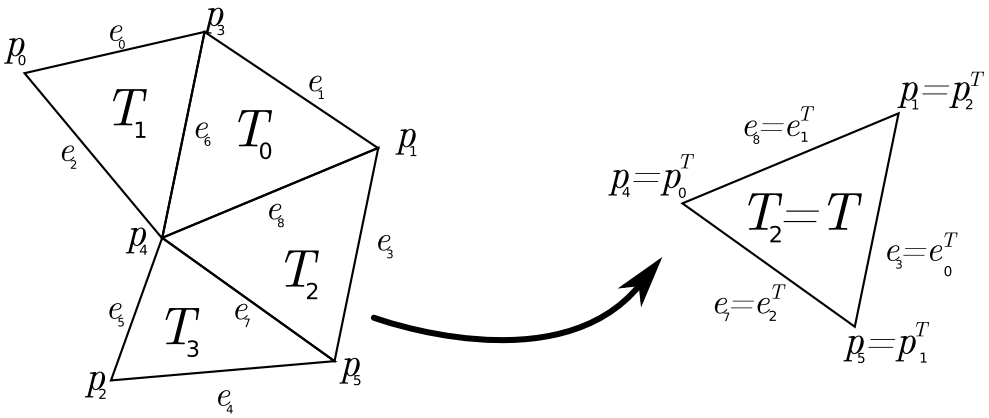



Abbildung 1.5: Globale und lokale Nummerierung.

## 1.3 Diskrete Funktionen

Wir betrachten zunächst stückweise lineare und stetige Finite Elemente Funktionen, d.h.

$$V_h := \left\{ v_h \in C^0(\Omega) \mid v_h|_T \in P_1(T) \; \forall T \in \mathcal{T}_h \right\}.$$

**Ziel.**
Wir benötigen eine Basis von $V_h$ und eine Vorschrift, um $v_h \in V_h$ auf einem gegebenen Element $T$ auszuwerten.

**Bemerkung.**
Bei der Implementierung einer Finite-Elemente-Methode stellt man fest, dass $v_h$ nie für ein $x \in \Omega$ ausgewertet werden muss, sondern immer nur $v_h(F_T(\hat{x}))$ für ein gegebenes Element $T \in \mathcal{T}_h$ und eine gegebene lokale Koordinate $\hat{x} \in \hat{T}$.

Daher konstruieren wir $v_h \in V_h$ durch: $v_h|_T =: v_T$ für $T \in \mathcal{T}_h$.

**Definition 1.8** (Basis der "Hütchenfunktionen")**.**
Eine Basis $\hat{\Phi}$ von $P_1(\hat{T})$ ist gegeben durch die Hütchenfunktionen $\hat{\varphi}_0$, $\hat{\varphi}_1$, $\hat{\varphi}_2$, definiert durch:

$$\hat{\varphi}_k(\hat{p}_l) := \delta_{kl}, \text{ für } 0 \leq k, l \leq 2.$$

Durch $\Phi^T = \left\{ {\varphi_0}^T, {\varphi_1}^T, {\varphi_2}^T \right\}$ mit ${\varphi_k}^T := \hat{\varphi}_k \circ {F_T}^{-1}$ wird eine Basis von $P_1(T)$ definiert.

Für $p \in \mathcal{E}^2$ mit $\mu^2(p) = i$ sei

$$\varphi_i := \begin{cases} 0, & p \notin \mathcal{E}^2(T), \\ {\varphi_k}^T, & p = {p_k}^T \text{ für ein } \; k \in \{0,1,2\} \text{ bzw. } \mu^{T,2}(k) = i \end{cases}$$

$\Longrightarrow \Phi = \{\varphi_1, \ldots, \varphi_{N_2}\}$ ist eine Basis von $V_h$.

**Bemerkung.**
Sei $T \in \mathcal{T}_h$, $\hat{x} \in \hat{T}$ und $\mu^{T,2}(k) = i$. Dann ist:

$$\varphi_i(F_T(\hat{x})) = {\varphi_k}^T(F_T(\hat{x})) = \hat{\varphi}_k(\hat{x}).$$

**Definition 1.9** (Lokale DOFs)**.**
Sei $v_h \in V_h$, dann ist $v_h = \sum_{i=1}^{N_2} v_i \varphi_i$. Es sind $(v_1, \ldots, v_{N_2})$ die DOFs von $v_h$. Zu $T \in \mathcal{T}_h$ sind ${v_k}^T := v_{\mu^{T,2}(k)}$ $(k = 0,1,2)$ die *lokalen Freiheitsgrade*. Es ist

$$v_h|_T = v_T = \sum_{i=0}^{2} {v_i}^T {\varphi_i}^T$$

bzw. $v_T(F_T(\hat{x})) = \sum_{i=0}^{2} {v_i}^T \hat{\varphi}_i(\hat{x})$.

**Allgemein:** $\nu^T : \hat{\Phi} \to \mathbb{N}$ mit $\nu_T(\hat{\varphi}_k) = \mu^2({p_k}^T)$ beschreibt eine Abbildung von lokalen DOFs auf globale DOFs.

Es ist

$$v_T(F_T(\hat{x})) = \sum_{\hat{\varphi} \in \hat{\Phi}} v_{\nu^T(\hat{\varphi})} \hat{\varphi}(\hat{x}).$$

Ist $\hat{\Phi} = (\hat{\varphi}_0, \ldots, \hat{\varphi}_k)$, so schreiben wir auch $\nu^T(k) = \nu^T(\hat{\varphi}_k)$.

## 1.4 Assemblieren des linearen Gleichungssystems

### 1.4.1 Die Rechte Seite

**Ziel.**
Berechnung von $b_i := \int_\Omega f\varphi_i$ für $i = 1, \dots, N_2$.

Falls $f$ keine "einfache" Funktion ist, benötigt man dafür numerische Quadraturen. Dazu transformieren wir die Integration zunächst auf eine Summe von Integrationen über dem Referenzelement.

Sei $g \in C^0(\overline{\Omega})$, dann gilt:

$$\begin{aligned}\int_\Omega g(x)dx &= \sum_{T\in\mathcal{T}_h} \int_T g(x)dx = \sum_{T\in\mathcal{T}_h} \int_{\hat{T}} g(F_T(\hat{x}))\ |\det DF_T(\hat{x})|\ d\hat{x} \\ &= \sum_{T\in\mathcal{T}_h} |\det A_T| \int_{\hat{T}} \hat{g}_T(\hat{x})d\hat{x},\end{aligned}$$

mit $\hat{g}_T = g \circ F_T \in C^0(\hat{T})$.

**Definition 1.10** (Quadraturen)**.**
Ein *Quadratur* $\hat{Q} = (W, X)$ mit $W = (\hat{w}_\alpha)_{\alpha=1}^r \subset \mathbb{R}, X = (\hat{x}_\alpha)_{\alpha=1}^r \subset \mathbb{R}^2$,

$$\hat{Q}(\hat{g}) := \sum_{\alpha=1}^r \hat{w}_\alpha \hat{g}(\hat{x}_\alpha),$$

approximiert für $\hat{g} \in C^0(\hat{T})$ das Integral über $\hat{T}$:

$$\hat{Q}(\hat{g}) \approx \int_{\hat{T}} \hat{g}(\hat{x})d\hat{x}.$$

$\hat{Q}$ heißt *exakt* auf einem Polynomraum $P_s(\hat{T})$, falls für alle $\hat{p} \in P_s(\hat{T})$ gilt:

$$\hat{Q}(\hat{p}) = \int_{\hat{T}} \hat{p}(\hat{x})d\hat{x}.$$

Für $g \in C^0(T)$ wird mittels

$$Q_T(g) := \sum_{\alpha=1}^r w_\alpha g(x_\alpha),$$

mit $x_\alpha = F_T(\hat{x}_\alpha)$ und $w_\alpha = |\det A_T|\hat{w}_\alpha$, eine Quadratur für $T \in \mathcal{T}_h$ induziert.

**Satz 1.11.**

*i) Ist $\hat{Q}$ exakt auf $P_s(\hat{T})$, so ist die induzierte Quadratur $Q_T$ exakt auf $P_s(T)$.*

*ii) Ist $\hat{Q}$ exakt auf $P_s(\hat{T})$ und sei $g \in C^\infty(\Omega)$, $Q(g) := \sum_{T\in\mathcal{T}_h} Q_T(g)$, so gilt:*

$$\left|\int_\Omega g(x)dx - Q(g)\right| \le Ch^{s+1},$$

*(mit z.B. $h := \max_{T\in\mathcal{T}_h} \operatorname{diam}(T)$).*

| $N$ | $logN$ | $N$ | $N \cdot logN$ | $N^2$ |
|---|---|---|---|---|
| $10^3$ | 7 | $10^3$ | $7 \cdot 10^3$ | $10^6$ |
| $10^4$ | 10 | $10^4$ | $10^5$ | $10^8$ |
| $10^5$ | 12 | $10^5$ | $12 \cdot 10^5$ | $10^{10}$ |

$10^8$ $\begin{cases} \frac{1}{100} \text{ Sekunde} & \rightsquigarrow \text{ 12 Tage} \\ \text{Speicher (8 byte)} & \rightsquigarrow \text{ 80 Mb} \end{cases}$

Abbildung 1.6: Daten für Aufwand zur Illustration

*Beweis:*

i) Nachrechnen; gilt, da det $A_T$ konstant

ii) Vgl. Vorlesung *Einführung in die Numerische Mathematik.*

□

**Bemerkung.**
Quadraturen mit hoher Ordnung können konstruiert werden durch die Forderung $\hat{Q}(\hat{p}) - \int_{\hat{T}} \hat{p}(\hat{x})dx = 0$ für ein $\hat{p}$ aus einer Basis von $P_s(\hat{T})$. Allerdings ergibt dies ein *nicht-lineares* gekoppeltes System algebraischer Gleichungen für $(\hat{x}_\alpha)$, $(\hat{w}_\alpha)$. In der Praxis ist schon $s = 10$ hoch.

Wir erhalten eine Approximation von $b_i$ der Form

$$b_i = \int_\Omega f\varphi_i \approx \sum_{T \in \mathcal{T}_h} Q_T(f\varphi_i).$$

Wir schreiben im weiteren $b_i = \sum_{T \in \mathcal{T}_h} Q_T(f\varphi_i),\ i = 1, \ldots, N_2$. Aufwand zur Berechnung von $b_i$ ist $O(|\mathcal{E}^2||\mathcal{E}^0|)$, d.h. quadratischer Aufwand.

Der optimale Aufwand zur Berechnung von $b$ ist sicherlich linear.

**Reduktion des Aufwandes:** Beitrag jedes Elementes zum *ganzen* Vektor $b$ am Stück berechnen.

Es ist:

$$b = \sum_{i=1}^{N_2} \sum_{T \in \mathcal{T}_h} Q_T(f \cdot \varphi_i)e_i = \sum_{T \in \mathcal{T}_h} \sum_{i=1}^{N_2} Q_T(f \cdot \varphi_i)e_i.$$

Mit

$$\varphi_i = \begin{cases} 0, & p_i \notin \mathcal{E}^2(T), \\ {\varphi_k}^T, & i = \mu^{T,2}(k) \end{cases}$$

folgt

$$\begin{aligned} b &= \sum_{T \in \mathcal{T}_h} \sum_{k=0}^{2} Q_T(f{\varphi_k}^T)e_{\mu^{T,2}(k)} \\ &= \sum_{T \in \mathcal{T}_h} \sum_{\hat{\varphi} \in \hat{\Phi}} Q_T(f(\hat{\varphi} \circ F_T))e_{\nu^T(\hat{\varphi})}, \end{aligned}$$

wobei $e_i$ den $i$-ten Einheitsvektor bezeichne.

Definieren wir nun die "lokalen rechten Seiten":

$$b^T = \left(b_k{}^T\right)_{k=0}^2, \text{ mit}$$
$$b_k{}^T = Q_T(f\varphi_k{}^T) \approx \int_T f\varphi_k{}^T,$$

dann ist $b = \sum_T \sum_{k=0}^2 b_k{}^T \overrightarrow{e}_{\nu^T(k)}$. Der Aufwand ist somit $O(|\mathcal{E}^0|)$.
**Bemerkung.**
Für $T \in \mathcal{T}_h$ und $\hat{\varphi} \in \hat{\Phi}$ ist

$$_k{}^T = Q_T(f\varphi_k{}^T) = \sum_{\alpha=1}^r |\det A_T| \hat{w}_\alpha f(F_T(\hat{x}_\alpha))\hat{\varphi}_k{}^T(F_T(\hat{x}_\alpha)).$$

**Implementierung:**

1) Berechne a-priori: $\hat{\varphi}_\alpha := \hat{\varphi}(\hat{x}_\alpha)$ für alle $\hat{\varphi} \in \hat{\Phi}$ ("Caching").
2) Setze $b = 0$. Für $T \in \mathcal{T}_h$: Berechne $d := |\det A_T|$, $y_\alpha := F_T(\hat{x}_\alpha)$, $f_\alpha := f(y_\alpha)$.
   Für $\hat{\varphi} \in \hat{\Phi}$: Berechne $\beta := \sum_{\alpha=1}^r \hat{w}_\alpha f_\alpha \hat{\varphi}_\alpha$ und schließlich (in C++ – Schreibweise)
$$b\left[\nu^T(\hat{\varphi})\right] += d\beta.$$

### 1.4.2 Die Steifigkeitsmatrix

Wie wir in Abschnitt 1.1 gesehen haben, ist die Steifigkeitsmatrix für das Poisson-Problem wie folgt gegeben

$$A = (a_{ij})_{1\le i,j\le N_2}, \qquad a_{ij} := \int_\Omega \nabla\varphi_j \cdot \nabla\varphi_i.$$

Mit der Strategie aus 1.4.1 betrachten wir zunächst lokale Steifigkeitsmatrizen:

$$a^T := \left(\int_T \nabla\varphi_k{}^T \cdot \nabla\varphi_l{}^T\right)_{l,k=0}^2.$$

Damit erhalten wir

$$A = \sum_T \sum_{k,l=0}^2 a_{kl}{}^T E_{\nu^T(k),\nu^T(l)},$$

mit der Matrix $E_{ij} = (e_{kl})_{k,l=0,\dots,2} \in \mathbb{R}^{N_2\times N_2}$ mit $e_{kl} = \delta_{ik}\delta_{jl}$.

Wieder erreichen wir durch die Lokalisierung eine Reduktion des Aufwands auf $O(|\mathcal{E}^0|)$ (anstelle von $O(|\mathcal{E}^0||\mathcal{E}^2|)$).

Zur Berechnung der Element-Steifigkeitsmatrizen $aT$ verwenden wir wieder die Transformation auf das Referenzelement:

$$\begin{aligned} a_{kl}{}^T &= \int_T \nabla\varphi_k{}^T \cdot \nabla\varphi_l{}^T \\ &= \int_T \underbrace{|\det A_T|}_{=\text{const}} \underbrace{A_T^{-\top}\nabla\hat{\varphi}_k}_{=\text{const}} \cdot \underbrace{A_T^{-\top}\nabla\hat{\varphi}_l}_{=\text{const}}, \end{aligned}$$

Dass die Ausdrücke konstant sind gilt natürlich nur, falls $\hat{\varphi}_k, \hat{\varphi}_l$ linear sind und $T$ ein Dreieck ist. Dabei haben wir verwendet, dass gilt ${\varphi_k}^T = \hat{\varphi}_k \circ F_T^{-1}$.

Im Gegensatz zum Rechenaufwand ist der Speicheraufwand zunächst von der Ordnung $O(|\mathcal{E}^0|^2)$, obwohl die Matrix $A$ dünn besetzt ist, d.h. man speichert im wesentlichen Nullen ab.
**Idee** (Compressed Row Storage (CRS))**.**
Ist $a_i$ die $i$-te Zeile von $A$, so ist $a_{ij} = 0$, falls keine Kante zwischen $p_i$ und $p_j$ existiert, d.h. es gibt kein Dreieck $T \in \mathcal{T}_h$, welches $p_i$ und $p_j$ als Knoten enthält (siehe Abb 1.7).

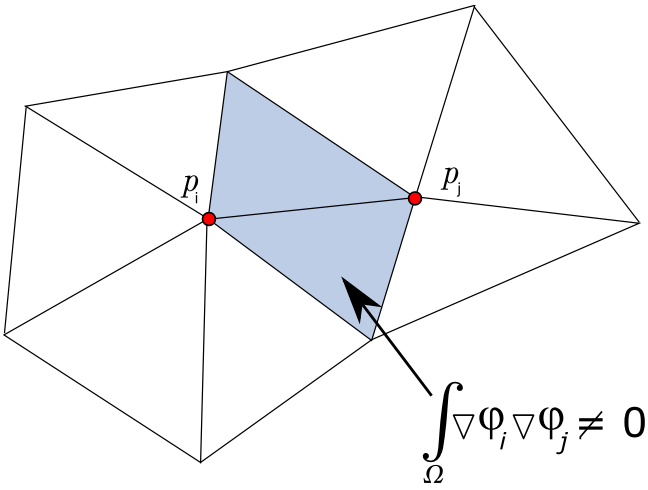



Abbildung 1.7: Gemeinsamer Träger von Basisfunktionen liefert Matrixeintrag. )

Also entspricht die Anzahl der Einträge $a_{ij} \neq 0$ der Anzahl der Kanten mit $p_i$ als Eckknoten.

**Ansatz:** Speichere nur Werte $a_{ij} \neq 0$ und zusätzlich die Spaltenposition $j$. Sei dazu $M > 0$ eine obere Schranke für die Anzahl der Codim-1-Entitäten, die einen Knoten gemeinsam haben (z.B. a-priori gewählt). Wir speichern Matrizen $S, C \in \mathbb{R}^{N_2 \times M}$, wobei die Zeilen $S_i, C_i$ wie folgt berechnet werden:

$$S_i = (a_{ij_1}, a_{ij_2}, \ldots, a_{ij_{M_i}}, 0\ ,\ \ldots\ ,\ 0),$$
$$C_i = (j_1,\ \ j_2,\ \ldots,\ j_{M_i}, -1, \ldots, -1).$$

Dabei sind $a_{ij_k}$ die Einträge ungleich Null.

Mit den Matrizen $S, C$ erhalten wir die Einträge der Matrix $A$ wie folgt

$$a_{ij} = \begin{cases} 0, & C_{ik} \neq j, k = 1, \ldots, M, \\ S_{ik}, & C_{ik} = j. \end{cases}$$

Der Speicheraufwand für CRS ist $O(|\mathcal{E}^0|)$.

## 1.5 Lösen des Linearen Gleichungssystems $AU = b$

**Bemerkung.**
Da $A$ dünn besetzt ist, eignet sich kein direkter Löser (es tritt ein "fill-in"-Effekt auf). Ein effizienter Ansatz sind Krylov-Unterraum-Methoden. Aus der Numerik II bekannt ist das cg-Verfahren für positiv definite symmetrische Matrizen. Ist $A$ nicht symmetrisch, existieren Verallgemeinerungen (z.B. gmres, bcgstab). Wir verweisen hier auf weiterführende Literatur, z.B. [12], [14].

**Bemerkung** (Weitere Verbesserung).
Mehrgitter-Löser und Vorkonditionierung (siehe [14]).
Bei den Krylov-Unterraum-Methoden wird die Matrix $A$ selbst nicht benötigt, sondern nur Matrix-Vektor-Multiplikation der Form $y = Ax$ für gegebenes $x \in \mathbb{R}^{N_2}$.

**Implementierung mit CRS-Matrizen:** Sei $x \in \mathbb{R}^{N_2}$ gegeben, $A$ eine CRS-Matrix und $y \equiv 0$. Dann sieht ein typischer Algorithmus zur Matrix-Vektor-Multiplikation wie folgt aus (Aufwand: $O(|\mathcal{E}^2|)$):

```
1 for( int i = 1; i < N_2; i++ )
2 {
3   k = 1;
4   while( C_ik != - 1 )
5   {
6     y_i = S_ik * x_C_ik;
7     k++;
8   }
9 }
```

## 1.6 Verallgemeinerte elliptische PDGLn

### Verallgemeinerung mit Termen niedrigerer Ordnung

**Definition** (Allgemeine elliptische Partielle Differentialgleichung).
Seien $a(x) \in \mathbb{R}^{n\times n}$, $b(x) \in \mathbb{R}^n$ und $c(x) \in \mathbb{R}$.

$$-\nabla \cdot (a(x)\nabla u(x)) + \nabla \cdot (b(x)u(x)) + c(x)u(x) = f(x).$$

Zusätzlich sei die Gleichung elliptisch (z.B. $a$ uniform symmetrisch und positiv definit, $b$ divergenzfrei und $c(x) \geq 0$).

**Finite Elemente Ansatz:** $u_h(x) = \sum_j u_j\varphi_j(x)$. Dann folgt:

$$\sum_j u_j \left( \int_\Omega a(x)\nabla\varphi_j\nabla\varphi_i - \int_\Omega (b(x)\varphi_j)\nabla\varphi_i + \int_\Omega c(x)\varphi_j\varphi_i \right) = \int_\Omega f(x)\varphi_i, \qquad \forall\varphi_i \in V_h \subset H_0^1(\Omega).$$

Dies führt auf ein LGS $AU = b$ mit der Matrix $A = S - G + M$,

$$\text{Steifigkeitsmatrix: } S := \left( \int_\Omega a(x)\nabla\varphi_j\nabla\varphi_i \right)_{1\leq i,j\leq N_2},$$

$$\text{Massematrix: } M := \left( \int_\Omega c(x)\varphi_j\varphi_i \right)_{1\leq i,j\leq N_2},$$

$$\text{Transportmatrix: } G := \left( \int_\Omega (b(x)\varphi_j)\nabla\varphi_i \right)_{1\leq i,j\leq N_2}.$$

Zur Berechnung der Matrixeinträge und zur Speicherung der Matrix können die Konzepte aus Abschnitt wie in 1.4 verwendet werden. Die Matrix $A$ ist dünn besetzt, aber im Allgemeinen nicht symmetrisch.

Hier benötigt man jetzt auch Quadraturformeln zur Berechnung von $A$, falls $a, b, c$ vom Ort abhängen.

### Allgemeine Randbedingungen

Bisher: $u = 0$ auf $\partial\Omega$. Betrachte nun:

$$\begin{aligned} -\nabla\cdot(a\nabla u) + \nabla\cdot(bu) + cu &= f && \text{in } \Omega, \\ u &= g_D && \text{auf } \partial\Omega_D, \ \text{(Dirichlet)} \\ (a\nabla u - bu)\cdot n &= g_N && \text{auf } \partial\Omega_N, \ \text{(Neumann)} \\ (a\nabla u - bu) + \alpha u &= g_R && \text{auf } \partial\Omega_R, \ \text{(Robin)}, \end{aligned}$$

wobei $\partial\Omega = \partial\Omega_D \cup \partial\Omega_N \cup \partial\Omega_R$, $\alpha \neq 0$, und $n$ die äußere Normale an $\partial\Omega$ ist.

**Schwache Formulierung:** Sei

$$\varphi \in H^1_{\partial\Omega_D} := \overline{\{\varphi \in C^\infty : \varphi = 0 \text{ auf } \partial\Omega_D\}}.$$

$$\begin{aligned} \int_\Omega f\varphi = &\int_\Omega a\nabla u\nabla\varphi - \int_{\partial\Omega} a\nabla u\cdot n\varphi \\ &- \int_\Omega bu\nabla\varphi + \int_{\partial\Omega} bu\cdot n\varphi + \int_\Omega cu\varphi \end{aligned}$$

$$\begin{aligned} &\underbrace{\int_\Omega (a\nabla u\nabla\varphi - bu\nabla\varphi + cu\varphi)}_{=:a(u,\varphi)} - \int_{\partial\Omega}(a\nabla u - bu)\cdot n\varphi \\ &= \underbrace{\int_\Omega f\varphi}_{=:b(\varphi)} + a(u,\varphi) - \underbrace{\int_{\partial\Omega_D}(a\nabla u - bu)\cdot n\varphi}_{=0,\ \text{da } \varphi|_{\partial\Omega_D}=0} \\ &\quad - \underbrace{\int_{\partial\Omega_N}(a\nabla u - bu)\cdot n\varphi}_{=\int_{\partial\Omega_N} g_N\varphi} - \underbrace{\int_{\partial\Omega_R}(a\nabla u - bu)\cdot n\varphi}_{=\int_{\partial\Omega_R}(g_R\varphi - \alpha u)\varphi} = b(\varphi). \end{aligned}$$

**Gesucht:** $u = \overline{g_D} + H^1_{\partial\Omega_D}$, mit $\overline{g_D} \in H^1$ mit $\text{spur}(\overline{g_D}) = g_D$ auf $\partial\Omega_D$, mit

$$a(u,\varphi) + \int_{\partial\Omega_R}\alpha u\varphi = b(\varphi) + \int_{\partial\Omega_N} g_N\varphi + \int_{\partial\Omega_R} g_R\varphi \ \forall\varphi \in H^1_{\partial\Omega_D}.$$

### Implementierung (der Randbedingungen)

Zur Implementierung der Randbedingung transformieren wir das Problem zunächst auf ein Problem mit Nullrandwerten auf dem Dirichletrand, d.h. wir setzen $v = u - \overline{g_D} \in H^1_{\partial\Omega_D}$ und erhalten für $v$:

$$\begin{aligned} a(v,\varphi) + \int_{\partial\Omega_R}\alpha v\varphi = &b(\varphi) + \int_{\partial\Omega_N} g_N\varphi + \int_{\partial\Omega_R} g_R\varphi \\ &+ a(\overline{g_D},\varphi) + \int_{\partial\Omega_R}\alpha\overline{g_D}\varphi, \qquad \forall\varphi \in H^1_{\partial\Omega_D}. \end{aligned}$$

Zur Implementierung dieser Formulierung sind folgende Modifikationen nötig:
**Modifikation der Massematrix:** Zur Berücksichtigung des Randintegralterms auf der Rechten Seite modifiziert sich die resultierende Massematrix $M$ zu:

$$M_{ij} := \int_\Omega c\varphi_i\varphi_j + \int_{\partial\Omega_R} \alpha\varphi_i\varphi_j.$$

**Modifikation der Rechten Seite:**

$$\begin{aligned} b_i := &\int_\Omega f\varphi_i + \int_{\partial\Omega_R} g_R\varphi_i + \int_{\partial\Omega_N} g_N\varphi_i \\ &+ a(\overline{g_{D,h}}, \varphi_i) + \int_{\partial\Omega_R} \alpha\overline{g_{D,h}}\varphi_i \end{aligned}$$

Dabei ist $\overline{g_{D,h}}$ im Falle von linearen Finiten Elementen definiert durch $\overline{g_{D,h}}(p_i) = g_D(p_i)$, falls $p_i \in \partial\Omega_D$ und $\overline{g_{D,h}}(p_i) = 0$ sonst.
**Dirichlet Randwerte:** Alle Freiheitsgrade, die auf $\partial\Omega_D$ "liegen" (im Fall linearer FE sind das die $u_i$ mit $p_i \in \partial\Omega_D$) sind festgelegt. Sie könnten aus dem Gleichungssystem $AU = b$ entfernt werden, z.B. durch Streichen der $i$-ten Zeile und Spalte. Im allgemeinen ist es aber einfacher, die Freiheitsgrade im System zu belassen und die Matrix $A$ und die rechte Seite $b$ anzupassen:

$$i \to \begin{pmatrix} & & & 0 & & & \\ & & & \vdots & & & \\ & & & 0 & & & \\ 0 & \dots & 0 & 1 & 0 & \dots & 0 \\ & & & 0 & & & \\ & & & \vdots & & & \\ & & & 0 & & & \\ & & & \uparrow & & & \\ & & & i & & & \end{pmatrix} \begin{pmatrix} \vdots \\ u_i \\ \vdots \end{pmatrix} = \begin{pmatrix} b_1 \\ \vdots \\ g_D(p_i) \\ \vdots \\ b_{N_2} \end{pmatrix}.$$

## 1.7 Das Discontinuous-Galerkin-Verfahren

Wir betrachten das Problem

$$-\Delta u = f \qquad \text{in } \mathbb{R}^2, \tag{1.4a}$$
$$u = 0 \qquad \text{auf } \partial\Omega. \tag{1.4b}$$

Ist $f \in L^2(\Omega)$, $\Omega$ glattes, konvexes Gebiet, so hat (1.4) genau eine Lösung in $H^2(\Omega) \cap H_0^1(\Omega)$.

Sei $\mathcal{T}_h$ eine Zerlegung von $\Omega$ (etwa ein konformes Dreiecksgitter) und sei

$$H^2(\mathcal{T}_h) := \left\{ v \in L^2(\Omega) \mid v|_T \in H^2(T)\ \forall T \in \mathcal{T}_h \right\}$$

der Raum der stückweise $H^2$-Funktionen.

Die Lösung $u$ von (1.4) ist in $H^2(\mathcal{T}_h)$, aber die Gleichung (1.4) ist auf $H^2(\mathcal{T}_h)$ nicht wohlgestellt.

Man kann nur $-\Delta u = f$ auf $T\ \forall T \in \mathcal{T}_h$ fordern. Dieses Problem hat viele Lösungen, so ist etwa für $f = 0$ nicht nur die Nulllösung eine Lösung, sondern etwa alle konstanten Funktionen. Die Lösung $u$ von (1.4) erfüllt zusätzlich noch, dass $u$ zwischen Elementen stetig ist ($u \in C^0(\Omega)$), und auch dass $\nabla u \cdot n$ zwischen zwei benachbarten Elementen keinen Sprung hat, wobei $n$ die Normale an die Kante zwischen zwei Elementen ist.

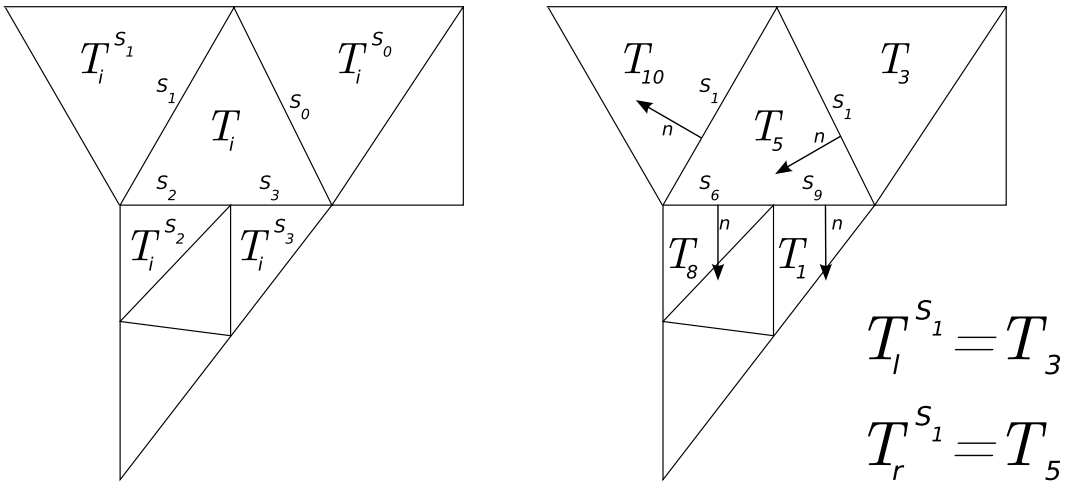



Abbildung 1.8: Beispiel Intersections

**Definition 1.12** (Intersections)**.**
Zu $T \in \mathcal{T}_h$ definieren wir die Menge der *Intersections* von $T$ mit den anderen Elementen des Gitters mittels

$$\mathcal{I}(T_i) := \left\{ S \subset \Omega, S = \overline{T_i} \cap \overline{T_i}^s, T_i^s \in \mathcal{T}_h, T_i^s \neq T_i \text{ und } |S| \neq 0 \text{ in } \mathbb{R}^{n-1} \right\}.$$

Zu $S \in \mathcal{I}(T_i)$ bezeichnen wir mit $T_i^s \in \mathcal{T}_h$ das Element mit $S = \overline{T_i} \cap \overline{T_i}^s$ das Nachbarelement an $T$ über $S$.

Da wir eine nicht überlappende Zerlegung von $\Omega$ annehmen, ist $S \subset \partial T_i \cap \partial T_i^s$. Ist $S \in \mathcal{I}(T_i)$, so ist $S \in \mathcal{I}(T_i^s)$.
**Beispiel.**
Ist $\mathcal{T}_h$ eine konforme Triangulierung, so ist $\mathcal{I}(T_i) = \{e_0^{T_i}, e_1^{T_i}, e_2^{T_i}\}$, falls $\partial T_i \cap \partial\Omega = \emptyset$.

Wir definieren die Menge aller Intersections von $\mathcal{T}_h$ durch :

$$\mathcal{I} := \left\{ S \in \mathcal{I}(T_i) \mid T_i \in \mathcal{T}_h \text{ mit } \mu^0(T_i) < \mu^0(T_i^s) \right\}.$$

Ist $S \in \mathcal{I}$, dann ist $S \in \mathcal{I}(T_i)$ mit $T_i \in \mathcal{T}_h$ und $S = \overline{T_i} \cap \overline{T_i}^s$.

Wir setzen: $T_S{}^l := T_i$ und $T_S{}^r := T_i^s$ und $n_S$ sei die Einheitsnormale an $S$, die von $T_S{}^l$ nach $T_S{}^r$ zeigt, d.h. $n_S$ ist die äußere Normale an $\partial T_S{}^l$. Wir definieren noch:

$$\mathcal{I}_\partial := \{ S \subset \partial\Omega \mid \exists T \in \mathcal{T}_h, \text{ sodass } S = \partial\Omega \cap \partial T \}.$$

Wir definieren zu $S \in \mathcal{I}_\partial$, $T_S{}^l = T$ und $n_S$ als die äußere Normale an $\partial\Omega$.
**Bemerkung.**
Bei einer konformen Triangulierung ist $\mathcal{I} = \mathcal{E}^1$, wobei durch $\mathcal{I}$ jeder Kante eindeutig eine Normale zugeordnet ist und die Elemente "vor" und "hinter" der Kante.

**Definition 1.13** (Sprung, Mittel)**.**
Sei $\varphi \in H^2(\mathcal{T}_h)$, $S \in \mathcal{I}$ und $x \in S$. Dann ist:

$$\varphi^-(x) := \lim_{\varepsilon \searrow 0} \varphi(x - \varepsilon n_S) = \lim_{\varepsilon \searrow 0} \varphi|_{T_S{}^l}(x - \varepsilon n_S),$$
$$\varphi^+(x) := \lim_{\varepsilon \searrow 0} \varphi(x + \varepsilon n_S) = \lim_{\varepsilon \searrow 0} \varphi|_{T_S{}^r}(x + \varepsilon n_S).$$

Des Weiteren definieren wir noch den *Sprung* von $\varphi$ über $S$ durch:

$$[\varphi](x) := \varphi^+(x) - \varphi^-(x),$$

und das *Mittel* von $\varphi$:

$$\{\varphi\}(x) := \frac{1}{2}(\varphi^+(x) + \varphi^-(x)).$$

Für $S \in \mathcal{I}_\partial, x \in S$ sei:

$$\begin{aligned}[\varphi](x) &:= -\varphi^-(x),\\ \{\varphi\}(x) &:= \varphi(x).\end{aligned}$$

**Lemma 1.14.**
*Sei $S \in \mathcal{I}, \varphi \in H^2(\mathcal{T}_h), x \in S$. Dann gilt*

*i) Ist zusätzlich $\varphi \in C^0({T_S}^l \cup {T_S}^r)$, so ist $[\varphi](x) = 0, \{\varphi\}(x) = \varphi(x)$.*

*ii) Ist $\psi \in H^2(\mathcal{T}_h)$, so gilt $[\varphi\psi](x) = [\varphi](x)\{\psi\}(x) + [\psi](x)\{\varphi\}(x)$.*

*iii) Ist $\psi \in C^0({T_S}^l \cup {T_S}^r)$, so folgt $[\varphi\psi](x) = [\varphi](x)\psi(x)$.*

*Beweis:* i) ist klar, iii) ergibt sich aus ii) und i).
zu ii)

$$\begin{aligned}[\varphi]\{\psi\} + [\psi]\{\varphi\} =& \frac{1}{2}(\varphi^+ - \varphi^-)(\psi^+ + \psi^-)\\ &+ \frac{1}{2}(\varphi^+ + \varphi^-)(\psi^+ - \psi^-)\\ =& \frac{1}{2}(2\varphi^+\psi^+ - 2\varphi^-\psi^-) = [\varphi\psi]\end{aligned}$$

□

**Lemma 1.15** (partielle Integration in $H^2(\mathcal{T}_h)$)**.**
*Sei $\varphi \in H^2(\mathcal{T}_h), v \in H^2(\mathcal{T}_h)^n$. Dann gilt:*

$$-\sum_T \int_T \nabla \cdot v\varphi = \sum_T \int_T v \cdot \nabla\varphi + \sum_{S \in \mathcal{I} \cup \mathcal{I}_\partial} \int_S [\varphi]\{v \cdot n_S\} + \sum_{S \in \mathcal{I}} \int_S [v \cdot n_S]\{\varphi\}.$$

*Beweis:*

$$\begin{aligned}-\sum_T \int_T \nabla \cdot v\varphi &= \sum_T \left( \int_T v \cdot \nabla\varphi - \int_{\partial T} v \cdot n\varphi \right)\\ &= \sum_T \int_T v \cdot \nabla\varphi - \sum_T \int_{\partial T} v \cdot n\varphi.\end{aligned}$$

In der zweiten Summe taucht jedes $S \in \mathcal{I}$ zweimal auf, einmal für $T = {T_S}^l$ und für $T = {T_S}^r$. Die

Normale $n$ an $\partial T_S{}^l$ ist gleich $n_S$ und für $T = T_S{}^r$ gilt: $n = -n_S$. Also folgt

$$\begin{aligned}-\sum_T \int_{\partial T} v \cdot n\varphi &= -\sum_T \int_{\partial T \cap \partial\Omega} v \cdot n\varphi \\ &\quad -\sum_S \int_S \left((v \cdot n_S\varphi)|_{T_S{}^l} - (v \cdot n_S\varphi)|_{T_S{}^r}\right) \\ &= \sum_{S \in \mathcal{I} \cup \mathcal{I}_\partial} \int_S [v \cdot n\varphi]\end{aligned}$$

Die Behauptung folgt mit Lemma 1.14 ii). □

Wir wenden Lemma 1.15 auf die Gleichung $-\Delta u = f$ auf $T$ an.

Sei $\varphi \in H^2(\mathcal{T}_h)$,

$$\begin{aligned}\Longrightarrow \sum_T \int_T f\varphi &= -\sum_T \int_T \Delta u\varphi \\ &= \sum_T \int_T \nabla u \cdot \nabla\varphi + \sum_{S \in \mathcal{I}} \int_S ([\nabla u \cdot n_S]\{\varphi\}) \\ &\quad + \sum_{S \in \mathcal{I} \cup \mathcal{I}_\partial} \int_S (\{\nabla u \cdot n_S\}[\varphi])\,.\end{aligned}$$

Ist $u$ Lösung von (1.4), so folgt $[\nabla u \cdot n_S] = 0$, d.h.

$$\int_\Omega f\varphi = \sum_T \int_T \nabla u \cdot \nabla\varphi + \sum_{S \in \mathcal{I} \cup \mathcal{I}_\partial} \int_S (\{\nabla u \cdot n_S\}[\varphi])\,. \tag{1.5}$$

Ist $u \in H^2(\Omega) \cap H_0^1(\Omega)$, so ist $u$ auch klassische Lösung von (1.5). Allerdings gilt die Umkehrung nicht. So ist etwa jede stückweise konstante Funktion Lösung von (1.5) mit $f = 0$, aber nur $u = 0$ ist Lösung von (1.4) mit $f = 0$.

Wir benötigen also weitere Terme, die berücksichtigen, dass die Lösung $u$ von (1.4) auch die folgenden Gleichungen erfüllt:

$$\begin{aligned}[u] &= 0 && \text{für alle } S \in \mathcal{I}, \\ [\nabla u \cdot n_S] &= 0 && \text{für alle } S \in \mathcal{I}.\end{aligned}$$

**Definition** (Kontinuierliches DG-Problem)**.**
Wir betrachten das folgende schwache Problem:

$$B_{\alpha,\beta}(u,\varphi) = I(\varphi) \qquad \text{für alle } \varphi \in H^2(\mathcal{T}_h), \tag{1.6}$$

mit $I(\varphi) := \int_\Omega f\varphi$ und für $\alpha \in \mathbb{R}, \beta \geq 0$:

$$\begin{aligned}B_{\alpha,\beta}(v,\varphi) := &\sum_T \int_T \nabla v \cdot \nabla\varphi + \sum_{S \in \mathcal{I} \cup \mathcal{I}_\partial} \int_S \{\nabla v \cdot n_S\}[\varphi] \\ &- \alpha \sum_{S \in \mathcal{I} \cup \mathcal{I}_\partial} \int_S [v]\{\nabla\varphi \cdot n_S\} + \beta \sum_{S \in \mathcal{I} \cup \mathcal{I}_\partial} {h_S}^{-1} \int_S [v][\varphi],\end{aligned}$$

wobei $h_S$ eine geeignete lokale Gitterfunktion sei (z.B. $h_S := \operatorname{diam}(S)$).

**Bemerkung.**
Ist $u$ die Lösung von (1.4), so ist $u$ Lösung von (1.6).

"Die Umkehrung gilt, falls $\beta > 0$" (ohne Beweis).

**Bemerkung** (Spezielle Bilinearformen/Ansätze)**.**

*Interior Penalty (IPG):* $\alpha = -1, \beta$ groß.
Nur falls $\alpha = -1$ ist $B_{\alpha,\beta}$ symmetrisch.

*Babuska, Oden, Baumann:* $\alpha = 1, \beta = 0$

*Stabilisiertes, nicht-symmetrisches Verfahren (NIPG):* $\alpha = 1, \beta > 0$.

$\beta > 0$ bestraft Unstetigkeiten in $u$.

**Bemerkung.**
Für $\alpha = -1$ ist $B_{\alpha,\beta}$ symmetrische Bilinearform; für $u \in C^0$ fallen alle zusätzlichen Terme weg.

**Definition** (Diskretes DG-Problem)**.**
Gesucht: $u_h \in V_h$ mit

$$B_{\alpha,\beta}(u_h, \varphi_h) = I(\varphi_h) \qquad \forall \varphi_h \in V_h$$

mit

$$V_h := \left\{ \varphi_h \in L^2(\Omega) \mid \varphi_h|_T \in P_{p_T}(T) \forall T \in \mathcal{T}_h \right\}.$$

Dabei sei $p_T \geq 1$ für alle $T \in \mathcal{T}_h$.

Wahl einer Basis von $V_h$: Sei ${\varphi_1}^T, \ldots, {\varphi_{d_i}}^T$ eine Basis von $P_{p_{T_i}}(T_i)$ mit $d_i = \dim(P_{p_{T_i}}(T_i))$. Dann ist die Basis $\Phi$ von $V_h$ gegeben durch

$$\Phi := \left\{ {\varphi_k}^{T_i} \mid T_i \in \mathcal{T}_h, 1 \leq k \leq d_i \right\}.$$

Die lokalen Basisfunktionen können wieder mittels $\hat{T}$ definiert werden:

$${\varphi_k}^{T_i} := \hat{\varphi}_{i,k} \circ F_{T_i}^{-1},$$

wobei $\hat{\varphi}_{i,0}, \ldots, \hat{\varphi}_{i,d_i}$ Basis ist von $P_{p_{T_i}}(\hat{T})$ ist.
**Beispiel.**

1) Lagrange-Basis auf $P_{p_{T_i}}(\hat{T})$ (s. Abbildung 1.9)
2) Orthonormalbasis zu den Monomen, d.h. $1, X, Y, X^2, Y^2, XY, \ldots$ orthonormalisieren mit Gram-Schmidt.

In beiden Fällen gilt $P_{p_{T_i}}(\hat{T}) \subset P_{p_{T_j}}(\hat{T})$, falls $p_{T_i} \leq p_{T_j}$.

**Implementierung:** Die Assemblierung des LGS geht analog zu klassischen Finite-Elemente-Verfahren.

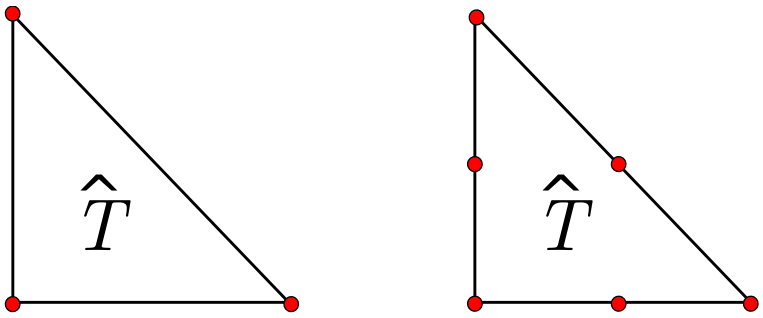



Abbildung 1.9: Stützstellen der Lagrange-Basisfunktionen für $p_{T_i} = 1$ (links) und $p_{T_i} = 2$ (rechts)

**Unterschied:** Die Systemmatrix $A$ ist eine schwachbesetzte *Blockmatrix*:

$$A = \begin{pmatrix} \begin{pmatrix} \dots \\ \dots \\ \dots \end{pmatrix} & \begin{pmatrix} \dots \\ \dots \\ \dots \end{pmatrix} & \dots & \begin{pmatrix} \dots \\ \dots \\ \dots \end{pmatrix} \\ \begin{pmatrix} \dots \\ \dots \\ \dots \end{pmatrix} & \begin{pmatrix} \dots \\ \dots \\ \dots \end{pmatrix} & \ddots & \begin{pmatrix} \dots \\ \dots \\ \dots \end{pmatrix} \\ \begin{pmatrix} \dots \\ \dots \\ \dots \end{pmatrix} & \begin{pmatrix} \dots \\ \dots \\ \dots \end{pmatrix} & \dots & \begin{pmatrix} \dots \\ \dots \\ \dots \end{pmatrix} \end{pmatrix},$$

mit Blöcken $A_{ij} \in \mathbb{R}^{d_i \times d_j}$ (siehe Übungsaufgabe).

### Vergleich DG versus CG:

**Nachteile:**

- Viel mehr Freiheitsgrade!
  Zum Beispiel bei linearen Ansatzfunktionen: CG $\to |\mathcal{E}^2|$, DG $\to 3|\mathcal{E}^0|$.
- Das resultierende LGS ist im Allgemeinen schwerer zu lösen. Für $a \neq -1$ kann beispielsweise das cg-Verfahren nicht eingesetzt werden. Stattdessen müssen teurere Verfahren wie z.B. bicgstab verwendet werden.

**Vorteile:**

- Flexibler Einsatz von Adaptivität, auch $p$-Adaption (d.h. $p_T$ ist nicht konstant), aber auch $h$-Adaption, da DOFs lokal auf den Elementen "leben".
- Nicht konforme Verfeinerung, d.h. auch non-matching Gitter sind einfach zu behandeln.
- Keine speziellen Basisfunktionen.
- Parallelisierung

# KAPITEL 2

# Local-Discontinuous-Galerkin-Verfahren

In diesem Abschnitt führen wir ein sehr allgemeines Konzept zur Approximation partieller Differentialgleichungen mit sogenannten *Local-Discontinuous-Galerkin-Verfahren* ein. Wir orientieren uns dabei an dem Artikel von Arnold, Brezzi, Cockburn und Marini *Unified analysis of discontinuous Galerkin methods for elliptic problems* [6].

Die Grundlegende Idee dieses Ansatzes ist die Umformulierung von partiellen Differentialgleichungen höherer Ordnung in ein System von Differentialgleichungen erster Ordnung. In diesem Kapitel werden wir diesen Ansatz anhand des Poisson-Problems exemplarisch durchführen.

**Lemma und Definition 2.1** (Laplace Problem als System 1. Ordnung – Gemischte Formulierung, Sattelpunktformulierung)**.**

*Sei $u$ Lösung des Laplace Problems* (1.1) *mit homogenen Dirichletrandwerten. Ist $u \in C^2(\Omega)$, so ist das Laplace-Problem äquivalent zu*

$$-\nabla \cdot \sigma = f \qquad \text{in } \Omega, \tag{2.1a}$$
$$\sigma = \nabla u \qquad \text{in } \Omega, \tag{2.1b}$$
$$u = 0 \qquad \text{auf } \partial\Omega. \tag{2.1c}$$

*Dabei haben wir den Fluss $\sigma$ als neue vektorwertige Variable eingeführt.*

Im nächsten Schritt wollen wir für das resultierende System erster Ordnung eine schwache Formulierung angeben.

**Definition 2.2** (Schwache Formulierung von (2.1))**.**
Ein Paar $(u, \sigma) \in H_0^1(\Omega) \times [H^1(\Omega)]^n$ heißt schwache Lösung von (2.1), falls für alle Teilmengen $K \subset \Omega$, $K$ offen und beschränkt mit Lipschitzrand, gilt:

$$\int_K \sigma \cdot \nabla\varphi = \int_K f\varphi + \int_{\partial K} \sigma \cdot n_K \varphi \qquad \forall \varphi \in H_0^1(\Omega), \tag{2.2a}$$
$$\int_K \sigma \cdot \psi = -\int_K u \nabla \cdot \psi + \int_{\partial K} u \cdot n_K \psi \qquad \forall \psi \in [H^1(\Omega)]^n. \tag{2.2b}$$

Dabei ist $n_K$ die äußere Normale an $\partial K$.

**Bemerkung.**

1) Man erhält die schwache Formulierung (2.2), indem man die erste Gleichung in (2.1) mit $\varphi$ und die zweite Gleichung mit $\psi$ multipliziert, dann beide Gleichungen über $K$ integriert und partiell integriert.

2) Man benötigt $u \in H_0^1(\Omega), \sigma \in [H^1(\Omega)]^n$ nur, damit die Randterme wohldefiniert sind.

3) Ist $u \in H^2(\Omega) \cap H_0^1(\Omega)$, eine schwache Lösung von (1.1), so ist $u$ auch eine schwache Lösung von (2.2) und umgekehrt.

## 2.1 Gitter in $n$D und LDG-Verfahren

**Definition 2.3** (Referenzelement in $n$D)**.**

1) Eine Menge $\hat{e}^0 \subset \mathbb{R}^n$ heißt *Referenzelement* in $n$D mit den Subentitätenmengen

$$\mathcal{E}^c(\hat{e}^0) := \{\hat{e}_0^c, \ldots, \hat{e}_{k_c}^c\},\ i \leq c \leq n,$$

falls gilt:

a) $\mathcal{H}_{n-c}(\hat{e}_j^c) \neq 0, \mathcal{H}_{n-c+1}(\hat{e}_j^c) = 0,\ \forall j = 0, \ldots, k_c,\ \forall 0 \leq c \leq n,$

b) $\hat{e}_j^c = \text{conv}(U^{c+1}(\hat{e}^0))$,
$U^{c+1}(\hat{e}^0) \subset \mathcal{E}^{c+1}(\hat{e}^0)\ \forall j = 0, \ldots, k_c,\ \forall 0 \leq c \leq n.$

2) $\forall d \leq n$ seien $\hat{M}^d$ Mengen von Referenzelementen in $\mathbb{R}^d$ mit folgenden Eigenschaften:

$$\forall \hat{e} \in \hat{M}^n\ \forall \hat{e}^c \in \mathcal{E}^c(\hat{e}) : \exists \hat{e} \in \hat{M}^{n-c} \text{ und eine bijektive Abbildung } \hat{F}(\hat{e}') = \hat{e}^c.$$

**Beispiel 2.4** (Referenzelemente)**.**
Das einzige Referenzelement in 1D ist das Einheitsintervall $[0, 1]$. In 2D gibt es zwei klassische Referenzelemente: Schließlich noch die gebräuchlichsten Referenzelemente in 3D:

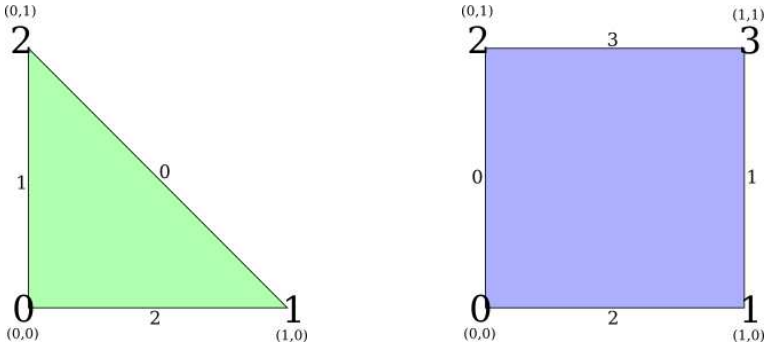



Abbildung 2.1: Dreieck(links), Quadrat (rechts)

*Die Abbildungen stammen von der offiziellen* DUNE*-Homepage (s.o.).*

**Definition 2.5** (Hierarchische Gitter in $n$D)**.**
Sei $\Omega \subset \mathbb{R}^n$ ein Gebiet mit stückweise glattem Rand $\partial\Omega$. Ein *hierarchisches Gitter* $\mathcal{T}_h$ mit $L + 1$

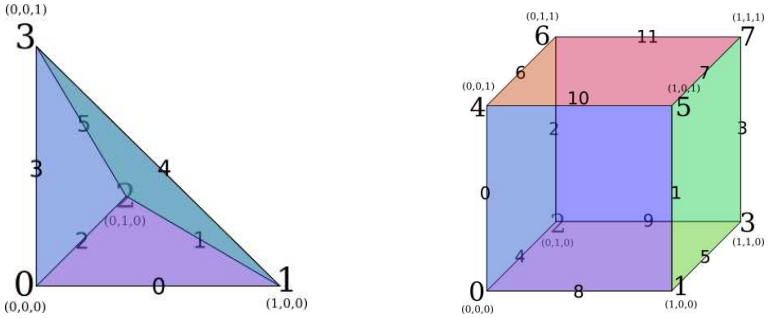



Abbildung 2.2: Tetraeder (links), Hexaeder (rechts)

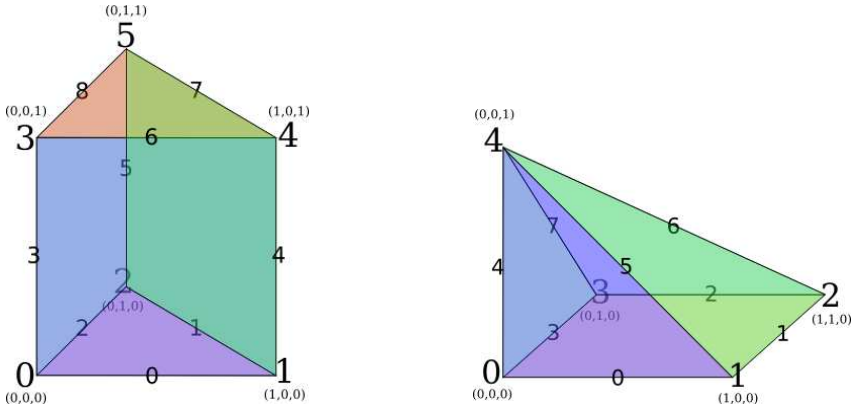



Abbildung 2.3: Prisma (links), Pyramide (rechts)

Leveln besteht aus $L+1$ Gittern

$$\mathcal{T}_h := \{\mathcal{T}_0, \mathcal{T}_1, \ldots, \mathcal{T}_L\}.$$

Jedes Gitter besteht aus Entitätenmengen ${\mathcal{E}_l}^c$ der Kodimension $c = 0, \ldots, n$,

$$\mathcal{T}_l := \{{\mathcal{E}_l}^0, {\mathcal{E}_l}^1, \ldots, {\mathcal{E}_l}^n\},$$
$${\mathcal{E}_l}^0 := \{{e_{l,0}}^c, {e_{l,1}}^c, \ldots, {e_{l,N(l,c)-1}}^c\}.$$

Wir definieren die *Levelindexmenge* für Entitäten der Kodimension $c$ als

$$I_l{}^c := \{0, 1, \ldots, N(l,c) - 1\}.$$

$\mathcal{T}_h$ heißt hierarchisches Gitter von $\Omega$, falls gilt:

1) die Entitäten der Kodimension 0 auf Level 0 definieren eine Partition von $\Omega$:

$$\bigcup_{i \in {I_0}^0} {\overline{e}_{0,i}}^0 = \overline{\Omega}, \qquad {\mathring{e}_{0,i}}^0 \cap {\mathring{e}_{0,j}}^0 = \emptyset \ (i \neq j).$$

2) Entitäten der Kodimension 0 bilden einen Baum. Wir fordern:

$$\forall l > 0, i \in {I_l}^0 : \exists! j \in {I_{l-1}}^0 : {e_{l,i}}^0 \subset {e_{l-1,j}}^0.$$

Dieses ${e_{l-1,j}}^0$ heißt Vater von ${e_{l,i}}^0$. Die Menge von Kindern einer Entität ${e_{k,i}}^0$ definieren wir durch

$$C({e_{k,i}}^0) := \{{e_{l,j}}^0 \mid {e_{l,j}}^0 \subset {e_{k,i}}^0 \ \forall l \leq L\}.$$

3) Für jede Entität ${e_{l,i}}^0$ der Kodimension 0 existiert ein Referenzelement $\hat{e}^0 \in \overline{M}^n$ und eine glatte bijektive Referenzabbildung:

$$F_{l,i} : \hat{e}^0 \to {e_{l,i}}^0$$

mit der Eigenschaft

$$F_{l,i}({\hat{e}_j}^c) = e^c_{l,{\mu_{l,i}}^c(j)} \qquad \forall 0 \le j \le k_l,\ 1 \le c \le n,$$

mit ${\mu_{l,i}}^c(j) : \{0, \dots, k\} \to I_l$.
Wir definieren:

$$I^c_{l,i} := {\mu_{l,i}}^c(\{0, \dots, k_c\}).$$

4) Es gilt die Eindeutigkeit: $\forall i, j, c, l : e^c_{l,i} = e^c_{l,j} \Rightarrow i = j$. Für $e^0_{l,i}$ definieren wir die Menge der "Intersections" mit anderen Elementen durch:

$$\mathcal{I}(e^0_{l,i}) := \left\{ \overline{e^0_{l,i}} \cap \overline{e^0_{l,j}} \mid \mathcal{H}_{n-1}(\overline{e^0_{l,i}} \cap \overline{e^0_{l,j}}) \neq 0,\ i \neq j \right\}.$$

**Beispiel 2.6.**
Hier sehen wir die Verfeinerung eines einfachen Makrogitters:

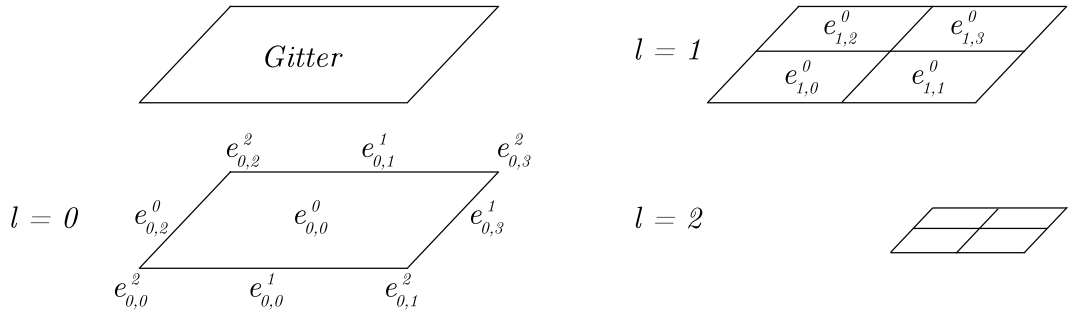



Abbildung 2.4: Ein zweimal verfeinertes Gitter

In einem Entitätenbaum kann man die Verwandtschaftsbeziehungen in einem verfeinerten Gitter festhalten:

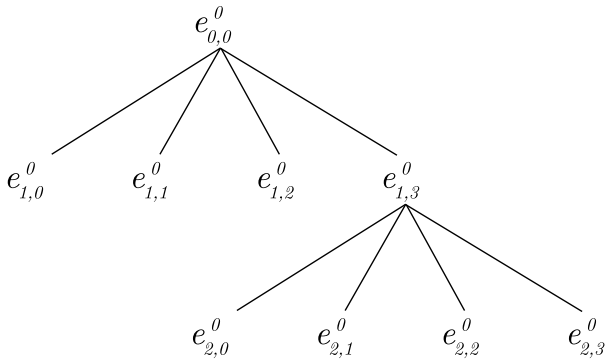



Abbildung 2.5: zugehöriger Entitätenbaum

**Definition 2.7** (Blattgitter von $\mathcal{T}_h$)**.**
Sei $\mathcal{T}_h$ ein hierarchisches Gitter von $\Omega \subset \mathbb{R}^n$. Dann ist das *Blattgitter* (engl. *leaf grid*) definiert durch:

$$\mathcal{T}_{\text{leaf}} := \{\mathcal{E}^0_{\text{leaf}}, \dots, \mathcal{E}^n_{\text{leaf}}\}$$

mit den Eigenschaften:

1) $\mathcal{E}^0_{\text{leaf}} = \{e^0_{l,j} \in \mathcal{E}^0_l \mid C(e^0_{l,j}) = \emptyset,\ l = 0, \dots, L\}$,
2) $\mathcal{E}^c_{\text{leaf}} \subset \bigcup_{l=0}^{L} \mathcal{E}^c_l$, sodass gilt: $\left[e^c_l \in \mathcal{E}^c_{\text{leaf}} \Rightarrow \exists e^0_l \in \mathcal{E}^0_{\text{leaf}} :\ e^0_l \in C(e^0_l)\right]$,
3) $\mathcal{T}_{\text{leaf}}$ ist ein hierarchisches Gitter im Sinne von Definition 2.5 mit $L = 0$. Die Indexmenge zu $\mathcal{E}^c_{\text{leaf}}$ bezeichnen wir mit $I^c_{\text{leaf}}\ \forall 0 \le c \le n$.

**Beispiel 2.8** (Entitätenbaum aus Beispiel 2.6)**.**
Es gilt: $\mathcal{E}^0_{\text{leaf}} = \left\{e^0_{1,0}, e^0_{1,1}, e^0_{1,2}, e^0_{2,0}, e^0_{2,1}, e^0_{2,2}, e^0_{2,3}\right\}$. Insbesondere ist $\mathcal{T}_{\text{leaf}}$ eine Partition von $\Omega$.

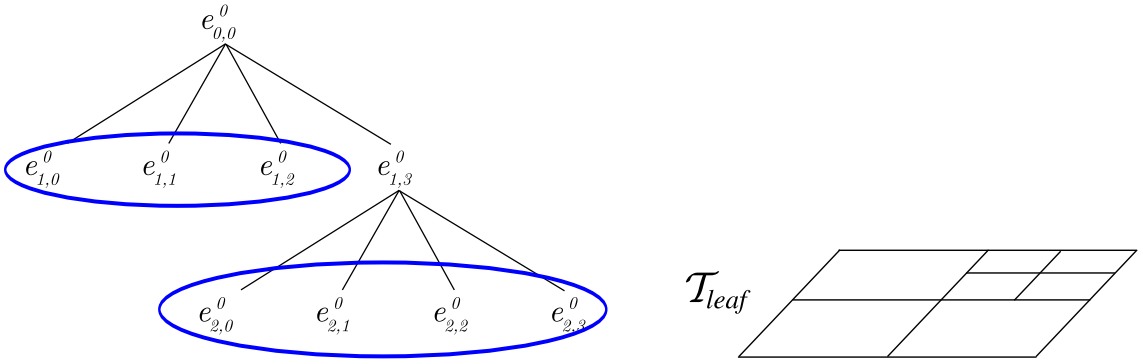



Abbildung 2.6: Entitätenbaum und Darstellung von $\mathcal{T}_{\text{leaf}}$ aus 2.6

**Definition 2.9** (Lokale Sobolevräume)**.**

1) Sei $\Omega \in \mathbb{R}^n$ und $\mathcal{T}_h$ ein hierarchisches Gitter auf $\Omega$. Wir definieren die *lokalen Sobolevräume* $H^m(\mathcal{T}_h)$ durch
$$H^m(\mathcal{T}_h) := \left\{v \in L^2(\Omega) \mid v|_{e^0} \in H^m(e^0) \forall e^0 \in \mathcal{E}_{\text{leaf}}\right\}.$$
Das heißt $H^m(\mathcal{T}_h)$ ist der Raum der stückweise $H^m$-Funktionen auf dem Blattgitter $\mathcal{T}_{\text{leaf}}$.
2) Sei $\Gamma := \bigcup_{e^1 \in \mathcal{E}^1_{\text{leaf}}} \overline{e^1}$ die Vereinigung der Kodim 1-Entitäten von $\mathcal{T}_{\text{leaf}}$. Wir definieren den Raum der Spuren von Funktionen in $H^1(\mathcal{T}_h)$ als
$$H_S(\Gamma) := \prod_{e^0 \in \mathcal{E}^0_{\text{leaf}}} L^2(\partial e^0).$$
Funktionen in $H_S(\Gamma)$ sind zweiwertig. Es gilt $\forall \varphi \in H_S(\Gamma) : \exists \overline{\varphi} \in H^1(\mathcal{T}_h)$ : Ist $x \in \Gamma \setminus \partial\Omega, x \in \overline{e^0_j}, \overline{e^0_i} \in \mathcal{E}^0_{\text{leaf}}, j < i$, so gilt:
$$\begin{aligned}
\varphi(x) &= \left\{\varphi^-(x), \varphi^+(x)\right\}, \\
\varphi^-(x) &:= \lim_{y \to x, y \in e^0_j} \overline{\varphi}(y), \\
\varphi^+(x) &:= \lim_{y \to x, y \in e^0_i} \overline{\varphi}(y) \quad \text{(s. Skizze unten)}.
\end{aligned}$$

Eine Funktion $\varphi \in H_S(\Gamma)$ liegt in $L^2(\Gamma)$, wenn für fast alle $x \in \Gamma$ gilt: $\varphi^-(x) = \varphi^+(x)$, d.h. $\varphi$ ist einwertig.

3) Für $\varphi \in H_S(\Gamma)$ definieren wir:

$$\begin{aligned} [\varphi](x) &:= \varphi^-(x) - \varphi^+(x) && \forall x \in \Gamma \setminus \partial\Omega, \\ \{\varphi\}(x) &:= \frac{1}{2}\left(\varphi^-(x) + \varphi^+(x)\right) && \forall x \in \Gamma \setminus \partial\Omega, \\ n(x) &:= n^-(x) && \forall x \in \Gamma. \end{aligned}$$

Die Lemmata 1.14 und 1.15 gelten analog für $\varphi \in H_S(\Gamma)$.

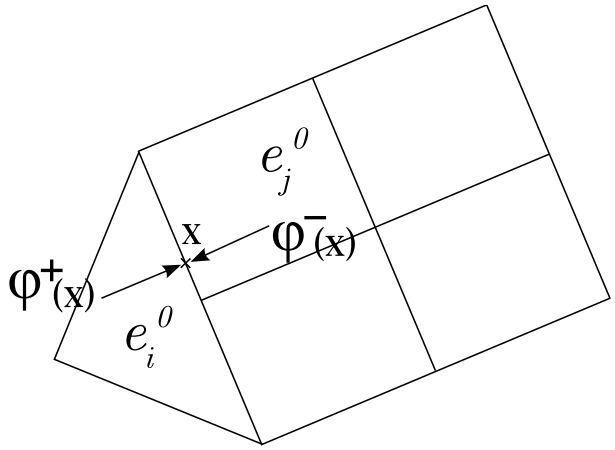



Abbildung 2.7: Skizze zu Definition 2.9: $\varphi^-(x), \varphi^+(x)$

**Definition 2.10** (Diskrete Räume)**.**
Wir definieren $V_h \subset L^2(\Omega), \Sigma_h \subset \left[L^2(\Omega)\right]^n$ auf $\mathcal{T}_h$ durch:

$$\begin{aligned} V_h = V_h^p &:= \left\{v \in L^2(\Omega) \mid v|_{e^0} \in \mathbb{P}_p(e^0) \forall e^0 \in \mathcal{E}^0_{\text{leaf}}\right\} \\ \Sigma_h = \Sigma_h^q &:= \left\{v \in \left[L^2(\Omega)\right]^n \mid v|_{e^0} \in \left[\mathbb{P}_q(e^0)\right]^n \forall e^0 \in \mathcal{E}^0_{\text{leaf}}\right\}. \end{aligned}$$

Dabei seien $p, q \in \mathbb{N}$ die maximalen Polynomgrade.

**Definition 2.11** (Numerische Flüsse)**.**
Der numerische skalare Fluss

$$\hat{u} = (\hat{u}_e)_{e \in \mathcal{E}^0_{\text{leaf}}}$$

und der vektorwertige numerische Fluss

$$\hat{\sigma} = (\hat{\sigma}_e)_{e \in \mathcal{E}^0_{\text{leaf}}}$$

auf $\mathcal{T}_h$ sind Abbildungen der Form:

$$\begin{aligned} \hat{u} : H^1(\mathcal{T}_h) &\longrightarrow H_S(\Gamma), \\ v &\mapsto \hat{u}(v), \\ \hat{\sigma} : H^2(\mathcal{T}_h) \times [H^1(\mathcal{T}_h)]^n &\longrightarrow [H_S(\Gamma)]^n, \\ (v, \psi) &\mapsto \hat{\sigma}(v, \psi). \end{aligned}$$

1) Die Flüsse heißen linear, falls $\hat{u}, \hat{\sigma}$ lineare Abbildungen sind.

2) Die Flüsse heißen *konsistent*, wenn für alle $v \in H^2(\Omega) \cap H_0^1(\Omega)$ gilt:

$$\hat{u}(v) = v|_\Gamma, \ \hat{\sigma}(v, \nabla v) = \nabla v|_\Gamma.$$

3) Die Flüsse heißen *erhaltend*, falls sie einwertig sind, d.h.

$$\hat{u} : H^1(\mathcal{T}_h) \longrightarrow L^2(\Gamma),$$
$$\hat{\sigma} : H^2(\mathcal{T}_h) \times [H^1(\mathcal{T}_h)]^n \longrightarrow [L^2(\Gamma)]^n.$$

**Definition 2.12** (Allgemeine Local-Discontinuous-Galerkin-Verfahren)**.**
Sei $\mathcal{T}_h$ ein hierarchisches Gitter auf $\Omega \subset \mathbb{R}^n$ und seien die numerischen Flüsse $\hat{u}, \hat{\sigma}$ auf $\mathcal{T}_h$ gegeben. Ein Paar $(u_h, \sigma_h) \in V_h^p \times \Sigma_h^q$ heißt Lösung des LDG-Verfahrens der Ordnung $(p, q)$ zu den Flüssen $(\hat{u}, \hat{\sigma})$ auf $\mathcal{T}_h$, falls für alle $e \in \mathcal{E}_{\text{leaf}}^0$ gilt:

$$\int_e \sigma_h \cdot \nabla \varphi_h = \int_e f\varphi_h + \int_{\partial e} \hat{\sigma}_e(u_h, \sigma_h) \cdot n_e \varphi_h, \qquad \forall \varphi_h \in V_h^p, \tag{2.3a}$$

$$\int_e \sigma_h \psi_h = -\int_e u_h \nabla \cdot \psi_h + \int_{\partial e} \hat{u}_e(u_h) n_e \cdot \psi_h, \qquad \forall \psi_h \in \Sigma_h^q. \tag{2.3b}$$

**Notation.**
Wir bezeichnen das Verfahren mit $LDG^{p,q}(\hat{u}, \hat{\sigma})$. Ist $p = q$, so schreiben wir $LDG^p(\hat{u}, \hat{\sigma})$. Ist $p > 0$ beliebig, so schreiben wir $LDG(\hat{u}, \hat{\sigma})$.

**Lemma 2.13** (Primale Formulierung)**.**
*Sei $(u_h, \sigma_h) \in V_h \times \Sigma_h$ Lösung von $LDG^{p,q}(\hat{u}, \hat{\sigma})$. Wir definieren die Bilinearform*

$$B_{(\hat{u},\hat{\sigma})} : H^2(\mathcal{T}_h) \times H^2(\mathcal{T}_h) \longrightarrow \mathbb{R}$$

*durch:*

$$\begin{aligned} B_{(\hat{u},\hat{\sigma})}(u, \varphi) := &\int_\Omega \nabla_h u \cdot \nabla_h \varphi \\ &+ \int_{\Gamma \setminus \partial\Omega} ([\hat{u}(u) - u]\{\nabla_h \varphi \cdot n\} - \{\hat{\sigma}(u, s(u)) \cdot n\}[\varphi]) \\ &+ \int_{\Gamma \setminus \partial\Omega} (\{\hat{u}(u) - u\}[\nabla_h \varphi \cdot n] - [\hat{\sigma}(u, s(u)) \cdot n]\{\varphi\}) \\ &+ \int_{\partial\Omega} (\hat{u}(u) - u)\nabla_h \varphi \cdot n - \hat{\sigma}(u, s(u)) \cdot n\varphi, \end{aligned}$$

*mit $s(u) := \nabla_h u - r([\hat{u} - u]) - l(\{\hat{u} - u\})$. Dabei sind die Lift-Operatoren $r, l : L^2(\Gamma) \to [H^2(\mathcal{T}_h)]^n$ definiert durch:*

$$\int_\Omega r(\varphi) \cdot \psi = -\int_{\Gamma \setminus \partial\Omega} \varphi\{\psi \cdot n\} - \int_{\partial\Omega} \varphi\psi \cdot n, \qquad \forall \psi \in [H^1(\mathcal{T}_h)]^n,$$

$$\int_\Omega l(\varphi) \cdot \psi = -\int_{\Gamma \setminus \partial\Omega} \varphi[\psi \cdot n], \qquad \forall \psi \in [H^1(\mathcal{T}_h)]^n.$$

*Dann gilt für $(u_h, \sigma_h)$:*

*1) $u_h$ löst:*

$$B_{(\hat{u},\hat{\sigma})}(u_h, \varphi_h) = \int_\Omega f\varphi_h, \qquad \forall \varphi_h \in V_h, \tag{2.4}$$

*2) $\sigma_h$ ist gegeben durch:*

$$\sigma_h = s(u_h). \tag{2.5}$$

*Beweis:* Analog zum Beweis von Lemma 1.15 zeigt man, dass für alle $\varphi \in H_S(\Gamma), \psi \in [H_S(\Gamma)]^n$ gilt

$$\sum_{e \in \mathcal{E}^0_{\text{leaf}}} \int_{\partial e} \varphi_e \cdot \psi_e \cdot \eta_e = \int_{\Gamma \setminus \partial\Omega} [\varphi]\{\psi \cdot n\} + \int_{\Gamma\setminus\partial\Omega} \{\varphi\}[\psi \cdot n] + \int_{\partial\Omega} \varphi\psi n.$$

Summieren wir (2.3) über alle $e \in \mathcal{E}^0_{\text{leaf}}$, so erhalten wir hiermit:

$$\begin{aligned} \int_\Omega \sigma_h \cdot \nabla_h \varphi_h &= \int_\Omega f\varphi_h + \int_{\Gamma\setminus\partial\Omega} (\{\hat{\sigma} \cdot n\}[\varphi_h] + [\hat{\sigma} \cdot n]\{\varphi_h\}) \\ &\quad + \int_{\partial\Omega} \hat{\sigma} n \varphi_h \qquad (2.6a) \\ \int_\Omega \sigma_h \cdot \psi_h &= -\int_\Omega u_h \nabla_h \cdot \psi_h + \int_{\Gamma\setminus\partial\Omega} ([\hat{u}]\{\psi_h \cdot n\} + \{\hat{u}\}[\psi_h \cdot n]) \\ &\quad + \int_\Omega \hat{u}\psi_h \cdot n \qquad (2.6b) \end{aligned}$$

Wir wollen zeigen: $\sigma_h = s(u_h)$. Für $\varphi_h, \psi_h$ gilt analog zu Lemma 1.15 die Formel für partielle Integration:

$$\begin{aligned} -\int_\Omega \nabla_h \cdot \psi_h \varphi_h &= \int_\Omega \psi_h \cdot \nabla_h \varphi_h - \int_{\Gamma\setminus\partial\Omega} (\{\psi_h \cdot n\}[\varphi_h] + [\psi_h \cdot n]\{\varphi_h\}) \\ &\quad - \int_{\partial\Omega} \psi_h \cdot n \varphi_h. \qquad (2.7) \end{aligned}$$

Wähle $\varphi_h = u_h$, dann folgt aus (2.6b):

$$\begin{aligned} \int_\Omega \sigma_h \cdot \psi_h &= \int_\Omega \nabla_h u_h \cdot \psi_h \\ &\quad + \int_{\Gamma\setminus\partial\Omega} ([\hat{u} - u_h]\{\psi_h \cdot n\} + \{\hat{u} - u_h\}[\psi_h \cdot n]) \\ &\quad + \int_{\partial\Omega} (\hat{u} - u_h)\psi_h \cdot n. \qquad (2.8) \end{aligned}$$

Mit der Definition der Liftoperatoren $r, l$ folgt dann $\sigma_h = s(u_h)$.
Mit (2.6b) und (2.6a) folgt weiter für $\psi_h = \nabla_h \varphi_h$:

$$B_{(\hat{u},\hat{\sigma})}(u_h, \varphi_h) = \int_\Omega f\varphi_h \qquad \forall \varphi_h \in V_h.$$

□

**Lemma 2.14** (Konsistenz und Galerkinorthogonalität)**.**
*Sei $u \in H^2(\Omega) \cap H_0^1(\Omega)$ eine schwache Lösung von (1.1). Sind die Flüsse $\hat{u}, \hat{\sigma}$ konsistent im Sinne von Definition 2.11, so ist LDG$(\hat{u}, \hat{\sigma})$ konsistent, d.h. es gilt $\forall \varphi \in H^2(\mathcal{T}_h)$:*

$$B_{(\hat{u},\hat{\sigma})}(u, \varphi) = \int_\Omega f\varphi.$$

*Außerdem gilt die Galerkin-Orthogonalität:*

$$B_{(\hat{u},\hat{\sigma})}(u - u_h, \varphi_h) = 0, \quad \forall \varphi_h \in V_h.$$

*Dabei bezeichne $u_h$ die Lösung von LDG$(\hat{u}, \hat{\sigma})$.*
*Beweis:* Die Galerkin-Orthogonalität folgt direkt aus der Konsistenz. Zur Konsistenz: Sind $\hat{u}, \hat{\sigma}$ konsistent, so folgt:

$$\begin{aligned} \hat{u}(u) &= u && \text{auf } \Gamma, \\ [\hat{u}] &= 0 && \text{auf } \Gamma, \\ \{\hat{u}\} &= u && \text{auf } \Gamma. \end{aligned}$$

Damit folgt: $s(u) = \nabla u$. Da $\hat{\sigma}$ konsistent ist, folgt weiter:

$$\hat{\sigma}(u, s(u)) = \hat{\sigma}(u, \nabla u) = \nabla u \qquad \text{auf } \Gamma$$

Damit reduziert sich $B_{(\hat{u},\hat{\sigma})}(u, \varphi)\ \forall \varphi \in H^2(\mathcal{T}_h)$ zu:

$$\begin{aligned} B_{(\hat{u},\hat{\sigma})}(u, \varphi) &= \int_\Omega \nabla u \nabla_h \varphi - \int_{\Gamma \setminus \partial\Omega} \nabla u \cdot n[\varphi] - \int_{\partial\Omega} \nabla u \cdot n\varphi \\ &\overset{\text{p.I.}}{=} -\int_\Omega \Delta u \varphi \\ &= \int_\Omega f\varphi, \end{aligned}$$

da nach Annahme gilt: $u \in H^2 \cap H_0^1$ löst $-\Delta u = f$. □

**Definition 2.15** (Adjungiert konsistent)**.**
$B_{(\hat{u},\hat{\sigma})}$ heißt *adjungiert konsistent*, falls gilt: Ist $v \in H^2(\Omega) \cap H_0^1(\Omega)$ eine Lösung von:

$$\begin{aligned} -\Delta v &= g && \text{in } \Omega, \\ v &= 0 && \text{auf } \partial\Omega, \end{aligned}$$

dann gilt: $\forall \varphi \in H^2(\mathcal{T}_h)$

$$B_{(\hat{u},\hat{\sigma})}(\varphi, v) = \int_\Omega g\varphi.$$

**Lemma 2.16** ($\hat{u}, \hat{\sigma}$ erhaltend $\Rightarrow B_{(\hat{u},\hat{\sigma})}$ adjungiert konsistent)**.**
*Seien $\hat{u}, \hat{\sigma}$ erhaltend im Sinne von Definition 2.11 und gelte*

$$\hat{u}|_{\partial\Omega} = 0,$$

*dann ist $B_{(\hat{u},\hat{\sigma})}$ adjungiert konsistent.*

*Beweis:* Sind $\hat{u}, \hat{\sigma}$ erhaltend, so folgt:

$$\begin{aligned} [\hat{u}] &= 0, & \{\hat{u}\} &= \hat{u}, \\ [\hat{\sigma} \cdot n] &= 0, & \{\hat{\sigma} \cdot n\} &= \hat{\sigma} \cdot n. \end{aligned}$$

Für $v \in H^2(\Omega) \cap H_0^1(\Omega)$ gilt:

$$\begin{aligned}[v] = [\nabla v \cdot n] &= 0,\\ \{v\} &= v,\\ \{\nabla v \cdot n\} &= \nabla v \cdot n.\end{aligned}$$

Mit $v|_{\partial\Omega} = \hat{u}|_{\partial\Omega} = 0$ folgt dann:

$$\begin{aligned}B_{(\hat{u},\hat{\sigma})}(\varphi, v) &= \int_\Omega \nabla_h \varphi \cdot \nabla v - \int_{\Gamma\setminus\partial\Omega} [\varphi] \nabla v \cdot n - \int_{\partial\Omega} \varphi \nabla v \cdot n\\ &\overset{\text{p.I.}}{=} -\int_\Omega \varphi \cdot \Delta v = \int_\Omega \varphi g\end{aligned}$$

□

**Beispiel 2.17** (Wahl der Flüsse und resultierende DG-Verfahren)**.**

Wir definieren Abbildungen

$$\alpha_j, \alpha_r : L^2(\Gamma) \longrightarrow [L^2(\Gamma)]^n$$

für gegebene $(\beta_e)_{e\in\mathcal{E}^1_{\text{leaf}}}$, $\beta_e \in \mathbb{R}^+$ durch:

1) $\alpha_j(\varphi) = \mu\varphi \cdot n$ auf $\Gamma$,
   $\mu : \Gamma \to \mathbb{R}$, $\mu|_e = \frac{\beta_e}{h_e}$ $\forall e \in \mathcal{E}^1_{\text{leaf}}$
   mit $h(e) := \text{diam}(e), e \in \mathcal{E}^1_{\text{leaf}}$.

2) $\alpha_r(\varphi)|_e := -\beta_e\{r_e(\varphi)\}$ $\forall e \in \mathcal{E}^1_{\text{leaf}}$
   mit dem Liftoperator $r_e : L^2(e) \to \Sigma_h$ definiert durch

$$\int_\Omega r_e(\varphi) \cdot \psi = -\int_e \varphi \cdot n\{\psi\}, \qquad \forall \psi \in \Sigma_h, \forall e \in \mathcal{E}^1_{\text{leaf}}.$$

Wir definieren die Verfahren $LDG_1 - LDG_9$:

| | | $\hat{u}_h$ | | $\hat{\sigma}_h$ | |
|---|---|---|---|---|---|
| Nr | Methode | $\Gamma \setminus \partial\Omega$ | $\partial\Omega$ | $\Gamma \setminus \partial\Omega$ | $\partial\Omega$ |
| 1 | Bassi-Rebay | $\{u_h\}$ | 0 | $\{\sigma_h \cdot n\}$ | $\sigma_h \cdot n$ |
| 2 | Brezzi et.al. | $\{u_h\}$ | 0 | $\{\sigma_h \cdot n\} - \alpha_r([u_h]) \cdot n$ | $\sigma_h \cdot n - \alpha_r(u_h) \cdot n$ |
| 3 | LDG | $\{u_h\} - \beta[u_h]$ | 0 | $\{\sigma_h \cdot n\} + \beta[\sigma_h \cdot n] - \alpha_j([u_h]) \cdot n$ | $\sigma_h \cdot n + \beta\sigma_h \cdot n - \alpha_j(u_h) \cdot n$ |
| 4 | Interior Penalty | $\{u_h\}$ | 0 | $\{\nabla_h u_h\} - \alpha_j([u_h]) \cdot n$ | $\nabla u_h \cdot n - \alpha_j(u_h) \cdot n$ |
| 5 | Bassi et.al. | $\{u_h\}$ | 0 | $\{\nabla_h u_h\} - \alpha_r([u_h]) \cdot n$ | $\nabla u_h \cdot n - \alpha_r(u_h) \cdot n$ |
| 6 | Baumann-Oden | $\{u_h\} \pm [u_h]$ | 0 | $\{\nabla_h u_h\}$ | $\nabla u_h \cdot n$ |
| 7 | NIPG | $\{u_h\} \pm [u_h]$ | 0 | $\{\nabla_h u_h\} - \alpha_j([u_h]) \cdot n$ | $\nabla u_h \cdot n - \alpha_j(u_h) \cdot n$ |
| 8 | Babuska-Zamel | $u_h^{\pm}$ | 0 | $-\alpha_j([u_h]) \cdot n$ | $-\alpha_j(u_h) \cdot n$ |
| 9 | Brezzi et.al. II | $u_h^{\pm}$ | 0 | $-\alpha_r([u_h]) \cdot n$ | $-\alpha_j(u_h) \cdot n$ |

**Lemma 2.18** (Primale Formulierung für die Methoden aus Bsp. 2.17)**.**
*Seien $LDG_1 - LDG_9$ die LDG-Verfahren aus Bsp. 2.17. Wir definieren für gegebene $(\beta_e)_{e\in\mathcal{E}^1_{leaf}}, \beta_e \in \mathbb{R}^+$:*

*1)*

$$\alpha^j(u_h, \varphi_h) := \int_{\Gamma\setminus\partial\Omega} \mu[u_h][\varphi_h] + \int_{\partial\Omega} \mu u_h \varphi_h,$$

*mit $\mu : \Gamma \longrightarrow \mathbb{R}$, $\mu|_e = \frac{\beta_e}{h_e} \forall e \in \mathcal{E}^1_{leaf}$.*

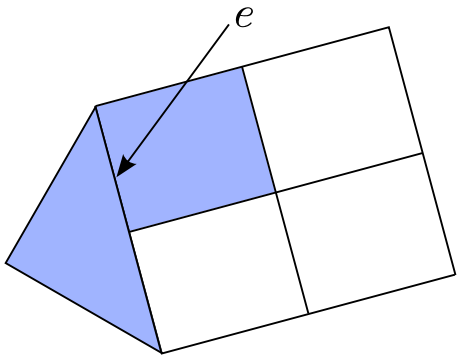



Abbildung 2.8: supp($r_e(\varphi)$)

*2)*

$$\begin{aligned}\alpha^r(u_h,\varphi_h) &:= \int_{\Gamma\setminus\partial\Omega} \alpha_r([u_h])\cdot[\varphi_h]\cdot n + \int_{\partial\Omega}\alpha_r(u_h)\cdot\varphi_h\cdot n\\ &= \sum_{e\in\mathcal{E}^1_{leaf}, e\not\subset\partial\partial\Omega} \beta_e \int_\Omega r_e([u_h])\cdot r_e([\varphi_h])\\ &\quad + \sum_{e\in\mathcal{E}^1_{leaf}, e\subset\partial\partial\Omega} \beta_e \int_\Omega r_e(u_h)\cdot r_e(\varphi_h).\end{aligned}$$

*3)*

$$\begin{aligned}\langle u_h,\varphi_h\rangle_\Omega &:= \int_\Omega u_h\cdot\varphi_h,\\ \langle u_h,\varphi_h\rangle_\Gamma &:= \int_\Gamma u_h\varphi_h.\end{aligned}$$

*Weiter setzen wir:*

$$\begin{aligned}B_\pm(w,v) :=& \langle\nabla_h w,\nabla_h v\rangle_\Omega - \langle\{\nabla_h w\cdot n\},[v]\rangle_{\Gamma\setminus\partial\Omega}\\ &\pm\langle[w],\{\nabla_h v\cdot n\}\rangle_{\Gamma\setminus\partial\Omega} - \langle\nabla_h w\cdot n, v\rangle_{\partial\Omega} \pm \langle w,\nabla_h v\cdot n\rangle_{\partial\Omega}.\end{aligned}$$

*Dann sind die primalen Formulierungen von $LDG_1 - LDG_9$ gegeben durch folgende Bilinearformen:*

| *Nr* | $B(\hat{u},\hat{\sigma})$ |
|---|---|
| *1* | $B_-(w,v) + \langle r([w]), r([v])\rangle_\Omega$ |
| *2* | $B_-(w,v) + \langle r([w]), r([v])\rangle_\Omega + \alpha^r(w,v)$ |
| *3* | $B_+(w,v) + \langle\beta[w],[\nabla_h v\cdot n]\rangle_{\Gamma\setminus\partial\Omega} + \langle\beta[v],[\nabla_h w\cdot n]\rangle_{\Gamma\setminus\partial\Omega}$ $+\langle r([w]) + l(\beta[w]), r([v]) + l(\beta[v])\rangle_\Omega + \alpha^j(w,v)$ |
| *4* | $B_-(w,v) + \alpha^j(w,v)$ |
| *5* | $B_-(w,v) + \alpha^r(w,v)$ |
| *6* | $B_+(w,v)$ |
| *7* | $B_+(w,v) + \alpha^j(w,v)$ |
| *8* | $\langle\nabla_h w,\nabla_h v\rangle_\Omega + \alpha^j(w,v)$ |
| *9* | $\langle\nabla_h w,\nabla_h v\rangle_\Omega + \alpha^r(w,v)$ |

*Beweis:* ohne Beweis, zum Teil Übungsaufgabe 1, Blatt 3. □

## 2.2 Stabilität und Fehlerabschätzungen

**Definition 2.19** (Beschränktheit und Stabilität).
Auf $V(h) := V_h + H^2(\Omega) \cap H_0^1(\Omega) \subset H^2(\mathcal{T}_h)$ definieren wir die gitterabhängige Norm:

$$|\!|\!| v |\!|\!| = \left( \sum_{e \in \mathcal{E}^0_{\text{leaf}}} |v|^2_{1,e} + h_e |v|^2_{2,e} + |v|^2_* \right)^{\frac{1}{2}},$$

mit $|v|^2_* = \sum_{e \in \mathcal{E}^1_{\text{leaf}}} \|r_e([v])\|^2_{0,\Omega}$.

a) Die Bilinearform $B_{(\hat{u},\hat{\sigma})}$ heißt beschränkt, falls gilt:

$$B_{(\hat{u},\hat{\sigma})}(w,v) \leq C_b |\!|\!| w |\!|\!| \, |\!|\!| v |\!|\!| \qquad \forall w, v \in V(h).$$

b) Die Bilinearform erfüllt die Stabilitätsbedingung, falls gilt:

$$B_{(\hat{u},\hat{\sigma})}(v,v) \geq C_s |\!|\!| v |\!|\!|^2 \qquad \forall v \in V_h.$$

**Lemma 2.20** (Beschränktheit für $LDG_1 - LDG_9$ aus Beispiel 2.17).
*Die Bilinearformen $B_{(\hat{u},\hat{\sigma})}$ zu den Verfahren $LDG_1 - LDG_9$ aus Bsp. 2.17 sind beschränkt.*

*Beweis:* Wir verwenden Lemma 2.18 und zeigen, dass alle vorkommenden Summanden in den Bilinearformen beschränkt sind:

1) $\langle \nabla_h w, \nabla_h v \rangle_\Omega$,
2) $\langle \{\nabla_h w \cdot n\}, [v] \rangle_{\Gamma \setminus \partial\Omega} + \langle \nabla_h w \cdot n, v \rangle_{\partial\Omega}$,
3) $\langle \{\nabla_h v \cdot n\}, [w] \rangle_{\Gamma \setminus \partial\Omega} + \langle \nabla_h v \cdot n, w \rangle_{\partial\Omega}$,
4) $\langle r([w]), r([v]) \rangle_\Omega$,
5) $\alpha^r(w,v)$,
6) $\alpha^j(w,v)$,
7) $\langle \beta[w], [\nabla_h v \cdot n] \rangle_{\Gamma \setminus \partial\Omega}$,
8) $\langle \beta[v], [\nabla_h w \cdot n] \rangle_{\Gamma \setminus \partial\Omega}$,
9) $\langle r([w]) + l(\beta[w]), r([v]) + l(\beta[v]) \rangle_\Omega$.

zu 1) Es ist

$$\begin{aligned} \langle \nabla_h w, \nabla_h v \rangle &= \sum_{e \in \mathcal{E}^0_{\text{leaf}}} \langle \nabla w, \nabla v \rangle \\ &\leq \sum_{e \in \mathcal{E}^0_{\text{leaf}}} |w|_{1,e} |v|_{1,e} \\ &\leq |w|_{1,h} |v|_{1,h}, \end{aligned}$$

mit $|w|^2_{1,h} := \sum_{e \in \mathcal{E}^0_{\text{leaf}}} |w|^2_{1,e}$. Mit $|w|_{1,h} \leq |\!|\!| w |\!|\!|$ folgt die Beschränktheit.

zu 2) Mit Skalierungsargumenten zeigt man, dass für alle $w \in H^2(e^0)$, $e^0 \in \mathcal{E}^0_{\text{leaf}}$, und $e^1 \subset \partial e^0, e^1 \in \mathcal{E}^1_{\text{leaf}}$, gilt:

$$\|\nabla w \cdot n\|^2_{0,e^1} \leq C \left( h^{-1}_{e^1} |w|^2_{1,e^0} + h_{e^1} |w|^2_{2,e^0} \right).$$

Dabei hängt $C$ nur vom kleinsten Winkel in $e^0$ ab. Damit gilt $\forall v \in L^2(e^1)$:

$$\begin{aligned} \int_{e^1} |\{\nabla w \cdot n\}[v]| &\leq \|h^{\frac{1}{2}}_{e^1} \nabla w \cdot n\|_{0,e_1} \|h_e^{-\frac{1}{2}}[v]\|_{0,e^1} \\ &\leq C \sum_{\substack{e^0 \in \mathcal{E}^0_{\text{leaf}}, \\ \overline{e^1} \subset \partial e^0}} \frac{1}{2} \left( |w|^2_{1,e^0} + h^2_{e^1} |w|_{2,e^0} \right)^{\frac{1}{2}} \cdot h_{e^1}^{-\frac{1}{2}} \|[v]\|_{0,e^1}. \end{aligned}$$

Insgesamt folgt durch Summation über $e \in \mathcal{E}^1_{\text{leaf}}$:

$$\begin{aligned} &\langle \{\nabla_h w \cdot n\}, [v] \rangle_{\Gamma \setminus \partial\Omega} + \langle \nabla_h w \cdot n, v \rangle_{\partial\Omega} \\ &\qquad \leq C |\!|\!| w |\!|\!| \left( \sum_{e \in \mathcal{E}^1_{\text{leaf}}} h_e^{-1} \|[v]\|^2_{0,e} \right)^{\frac{1}{2}}. \end{aligned}$$

Mit der Definition von $r_e$ und Skalierungsargumenten zeigt man schließlich: $\exists C_1, C_2 > 0$, die nur vom kleinsten Winkel von $\mathcal{T}_h$ und dem Polynomgrad $p$ abhängen, so dass :

$$\begin{aligned} C_1 \|r_e(\varphi_h)\|^2_{0,\Omega} &\leq h_e^{-1} \|\varphi_h\|^2_{0,e} \\ &\leq C_2 \cdot \|r_e(\varphi_h)\|^2_{0,\Omega}. \end{aligned}$$

Somit folgt:

$$C_1 |\varphi_h|^2 \leq \sum_{e \in \mathcal{E}^1_{\text{leaf}}} h_e^{-1} \|\varphi_h\|^2_{0,e} \leq C_2 |\varphi_h|^2. \tag{2.9}$$

Damit folgt die Beschränktheit von 2).

zu 3) analog zu 2) mit den Rollen von $v, w$ vertauscht.

zu 4)

$$\begin{aligned} \langle r([w]), r([v]) \rangle_\Omega &\leq \|r([w])\|_{0,\Omega} \|r([v])\|_{0,\Omega}. \\ &\leq C |w|_* |v|_*, \end{aligned}$$

Dabei ist $C$ die maximale Anzahl von Intersections für eine Codim-0-Entität.

zu 5)

$$\begin{aligned} \alpha^r(w,v) &= \sum_{e \in \mathcal{E}^1_{\text{leaf}}} \int_\Omega \beta_e r_e([w]) r_e([v]) \\ &\leq \max_{e \in \mathcal{E}^1_{\text{leaf}}} \beta_e |w|_* |v|_*. \end{aligned}$$

zu 6) analog zu 5) unter Verwendung von (2.8).

zu 7)-9) analoge Argumente wie in 1)-6).

Details findet man in [6]. □

**Lemma 2.21** (Stabilität $LDG_2 - LDG_5, LDG_7 - LDG_9$)**.**
*Die Bilinearformen $B_{(\hat{u},\hat{\sigma})}$ zu den Verfahren $LDG_2 - LDG_5, LDG_7 - LDG_9$ aus Beispiel 2.17 erfüllen die Stabilitätsbedingungen, falls*

$$\beta_{min} := \min_{e \in \mathcal{E}^1_{leaf}} \beta_e$$

*groß genug ist.*
*Beweis:* Alle Verfahren $LDG_2 - LDG_5, LDG_7 - LDG_9$ haben eine Bilinearform, die sich schreiben lässt als

$$B_{(\hat{u},\hat{\sigma})}(v,v) = \|\nabla_h v\|^2_{0,\Omega} + \alpha^{r/j}(v,v) + b(v,v),$$

mit $\alpha^{r/j}(v,v) = \alpha^r(v,v)$ oder $\alpha^j(v,v)$ und $b(v,v)$ gegeben durch den Rest.

Nach Lemma 2.20 wissen wir

$$|b(v,v)| \leq C|||v||| \, |v|_*.$$

Damit folgt unter Verwendung der Äquivalenz (2.9)

$$B_{(\hat{u},\hat{\sigma})}(v,v) \geq |v|^2_{1,h} + C_{r/j}\beta_{\min}|v|^2_* - C|||v||| \, |v|_*,$$

wobei

$$C_{r/j} := \begin{cases} 1, & \text{falls } \alpha^{r/j} = \alpha^r, \\ C_1, & \text{falls } \alpha^{r/j} = \alpha^j, \end{cases}$$

ist.

Da $V_h$ endlich dimensional ist, folgt mit Skalierungsargumenten $\exists C_3, C_4 > 0$, mit

$$C_3|||v|||^2 \leq |v|^2_{1,h} + |v|^2_* \leq C_4|||v|||^2.$$

Damit folgt mit der Youngschen Ungleichung ($ab \leq \varepsilon a^2 + \frac{1}{4\varepsilon}b^2$):

$$\begin{aligned} B_{(\hat{u},\hat{\sigma})} &\geq |v|^2_{1,h} * c_{r/j}\beta_{\min}|v|^2_* - \varepsilon\frac{1}{C_3}(|v|^2_{1,h} + |v|^2_*) - \frac{C^2}{4\varepsilon}|v|^2_* \\ &(1 - \frac{\varepsilon}{C_3})|v|^2_{1,h} + \left(C_{r/j}\beta_{\min} - \frac{\varepsilon}{C_3} - \frac{C^2}{4\varepsilon}\right)|v|^2_* \\ &\overset{\varepsilon = C_3/2}{\geq} \min\left\{\frac{1}{2}, C_{r/j}\beta_{\min} - \frac{C^2}{2C_3}\right\} \cdot (|v|^2_{1,h} + |v|^2_*) \\ &\geq C_3 \min\left\{\frac{1}{2}, C_{r/j}\beta_{\min} - \frac{C^2}{2C_3}\right\}|||v|||^2. \end{aligned}$$

Also folgt, dass alle betrachteten Verfahren stabil sind, falls $\beta_{\min}$ groß genug ist. □

**Bemerkung 2.22.**

1) Scharfe Abschätzungen für $\beta_{\min}$ können durch detaillierte Abschätzungen gewonnen werden.
2) Die Verfahren $LDG_1$ und $LDG_6$ sind nicht stabil im Sinne von 2.19.

**Theorem 2.23** (Fehlerabschätzungen für konsistente und stabile LDG-Verfahren)**.**
*Sei $\Omega \subset \mathbb{R}^n$ ein Gebiet mit polygonalem Rand und $\mathcal{T}_h$ ein hierarchisches Gitter auf $\Omega$, das die "kleinste Winkel" Bedingung erfüllt. Seien $\hat{u}, \hat{\sigma}$ konsistente numerische Flüsse und $B_{(\hat{u},\hat{\sigma})}$ sei beschränkt und erfülle die Stabilitätsbedingung aus Definition (2.19). Sei $(u_h, \sigma_h) \in V_h^p \times \Sigma_h^p$ eine Lösung von $LDG^p(\hat{u}, \hat{\sigma})$, dann gilt:*

$$|||u - u_h||| \leq C \cdot \inf_{v_h \in V_h} |||u - v_h|||.$$

*Dabei sei $u \in H^2(\Omega) \cap H_0^1(\Omega)$ die Lösung von (1.1). Ist $u \in H^{p+1}(\Omega)$, so folgt mit einer Konstanten $C > 0$*

$$\|\!|u - u_h|\!\| \leq C \cdot h^p |u|_{p+1,\Omega}. \tag{2.10}$$

*Dabei ist $h := \max_{e \in \mathcal{E}^0_{leaf}} h_e, h_e := \operatorname{diam}(e)$.*

*Beweis:*

(i) $\|\!|u - u_h|\!\| \leq \|\!|u - v_h|\!\| + \|\!|v_h - u_h|\!\|$,

(ii)

$$\begin{aligned}
\|\!|v_h - u_h|\!\|^2 &\leq \frac{1}{C_s} B_{(\hat{u},\hat{\sigma})}(v_h - u_h, v_h - u_h) \\
&\overset{\text{Kons.}}{=} \frac{1}{C_s} B_{(\hat{u},\hat{\sigma})}(v_h - u, v_h - u_h) \\
&\leq \frac{C_b}{C_s} \|\!|u - v_h|\!\| \, \|\!|v_h - u_h|\!\|.
\end{aligned}$$

$$\begin{aligned}
\Rightarrow \|\!|v_h - u_h|\!\| &\leq \frac{C_b}{C_s} \|\!|u - v_h|\!\|, \\
\Rightarrow \|\!|u - u_h|\!\| &\leq \left(1 + \frac{C_b}{C_s}\right) \inf_{v_h \in V_h} \|\!|u - v_h|\!\| \\
&\leq \left(1 + \frac{C_b}{C_s}\right) C_I \cdot h^p |u|_{p+1,\Omega},
\end{aligned}$$

wenn man eine entsprechende Interpolationsabschätzung benutzt.

□

**Theorem 2.24** (Aubin-Nitsche Trick für adjungiert konsistente Verfahren)**.**
*Es gelten die Voraussetzungen aus Theorem 2.23 und zusätzlich seien $\hat{u}, \hat{\sigma}$ erhaltend, $\hat{u} = 0$ auf $\partial\Omega$. Dann gilt im Falle $u \in H^{p+1}(\Omega)$:*

$$\|u - u_h\|_{0,\Omega} \leq C h^{p+1} |u|_{p+1,\Omega}.$$

*Beweis:* Sei $v \in H^2(\Omega) \cap H_0^1(\Omega)$ Lösung des adjungierten Problems

$$\begin{aligned}
-\Delta v &= u - u_h && \text{in } \Omega, \\
v &= 0 && \text{auf } \partial\Omega.
\end{aligned}$$

Dann folgt aus der adjungierten Konsistenz von $B_{(\hat{u},\hat{\sigma})}$ nach Lemma 2.14:

$$B_{(\hat{u},\hat{\sigma})}(\varphi_h, v) = \langle u - u_h, \varphi_h \rangle \qquad \forall \varphi_h \in V(h). \tag{2.11}$$

Sei $v_I$ eine stückweise lineare Interpolierende von $v$ und setze $\varphi_h = u - u_h$ in (2.10), so folgt:

$$\begin{aligned}
\|u - u_h\|^2_{0,\Omega} &= B_{(\hat{u},\hat{\sigma})}(u - u_h, v) \\
&\overset{\text{Kons.}}{=} B_{(\hat{u},\hat{\sigma})}(u - u_h, v - v_I) \\
&\overset{\text{Stab.}}{\leq} C_b \|\!|u - u_h|\!\| \, \|\!|v - v_I|\!\| \\
&\leq C h |v|_{2,\Omega} \|\!|u - u_h|\!\| \\
&\overset{2.23}{\leq} C h |v|_{2,\Omega} h^p |u|_{p+1,\Omega} \\
&\overset{\text{ellipt. Reg.}}{\leq} C \|u - u_h\|_{0,\Omega} h^{p+1} |u|_{p+1,\Omega}.
\end{aligned}$$

□

**Korollar 2.25** (Stabilitäts- und Approximationseigenschaften der LDG-Verfahren aus Beispiel 2.17)**.**

| *Nr.* | *Konsistenz* | *adjungiert konsistent* | *Stabilität* | *Bedingung* | $\|\|\|u-u_h\|\|\|$ | $\|u-u_h\|_{0,\Omega}$ |
|---|---|---|---|---|---|---|
| *1* | ✓ | ✓ | – | – | $(h^p)^\dagger$ | $(h^{p+1})^\dagger$ |
| *2* | ✓ | ✓ | ✓ | $\beta_{min} > 0$ | $h^p$ | $h^{p+1}$ |
| *3* | ✓ | ✓ | ✓ | $\beta_{min} > 0$ | $h^p$ | $h^{p+1}$ |
| *4* | ✓ | ✓ | ✓ | $\beta_{min} > \eta^*$ | $h^p$ | $h^{p+1}$ |
| *5* | ✓ | ✓ | ✓ | $\beta_{min} > 3$ | $h^p$ | $h^{p+1}$ |
| *6* | ✓ | – | – | – | $\begin{cases} p=1: & - \\ p\geq 2: & h^p \end{cases}$ | $\begin{matrix} - \\ h^p \end{matrix}$ |
| *7* | ✓ | – | ✓ | $\beta_{min} > 0$ | $h^p$ | $h^p$ |
| *8* | – | – | ✓ | $\beta_{min} \sim h^{-2p}$ | $h^p$ | $h^{p+1}$ |
| *9* | – | – | ✓ | $\beta_{min} \sim h^{-2p}$ | $h^p$ | $h^{p+1}$ |

$+$ *Beschränktheit für alle Verfahren.*

$^\dagger$*: gilt für* $f \in V_h^{p+1}$.

## 2.3 A-posteriori-Fehlerabschätzungen

**Theorem 2.26** (A-posteriori-Fehleridentität für konsistente und adjungiert konsistente LDG-Verfahren)**.**
*Es seien* $\hat{u}, \hat{\sigma}$ *konsistente und erhaltende Flüsse mit* $\hat{u} = 0$ *auf* $\partial\Omega$*. Weiter nehmen wir an, dass für alle rechten Seiten* $g \in L^2(\Omega)$ *eine Lösung* $v \in H^2(\Omega) \cap H_0^1(\Omega)$ *des adjungierten Problems*

$$\begin{aligned} -\Delta v &= g && \text{in } \Omega \\ v &= 0 && \text{auf } \partial\Omega \end{aligned}$$

*existiert. Ist* $u \in H^2(\Omega) \cap H_0^1(\Omega)$ *Lösung von* (1.1) *und* $(u_h, \sigma_h) \in V_h^p \times \Sigma_h^q$ *eine Lösung von* $LDG^p(\hat{u}, \hat{\sigma})$ *auf* $\mathcal{T}_h$*, so gilt:*

$$\|u - u_h\|_{0,\Omega}^2 = \langle f, v - v_h\rangle_\Omega - B_{(\hat{u},\hat{\sigma})}(u_h, v - v_h)$$

*für alle* $v_h \in V_h$.

*Beweis:* Wir wählen $g = u - u_h$ als rechte Seite des dualen Problems. Dann folgt aus der adjungierten Konsistenz:

$$B_{(\hat{u},\hat{\sigma})}(\varphi_h, v) = \langle u - u_h, \varphi_h\rangle_\Omega \qquad \forall \varphi_h \in V(h).$$

Mit $\varphi_h = u - u_h$ folgt weiter:

$$\begin{aligned} \|u - u_h\|_{0,\Omega}^2 &= B_{(\hat{u},\hat{\sigma})}(u - u_h, v) \\ &= B_{(\hat{u},\hat{\sigma})}(u - u_h, v - v_h) \\ &= \langle f, v - v_h\rangle - B_{(\hat{u},\hat{\sigma})}(u_h, v - v_h). \end{aligned}$$

□

**Korollar 2.27** (A-posteriori-Fehlerabschätzung für konsistente und adjungiert konsistente LDG–Verfahren)**.**
*Es gelten die Voraussetzungen aus Theorem 2.26. Dann gilt:*

$$\|u - u_h\|_{0,\Omega} \leq C \cdot \eta(u_h),$$

$\eta^2(u_h) := \sum_{e \in \mathcal{E}^0_{leaf}} \eta_e^2$ *mit den Fehlerindikatoren* $\eta_e$ *definiert durch*

$$\begin{aligned}
\eta_e^2 := {} & h_e^4 \|f + \Delta_h u_h\|_{0,\Omega}^2 \\
& + \frac{1}{2} \sum_{e \in \mathcal{E}^0_{leaf}} \sum_{e^1 \subset \partial e \setminus \partial\Omega} \left( h_e \|[\hat{u} - u_h]\|_{0,e^1}^2 + h_e^3 \|\{(\hat{\sigma} - \nabla_h u_h) \cdot n\}\|_{0,e^1}^2 \right) \\
& + \frac{1}{2} \sum_{e \in \mathcal{E}^0_{leaf}} \sum_{e^1 \subset \partial e \setminus \partial\Omega} \left( h_e \|\{\hat{u} - u_h\}\|_{0,e^1}^2 + h_e^3 \|[(\hat{\sigma} - \nabla_h u_h) \cdot n]\|_{0,e^1}^2 \right) \\
& + \sum_{e \in \mathcal{E}^0_{leaf}} \sum_{e^1 \subset \partial e \cap \partial\Omega} \left( h_e \|\hat{u} - u_h\|_{0,e^1}^2 + h_e^3 \|(\hat{\sigma} - \nabla_h u_h) \cdot n\|_{0,e^1}^2 \right).
\end{aligned}$$

*Die Konstante* $C > 0$ *hängt vom Polynomgrad* $p$ *und vom kleinsten Winkel des Gitters ab.*

*Beweis:* Nach Theorem 2.26 gilt:

$$
\begin{aligned}
\|u-u_h\|_{0,\Omega}^2 &= \langle f, v-v_h\rangle - B_{(\hat{u},\hat{\sigma})}(u_h, v-v_h)\\
&= \langle f, v-v_h\rangle_\Omega - \langle \nabla_h u_h, \nabla_h(v-v_h)\rangle_\Omega\\
&\quad - \left(\int_{\Gamma\setminus\partial\Omega} [\hat{u}-u_h]\{\nabla_h(v-v_h)\cdot n\} - \{\hat{\sigma}\cdot n\}\cdot[v-v_h]\right)\\
&\quad - \int_{\Gamma\setminus\partial\Omega} (\{\hat{u}-u_h\}[\nabla_h(v-v_h)\cdot n] - [\hat{\sigma}\cdot n]\{v-v_h\})\\
&\quad - \int_{\partial\Omega} ((\hat{u}-u_h)\nabla_h(v-v_h)\cdot n - \hat{\sigma}\cdot n(v-v_h))\\
&\overset{\text{p.I.}}{=} \langle f+\Delta_h u_h, v-v_h\rangle_\Omega\\
&\quad - \int_{\Gamma\setminus\partial\Omega} ([\hat{u}-u_h]\{\nabla_h\cdot(v-v_h)\cdot n\}\\
&\qquad\qquad -\{(\hat{\sigma}-\nabla_h u_h)\cdot n\}\cdot[v-v_h])\\
&\quad - \int_{\Gamma\setminus\partial\Omega} (\{\hat{u}-u_h\}[\nabla_h(v-v_h)\cdot n]\\
&\qquad\qquad -[(\hat{\sigma}-\nabla_h u_h)\cdot n]\{v-v_h\})\\
&\quad - \int_{\partial\Omega} ((\hat{u}-u_h)\nabla_h(v-v_h)\cdot n - (\hat{\sigma}-\nabla_h u_h)\cdot n(v-v_h))\\
&\leq \sum_{e\in\mathcal{E}^0_{\text{leaf}}} \|f+\Delta_h u_h\|_{0,e}\cdot\|v-v_h\|_{0,e}\\
&\quad + \sum_{\substack{e^1\in\mathcal{E}^1_{\text{leaf}},\\ e^1\not\subset\partial\Omega}} \big(\|[\hat{u}-u_h]\|\cdot\|\{(\nabla v_h - v_h)\cdot n)\}\|_{0,e^1}\\
&\qquad\qquad +\|\{(\hat{\sigma}-\nabla_h u_h)\cdot n\}\|_{0,e^1}\|[v-v_h]\|_{0,e^1}\big)\\
&\quad + \sum_{\substack{e^1\in\mathcal{E}^1_{\text{leaf}},\\ e^1\not\subset\partial\Omega}} \big(\|\{\hat{u}-u_h\}\|\cdot\|[(\nabla v_h - v_h)\cdot n)]\|_{0,e^1}\\
&\qquad\qquad +\|[(\hat{\sigma}-\nabla_h u_h)\cdot n]\|_{0,e^1}\|\{v-v_h\}\|_{0,e^1}\big)\\
&\quad + \sum_{\substack{e^1\in\mathcal{E}^1_{\text{leaf}},\\ e^1\subset\partial\Omega}} \big(\|\hat{u}-u_h\|_{0,e^1}\cdot\|(\nabla v_h - v_h)\cdot n)\|_{0,e^1}\\
&\qquad\qquad +\|(\hat{\sigma}-\nabla_h u_h)\cdot n\|_{0,e^1}\|v-v_h\|_{0,e^1}\big)\,.
\end{aligned}
$$

Sei nun $I_h : H^2(\Omega) \to V_h^{p\geq 1}$ eine Interpolierende mit den Eigenschaften:

$$
\begin{aligned}
\|v-I_h v\|_{0,e} &\leq C h_e^s\cdot|v|_{s,e},\ \forall e\in\mathcal{E}^0_{\text{leaf}},\\
\|v-I_h v\|_{0,e^1} &\leq C h_{e^1}^{s-\frac{1}{2}}|v|_{s,e^1},\ \forall e^1\in\mathcal{E}^1_{\text{leaf}}, e^1\subset\partial e.
\end{aligned}
$$

Dabei sei $s \leq \min\{p+1, 2\}$. Wählt man $v_h = I_h v$, so folgt

$$\begin{aligned}\|u-u_h\|_{0,\Omega}^2 \leq C \sum_{e\in\mathcal{E}_{\text{leaf}}^0} \Big( & h_e^2 \|f+\Delta_h u_h\|_{0,e}|v|_{2,e} \\ & + \frac{1}{2}\sum_{e\in\mathcal{E}_{\text{leaf}}^0}\sum_{e^1\subset\partial e\setminus\partial\Omega}\left(h_e^{\frac{1}{2}}\|[\hat{u}-u_h]\|_{0,e^1} + h_e^{\frac{3}{2}}\|\{(\hat{\sigma}-\nabla_h u_h)\cdot n\}\|_{0,e^1}\right)\cdot|v|_{2,e} \\ & + \frac{1}{2}\sum_{e\in\mathcal{E}_{\text{leaf}}^0}\sum_{e^1\subset\partial e\setminus\partial\Omega}\left(h_e^{\frac{1}{2}}\|\{\hat{u}-u_h\}\|_{0,e^1} + h_e^{\frac{3}{2}}\|[(\hat{\sigma}-\nabla_h u_h)\cdot n]\|_{0,e^1}\right)\cdot|v|_{2,e} \\ & + \sum_{e\in\mathcal{E}_{\text{leaf}}^0}\sum_{e^1\subset\partial e\cap\partial\Omega}\left(h_e^{\frac{1}{2}}\|\hat{u}-u_h\|_{0,e^1} + h_e^{\frac{3}{2}}\|(\hat{\sigma}-\nabla_h u_h)\cdot n\|_{0,e^1}\right)\cdot|v|_{2,e}\Big).\end{aligned}$$

Mit Cauchy-Schwarz gilt damit:

$$\begin{aligned}\|u-u_h\|_{0,\Omega}^2 &\overset{\text{C.S.}}{\leq} C\cdot\eta(u_h)|v|_{2,\Omega} \\ &\overset{\text{ellipt. Reg.}}{\leq} C\cdot\eta(u_h)\|u-u_h\|_{0,\Omega}.\end{aligned}$$

Kürzen liefert die Behauptung. □

**Beispiel 2.28** (A-posteriori-Fehlerabschätzung für das IP-Verfahren ($LDG_4$))**.**
Für das $LDG_4$-Verfahren aus Beispiel 2.17 gilt:

$$\begin{aligned}\|u-u_h\|_{0,\Omega} &\leq C\eta(u_h), \\ \eta^2(u_h) &= \sum_{e\in\mathcal{E}_{\text{leaf}}^0}\eta_e^2,\end{aligned}$$

mit

$$\begin{aligned}\eta_e^2 =& h_e^4\|f+\Delta_h u_h\|_{0,\Omega}^2 \\ &+ \sum_{e\in\mathcal{E}_{\text{leaf}}^0}\sum_{e^1\subset\partial e\setminus\partial\Omega}\left((1+\beta_{e^1}^2)h_e\|[u_h]\|_{0,e^1}^2 + h_e^3\|[\nabla_h u_h\cdot n]\|_{0,e^1}^2\right) \\ &+ \sum_{e\in\mathcal{E}_{\text{leaf}}^0}\sum_{e^1\subset\partial e\cap\partial\Omega}(1+\beta_{e^1}^2)h_e\|u_h\|_{0,e^1}^2.\end{aligned}$$

Dabei hängt die Konstante $C$ vom Polynomgrad $p$ ab.

*Beweis:* Für $LDG_4$ sind die numerischen Flüsse gegeben durch

$$\begin{aligned}\hat{u} &= \begin{cases}\{u_h\}, & \text{auf } \Gamma\setminus\partial\Omega, \\ 0, & \text{auf } \partial\Omega,\end{cases} \\ \hat{\sigma} &= \begin{cases}\{\nabla_h u_h\} - \alpha_j([u_h])\cdot n, & \text{auf } \Gamma\setminus\partial\Omega, \\ \nabla_h u_h - \alpha_j(u_h)\cdot n, & \text{auf } \partial\Omega.\end{cases}\end{aligned}$$

Also folgt

(i)

$$\begin{aligned}[\hat{u}-u_h] &= -[u_h], && \text{auf } \Gamma\setminus\partial\Omega, \\ \{\hat{u}-u_h\} &= 0, && \text{auf } \Gamma\setminus\partial\Omega, \\ \hat{u}-u_h &= -u_h, && \text{auf } \partial\Omega.\end{aligned}$$

(ii)

$$\begin{aligned} \{(\hat{\sigma} - \nabla_h u_h) \cdot n\}|_e &= -\frac{\beta_e}{h_e}[u_h], && e \subset \Gamma \setminus \partial\Omega, \\ [(\hat{\sigma} - \nabla_h u_h) \cdot n] &= -[\nabla_h u_h \cdot n], && \text{auf } \Gamma \setminus \partial\Omega, \\ (\hat{\sigma} - \nabla_h u_h) \cdot n|_e &= -\frac{\beta_e}{h_e} u_h, && e \subset \Gamma \cap \partial\Omega. \end{aligned}$$

Einsetzen in die Fehlerabschätzung aus Korollar 2.27 ergibt die Behauptung. □

**Bemerkung 2.29.**

1) Auch für nicht adjungiert konsistente Verfahren können mit Hilfe der beschriebenen Vorgehensweise A-posteriori-Fehlerabschätzungen bewiesen werden.
(*siehe [19] für NPG*)

2) Durch eine Zerlegung des $H^1$-Fehlers in konformen und nicht-konformen Anteil können unter Verwendung der Helmholtz-Zerlegung auch A-posteriori-Fehlerabschätzungen in der Energienorm hergeleitet werden.
(*siehe [5] und [8]*)

3) Korollar 2.27 gibt eine obere Abschätzung des Fehlers, eine entsprechende untere Abschätzung ist bisher nicht bewiesen worden.

## 2.4 Adaptives LDG-Verfahren

**Ziel.**
Startend mit einem "groben" Gitter $\mathcal{T}_h^0$ soll ein lokal verfeinertes Gitter $\mathcal{T}_h^j$ konstruiert werden, so dass für eine vorgegebene Toleranz $TOL$ gilt:
Ist $(u_h^j, \sigma_h^j)$ Lösung eines LDG-Verfahrens auf Gitter $\mathcal{T}_h^j$, so gilt

$$\|u - u_h\|_{\#} \leq TOL.$$

**Annahme.**
Für das betrachtete LDG-Verfahren existiere eine A-posteriori-Fehlerabschätzung der Form

$$\begin{aligned} \|u - u_h\|_{\#} &\leq C_{\#}\eta, \\ \eta^2 &= \sum_{e \in \mathcal{E}^0_{\text{leaf}}} \eta_e^2. \end{aligned} \tag{2.12}$$

**Notation.**
$(\mathcal{E}^0_{\text{leaf}})^j \mathrel{\hat{=}}$ Blattgitter der Kodimension 0 von Gitter $\mathcal{T}_h^j$,
$M^j := \left|(\mathcal{E}^0_{\text{leaf}})^j\right| \mathrel{\hat{=}}$ Anzahl der Blattentitäten von $\mathcal{T}_h^j$,
$TOL^j := \left(\frac{TOL}{C_{\#}}\right)^2 \frac{1}{M^j}$.

Mit (2.12) und der Definition von $TOL^j$ folgt: Ist

$$\left(\eta_e^j\right)^2 \leq TOL^j \qquad \forall e \in \mathcal{E}^0_{\text{leaf}},$$

dann gilt

$$C\eta^j \leq TOL.$$

Basierend auf dieser Feststellung definieren wir nun für ein gegebenes $\theta \in (0, \frac{1}{2})$ folgenden adaptiven Algorithmus:

**Definition 2.30** (Adaptives LDG-Verfahren)**.**
Gegeben: $\mathcal{T}_h^0, \theta, TOL, LDG(\hat{u}, \hat{\sigma})$
Berechne: $(u_h^0, \sigma_h^0)$ und $\eta^0, \eta_e^0$ auf $\mathcal{T}_h^0$ und setze $j = 0$.
Solange $(C_\# \eta^j > TOL)\{$

1) $\forall e \in \left(\mathcal{E}_{\text{leaf}}^0\right)^j$:
   falls $\left(\eta_e^j\right) > TOL^j$ : markiere $e$ zur Verfeinerung,
   falls $\left(\eta_e^j\right) < \theta TOL^j$ : markiere $e$ zur Vergröberung.
2) Konstruiere $\mathcal{T}_h^{j+1}$ aus $\mathcal{T}_h^j$ indem alle zur Verfeinerung markierten Entitäten verfeinert werden und alle zur Vergröberung markierten Entitäten vergröbert werden, falls dies möglich ist.
3) Berechne $(u_h^{j+1}, \sigma_h^{j+1}), \eta^{j+1}, \eta_e^{j+1}$ auf $\mathcal{T}_h^{j+1}$.
4) Setze $j = j + 1$.

$\}$

## 2.5 LDG-Verfahren für nichtlineare Probleme

**Definition 2.31** (Allgemeines nichtlineares Problem in Divergenzform)**.**
Sei $\Omega \subset \mathbb{R}^n$ ein Gebiet mit polygonalem Rand und gelte $\partial\Omega = \overline{\Gamma}_D \cup \overline{\Gamma}_N$ mit $\mathring{\Gamma}_D \cap \mathring{\Gamma}_N = \emptyset$ und $\Gamma_D \neq \emptyset$. Seien $a : \Omega \times \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^n$ und $f : \Omega \times \mathbb{R} \to \mathbb{R}$ gegeben. Wir suchen eine Lösung $u \in H^1(\Omega)$ mit

$$-\nabla \cdot a(\cdot, u(\cdot), \nabla u(\cdot)) = f(\cdot, u(\cdot)) \qquad \text{in } \Omega, \tag{2.13a}$$

$$u = g_D \qquad \text{auf } \Gamma_D, \tag{2.13b}$$

$$a(\cdot, u(\cdot), \nabla u(\cdot)) \cdot n = g_N \qquad \text{auf } \Gamma_N. \tag{2.13c}$$

**Beispiel 2.32.**

**A)** Lineare Advektions-Diffusionsgleichung:

$$-\nabla \cdot (D\nabla u) + \nabla \cdot (\overrightarrow{b} \cdot u) = \overline{f}$$

$$\Rightarrow \quad \begin{aligned} a(x, v, p) &= Dp - \overrightarrow{b}\, v, \\ f(x, v) &= \overline{f}(x). \end{aligned}$$

**B)** Mean-Curvature-flow:

$$-\nabla \cdot \left(\frac{\nabla u}{|\nabla u|}\right) = 0$$

$$\Rightarrow \quad \begin{aligned} a(x, v, p) &= \frac{p}{|p|}, \\ f(x, v) &= 0. \end{aligned}$$

**C)** Hyperbolische Erhaltungsgleichungen:

$$-\nabla \cdot F(u) = 0$$

$$\begin{aligned} \Rightarrow \quad a(x,v,p) &= F(v), \\ f(x,v) &= 0. \end{aligned}$$

**Definition 2.33** (Allgemeines Problem als System 1. Ordnung)**.**
Wir führen neue Variablen ein:

$$\begin{aligned} \theta &= \nabla u, \\ \sigma &= a(\cdot, u, \theta). \end{aligned}$$

Dann ist $(u, \theta, \sigma)$ eine Lösung vonm Problem in Definition 2.31, falls gilt:

$$\left.\begin{aligned} \sigma &= a(\cdot, u, \theta) \\ \theta &= \nabla u \\ -\nabla\sigma &= f(\cdot, u) \end{aligned}\right\} \quad \text{auf } \Omega, \tag{2.14}$$

$$\begin{aligned} u &= g_D, && \text{auf } \Gamma_D, \\ \sigma \cdot n &= g_N, && \text{auf } \Gamma_N. \end{aligned}$$

**Definition 2.34** (Schwache Formulierung von (2.14))**.**
Sei $\mathcal{T}_h$ ein hierarchisches Gitter auf $\Omega$. Dann heißt $(u, \theta, \sigma) \in H^1(\Omega) \times [L^2(\Omega)]^n \times [H^1(\Omega)]^n$ schwache Lösung von (2.14), falls für alle $e \in \mathcal{E}^0_{\text{leaf}}$ gilt:

$$\begin{aligned} \int_e a(\cdot, u, \theta) \cdot \psi - \int_e \sigma \cdot \psi &= 0, && \forall \psi \in [L^2(\Omega)]^n, \\ \int_e \theta \cdot \psi &= -\int_e u \cdot \nabla\psi + \int_{\partial e} u\psi \cdot n, && \forall \psi \in [H^1(\Omega)]^n, \\ \int_e \sigma \cdot \nabla\varphi &= \int_{\partial e} \sigma \cdot n\varphi + \int_e f(\cdot, u) \cdot \varphi, && \forall \varphi \in H^1(\Omega), \end{aligned}$$

und

$$\begin{aligned} u &= g_D, && \text{auf } \Gamma_D, \\ \sigma \cdot n &= g_N, && \text{auf } \Gamma_N. \end{aligned}$$

**Definition 2.35** (Allgemeines LDG-Verfahren für nichtlineare Probleme)**.**
Sei $\mathcal{T}_h$ ein hierarchisches Gitter auf $\Omega \subset \mathbb{R}^n$ und seien die numerischen Flüsse $\hat{u}, \hat{\sigma}$ auf $\mathcal{T}_h$ gegeben. Ein Tripel $(u_h, \theta_h, \sigma_h) \in V_h^p \times \Sigma_h^q \times \Sigma_h^r$ heißt Lösung des *LDG*-Verfahrens der Ordnung $(p, q, r)$ zu den Flüssen $\hat{u}, \hat{\sigma}$, falls für alle $e \in \mathcal{E}^0_{\text{leaf}}$ gilt:

$$\begin{aligned} \int_e (a(\cdot, u_h, \theta_h) - \sigma_h) \cdot \psi_h &= 0, && \forall \psi_h \in \Sigma_h^q, \\ \int_e \theta_h \cdot \psi_h &= -\int_e u_h \cdot \nabla\psi_h + \int_{\partial e} \hat{u}_e \psi_h \cdot n, && \forall \psi_h \in \Sigma_h^r, \\ \int_e \sigma_h \cdot \nabla\varphi_h &= \int_e f(\cdot, u_h) \cdot \varphi_h + \int_{\partial e} \hat{\sigma}_e \cdot n\varphi_h, && \forall \varphi_h \in V_h^p. \end{aligned}$$

Wir bezeichnen das Verfahren mit $LDG^{p,q,r}(\hat{u},\hat{\sigma})$ (entsprechend $LDG(\hat{u},\hat{\sigma})$, falls $p=q=r$.) Das Verfahren heißt konsistent mit den Randbedingungen, falls gilt:

$$\begin{aligned} \hat{u} &= g_D, && \text{auf } \Gamma_D, \\ \hat{\sigma} &= g_N, && \text{auf } \Gamma_N. \end{aligned}$$

# KAPITEL 3

# Interface-Programmierung in C++

**Elemente des Programmentwurfs:**

1) "Proof of concept":
   Beispiele: Template-Klassen für einen festen Datentyp implementieren und testen. Zunächst einfache Verfahren implementieren und testen (z.B. 1D, satt 3D). Dabei immer auf fertige Bibliotheken zurückgreifen (etwa STL).

2) "Profiling":
   Aufspüren von Engpässen im Code. Nicht den Teil optimieren, der nur 1% der Laufzeit ausmacht (Tools: gprof,...)

3) Optimierungsoptionen des Compilers studieren.
   Beispiele: loopunrolling, Return-value-Optimierung, inline-Grenzen,...
   Die Optimierungsoptionen können sehr von dem verwendeten Compiler abhängen.

## 3.1 Funktionsweise des Compilers

**Definition** (Statisches und dynamisches Binden)**.**
In C++ muss jeder Methode/Funktion und jeder Variablen ein Typ zugewiesen werden. Dies bezeichnet man als den *statischen Typ* und sagt, dass die Variablen/Methoden statisch gebunden sind. Dies ermöglicht starke Typprüfung durch den Compiler. Jede Variable hat auch einen *dynamischen Typ*, der erst zur Laufzeit feststeht. Nur bei Instanzen von Klassen mit virtuellen Funktionen unterscheiden sich diese Konzepte.

### 3.1.1 Speicheraufbau

#### Aus Programmierersicht

**Definition** (Heap, Stack und statischer Speicherbereich)**.**
Der *statische Speicherbereich* enthält alle global definierten Variablen und statische Klassenvariablen. Der *Stack* enthält alle lokalen Variablen sowie Funktionsparameter und Programmflussinformationen. Der *Heap* enthält die dynamisch ausgelegten Klassen. Klassentemplates erzeugen keinen Code. Sie sind nur Schablonen und erst bei der Instanzierung wird Code erzeugt. Code wird ein-

mal pro Templateparameter erzeugt, d.h. die Kombination `A<int>` ist ein völlig anderer Typ als `A<double>`.

**Beispiel.**
Code:

```
1  class A
2  {
3    static double a;
4    double b;
5    void foo();
6  };
7
8  void foo()
9  {
10   double* heap = new double[100];
11   double x,y;
12   A a;
13 }
```

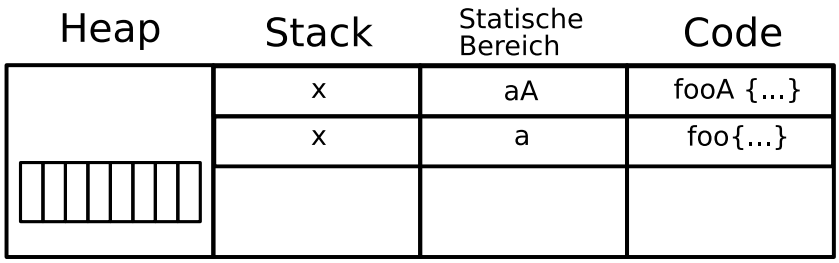
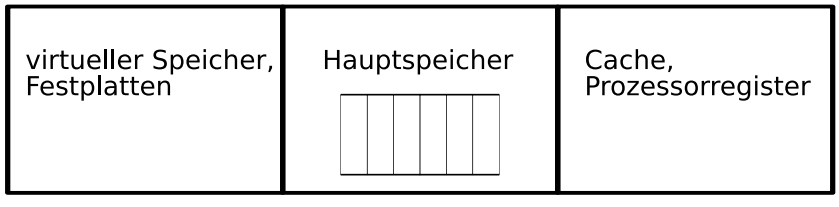



Abbildung 3.1: Speicheraufbau

### Aus Hardwaresicht

Der Speicher ist hierarchisch organisiert in Prozessorregister, Level-1-Cache, Level-2-Cache, Hauptspeicher und virtueller Speicher (Festplatte). Daten müssen stets über diese Hierarchie nachgeladen werden.

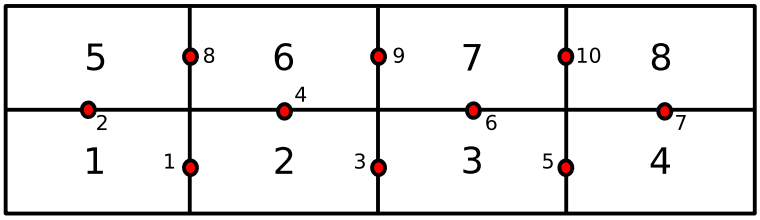
Der Cache enthält einige Seiten (pages) des Speichers. Ist ein benötigtes Datum im Cache (*cache hit*), so ist der Zugriff effizient. Andernfalls spricht man von *cache miss* und die Seite muss aus dem Hauptspeicher geladen werden.




Der Cache enthält einige Seiten des Hauptspeichers. Ist ein benötigtes Datum im Cache (*cache hit*), so ist der Zugriff effizient. Andernfalls spricht man von *cache miss* und die Seite muss aus dem Hauptspeicher geladen werden.

**Definition** (Cacheeffizienz)**.**
Eine Cache-effiziente Implementierung minimiert die Anzahl der cache misses.

**Beispiel** (Berechnung von Flüssen über Elementkanten)**.**

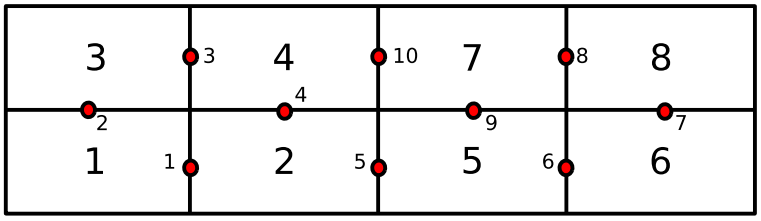



Annahme: Cache hat zwei Seiten und zwei Zeilen pro Seite. Wichtig: Transparenz für Benutzer.

In diesem Beispiel erhalten wir folgendes Schema:

| Fluss | Cache | | Miss |
|---|---|---|---|
| 1,2 | 1 2 | 5 6 | 2 |
| 3 | 1 2 | 3 4 | 1 |
| 4 | 1 2 | 5 6 | 1 |
| 5 | 1 2 | 3 4 | 1 |
| 6,7 | 7 8 | 3 4 | 1 |
| 8,9,10 | 7 8 | 5 6 | 1 |
| | | | 7 |

Eine Umnummerierung der Entitäten kann zu einer Verbesserung der Cacheeffizienz führen, wie folgendes Beispiel zeigt:




Wir erhalten das effizientere Schema:

| Fluss | Cache | | Miss |
|---|---|---|---|
| 1,2,3,4 | 1 2 | 3 4 | 2 |
| 5 | 1 2 | 5 6 | 1 |
| 6,7,8,9 | 7 8 | 5 6 | 1 |
| 10 | 7 8 | 5 6 | 1 |
| | | | 5 |

**Bemerkung.**
Bei unstrukturierten Gittern wird Cacheeffizienz in der Regel ignoriert.

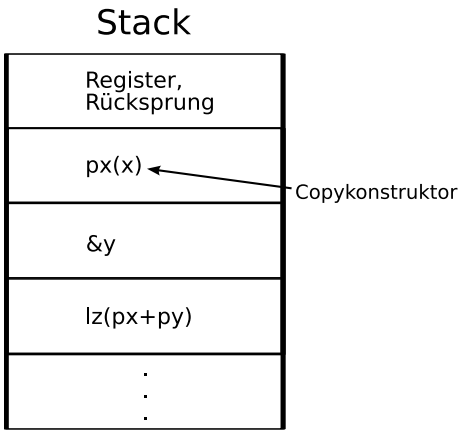
Der AMR-Ansatz (adaptive mesh refinement, blockstrukturiert) versucht Cacheeffizienz umzusetzen.

### 3.1.2 Funktionen/Methodenaufrufe

Dazu betrachten wir folgendes Codefragment:

```
1  double foo( double px, double& py )
2  {
3    double lz = px = py;
4    return lz;
5  }
6  int main ()
7  {
8    double x, y, z;
9    z = foo( x, y );
10 }
```

Die Funktion wird in einem Stapel oder Stack wie folgt realisiert:




Der Aufruf des =-Operators auf `z` führt zum Aufräumen des Stacks. Bei der Anweisung `double z = foo(x,y);` würde hingegen der Copy-Konstuktor aufgerufen werden.
Bei Klassenmethoden wird aus `A::foo(x,y); fooA(A*, double x, double y);`. Mit `this` kann auf `A*` zugegriffen werden.

**Definition** (Return-value-Optimierung)**.**
Der Compiler wandelt `double foo(double, & double)` um in `void foo(double, &double, * double)`. Dadurch wird das Anlegen einer lokalen Instanz (hier `lz`) und der Copy-Konstruktor vermieden. Der =-Operator wird auf jeden Fall aufgerufen.

Probleme bei Funktionsaufrufen:

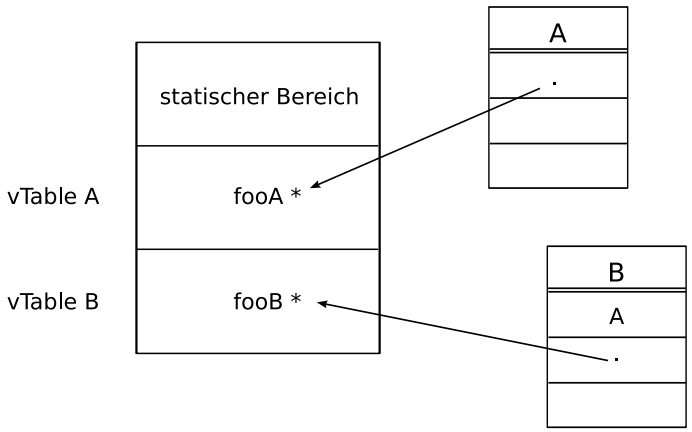
- Kopien auf den Stack
- Cache miss
- Pipelining kann kaputt gehen (spielt mit eine Rolle bei kleinen Funktionen)

**Definition** (Inlining von Funktionen)**.**
Der Compiler kann (muss aber nicht) den Code von `foo` an die Stelle des Aufrufs einfügen. Die Benutzung von inline kann (muss aber nicht) zu schnelleren/größeren Programmen führen.
Was der Compiler inlinet, kann durch Compilerflags beeinflusst werden.

**Definition** (Virtuelle Funktionen)**.**
Für jede Klasse mit virtuellen Methoden wird eine *vTable* im statischen Speicherbereich angelegt und pro Instanz ein Pointer auf die vTables des dynamischen Typs. Die vTable besteht aus Funktionspointern.

**Beispiel** (virtuelle Funktionen)**.**
Wir betrachten folgendes Codebeispiel:

```
1 class A
2 {
3   virtual void foo();
4 };
5
6 class B: public A
7 {
8   virtual void foo();
9 };
```




Durch den Pointer auf die vTable kann der dynamische Typ bestimmt werden und die richtige virtuelle Funktion aufgerufen werden.

**Bemerkung.**
Virtuelle Funktionen erzeugen kaum zusätzlichen Speicheraufwand.

Abarbeitung eines Methodenaufrufs:

1) Holen des Pointers auf die vTable,
2) Laden der vTable,
3) Aufruf der Funktionen durch den Funktionsparameter.

Der Effizienzverlust (besonders bei kleinen Funktionen) liegt an fehlender Möglichkeit zum Inlinen.

### 3.1.3 Objectfiles

Objectfiles enthalten alle im Compilierungsmodul erzeugten globalen Symbole. Aber nicht:

- `static` (C-Konstrukt! Nicht in C++ verwenden),
- inline definierte Methoden (daher müssen inline-Funktionen überall eingebunden werden, wo sie benutzt werden sollen. Nicht inline-definierte Funktionen dürfen nicht im Header stehen. Inline-Funktionen müssen im Header stehen.)

Des Weiteren enthalten Objectfiles die unaufgelösten Synmbole (z.B. `extern`,...).

Linken: Verbinden unaufgelöster Symbole mit den exportierten Symbolen. Bibliotheken sind einfach eine Ansammlung von Objectfiles.

## 3.2 Standardbibliotheken

In den Standard aufgenommen wurde 1997 die Standard Template Library (STL). Sie definiert:

1) Container,
2) Iteratoren,
3) Algorithmen.

### 3.2.1 Container

Container sind Speicherstrukturen für gleichartige Objekte. Der Objekttyp wird über Template Argumente festgelegt.

Bei der STL werden keine Datenstrukturen vorgeschrieben, sondern Komplexitäten von elementaren Operationen und Arten des Zugriffs auf die Elemente.
**Beispiel** (`vector<T>`)**.**
Vorgeschrieben:

- Objekte liegen konsekutiv im Speicher,
- `&(v[0])` liefert die Adresse des ersten Elementes,
- aber `&(v[0])` $\neq$ `&(v.begin())`!

Überblick über vorgeschriebene Komplexitäten:

| **Sequenzen** | `vector` | `dequeue` | `list` (einfach verkettet) |
|---|---|---|---|
| Einfügen/Löschen am Ende | $\mathcal{O}(1)$ | $\mathcal{O}(1)$ | $\mathcal{O}(1)$ |
| Einfügen/Löschen am Anfang | $\mathcal{O}(N)$ | $\mathcal{O}(1)$ | $\mathcal{O}(1)$ |
| Einfügen/Löschen an beliebiger Position | $\mathcal{O}(N)$ | $\mathcal{O}(N)$ | $\mathcal{O}(1)$ |
| wahlfreier Zugriff | $\mathcal{O}(1)$ | $\mathcal{O}(1)$ | $\mathcal{O}(N)$ |

### Assoziative Container

Merkmal: “Schnelle Suche, kein wahlfreier Zugriff”

| `set`, | `multiset`, | `map`, | `multimap` |
|---|---|---|---|
| ↑ | ↑ | ↑ | ↑ |
| key | key | (key, value) | (key, value) |
| (eindeutig) | | (eindeutig) | |

Beispiel eines Codefragments:

```
1  using namespace std;
2  {
3    map< const string, int > kurs;
4    map< const string, int>::iterator it;
5
6    kurs["A"] = 13;
7    kurs["B"] = 23;
8    kurs["C"] = 16;
9
10   string key( "B" );
11   it = kurs.find( key );
12   cout << "Anzahl␣Kursteilnehmer:␣" << el->second << endl;
13 }
```

Elemente eines Containers gehören zu Klassen mit

- Default-Konstruktor,
- Copy-Konstruktor,
- =-Operator.

Bei assoziativen Containern muss für den Schlüssel der <-Operator vorhanden sein.
Es ist kein Container mit Referenzen möglich: Anstelle von `vector<T &>` muss man `vector<T *>` benutzen.

### 3.2.2 Iteratoren

Container stellen Iteratoren zur Verfügung: `begin()`, `end()`.

Iteratoren in der STL sind verallgemeinerte C-Pointer, d.h. sie enthalten keine eigenen Methoden, sondern nur Operatoren `++`, `*`, `->`, `==`,...

**Alternatives Design Pattern**:

```
1  template< Container >
2  class Iterator
3  {
4    void first();
5    void next();
6    bool done();
```

```
7    T& item();
8  };
```

**Spezialisierung von Template-Argumenten**:

```
1  template<>
2  class Iterator< vector >
3  {
4  //...
5  };
```

## 3.3 Von der abstrakten Gitterdefinition zum DUNE-Interface

**Ziel.**
Umsetzung der abstrakten Gitterdefinitionen (Def. 2.3 - 2.7) in C++-Klassen.

### 3.3.1 Verwendete Techniken und Designelemente

- generische Programmierung (Template-Techniken in C++),
- statischer Polymorphismus (Barton-Nackmann und Engine-Konzept),
- Dimensionsunabhängigkeit,
- Unabhängigkeit von Datenstruktur, d.h. die Schnittstelle schreibt nur das Verhalten vor,
- schlankes Interface zur Implementierung und breites Interface zur Verwendung (Interface- und Defaultklassen).

(Details zur Implementierung siehe Abschnitt 3.4.)

### 3.3.2 Überblick über die Klassen der DUNE-Gitterschnittstelle

1) `Grid<dim, dimworld>`: Diese Klasse repräsentiert das hierarchische Gitter $\mathcal{T}_h$ (vgl. Def. 2.5). `dim` gibt die Dimension des Gitters an und `dimworld` die Dimension des Raumes, in den das Gitter eingebettet ist. `Grid` ist ein Container von Entitäten und bietet Iteratoren zum Durchlaufen der Entitäten $\mathcal{E}_l^c$ und $\mathcal{E}_{\text{leaf}}^c$ an (vgl. Def. 2.5, 2.7).
2) `Entity<codim, dim>`: Schnittstelle zu Entitäten der Kodimension `codim` eines Gitters der Dimension `dim`. `Entity` enthält alle topologischen Informationen einer Entität, wie z.B. Vater-Kind-Relationen und Iteratoren über die Intersections.
3) `Geometry<mydim, dimworld>`: Enthält die geometrischen Informationen einer Entität, insbesondere die Referenzabbildung $F : \hat{e} \to e$ (vgl. Def. 2.5).
4) Die Iteratorklassen
   a) `LevelIterator<codim>`: durchläuft die Menge $\mathcal{E}_l^{\texttt{codim}}$,
   b) `LeafIterator<codim>`: durchläuft die Menge $\mathcal{E}_{\text{leaf}}^{\texttt{codim}}$,
   c) `HierarchicIterator`: durchläuft die Menge $C(e)$, startend auf einer Entiät $e \in \mathcal{E}_l^0$,

d) `IntersectionIterator`: durchläuft die Menge $\mathcal{I}(e)$, startdend auf einer Entität $e \in \mathcal{E}_l^0$,

e) `EntityPointer`: IteratorBasisklasse ohne `increment`.

5) Index- und Id-Mengen

a) `LevelIndexSet`: repräsentiert die Indexmenge $I_l^c$,

b) `LeafIndexSet`: repräsentiert die Indexmenge $I_{\text{leaf}}^c$,

c) `globalIdSet`: eine eindeutige Id für Entitäten (auch nach Gitteradaption),

d) `localIdSet`: lokal eindeutige Id für Entitäten auf einem Prozessor.

6) a) `ReferenceElement<dim>`: definiert ein Referenzelement $\hat{e}$ der Dimension `dim` durch die Subentitätenmengen $\mathcal{E}^c(\hat{e})$ (vgl. Def. 2.3),

b) `ReferenceElementContainer<dim>`: repräsentiert eine Menge von Referenzelementen $\hat{M}$ der Dimension `dim` (vgl. Def. 2.3).

7) Klassen für parallele Kommunikation:

a) `Datahandle`,

b) `CollectiveCommunication`

(ohne Beschreibung).

### 3.3.3 Die Grid-Klasse und ihre Verwendung

**1) Wichtige Methoden (Auswahl)**

Allgemein

`size(codim)`: liefert Anzahl der Entitäten der Kodimension `codim` auf Blattgitter,

`size(level, codim)`: liefert Anzahl der Entitäten der Kodimension `codim` auf dem Level `level`,

`maxLevel()`: gibt den maximalen Level des Gitters an.

Iteratoren

`<codim>lbegin(level)`: liefert einen LevelIterator, zeigt auf erste Entität von $\mathcal{E}_{\texttt{level}}^{\texttt{codim}}$,

`<codim>lend(level)`: liefert einen LevelIterator, zeigt auf eins hinter der letzten Entität von $\mathcal{E}_{\texttt{level}}^{\texttt{codim}}$,

`<codim>leafbegin(),<codim>leafend()`: liefert Iterator für Blattgitter $\mathcal{E}_{\texttt{leaf}}^{\texttt{codim}}$.

Index- und IdSets

`levelIndexSet()`, `leafIndexSet()`, `globalIdSet()`, `localIdSet()` liefern Objekte der entsprechenden IndexSets bzw. IdSet-Klassen.

Adaptivität

`globalRefine(refCount)`: verfeinert Gitter `refCount`-mal ,

markiert Entität $e$ zur Verfeinerung/Vergröberung.
Dabei gilt:

`mark(refCount,e)`: $\text{refCount} = \begin{cases} 1 \Rightarrow & \text{verfeinern} \\ -1 & \Rightarrow \text{vergröbern} \end{cases}$

`preAdapt()`
`adapt()`
`postAdapt()`
} Umbau des Gitters nach Markierung,

(Diese Methoden werden immer in dieser Reihenfolge aufgerufen.)

Parallele Kommunikation

`communicate(dataHandle,interface,direction)`: Datenkommunikation,

`comm()`: liefert ein Objekt vom Typ CollectiveCommunication.

**2) Verwendung der Iteratoren**

**Beispiel** (Durchlauf der Blattentitäten der Kodimension 0)**.**

```
1  template< GridImp >
2  void traverseElements( GridImp& grid )
3  {
4    typedef typename GridImp::template Codim< 0 > LeafIterator
5      ELIteratorType;
6    ELIteratorType it = grid.template leafbegin<0>();
7    for( it; it!= grid.template leafend<0>(); ++it )
8    {
9      //...
10   }
11 }
```

Eine Anleitung zur Verwendung der Gitterschnittstelle DUNE findet man in [7].

**3) Kopplung von Daten an Gitterentitäten:**

Die Index- und IdSets liefern bijektive Abbildungen von Entitätenmengen in geordnete Identifizierungsmengen, z.B.

$$\begin{aligned} i_l^c : &\quad \mathcal{E}_l^c \longrightarrow I_l^c \quad \text{in } \texttt{LevelIndexset}, \\ i_{\text{leaf}}^c : &\quad \mathcal{E}_l^c \longrightarrow I_{\text{leaf}}^c \quad \text{in } \texttt{LeafIndexset}. \end{aligned}$$

Ist

$$D(\mathcal{E}') := \left\{ d_e \mid e \in \mathcal{E}' \right\}$$

eine Datenmenge, die den Entitäten $e \in \mathcal{E}'$ zugeordnet ist, und ist $i : \mathcal{E}' \to I_{\mathcal{E}'}$ eine bijektive Abbildung in eine geordnete Identifizierungsmenge, dann gilt

$$d_e = d(i(e)),$$

falls $d : I_{\mathcal{E}'} \to D(\mathcal{E}')$ eine Abbildung auf die Datenmenge ist.

### 3.3.4 Die Entity-Klasse

**1) Die Methoden für alle Kodimensionen**

`level()`: gibt den Level der Entität zurück,

`geometry()`: gibt Objekt vom Typ Geometry zurück.

**2) Methoden für Entitäten der Kodimension 0**

`<codim>entity(i)`: gibt die `i`-te Subentität der Kodimension `codim` zurück,

`ileafbegin()`, `ileafend()`, `ilevelbegin()`, `ilevelend()` } liefern IntersectionIterator,

`father()`: liefert Vaterentität, falls vorhanden,

`geometryInFather()`: liefert ein Objekt vom Typ Geometry mit einer Abbildung der Referenzelements in das Referenzelements des Vaters.

`hbegin()`, `hend()` } liefern HierarchicIterator.

`wasRefined ()`, `mightBeCoarsened ()` } Informationen zur Gitteradaption.

### 3.3.5 Die Sicht auf ein Gitter: `GridParts` und `GridView`

Da man zur Programmierung in der Regel nur eine Sicht auf das hierarchische Gitter verwendet (z.b. Leaf- oder Level-Sicht) gibt es in DUNE-FEM die Klasse `GridParts` und ab der Version 1.2 in DUNE-GRID die Klasse `GridView`.
Diese Klassen exportieren eine definierte Sicht auf das Gitter und bieten Iteratoren und Indexmengen für diese Sicht an. Insbesondere wird mit

`ibegin(&e)`,
`iend(&e)`

ein Intersection-Iterator für die Entität $e$ in dieser Sicht angeboten und mit

`begin()`,
`end()`

ein Iterator für die Kodimension-0-Entitäten dieser Sicht.
Um bei der Programmierung unabhängig von der Sicht zu sein, sollte stets dieses Konzept verwendet werden.

### 3.3.6 Die Geometrie-Klasse

`type()`: gibt den GeometryType der Geometrie zurück (z.B. `cube`, `simplex`, `prism`, `pyramid`),

`corners()`: gibt die Anzahl der Ecken einer Entität an,

`operator[](i)`: gibt die Koordinate der `i`-ten Ecke an,

`global(x_local)`: gibt die globalen Koordinaten der lokalen Koordinaten `x_local` an. Entspricht damit der Referenzabbildung $F$.

`local(x_global)`: entspricht $F^{-1}$,

`checkInside(x_local)`: gibt true/false zurück, je nachdem ob `x_local` im Element liegt, oder nicht,

`integrationElement(x_local)`: liefert $\sqrt{\det\left(\nabla F(\texttt{x\_local})^T \nabla F(x\_\texttt{local})\right)}$,

`volume()`: liefert Volumen der Entität,

`jacobianInverseTransposed(x_local)`: liefert $\nabla F(x\_\texttt{local})^{-T}$.

### 3.3.7 Die Iteratorenklassen

**1) Methoden von LevelIterator und LeafIterator**

`operator++()`: Preincrement Operator, liefert Iterator auf die nächste Entität,

`operator*()`: liefert eine Referenz auf eine Entity,

`operator->()`: liefert einen Pointer auf eine Entity,

`operator ==(ep)`, `operator !=(ep)`: Vergleich eines Iterators mit `this`, `ep` ist vom Typ `EntityPointer`.

**2) Methoden der IntersectionIterator Klasse**

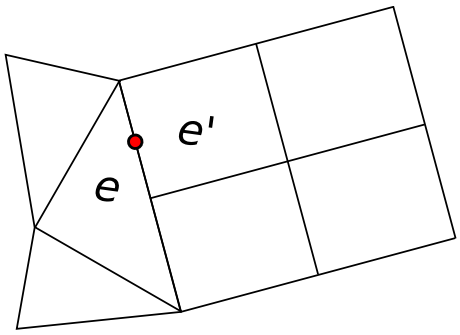



Abbildung 3.2: "inside" Entity $e$ und "outside" Entity $e'$

`operator++()`: Preincrement operator,

`boundary()`: `true`, falls Intersection am Rand (auch innerer Rand), sonst `false`,

`neighbor()`: `true`, falls Nachbar existiert,

`inside()`: liefert "inside"-Entity $e$,

`outside()`: liefert "outside"-Entity $e'$,

`intersectionSelfLocal()`: liefert ein Objekt vom Typ Geometry mit einer Abbildung von lokalen Koordinaten der Intersection auf lokale Koordinaten der "inside" Entity,

`intersectionNeighborLocal()`: liefert ein Objekt vom Typ `Geometry` mit einer Abbildung von lokalen Koordinaten der Intersection auf lokale Koordinaten der "outside"-Entity,

`intersectionGlobal()`: liefert Abbildung von lokalen Koordinaten auf globale Koordinaten,

`numberInSelf()`: liefert lokale Nummer der Codim-1-Subentität von "inside", die die Intersection enthält,

`numberInNeighbor()`: liefert lokale Nummer der Codim-1-Subentität von "outside", die die Intersection enthält,

`outerNormal(x_local)`: äußere Normale, gewichtet mit dem Volumen der Intersection,

`unitOuterNormal(x_local)`: äußere Einheitsnormale.

### 3.3.8 Die Referenzelement Klassen

**1) Methoden von ReferenceElement**

`size(codim)`: liefert die Anzahl der Subentitäten der Kodimension `codim` (entspricht $k_{\texttt{codim}}$ in Def. 2.3),

`size(i,codim,cc)`: liefert die Anzahl der Subentitäten der Kodimension `cc` von $\hat{e}_{\texttt{i}}^{\texttt{codim}}$,

`subEntity(i,c,ii,cc)`: liefert Nummer der `i`-ten Subentität der Kodimension `cc` von $\hat{e}_{\texttt{i}}^{\texttt{c}}$,

`position(i,c)`: liefert die Koordinate des Schwerpunkts der Entität $\hat{e}_{\texttt{i}}^{\texttt{c}}$,

`global(x_local,i,c)`: bildet lokale Koordinate `x_local` der Subentität $\hat{e}_{\texttt{i}}^{\texttt{c}}$ auf Koordinate des Referenzelementes ab,

`type(i,c)`: liefert den `GeometryType` von $\hat{e}_{\texttt{i}}^{\texttt{c}}$,

`volume()`: liefert Volumen des Referenzelements.

**2) Methode der Klasse ReferenceElementContainer**

`operator()(geometryType)`: liefert das Referenzelement vom Typ `GeometryType`.

### 3.3.9 Diskrete Funtionen in DUNE-FEM

**1) Abstrakte Definitionen**

Ein Funktionenraum $V$ ist eine Menge von Abbildungen von $D = \mathbb{K}_D^d$ in den Wertebereich $R = \mathbb{K}_R^n$, d.h.

$$V := \left\{ \mu : \mathbb{K}_D^d \to \mathbb{K}_R^n \right\}.$$

$\mathbb{K}_D, \mathbb{K}_R$ sind Körper (engl. field) und $d, n$ die Dimensionen von $D, R$. Ein diskreter Funktionenraum $V_h$ der Dimension $m$ ist ein Teilraum eines Funktionenraums mit der Eigenschaft, dass alle Funktionen in $V_h$ lokal auf den Entitäten $e \in \mathcal{E}_{\text{leaf}}^0$ des Gitters definiert sind.

Ist $\hat{e}$ das Referenzelement von $e \in \mathcal{E}_{\text{leaf}}^0$ und $F_e : \hat{e} \to e$ die Referenzabbildung, so definieren wir die Menge der lokalen Basisfunktionen $V_{\hat{e}}$ auf $\hat{e}$ durch:

$$V_{\hat{e}} := \left\{ \varphi_1, \ldots, \varphi_{\dim(V_e)} \right\},$$

wobei $V_e := \operatorname{span} \left\{ \varphi \circ F_e^{-1} \mid \varphi \in V_{\hat{e}} \right\}$.

$V_h$ ist dann gegeben durch:

$$V_h := \left\{ u_h \in V : u_h|_e := u_e := \sum_{\varphi \in V_{\hat{e}}} g_e(u_{e,\varphi}) \varphi \circ F_e^{-1} \; \forall e \in \mathcal{E}_{\text{leaf}}^0 \right\}.$$

Wir nennen $V_e$ den lokalen Funktionenraum, $u_e \in V_e$ eine lokale Funktion und $DOF_e := \{u_{e,\varphi} \mid \varphi \in V_{\hat{e}}\}$ die Menge der lokalen Freiheitsgrade.
Ist $DOF := \{u_i \mid i = 1, \ldots, m\}$ die Menge der globalen Freiheitsgrade einer Funktion $u_h \in V_h$, so liefert der diskrete Funktionenraum eine Abbildung

$$g_e : DOF_e \longrightarrow DOF.$$

**2) Überblick über die Schnittstellenklassen für diskrete Funktionen**

1) `FunctionSpace<DomainField,RangeField, dimD, dimR>`: repräsentiert den Funktionenraum $V$ mit $\mathbb{K}_D =$ `DomainField`, $\mathbb{K}_R =$ `RangeField`, $d =$ `dimD`, $n =$ `dimR`.

2) `Function<FunctionSpace>`: repräsentiert eine Funktion $u \in V$. Wichtigste Methode ist `evaluate(x, ret)`: `ret` $= u($`x`$)$, für $u \in V$.

3) `DiscreteFunctionSpace<FunctionSpaceTraits>`: repräsentiert $V_h$ und wird mit dem Funktionenraum $V =$ `FunctionSpace` (s.d. $V_h \subset V$), dem Gitter $\mathcal{T}_h =$ `Grid` und der Menge der lokalen Basisfunktionen $V_{\hat{e}} =$ `BaseFunctionSet` parametrisiert. Die Klasse liefert einen Iterator über Kodimension-0-Entitäten des Gitters ($\mathcal{E}^0_{\text{leaf}}$), sowie folgende Methoden:

   a) `mapToGlobal(e,nLocal)`: entspricht der Abbildung $g_{\text{e}}$ und bildet die lokale DOF-Nummer `nLocal` auf die zugehörige globale DOF-Nummer ab,

   b) `getBaseFunctionSet(e)`: gibt $V_{\hat{\text{e}}}$ zu `e` zurück.

4) `DiscreteFunction<Traits>`: Eine diskrete Funktion $u_h \in V_h$ wird durch den diskreten Funktionenraum $V_h =$ `DiscreteteFunctionSpace` und den Typ der lokalen Funktion $u_e \in V_e, V_e =$ LocalFunction parametrisiert. Die wichtigste Methode ist:
`localFunction(e, lf)`: gibt die lokale Funktion `lf` $= u_{\text{e}}$ zurück.

5) `LocalFunction<Traits>`: repräsentiert die lokale Funktion $u_e$. Die wichtigsten Methoden sind:

   a) `numDofs()`: gibt $\dim(V_e)$ zurück,

   b) `operator[](i)`: gibt den `i`-ten Freiheitsgrad zurück (entspricht $u_{e,\varphi_{\text{i}}}$),

   c) `evaluate(x,ret)`: berechnet `ret` $= u_e($`x`$)$,

   d) `jacobian(x,ret)`: berechnet `ret` $= \nabla u_e($`x`$)$,

Die Dokumentation zu den in DUNE und DUNE-FEM enthaltenen Klassen findet sich auf den Internetseiten der Projekte [4, 3].

## 3.4 Weiterführende Techniken

In diesem Abschnitt wollen wir uns mit weiterführenden Template-Techniken beschäftigen, die in DUNE und DUNE-FEM verwendet werden. Wir beginnen mit Techniken zur Realisierung eines statischen Polymorphismus.

### 3.4.1 Das "Engine"-Konzept

Wir erläutern das Konzept an einem Beispiel, bei dem eine Basisklasse zwei abgeleitete Klassen hat:

```
1  class ImplA
2  {
3  };
4
5  class ImplB
6  {
7  };
8
9  template<class T_engine>
10 class Base
11 {
12   private:
13     T_engine engine;
14 };
15
16 // Beispielroutine, die eine beliebige abgeleitete Klasse nimmt
17 template< class T_engine >
18 double foo( Base< T_engine >& A );
19
20 // Verwendung
21 Base< ImplA > A;
22 foo( A );
```

Problem: Bei dieser Technik müssen die Methoden aller abgeleiteten Klassen identisch sein. Dies kann mit dem folgenden Konstrukt umgangen werden.

### 3.4.2 Curiously recurring template pattern (CRTP)

Diese Methodik ermöglicht Vererbung ohne virtuelle Funktionen. Die prinzipielle Idee liegt darin, dass die abstrakte Basisklasse die abgeleitete Klasse als Template-Parameter erhält. Die Basisklasse kann dadurch einen Methode `impl()` zur Verfügung stellen die eine statische Typumwandlung einer Instanz auf ihren abgeleiteten Typ durchführt.
**Beispiel.**

```
1  template< typename Impl >
2  class Base
3  {
4    public:
5      Impl& impl()
6      {
7        return static_cast< Impl& >(* this );
8      }
9
10     void aMethod()
11     {
12       impl().aMethod();
```

```
13      }
14  };
15
16  class Implementation
17    : public Base< Implementation >
18  {
19    public:
20      void aMethod ()
21      {
22        // ...
23      }
24  };
```

### 3.4.3 CRTP und Traits

Die Anwendung von CRTP auf abgeleitete Klassen mit Template-Parametern stellt eine weitere Schwierigkeit dar, da rekursive Abhängigkeiten entstehen, falls in den Methoden der Basisklasse auf diese Template-Parameter zugegriffen werden muss.
**Beispiel** (rekursive Abhängigkeiten)**.**

```
1  template< typename Impl >
2  class Base
3  {
4    public:
5      typedef typename Impl::T_Type T_Type;
6      // ...
7  };
8
9  template< typename T >
10 class Implementation
11   : public Base< Implementation< T > >
12 {
13   public :
14     typedef T T_Type ;
15     // ...
16 };
```

Die Lösung ist eine Vordeklaration des Typs in einer Traitsklasse wie folgt:

```
1  template< typename I >   // Traitsklasse fuer die Basisklasse
2  struct Traits
3  {
4  };
5
6  template< typename Impl >
7  class Base
8  {
9    public :
```

```
10      typedef typename TypeInfo< Impl >::T_Type T_Type;
11      // ...
12  };
13  template< typename T >   // Vordeklaration
14  class Implementation;
15
16  template< typename T >   // Traitsklasse fuer die abgeleitete Klasse
17  struct Traits< Implementation< T > >
18  {
19    typedef T T_Type;
20  };
21
22  template< typename T >
23  class Implementation
24    : public Base< Implementation< T > >
25  {
26    // ...
27  };
```

### 3.4.4 Template Meta Programming

**Idee.**
Der Compiler soll Code generieren

1994 wurde entdeckt: Der C++-Compiler ist Turing-vollständig!

Als Beispiel geben wir an, wie die Berechnung der Fakultät mit Template Meta Programming implementiert werden kann:

```
1  #include <iostream>
2
3  template< int N >
4  struct Factorial
5  {
6    static const int value = Factorial< N - 1 >::value * N;
7  };
8
9  template<>
10 struct Factorial< 1 >
11 {
12   static const int value = 1;
13 };
14
15 int main()
16 {
17   std::cout << Factorial< 10 >::value << std::endl;
18 }
```

In Zeile drei bis sieben wird ein Template definiert. Zeile neun leitet eine Template Spezialisierung ein. Diese wird als Abbruchbedingung für die Berechnung der Fakultät benutzt. Dieses Template wird in Zeile 17 mit der Zahl 10 verwendet. Der Compiler erzeugt die Definition aus dem Template beim Compilieren. Das obige Programm berechnet die Fakultäten zur Compilierungszeit und benutzt die Werte, als wären es vordefinierte Konstanten.

### 3.4.5 Expression Templates

**Problem:** Paarweise Auswertung von Operatoren bei Verkettungen.
**Beispiel.**

```
1 Vector< double > a, b, c, d;
2 a = b + c + d;
```

Dies führt zu einem Code, der wie folgt aussieht:

```
1 double* _t1 = new double[N];
2 for( int i = 0; i < N; i++ )
3         _t1[i] = b[i] + c[i];
4 double* _t2 = new double[N];
5 for( int i = 0; i < N; i++)
6         _t2[i] = _t1[i] + d[i];
7 for( int i = 0; i < N; i++ )
8         a[i] = _t2[i];
9 delete[] _t1;
10 delete[] _t2;
```

Man hätte in diesem Fall jedoch gerne eine Realisierung der Form:

```
1 for( int i = 0; i < N; i++ )
2         a[i] = b[i] + c[i] + d[i];
```

Die Idee von Expression Templates ist es, das Überladen von Operatoren zu verwenden, um Abarbeitungsbäume aufzubauen. Demnach kann z.B. `b+c+d` in folgenden Typ verwandelt werden:

```
1     X< Array, plus, X< Array, plus, Array > >
```

Dies kann wir folgt geschehen:

```
1 struct plus{};  // Repraesentiert Addition
2 class Array{};  // Eine Vektorklasse
3
4 // X repraesentiert einen Knoten im Abarbeitungsbaum
5 template< typename LeftArg, typname Op, typename RightArg >
6 class X{};
7
8 // Ueberladung des ``+'' Operators
9 template< T >
10 X< T, plus, Array > operator+( T, Array )
11 {
12   return X< T, plus, Array >;
13 }
```

Damit erhalten wir für `Array b, c, d`

```
1 b + c + d = X< Array, plus, Array >() + d;
2 b + c + d = X< X< Array, plus, Array >, plus, Array >();
```

Dabei wird noch keine Rechenoperation ausgeführt. Erst wenn der =-Operator aufgerufen wird, wird die Berechnung angestoßen und kann optimiert umgesetzt werden, da dann der Abarbeitungsbaum vollständig bekannt ist.

In DUNE-FEM wird dieses Konzept verwendet, um diskrete Operatoren zu verketten und nur einen Gitterdurchlauf zu benötigen.

# KAPITEL 4

# Evolutionsgleichungen

In diesem Kapitel beschäftigen wir uns mit der Diskretisierung und Implementierung von Discontinuous-Galerkin-Verfahren für zeitabhängige Probleme. Dabei verfolgen wir den Ansatz, dass wir zunächst im Ort diskretisieren und für die Diskretisierung der resultierenden zeitabhängigen Systeme Diskretisierungsverfahren für Anfangswertprobleme anwenden. Dieser Ansatz wird "Method of Lines" genannt und im ersten Abschnitt anhand einfacher Ortsdiskretisierungen vorgestellt. Anschließend gehen wir auf Local Discontinuous-Galerkin-Verfahren zur Ortsdiskretisierung eine. Eine weitergehende Vorstellung von Zeitdiskretisierungsverfahren für Anfangswertprobleme und Aspekte der Programmierung in DUNE runden das Kapitel ab.

**Definition 4.1** (Evolutionsprobleme)**.**
Sei $\Omega \subset \mathbb{R}^n$ ein Gebiet mit Lipschitz-Rand und $L : V \to W$ ein Differentialoperator, der Elemente eines Funktionenraums $V$ auf Elemente eines Funktionenraums $W$ abbildet. Dann betrachten wir für eine Funktion $u(t,\cdot) \in V$ für fast alle $t \in (0,T)$ ein Anfangswertproblem der Form:

$$\partial_t u(t,x) + L[u(t,\cdot)](x) = 0 \qquad \text{für alle } (t,x) \in (0,T) \times \Omega. \tag{4.1}$$

Damit (4.1) wohlgestellt ist, benötigen wir geeignete Randbedingungen auf $(0,T) \times \partial\Omega$ und Anfangsbedingungen der Form

$$u(0,x) = u_0(x) \qquad \text{für } x \in \Omega.$$

**Beispiel 4.2.**
Je nach Definition des Ortsoperators $L$, sind in der Definition (4.1) zum Beispiel folgende Evolutionsprobleme enthalten:

$$\begin{aligned} \partial_t u - \Delta u &= f && \text{(Wärmeleitungsgleichung)} \\ \partial_t u + \nabla \cdot f(u) - \varepsilon \Delta u &= f && \text{(Konvektions-Diffusionsgleichung)} \\ \partial_t u - \nabla \cdot f(u) &= 0 && \text{(Erhaltungssgleichung)} \end{aligned}$$

Die ersten beiden partiellen Differentialgleichungen sind von parabolischem Typ, während die dritte Differentialgleichung hyperbolisch ist.

**Annahme.**
Wir gehen im Weiteren davon aus, dass das Problem (4.1) wohlgestellt ist.

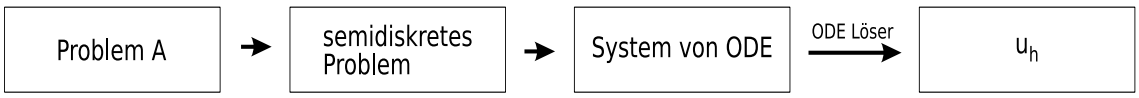



Abbildung 4.1: Method of Lines: Schritte der Diskretisierung

## 4.1 Method of lines

**Idee.**
Bezüglich einer Koordinate (hier die Zeit) haben wir es mit einem gewöhnlich Anfangswertproblem zu tun, für welche sehr gute numerische Methoden und Softwarepakete existieren.

Die Diskretisierung erfolgt daher in zwei Schritten (s. Abb. 4.1). Zunächst wird das Evolutionsproblem im Ort diskretisiert. Dies führt auf ein semidiskretes Problem (diskret im Ort, kontinuierlich in der Zeit), dass wir mit Hilfe eines diskreten approximierenden Ortsoperators $L_h : V_h \to W_h$ wie folgt schreiben können:

**Definition 4.3** (Semidiskretes Evolutionsproblem)**.**
Sei $\Omega \subset \mathbb{R}^n$ ein Gebiet mit Lipschitz-Rand und $L_h : V_h \to W_h$ eine diskrete Approximation des Differentialoperators $L$. Dann betrachten wir für eine diskrete Funktion $\overline{u}_h(t,\cdot) \in V_h$ für fast alle $t \in (0,T)$ ein diskretes Evolutionsproblem der Form:

$$\partial_t \overline{u}_h + L_h[\overline{u}_h(t,\cdot)](x) = 0 \tag{4.2}$$

Natürlich kommen auch hier geeignete Rand- und Anfangsbedingungen hinzu.

**Variationsformulierung der Evolutionsprobleme:**

In typischen Anwendungen ist $W$ ein Dualraum eines Sobolevraumes $\bar{W}$ (z.B. $W = H^{-1}(\Omega)$, falls $L = -\Delta$). Daher ist $L(u(t,\cdot))$ nur durch seine Aktion auf Elemente aus $\bar{W}$ definiert. Analoges gilt auch im diskreten Kontext.

Ohne Einschränkung gelte $V_h := \operatorname{span}\{\varphi_1, \ldots, \varphi_r\}$, $\varphi_i : \Omega \to \mathbb{R}$ und sei $\overline{u}_h(t,\cdot) \in V_h$, d.h. $\overline{u}_h = \sum_{j=1}^r \alpha_j(t)\varphi_j(x), \alpha_j : [0,T) \to \mathbb{R}$. Des Weiteren sei $\bar{W}_h$ mit $\dim(\bar{W}_h) = \dim(V_h)$ gegeben durch $\bar{W}_h := \operatorname{span}\{\psi_1, \ldots, \psi_r\}$, $\psi_i : \Omega \to \mathbb{R}$. Die Variationsformulierung von (4.2) ist dann gegeben durch:

$$\langle \partial_t \overline{u}_h, \psi_i \rangle + \langle L_h[\overline{u}_h(t,\cdot)](\cdot), \psi_i \rangle = 0, \qquad \forall\ i = 1, \ldots, r. \tag{4.3}$$

Mit $\partial_t \overline{u}_h = \sum_{j=1}^r \alpha_j'(t)\varphi_j(x)$ erhalten wir

$$\sum_{j=1}^r \alpha_j'(t)\langle \varphi_j, \psi_i \rangle = -\langle L_h[\overline{u}_h(t,\cdot)], \psi_i \rangle, \qquad \forall i = 1, \ldots, r.$$

Setzen wir voraus, dass im Diskreten $L_h[\overline{u}_h(t,\cdot)] \in V_h$ gilt, so können wir auch $L_h[\overline{u}_h(t,\cdot)]$ mit der Basis von $V_h$ wie folgt darstellen $L_h[\overline{u}_h(t,\cdot)] = \sum_{j=1}^r w_j(t)\varphi_j(x)$ und erhalten

$$\sum_{j=1}^r \alpha_j'(t)\langle \varphi_j, \psi_i \rangle = -\sum_{j=1}^r w_j(t)\langle \varphi_j, \psi_i \rangle.$$

Wir definieren die Massematrix $M \in \mathbb{R}^{r\times r}$ durch

$$M := (\langle \varphi_j, \psi_i\rangle)_{1\le i,j\le r}$$

sowie

$$\begin{aligned}\overline{U}(t) &:= (\alpha_j(t))_{1\le j\le r},\\ a(t,(\overline{U}(t)) &:= -(w_j(t))_{1\le j\le r}.\end{aligned}$$

Dann ist (4.3) äquivalent zu

$$M\ \overline{U}'(t) = M\ a(t,\overline{U}(t)),$$

und somit, falls $M$ regulär ist

$$\overline{U}'(t) = a(t,\overline{U}(t)). \tag{4.4}$$

Wir erhalten also ein System von gewöhnlichen Differentialgleichungen für den Koeffizientenvektor $\overline{U} = (\alpha_1,\dots,\alpha_r)$.

Zur Definition der diskreten Anfangswerte sei $\pi_h : V \to V_h$ ein Projektionsoperator.

Wir können dann die diskreten Anfangswerte definieren als $\overline{u}_h(0,x) = \pi_h[u_0](x)$ für $x \in \Omega$ bzw. in Variationsform

$$\langle \overline{u}_h(0,\cdot),\psi_i\rangle = \langle \pi_h[u_0](\cdot),\psi_i\rangle, 1\le i\le r.$$

Dies ist äquivalent zu

$$\sum_{j=1}^{r}\alpha_j(t)M_{ji} = \langle \pi_h[u_h],\psi_i\rangle$$

und wir können folglich setzen

$$\overline{U}(0) := M^{-1}\langle \pi_h[u_h],\psi_i\rangle.$$

In den Anwendungen gilt häufig $\bar{W}_h = V_h$, der Ansatz und Testraum sind dann identisch und die Definition der Massematrix stimmt mit der klassischen Definition überein.
Als Projektion $\pi_h$ kann beispielsweise die Lagrangeinterpolation oder auch die $L^2$-Projektion gewählt werden.

Im Folgenden wollen wir einfache Beispiele von Ortsdiskretisierungen anschauen, die in diesem Kontext behandelt werden können.
**Beispiel 4.4** (Finite Differenzen Diskretisierung in 1D)**.**
Sei $\Omega = [a,b], L[v] = -\Delta v$ und

$$L_h[v_h] = -\sum_{j=1}^{r}\frac{v_{j+1}-2v_j+v_{j-1}}{h^2}\varphi_j,$$

mit $v_h(x) = \sum_{j=1}^{r} v_j\varphi_j(x)$, $\varphi_j$ Basisfunktion mit $\varphi_j(x_i) = \delta_{ij}$ zu einer uniformen Zerlegung $\Delta_h$ von $[a,b]$. Also folgt

$$\begin{aligned}L_h[\overline{u}_h(t,\cdot)](x) &= -\sum_{j=1}^{r}\frac{\alpha_{j+1}(t)-2\alpha_j(t)+\alpha_{j-1}(t)}{h^2}\varphi_j(x)\\ &=: -\sum_{j=1}^{r} w_j(t)\varphi_j(x)\end{aligned}$$

$$\Rightarrow a(t,\alpha_1(t),\dots,\alpha_r(t)) = \left(\frac{\alpha_{j+1}(t)-2\alpha_j(t)+\alpha_{j-1}(t)}{h^2}\right)_{1\le j\le r}.$$

Damit erhalten wir das System von ODE's

$$\alpha_j'(t) = \frac{1}{h^2}(\alpha_{j+1}(t)-2\alpha_j(t)+\alpha_{j-1}(t))$$

für $j = 1,\dots,r$, wobei geeignete Randbedingungen zur Definition von $\alpha_0(t), \alpha_{r+1}(t)$ verwendet werden müssen.

**Beispiel 4.5** (Finite Elemente Diskretisierung in nD)**.**
Sei $\Omega \subset \mathbb{R}^n$ ein Gebiet mit Lipschitz-Rand und $L[u] := -\Delta u$. Für gegebenes $v_h \in V_h := \operatorname{span}(\varphi_1,\dots,\varphi_r)$ wird $w_h := L_h[v_h]$ definiert durch

$$a(v_h,\varphi_i) = (w_h,\varphi_i), \qquad 1 \le i \le r,$$

wobei $a(\cdot,\cdot)$ die Standard Bilinearform zum Laplace Operator ist, d.h.

$$\begin{aligned} a(v_h,\varphi_i) &= \int_\Omega \nabla v_h \cdot \nabla\varphi_i \\ &= \sum_{j=1}^{r} v_j \int_\Omega \nabla\varphi_j \nabla\varphi_i \\ &=: SV, \end{aligned}$$

wobei $S := \left(\int_\Omega \nabla\varphi_j\nabla\varphi_i\right)_{1\le i,j\le r}$ die Steifigkeitsmatrix und $V = (v_j)_j$ der (gegebene) Koeffizientenvektor von $v_h$ ist.

Analog folgt mit der Massematrix $M$ und dem Koeffizientenvektor $W$ von $w_h$

$$(w_h,\varphi_i) = MW.$$

Wir erhalten also folgende Darstellung für den Koeffizientenvektor von $w_h$

$$W = M^{-1}SV.$$

Betrachten wir nun die zeitabhängige parabolische Wärmeleitungsgleichung

$$\partial_t u - \Delta u = 0,$$

so erhalten wir für den Koeffizientenvektor $U(t)$, der semidiskreten Finite Elemente Approximation $\bar{u}_h(t,\cdot)$ das folgende System linearer gewöhnlicher Differentialgleichungen:

$$\begin{aligned} \Rightarrow U'(t) &= -W(t) \\ &= -M^{-1}SU(t) \\ &=: a(t,U(t)) \end{aligned}$$

**Mögliche Zeitdiskretisierungen**:
Explizites Eulerverfahren

$$U^{n+1} = U^n - \Delta t \cdot M^{-1}SU^n = (I - \Delta t M^{-1}S)U^n,$$

Implizites Eulerverfahren

$$U^{n+1} = U^n - \Delta t \cdot M^{-1} S U^{n+1},$$

Tatsächlich würde man in diesem Fall aber folgende Darstellung wählen:

$$(M + \Delta t S)\, U^{n+1} = M U^n.$$

Dies ist ein lineares Gleichungssystem für $U^{n+1}$.

**Beispiel 4.6** (Erhaltungsgleichungen in 1D)**.**
Sei $L[v] = \partial_x f(v)$ approximiert durch den diskreten Operator $L_h[v_h] = \frac{1}{h}(g(v_{i+1}, v_i) - g(v_i, v_{i-1}))$, wobei $v_h$ eine stückweise konstante Funktion mit Werten $v_i$ auf einer Gitterzelle $(x_{i-\frac{1}{2}}, x_{i+\frac{1}{2}})$, $x_i = ih$ sei. Die numerische Flussfunktion $g \in C^{0,1}(\mathbb{R} \times \mathbb{R})$ erfülle zudem die Konsistenzbedingung $g(u, u) = f(u)$.
Dann erhalten wir für den Koeffizientenvektor $U(t)$ der Semidiskretisierung $\bar{u}_h(t, \cdot)$ von

$$\partial_t \bar{u}_h + L_h[u_h(t, \cdot)] = 0$$

das folgende nicht-lineare System gewöhnlicher Differentialgleichungen:

$$U_i'(t) = -\frac{1}{h}(g(U_{i+1}(t), U_i(t)) - g(U_i(t), U_{i-1}(t))),\ 1 \le i \le r.$$

## 4.2 Zeitdiskretisierung

Die "Method of Lines" führt auf ein System von gewöhnlichen Differentialgleichungen. Wir betrachten daher in diesem Abschnitt allgemein Anfangswertprobleme der folgenden Art.

**Definition 4.7** (System von gewöhnlichen DGLn)**.**
Sei

$$L : \mathbb{R}^N \times \mathbb{R}^+ \longrightarrow \mathbb{R}^N,$$

ein "diskreter" Ortsoperator, so betrachten wir für $U : \mathbb{R}^+ \to \mathbb{R}^N$ das folgende Anfangswertproblem

$$\frac{d}{dt} U(t) = L(U(t), t) \qquad \text{für } t \in (t^n, t^{n+1}), \tag{4.5}$$

mit der Anfangsbedingung

$$U(t^n) = U^n. \tag{4.6}$$

Dabei sei $t^{n+1} := t^n + \Delta t,\ Deltat > 0$.

**Bemerkung.**
Bei der Diskretisierung von Evolutionsgleichungen entspricht $N = \dim(V_h)$ der Dimension des diskreten Lösungsraums für den Ort und der Vektor $U$ enthält die Freiheitsgrade von $V_h$ als Komponenteneinträge.

### 4.2.1 Explizite SSP-Runge-Kutta-Verfahren

In diesem Abschnitt wollen wir uns mit einer Klasse von expliziten Runge-Kutta Verfahren beschäftigen, die besondere Stabilitätseigenschaften besitzen, den sogenannten "Strongly Stability Preserving" Runge-Kutta-Verfahren.

Literatur: C.-W. Shu, *A survey of strong stability preserving high order time discretisations*, [20].

**Definition 4.8** (Explizites $m$-Schritt Runge-Kutta-Verfahren)**.**
Ein allgemeines $m$-Schritt Runge-Kutta-Verfahren zur approximativen Lösung von (4.5), (4.6) ist gegeben durch

$$\begin{aligned}\tilde{U}^l &:= U^n + \Delta t \sum_{k=1}^{l-1} a_{lk} L^k,\ l = 1, \dots m,\\ L^l &:= L(\tilde{U}^l, t^n + c_l \Delta t),\\ U^{n+1} &:= U^n + \Delta t \sum_{k=1}^{m} b_k L^k.\end{aligned}$$

Das Verfahren heißt konsistent, falls gilt

$$\sum_{k=1}^{m} b_k = 1,$$

$$c_l = \sum_{k=1}^{l-1} a_{lk} \in [0,1].$$

Das explizite Runge-Kutta-Verfahren ist durch Angabe der Werte $b_k, k = 1, \dots, m$ und $a_{lk}, l = 2, \dots, m, k < l$ definiert.

Die Werte können in Form eines *Butchertableaus* angegeben werden.

$$\begin{array}{c|cccc} c_1 & 0 & & & 0 \\ c_2 & a_{21} & \ddots & & \\ \vdots & \vdots & \ddots & \ddots & \\ c_m & a_{2m} & \dots & a_{m,m-1} & 0 \\ \hline & b_1 & \dots & b_{m-1} & b_m \end{array}$$

**Definition 4.9** (Explizites Eulerverfahren)**.**
Das Euler-Verfahren, definiert durch

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

ist *das* 1-Schritt Runge-Kutta-Verfahren der Konsistenzordnung 1.

**Definition 4.10** (SSP-Verfahren)**.**
Sei $\|\cdot\|$ eine (Semi-)Norm auf $\mathbb{R}^N$. Ein numerisches Verfahren zur Approximation von (4.5), (4.6) heißt stabil bzgl. $\|\cdot\|$, falls gilt

$$\|U^{n+1}\| \le \|U^n\|(1 + \mathcal{O}(\Delta t)).$$

Ein numerisches Verfahren heißt SSP-Verfahren, falls gilt:

$$\left(\begin{array}{c}\text{Das Eulerverfahren ist}\\ \text{stabil für } \Delta t \le \Delta t_0.\end{array}\right) \Longrightarrow \left(\begin{array}{c}\exists c > 0, \text{ so dass das Verfahren}\\ \text{stabil ist für } \Delta t \le c\Delta t_0.\end{array}\right)$$

Die Konstante $c > 0$ heißt CFL-Koeffizient des SSP-Verfahrens.

**Definition 4.11** (SSP-Runge-Kutta-Verfahren mit bis zu 4 Schritten)**.**

i) 1-Schritt-Verfahren der Ordnung 1:

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

ii) 2-Schritt-Verfahren der Ordnung 2:

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

iii) 3-Schritt-Verfahren der Ordnung 3:

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

iv) 4-Schritt-Verfahren der Ordnung 4:

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

**Bemerkung 4.12.**
Die expliziten Runge-Kutta-Verfahren liefern zu gegebenem $U^n \in R^N$ lediglich eine Approximation $U^{n+1}$ von $U(t^{n+1})$. Die Runge-Kutta Schritte $\tilde{U}_l$ können *nicht* als Approximation von $U(t)$ für $t \in [t^n, t^{n+1}]$ interpretiert werden.

**Frage**: Kann man mit Hilfe eines expliziten Runge-Kutta-Verfahrens eine Approximation $U_h : [t^n, t^{n+1}] \to \mathbb{R}^N$ definieren, so dass

$$U_h(t^n) = U^n,$$
$$U_h(t^{n+1}) = U^{n+1},$$

und

$$\|U - U_h\|_{L^\infty(t^n, t^{n+1})} \le c \cdot \Delta t^{\tilde{p}+1},$$

für ein $\tilde{p} \le m$?

**Satz 4.13** (Natural Continuous Extension)**.**
*Jedes $m$-Schritt Runge-Kutta-Verfahren der Ordnung $\tilde{m}$ besitzt eine "natürliche stetige Erweiterung" $u_h$ vom Polynomgrad $\tilde{p}$ mit*

$$\frac{m+1}{2} \le \tilde{p} \le \min\{m^*, \tilde{m}\},$$

*wobei $m^*$ die Anzahl der disjunkten Koeffizienten $c_l$ ist in dem Sinne, dass es $m$ Polynome*

$$b_l \in \mathbb{P}^{\tilde{p}}(0,1),\ l = 1, \dots, m$$

*gibt, so dass gilt*

$$\begin{aligned} U_h(t^n) &= U^n, \\ U_h(t^{n+1}) &= U^{n+1}, \\ U_h(t^n + s\Delta t) &:= U^n + \Delta t \sum_{k=1}^{n} b_k(s) L^k,\ 0 \le s \le 1. \end{aligned} \tag{4.7}$$

*Falls die exakte Lösung $U$ von (4.5), (4.6) glatt genug ist, gilt für $U_h$*

$$\| \frac{d^l}{dt^l}(U - U_h) \|_{L^\infty(t^n, t^{n+1})} \le c \cdot \Delta t^{\tilde{p}+1-l}.$$

*Beweis:* siehe [22]. □

**Definition 4.14** (NCE-SSP-RK-Verfahren mit bis zu 4 Schritten)**.**

i) 1 Schritt SSP-RK-Verfahren der Ordnung 1 ($m = \tilde{m} = \tilde{p} = 1$):

| | |
|---|---|
| 0 | |
| | $b_1(s) = s$ |

ii) 2-Schritt-SSP-RK-Verfahren der Ordnung 2 ($m = \tilde{m} = \tilde{p} = 2$):

| | | |
|---|---|---|
| 0 | | |
| 1 | 1 | |
| | $\frac{1}{2}$ | $\frac{1}{2}$ |

$$\begin{aligned} b_1(s) &= (b_1 - 1)s^2 + s, \\ b_2(s) &= b_2 s^2. \end{aligned}$$

iii) 3-Schritt-SSP-RK-Verfahren der Ordnung 3 ($m = \tilde{m} = 3, \tilde{p} = 2$):

| | | | |
|---|---|---|---|
| 0 | | | |
| 1 | 1 | | |
| $\frac{1}{2}$ | $\frac{1}{4}$ | $\frac{1}{4}$ | |
| | $\frac{1}{6}$ | $\frac{1}{6}$ | $\frac{2}{3}$ |

$$\begin{aligned} b_i(s) &= 3(2c_i - 1)b_i s^2 + 2(2 - 3c_i)b_i s, \\ &\quad \textit{für } i = 1, 2, 3. \end{aligned}$$

**Bemerkung**: Es gibt kein SSP-Runge-Kutta-Verfahren mit $m = \tilde{m} = 3$ und $\tilde{p} = 3$!

iv) 4-Schritt-SSP-RK-Verfahren der Ordnung 3 mit "stage reuse" ($m = 4, \tilde{m} = \tilde{p} = 3$):

| | | | | |
|---|---|---|---|---|
| 0 | | | | |
| 1 | 1 | | | |
| $\frac{1}{2}$ | $\frac{1}{4}$ | $\frac{1}{4}$ | | |
| 1 | $\frac{1}{6}$ | $\frac{1}{6}$ | $\frac{2}{3}$ | |
| | $\frac{1}{6}$ | $\frac{1}{6}$ | $\frac{2}{3}$ | 0 |

$$\begin{aligned} b_1(s) &= \tfrac{2}{3}s^3 - \tfrac{3}{2}s^2 + s, \\ b_2(s) &= -\tfrac{1}{3}s^3 + \tfrac{1}{2}s^2, \\ b_3(s) &= -\tfrac{4}{3}s^3 + 2s^2, \\ b_4(s) &= s^3 - s^2. \end{aligned}$$

v) 4-Schritt-SSP-RK-Verfahren der Ordnung 4 ($m = \tilde{m} = 4, \tilde{p} = 3$):

| | | | | |
|---|---|---|---|---|
| 0 | | | | |
| $\frac{1}{2}$ | $\frac{1}{2}$ | | | |
| 1 | $\frac{1}{2}$ | $\frac{1}{2}$ | | |
| $\frac{1}{2}$ | $\frac{1}{6}$ | $\frac{1}{6}$ | $\frac{1}{6}$ | |
| | $\frac{1}{6}$ | $\frac{1}{6}$ | $\frac{1}{6}$ | $\frac{1}{2}$ |

$$\begin{aligned} b_1(s) &= 2(1 - 4b_1)s^3 + 3(3b_1 - 1)s^2 + s, \\ b_i(s) &= 4(3c_i - 2)b_i s^3 + 3(3 - 4c_i)b_i s^2, \\ &\quad \textit{für } i = 2, 3, 4. \end{aligned}$$

**Bemerkung**: Es gibt kein SSP-Runge-Kutta-Verfahren mit $m = \tilde{m} = 4$ und $\tilde{p} = 4$!

**Bemerkung 4.15** (ODE für NCE-RK-Verfahren).

1) NCE-Verfahren 4. und 5. Ordnung findet man in [Zennaro '86, Owren-Zennaro '95]

2) Sei $U_h$ gegeben als Lösung eines $m$-Schritt NCE-RK-Verfahrens von (4.5), (4.6). Setzt man

$$L_h(U_h, t) := \sum_{k=1}^{m} b'_k \left( \frac{t - t^n}{\Delta t} \right) L^k,$$

so folgt aus (4.7)

$$\begin{aligned} \frac{d}{dt} U_h &= L_h(U_h, t), \\ U_h(t^n) &= U^n. \end{aligned}$$

Die NCE-Verfahren erlauben also eine kontinuierliche Interpretation der RK-Diskretisierungen.

### 4.2.2 Implizite Runge-Kutta-Verfahren und Kollokationsverfahren

Literatur: Strehmel/Weiner, *Numerik gewöhnlicher Differentialgleichungen* [21].

**Definition 4.16** (Implizites $m$-Schritt Runge-Kutta-Verfahren).
Ein implizites Runge-Kutta-Verfahren ist definiert durch ein Butchertableau

$$\begin{array}{c|c} c & a \\ \hline & b \end{array}$$

mit $c, b \in \mathbb{R}^m, a \in \mathbb{R}^{m \times m}$. Im Gegensatz zum expliziten Runge-Kutta-Verfahren kann für die Einträge $a_{l,k}$ gelten

$$a_{l,k} \neq 0, \qquad \forall l, k = 1, \ldots . m$$

Ein implizites Runge-Kutta-Verfahren führt auf ein System von (nicht linearen) Gleichungen mit $m \cdot N$ Unbekannten.

**Definition 4.17** (Kollokationsverfahren).
Eine Approximation $U_h \in \left[\mathbb{P}^m(t^n, t^{n+1})\right]^N$ heißt Lösung des Kollokationsverfahrens mit den Kollokationspunkten $t^n + c_i \Delta t, i = 1, \ldots, m$ für das Problem (4.5), (4.6), falls gilt

$$\begin{aligned} \frac{d}{dt} U_h(t^n + c_i \Delta t) &= L(U_h(t^n + c_i \Delta t), t^n + c_i \Delta t), i = 1, \ldots, m, \\ U_h(t^n) &= U^n. \end{aligned}$$

Wir setzen in diesem Fall $U^{n+1} = U_h(t^{n+1})$.

Ein Kollokationsverfahren ist durch die Angabe von $m$ und den Kollokationspunkten $c_i \in [0, 1], i = 1, \ldots, m$, definiert.

**Theorem 4.18** (Kollokationsverfahren sind stetige implizite RK-Verfahren (SIRK)).
*Ein Kollokationsverfahren mit $m$ Stufen und Kollokationspunkten $c_i, i = 1, \ldots, m$, ist äquivalent zu*

*einem RK-Verfahren definiert durch*

$$\begin{aligned} c_i, i &= 1, \dots, m, \\ a_{ij} &= \int_0^{c_i} l_j(\theta) d\theta, \\ b_j &= \int_0^1 l_j(\theta) d\theta, \end{aligned}$$

*wobei* $\{l_1, \dots, l_m\}$ *die Lagrange-Basis von* $\mathbb{P}^{m-1}(0,1)$ *zu den Lagrangepunkten* $c_1, \dots, c_m$ *ist, d.h.*

$$l_i(c_j) = \delta_{ij}, \qquad i, j = 1, \dots m.$$

*Beweis:* Siehe Deuflhard/Bornemann, *Scientific computing with ordinary differential equations*, [11]. □

**Theorem 4.19** (Ordnung von Kollokations-RK-Verfahren)**.**
*Ein implizites Runge-Kutta-Verfahren, das durch ein Kollikationsverfahren gegeben ist, hat bei genügend glattem L die Konsistenzordnung p, g.d.w. die Quadraturformel mit Punkten* $c_i, i = 1, \dots, m$, *und Gewichten* $b_i, i = 1, \dots, m$, *die Ordnung p hat.*
*Beweis:* Siehe [11] □

**Beispiel 4.20** (Gauß-Kollokationsverfahren)**.**

i) $m = 1, p = 2$

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

ii) $m = 2, p = 4$

$$\begin{array}{c|cc} \frac{1}{2} - \frac{1}{\sqrt{12}} & \frac{1}{4} & \frac{1}{4} - \frac{1}{\sqrt{12}} \\ \frac{1}{2} + \frac{1}{\sqrt{12}} & \frac{1}{4} + \frac{1}{\sqrt{12}} & \frac{1}{4} \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array}$$

### 4.2.3 IMEX-RK-Verfahren

Nachdem wir explizite und implizite Runge-Kutta-Verfahren eingeführt haben, wollen wir im Folgenden gemischt implizite-explizite Verfahren studieren, sogenannte IMEX-RK-Verfahren.

Literatur: Pareschi/Russo, *Implicit-explicit Runge-Kutta schemes and applications to hyperbolic systems with relaxation*, [17].
**Motivation** (Zeitdiskretisierung für Konvektions-Diffusionsprobleme)**.**
Bei der Ortsdiskretisierung von Konvektions-Diffusionsprobleme der Form

$$\partial_t u + \nabla \cdot f(u) - \varepsilon \Delta u = 0$$

erhält man ein System von partiellen Differentialgleichungen der Form

$$\bar{U}'(t) = -L_c[\bar{U}(t)] - L_d[\bar{U}(t)].$$

Dabei ist $L_c$ eine Diskretisierung des Konvektionsoperators und $L_d$ eine Diskretisierung des Diffusionsoperators. Während die Lipschitz-Konstante von $L_c$ proportional zu $\frac{1}{h}$ ist, ist die Lipschitz-Konstante von $L_d$ proportional zu $\frac{1}{h^2}$. Dies legt nahe, die Konvektion explizit und die Diffusion implizit zu behandeln.

**Definition 4.21** (IMEX-RK Verfahren)**.**
Wir betrachten Anfangswertprobleme der Form (4.5), (4.6), wobei die rechte Seite $L : \mathbb{R}^N \times \mathbb{R}^+ \longrightarrow \mathbb{R}^N$ gegeben ist durch

$$L = L_e + L_i.$$

Ein implizites-explizites Runge-Kutta-Verfahren (IMEX) ist von der Form

$$\begin{aligned}
\tilde{U}^l &:= U^n + \Delta t \left( \sum_{k=1}^{l-1} a_{lk}^e L_e^k + \sum_{k=1}^{m} a_{lk}^i L_i^k \right), \\
L_{e/i}^l &:= L_{e/i}(\tilde{U}^l, t^n + c_l^{e/i} \Delta t), \\
U^{n+1} &:= U^n + \Delta t \sum_{k=1}^{m} \left( b_k^e L_e^k + b_k^i L_i^k \right).
\end{aligned}$$

Das Verfahren ist charakterisiert durch die Angabe von zwei Butchertableaus:

$$\begin{array}{c|c} c^e & a^e \\ \hline & b^e \end{array} \qquad \begin{array}{c|c} c^i & a^i \\ \hline & b^i \end{array}$$

Dabei gehört das erste Tableau zu einem expliziten, das zweite zu einem impliziten Runge-Kutta-Verfahren.

**Bemerkung.**
Um zu garantieren, dass der Operator $L_e$ tatsächlich nur explizit ausgewertet wird, benötigen wir folgende notwendige Bedingung:

$$\begin{aligned}
&\text{a) } a_{lk}^e = 0, \quad \text{für } k \geq l, \\
&\text{b) } a_{lk}^i = 0, \quad \text{für } k > l.
\end{aligned}$$

Ein Runge-Kutta-Verfahren mit der Eigenschaft a) ist ein explizites Runge-Kutta-Verfahren. Ein Runge-Kutta-Verfahren mit der Eigenschaft b) heißt diagonal-implizites Runge-Kutta-Verfahren (DIRK-Verfahren).

Gilt für ein DIRK-Verfahren

$$a_{ii} = \gamma, \qquad \forall i = 1, \ldots, N,$$

so heißt das Verfahren *einfach diagonal-implizites* Verfahren oder *singly diagonally implicit Runge-Kutta*-Verfahren (SDIRK).

**Algorithmus 4.22** (IMEX-DIRK-Verfahren)**.**

Für $l = 1, \ldots, m$ {
    A) Berechne: 1) falls (l-1 $\geq$ 1):
    $L_e^{l-1} = L_e(\tilde{U}^{l-1}, t^n + c_{l-1}^e \Delta t)$
    2) $\hat{U}^l = U^n + \Delta t \sum_{k=1}^{l-1} a_{lk}^e L_e^k$

    B) Berechne: 1) falls (l-1 $\geq$ 1):
    $L_i^{l-1} = L_i(\tilde{U}^{l-1}, t^n + c_{l-1}^i \Delta t)$
    2) Berechne $\tilde{U}^l$ als Lösung von
    $\tilde{U}^l = \hat{U}^l + \Delta t \sum_{k=1}^{l-1} a_{lk}^i L_i^k + \Delta t a_{ll}^i L_i(\tilde{U}^l, t^n + c_l^i \Delta t)$
}

Setze $U^{n+1} = U^n + \Delta t \sum_{k=1}^{m} (b_k^e L_e^k + b_k^i L_i^k)$.

**Bemerkung.**
Durch die Verwendung eines DIRK-Verfahrens für den impliziten Operator muss lediglich in Teil B 2) ein implizites Problem gelöst werden.

**Beispiel 4.23** (SSP-DIRK-Verfahren)**.**
Wir geben in diesem Beispiel IMEX-Verfahren an, die sich aus expliziten SSP-RK-Verfahren und impliziten DIRK-Verfahren zusammensetzen.
Wir bezeichnen diese Verfahren mit SSP-DIRK $(m_1, m_2, p)$ mit folgender Notation:

$m_1$ : Anzahl der Stufen des SSP-RK-Verfahrens,
$m_2$: Anzahl der Stufen des DIRK-Verfahrens,
$p$ : Konsistenzordnung des Gesamtverfahrens.

1) SSP-DIRK (1,1,1):

| | |
|---|---|
| 0 | 0 |
| | 1 |

| | |
|---|---|
| 1 | 1 |
| | 1 |

2) SSP-DIRK (2,2,2):

| | | |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 1 | 0 |
| | $\frac{1}{2}$ | $\frac{1}{2}$ |

| | | |
|---|---|---|
| $\gamma$ | $\gamma$ | 0 |
| 1-$\gamma$ | 1-2$\gamma$ | $\gamma$ |
| | $\frac{1}{2}$ | $\frac{1}{2}$ |

mit $\gamma := 1 - \frac{1}{\sqrt{2}}$.

3) SSP-DIRK (2,3,2):

| | | | |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| | 0 | $\frac{1}{2}$ | $\frac{1}{2}$ |

| | | | |
|---|---|---|---|
| $\frac{1}{2}$ | $\frac{1}{2}$ | | |
| 0 | $-\frac{1}{2}$ | $\frac{1}{2}$ | |
| 1 | 0 | $\frac{1}{2}$ | $\frac{1}{2}$ |
| | 0 | $\frac{1}{2}$ | $\frac{1}{2}$ |

4) SSP-DIRK (3,3,2):

| | | | |
|---|---|---|---|
| 0 | 0 | | |
| $\frac{1}{2}$ | $\frac{1}{2}$ | 0 | |
| 1 | $\frac{1}{2}$ | $\frac{1}{2}$ | 0 |
| | $\frac{1}{3}$ | $\frac{1}{3}$ | $\frac{1}{3}$ |

| | | | |
|---|---|---|---|
| $\frac{1}{4}$ | $\frac{1}{4}$ | | |
| $\frac{1}{4}$ | 0 | $\frac{1}{4}$ | |
| 1 | $\frac{1}{3}$ | $\frac{1}{3}$ | $\frac{1}{3}$ |
| | $\frac{1}{3}$ | $\frac{1}{3}$ | $\frac{1}{3}$ |

5) SSP-DIRK (3,4,3):

| | | | | |
|---|---|---|---|---|
| 0 | 0 | | | |
| 0 | 0 | 0 | | |
| 1 | 0 | 1 | 0 | |
| $\frac{1}{2}$ | 0 | $\frac{1}{4}$ | $\frac{1}{4}$ | 0 |
| | 0 | $\frac{1}{6}$ | $\frac{1}{6}$ | $\frac{2}{3}$ |

| | | | | |
|---|---|---|---|---|
| $\alpha$ | $\alpha$ | | | |
| 0 | $-\alpha$ | $\alpha$ | | |
| 1 | 0 | $1-\alpha$ | $\alpha$ | |
| $\frac{1}{2}$ | $\beta$ | $\eta$ | $\frac{1}{2}-\alpha-\beta-\eta$ | $\alpha$ |
| | 0 | $\frac{1}{6}$ | $\frac{1}{6}$ | $\frac{2}{3}$ |

mit $\alpha := 0.24169426078821$,
$\beta := 0.06042356519705$,
$\eta := 0.12915286960590$.

**Algorithmus 4.24** (Lösung von IMEX-Verfahren mit dem Newton-Algorithmus)**.**
Wir betrachten die implizite Gleichung aus dem Algorithmus der IMEX-DIRK-Verfahren

$$\tilde{U}^l = \hat{U}^l + \Delta t \sum_{k=1}^{l-1} a_{lk}^i L_i^k + \Delta t a_{ll}^i L_i(\tilde{U}^l, t^n + c_l^i \Delta t)$$

und schreiben dies in der Form

$$F(\tilde{U}^l) = 0.$$

Dieses nichtlineare Gleichungssystem wollen wir mit Hilfe des Newton-Verfahrens lösen:

$\tilde{U}^{l,0} := U^n$
Für $i = 1, \ldots$:
1) $DF(\tilde{U}^{l,i})\ V = -F(\tilde{U}^{l,i})$,
2) $\tilde{U}^{l,i+1} = \tilde{U}^{l,i} + V$.

Dabei ist 1) ein lineares Gleichungssystem für den Defekt $V$, wobei $DF(\tilde{U}^{l,i})$ die Jacobi-Matrix von $F$ in $\tilde{U}^{l,i}$ bezeichnet. 1) ist also ein lineares Gleichungssystem, das z.B. mit einem iterativen Löser gelöst werden kann.

**Problem**: Das Verfahren benötigt in jeder Iteration die Jacobi-Matrix $DF(\tilde{U}^{l,i})$.

Mögliche Vereinfachungen sind

1) Ersetze überall $DF(\tilde{U}^{l,i})$ durch $DF(U^n)$.

2) Approximiere $DF(U^n)$ durch Differenzenquotienten

$$\begin{aligned} DF(U^n)_{ij} &= \frac{\partial F_i(U^n)}{\partial U_j} \\ &\approx \frac{F_i(U^n + \delta e_j) - F_i(U^n)}{\delta}. \end{aligned}$$

3) In iterativen Verfahren zur Lösung von 1) benötigt man nur Produkte der Form

$$DF(\tilde{U}^{l,i}) \cdot V.$$

Dies entspricht einer Richtungsableitung von $F$ in $\tilde{U}^{l,i}$ in Richtung $V$ und kann Approximiert werden durch

$$\frac{F_i(\tilde{U}^{l,i} + \delta V) - F_i(\tilde{U}^{l,i})}{\delta}.$$

Auf diese Weise erhält man ein Matrix-freies Lösungsverfahren, dass lediglich auf Auswertungen des nichtlinearen Operators beruht.

**Alternative: (Nichtlineares SOR-Verfahren)**

**Idee.**
Löse iterativ zeilenweise nichtlineare Probleme

Für $i = 1, \ldots$:
Für $k = 1, \ldots, N$:
$F_k(\tilde{U}_1^{l,i+1}, \ldots, \tilde{U}_{k-1}^{l,i+1}, U_k, \tilde{U}_{k+1}^{l,i}, \ldots, \tilde{U}_N^{l,i}) = 0$
$\tilde{U}_k^{l,i+1} := \tilde{U}_k^{l,i} + \omega(U_k - \tilde{U}_k^{l,i})$

Zur Lösung der skalaren nichtlinearen Teilprobleme kann z.B. ein Sekanten-Verfahren verwendet werden.

## 4.3 Semidiskrete Discontinuous-Galerkin-Verfahren

In diesem Abschnitt beschäftigen wir uns mit semidiskreten Discontinuous-Galerkin-Verfahren zur Approximation von Evolutionsgleichungen. Wir starten mit einer Motivation anhand eines linearen Transportproblems, betrachten dann nichtlineare Erhaltungsgleichungen und schließlich die Wärmeleitungsgleichung. Ein Ausblick auf Konvektions-Diffusionsprobleme und Systeme von Evolutionsgleichungen schließen den Abschnitt ab.

### 4.3.1 Motivation: Lineare Transportprobleme

In diesem Abschnitt betrachten wir semidiskrete Discontinuous-Galerkin-Verfahren für lineare Transportprobleme. An diesem einfachen Beispiel wollen wir spezielle Eigenschaften von Discontinuous-Galerkin-Verfahren untersuchen, die in diesem Zusammenhang von besonderem Interesse sind. Insbesondere werden wir dabei auf den Zusammenhang von Discontinuous-Galerkin-Verfahren und Finite-Volumen-Verfahren eingehen. Die Darstellung folgt einem einführenden Artikel von Cockburn [9].

**Definition 4.25** (Lineares Transportproblem)**.**
Sei $b : \mathbb{R}^d \times \mathbb{R}^+ \to \mathbb{R}^d$ ein gegebenes Geschwindigkeitsfeld und $u_0 : \mathbb{R}^d \to \mathbb{R}$ gegebene Anfangsdaten. Dann heißt $u : \mathbb{R}^d \times \mathbb{R}^+ \to \mathbb{R}$ Lösung des linearen Transportproblems, falls $u$ hinreichend glatt ist und folgendes Cauchyproblem erfüllt

$$\partial_t u + \nabla \cdot (bu) = 0 \qquad \text{in } \mathbb{R}^d \times (0,T), \tag{4.8a}$$

$$u(\cdot,t) = u_0 \qquad \text{auf } \mathbb{R}^d. \tag{4.8b}$$

Analog zu Kapitel 2 sei $\mathcal{T}_h$ ein Gitter auf $\mathbb{R}^d$. Wir verwenden den diskreten Raum

$$V_h = V_h^p := \left\{ v \in L^2(\Omega) \mid v|_{e^0} \in \mathbb{P}_p(e^0) \forall e^0 \in \mathcal{E}^0_{\text{leaf}} \right\}$$

zur Definition einer Discontinuous-Galerkin-Diskretisierung.

**Definition 4.26** (DG-Verfahren für Transportprobleme)**.**
Sei $\mathcal{T}_h$ ein Gitter auf $\mathbb{R}^d$ und $\hat{\sigma} = (\hat{\sigma}_e)_{e \in \mathcal{E}^0_{\text{leaf}}}$ ein gegebener erhaltender vektowertiger numerischer Fluss. Dann heißt $u_h(\cdot,t) \in V_h$ semidiskrete Discontinuous Galerkin Approximation mit numerischen Fluss $\hat{\sigma}$ des Transportproblems (4.8a), (4.8b), falls $\forall e \in \mathcal{E}^0_{\text{leaf}}, \forall v_h \in V_h$ gilt

$$\begin{aligned} \int_e \partial_t u_h(\cdot,t) v_h - \int_e b u_h(\cdot,t) \cdot \nabla v_h + \int_{\partial e} \hat{\sigma}_e(u_h(\cdot,t)) \cdot n v_h &= 0, \\ u_h(\cdot,0) &= \pi_h(u_0). \end{aligned}$$

Wir wollen uns nun mit der Frage beschäftigen, wie der numerische Fluss $\hat{\sigma}$ gewählt werden muss, damit das resultierende DG-Verfahren stabil ist. Dazu wählen wir $v_h = u_h$ in der Definition des Verfahrens und erhalten nach Addition über alle Entitäten

$$\begin{aligned} \int_{\mathbb{R}^d} \partial_t u_h(\cdot,t) u_h(\cdot,t) - \int_{\mathbb{R}^d} b u_h(\cdot,t) \cdot \nabla u_h(\cdot,t) &= 0, \\ \frac{d}{dt} \int_{\mathbb{R}^d} \frac{1}{2} |u_h(\cdot,t)|^2 + \frac{1}{2} \int_{\mathbb{R}^d} \nabla \cdot b |u_h(\cdot,t)|^2 + \Theta_h(t) &= 0. \end{aligned} \tag{4.9}$$

Der Term $\Theta_h(t)$ entsteht bei der lokalen partiellen Integration des zweiten Terms

$$\begin{aligned}\int_e bu_h(\cdot,t)\cdot\nabla u_h(\cdot,t) &= -\int_e \nabla\cdot(bu_h(\cdot,t))u_h(\cdot,t)+\int_{\partial e} b\cdot n|u_h(\cdot,t)|^2\\ &= -\int_e \nabla\cdot b|u_h(\cdot,t)|^2-\int_e bu_h(\cdot,t)\cdot\nabla u_h(\cdot,t)\\ &\quad +\int_{\partial e} b\cdot n|u_h(\cdot,t)|^2\end{aligned}$$

und ist wie folgt definiert

$$\Theta_h(t)=\sum_{e\in\mathcal{E}^0_{\text{leaf}}}\Big(-\frac{1}{2}\int_{\partial e} b\cdot n|u_h(\cdot,t)|^2+\int_{\partial e}\hat{\sigma}(u_h)(\cdot,t)\cdot nu_h(\cdot,t)\Big).$$

Durch Integration über $0,T$ erhalten wir aus (4.9) weiter

$$\begin{aligned}\frac{1}{2}\int_{\mathbb{R}^d}|u_h(\cdot,T)|^2 \quad &+\frac{1}{2}\int_0^T\int_{\mathbb{R}^d}\nabla\cdot b|u_h(\cdot,t)|^2+\int_0^T\Theta_h(t)\\ &=\frac{1}{2}\int_{\mathbb{R}^d}|u_h(\cdot,0)|^2. \qquad (4.10)\end{aligned}$$

Wir wollen nun voraussetzen, dass für das Geschwindigkeitsfeld gilt $\nabla\cdot b\geq 0$. Dann folgt aus dieser Gleichung ein Stabilitätsresultat für das diskrete Verfahren, falls der numerische Fluss so gewählt werden kann, dass gilt $\Theta(t)\geq 0$. Dies wollen wir nun genauer untersuchen. Mit der Notation aus Kapitel 2 ($[\cdot]$ bezeichnet den Kantensprung) folgt

$$\begin{aligned}\Theta_h(t) &= \sum_{e^1\in\Gamma}\int_{e^1}[\hat{\sigma}(u_h)(\cdot,t)\cdot nu_h(\cdot,t)-\frac{1}{2}\,b\cdot nu_h^2(\cdot,t)]\\ &= \sum_{e^1\in\Gamma}\int_{e^1}\Big(\hat{\sigma}(u_h)(\cdot,t)\cdot n[u_h(\cdot,t)]-\frac{1}{2}\,b\cdot n\,[u_h^2(\cdot,t)]\Big)\\ &= \sum_{e^1\in\Gamma}\int_{e^1}\Big(\hat{\sigma}(u_h)(\cdot,t)-\frac{1}{2}\,b[u_h(\cdot,t)]\Big)\cdot n\,[u_h(\cdot,t)]\end{aligned}$$

Durch diese Beobachtung erhalten wir folgendes Lemma

**Lemma 4.27** (Stabile DG-Verfahren für Transportprobleme)**.**
*Wir betrachten lineare Transportprobleme der Form* (4.8a), (4.8b) *mit* $\nabla\cdot b\geq 0$. *Dann ist das DG-Verfahren aus Definition 4.26* $L^2$*-stabil, falls für den numerischen Fluss* $\hat{\sigma}$ *gilt*

$$\hat{\sigma}(u_h)(\cdot,t)=b\{u_h(\cdot,t)\}+C[u_h]n$$

*mit einer positiv semi-definiten Matrix* $C\in\mathbb{R}^{d\times d}$.

*Beweis:* Mit der Voraussetzung an den numerischen Fluss $\hat{\sigma}$ folgt

$$\Theta_h(t)=\sum_{e^1\in\Gamma}\int_{e^1}Cn\cdot n[u_h]^2.$$

Damit folgt die $L^2$-Stabilität aus der positiv semi Definitheit von $C$ und der Gleichung (4.10). □

**Beispiel 4.28** (Stabile numerische Flussfunktionen)**.**

1) **Upwind-Fluss:**
Wählt man $C = \frac{1}{2}\,|b \cdot n| Id$, so erhält man den klassischen Upwind-Fluss

$$\hat{\sigma}(u_h)(x,t) = b \lim_{\varepsilon \searrow 0} u_h(x - \varepsilon b).$$

2) **Lokaler Lax-Friedrich-Fluss:**
Wählt man $C = \frac{1}{2}\,|b| Id$, so erhält man den lokalen Lax-Friedrich Fluss

$$\hat{\sigma}(u_h) = b\{u_h\} + \frac{1}{2}|b|\,[u_h]\,n$$

**Bemerkung 4.29** (Vergleich mit Finite-Volumen-Verfahren)**.**

1) Wählen wir $V_h = V_h^0$, so ist das DG-Verfahren äquivalent zu einem Finite-Volumen-Verfahren. DG-Verfahren mit $V_h^p, p > 0$ können daher auch als Verallgemeinerung von Finite-Volumen-Verfahren interpretiert werden. Im Gegensatz zu klassischen Finite-Volumen-Verfahren erzielen DG-Verfahren jedoch auf eine sehr einfache Weise höherer Konvergenzordnungen.

2) Die lokale Massenerhaltung von Finite-Volumen-Verfahren, bleibt bei DG-Verfahren erhalten (Testen mit stückweise Konstanten Funktionen).

3) Der Term $\Theta_h(t) = \sum_{e^1 \in \Gamma} \int_{e^1} Cn \cdot n[u_h]^2$ kann als numerische Viskosität des DG-Verfahrens interpretiert werden.

4) Das lokale Residuum $\partial_t u_h + \nabla \cdot (bu_h)$ ist stark korrelliert mit den Kantensprüngen. Aus der Definition des DG-Verfahrens erhalten wir

$$\int_e R(u_h)v_h := \int_e (\partial_t u_h + \nabla \cdot (bu_h))v_h = \int_e (bu_h \cdot nv_h - \hat{\sigma}(u_h) \cdot nv_h).$$

Für den Upwind-Fluss erhalten wir hieraus

$$\int_e R(u_h)v_h = \int_{\partial e^-} b[u_h] \cdot nv_h.$$

Dabei bezeichnet $\partial e^- = \{x \in \partial e \mid\ b(x) \cdot n(x) \leq 0\}$ den Einflussrand von $e$. Dies zeigt, dass das lokale Residuum in diesem Fall linear von dem Sprung von $u_h$ auf dem Einflussrand abhängt. Dies zeigt auch, dass wir erwarten können, dass die Sprünge mit wachsendem Polynomgrad $p$ kleiner werden. Insbesondere hängt also auch die numerische Viskosität vom verwendeten Polynomgrad ab!

### 4.3.2 DG-Verfahren für nichtlineare Erhaltungsgleichungen

Als Erweiterung von linearen Transportproblemen, betrachten wir in diesem Abschnitt semidiskrete Discontinuous-Galerkin-Verfahren für nichtlineare skalare Erhaltungsgleichungen.

**Definition 4.30** (Nichtlineare Erhaltungsgleichungen)**.**
Seien $f \in C^1(\mathbb{R}; \mathbb{R}^d)$, $u_0 : \mathbb{R}^d \to \mathbb{R}$ gegeben. Dann heißt $u : \mathbb{R}^d \times \mathbb{R}^+ \to \mathbb{R}$ Lösung der nichtlinearen Erhaltungsgleichung zu der Flussfunktion $f$ und den Anfangsdaten $u_0$, falls $u$ hinreichend glatt ist und folgendes Cauchyproblem erfüllt

$$\partial_t u + \nabla \cdot f(u) = 0 \qquad \text{in } \mathbb{R}^d \times (0,T), \tag{4.11a}$$

$$u(\cdot, 0) = u_0 \qquad \text{auf } \mathbb{R}^d. \tag{4.11b}$$

Wir werden uns später speziell mit Entropielösungen solcher Erhaltungsprobleme beschäftigen, aber zunächst unter der Annahme genügender Regularität eine Discontinuous-Galerkin-Diskretisierung motivieren.

**Definition 4.31** (DG-Verfahren für Erhaltungsgleichungen)**.**
Sei $\mathcal{T}_h$ ein Gitter auf $\mathbb{R}^d$ und $\hat{\sigma} = (\hat{\sigma}_e)_{e\in\mathcal{E}^0_{\text{leaf}}}$ ein gegebener erhaltender vektowertiger numerischer Fluss. Dann heißt $u_h(\cdot,t) \in V_h$ semidiskrete Discontinuous-Galerkin-Approximation mit numerischen Fluss $\hat{\sigma}$ der Erhaltungsgleichung (4.11a), (4.11b), falls $\forall e \in \mathcal{E}^0_{\text{leaf}}, \forall v_h \in V_h$ gilt

$$\begin{aligned}\int_e \partial_t u_h(\cdot,t)v_h - \int_e f(u_h(\cdot,t))\cdot\nabla v_h + \int_{\partial e}\hat{\sigma}_e(u_h)\cdot n v_h &= 0,\\ u_h(\cdot,0) &= \pi_h(u_0).\end{aligned}$$

Analog zum Fall der linearen Transportprobleme, ist auch diese Diskretisierung lokal erhaltend und kann als Verallgemeinerung von Finite-Volumen-Verfahren gesehen werden. Wählen wir $V_h = V_h^p$, so ist das Verfahren formal von der Ordnung $p + \frac{1}{2}$ (siehe Zhang, Shu [23]). Typische numerische Flussfunktionen können analog zu Finite-Volumen-Verfahren gewählt werden.

**Beispiel 4.32** (Stabile numerische Flussfunktionen)**.**
Die numerische Flussfunktion $\hat{\sigma}$ kann so gewählt werden, dass sie nur von den Spuren der Funktion $u_h$ auf den beiden benachbarten Entitäten abhängt. Wir setzten also

$$\hat{\sigma}(u_h)|_{e^1}(\cdot,t)\cdot n := g_n(u_h^-(\cdot,t), u_h^+(\cdot,t))$$

mit einer numerischen Flussfunktion $g \in C^{0,1}(\mathbb{R}\times\mathbb{R})$. Analog zur Definition bei Finite-Volumen-Verfahren können wir beispielsweise folgende numerische Flussfunktionen verwenden:

1) **Godunov-Fluss**

$$g_n(v,w) = \begin{cases} \min_{v\le u\le w} f(u)\cdot n, & \text{falls } v \le w,\\ \max_{w\le u\le v} f(u)\cdot n, & \text{sonst.}\end{cases},$$

2) **Engquist-Osher-Fluss**

$$g_n(v,w) = \int_0^w \min\{f'(s)\cdot n, 0\}ds + \int_0^v \max\{f'(s)\cdot n, 0\}ds + f(0),$$

3) **Lax-Friedrich-Fluss**

$$g_n(v,w) = \frac{1}{2}\left((f(v)+f(w))\cdot n - C(w-v)\right), \quad \text{mit } C := \max_s |f'(s)|.$$

Für Finite-Volumen-Verfahren haben wir gesehen, dass monotone numerische Flüsse zu stabilen Verfahren führen, falls die sogenannte CFL-Bedingung erfüllt ist (siehe Skript zur Vorlesung *Numerik partieller Differentialgleichungen II*). Dabei nennen wir einen numerischen Fluss $g_n \in C^{0,1}(\mathbb{R}\times\mathbb{R})$ monoton, falls er im ersten Argument monoton wachsend und im zweiten Argument monoton fallend ist.

Wir werden nun dieses Resultat erneut betrachten und feststellen, dass dies im Kontext von Discontinuous-Galerkin-Verfahren nur für stückweise konstante Ansatzfunktionen gilt. Dazu betrachten wir das Discontinuous-Galerkin-Verfahren aus Definition 4.31 in einer Raumdimension auf einem uniformen

Gitter mit Entitäten $e := I_j := (x_{j-\frac{1}{2}}, x_{j+\frac{1}{2}}), x_j := hj$. Wählen wir dann als Testfunktion $v_h = 1$, so erhalten wir

$$\int_{I_j} \partial_t u_h + \Big(g(u^-_{j+\frac{1}{2}}, u^+_{j+\frac{1}{2}}) - g(u^-_{j-\frac{1}{2}}, u^+_{j-\frac{1}{2}})\Big) = 0.$$

Wir wollen nun im Kontext von SSP-Runge-Kutta-Verfahren die Stabilität eines Euler-Schrittes analysieren, d.h. wir betrachten den Übergang

$$u_h \to w_h = u_h + \Delta t L_h(u_h).$$

genauer erhalten wir für $v_h = 1$:

$$\int_{I_j} w_h = \int_{I_j} u_h - \Delta t\Big(g(u^-_{j+\frac{1}{2}}, u^+_{j+\frac{1}{2}}) - g(u^-_{j-\frac{1}{2}}, u^+_{j-\frac{1}{2}})\Big).$$

Dabei bezeichnet $u^-_{j+\frac{1}{2}}$ den Grenzwert von links in $x_{j+\frac{1}{2}}$ und $u^+_{j+\frac{1}{2}}$ den Grenzwert von rechts in $x_{j+\frac{1}{2}}$. Bezeichnen wir weiter mit $\bar{v}_j := \frac{1}{h}\int_{I_j} v_h$ den Mittelwert auf einer Zelle $I_j$, so erhalten wir

$$\bar{w}_j = \bar{u}_j - \frac{\Delta t}{h}\Big(g(u^-_{j+\frac{1}{2}}, u^+_{j+\frac{1}{2}}) - g(u^-_{j-\frac{1}{2}}, u^+_{j-\frac{1}{2}})\Big). \tag{4.12}$$

Jetzt können wir analog zur Analyse von Finite-Volumen-Verfahren vorgehen und erhalten folgendes Lemma:

**Lemma 4.33** (TV-Stabilität für das DG-Verfahren)**.**
*Sei $u_h \in V_h$ mit kompaktem Träger gegeben und $w_h \in V_h$ definiert durch Gleichung* (4.12).
*Wir definieren eine diskrete TV-Norm durch*

$$|\bar{u}_h|_{TV} := \sum_{j\in\mathbb{Z}} |\bar{u}_{j+1} - \bar{u}_j|.$$

*Ein Euler-Schritt für die DG-Diskretisierung 4.31 ist Total-Variation-verringernd (d.h. es gilt $|\bar{w}_h|_{TV} \leq |\bar{u}_h|_{TV}$), falls für*

$$\begin{aligned} c_{j+\frac{1}{2}} &:= -\frac{\Delta t}{h}\frac{g(u^-_{j+\frac{1}{2}}, u^+_{j+\frac{1}{2}}) - f(\bar{u}_j)}{\bar{u}_{j+1} - \bar{u}_j}, \\ d_{j-\frac{1}{2}} &:= -\frac{\Delta t}{h}\frac{g(u^-_{j-\frac{1}{2}}, u^+_{j-\frac{1}{2}}) - f(\bar{u}_j)}{\bar{u}_j - \bar{u}_{j-1}}, \end{aligned}$$

*folgende Eigenschaften erfüllt sind:*

*1)* $c_{j+\frac{1}{2}}, d_{j-\frac{1}{2}} \geq 0, \forall j \in \mathbb{Z}$, *2)* $c_{j+\frac{1}{2}} + d_{j+\frac{1}{2}} \leq 1, \forall j \in \mathbb{Z}$.

*Beweis:* Mit der Notation des Lemmas lässt sich (4.12) schreiben als

$$\bar{w}_j = \bar{u}_j + c_{j+\frac{1}{2}}\Delta_+\bar{u}_j - d_{j-\frac{1}{2}}\Delta_-\bar{u}_j.$$

Dabei sind $\Delta_+\bar{u}_j := \bar{u}_{j+1} - \bar{u}_j$ und $\Delta_-\bar{u}_j := \bar{u}_j - \bar{u}_{j-1}$. Das Lemma folgt dann direkt aus der Betrachtung von $\sum_{j\in\mathbb{Z}} |w_{j+1} - w_j|$ (siehe Lemma 1.2.25 im Skript zur Vorlesung *Numerik partieller Differentialgleichungen II*). □

Die Eigenschaft 2) folgt für die DG-Approximation automatisch, falls die Zeitschrittweite $\Delta t$ hinreichend klein gewählt wird. Kritischer ist die Eigenschaft 1). Betrachten wir dazu zunächst den

Spezialfall $u_h \in V_h^0$.
Für stückweise konstante Funktionen gilt $u^+_{j+\frac{1}{2}} = u_{j+1} = \bar{u}_{j+1}$ und $u^-_{j+\frac{1}{2}} = u_j = \bar{u}_j$. Daher erhalten wir

$$\begin{aligned} c_{j+\frac{1}{2}} &:= -\frac{\Delta t}{h}\frac{g(u_j, u_{j+1}) - f(u_j)}{u_{j+1} - u_j}, \\ d_{j-\frac{1}{2}} &:= -\frac{\Delta t}{h}\frac{g(u_{j-1}, u_j) - f(u_j)}{u_j - u_{j-1}}, \end{aligned}$$

Wir sehen nun, dass in diesem Fall die TV-Stabilität direkt aus der Konsistenz ($g(u,u) = f(u)$) und der Monotonie ($g(\nearrow, \searrow)$) des numerischen Flusses folgt, da gilt

$$\begin{aligned} \frac{g(u_j, u_{j+1}) - g(u_j, u_j)}{u_{j+1} - u_j} \leq 0, \\ \frac{g(u_{j-1}, u_j) - g(u_j, u_j)}{u_j - u_{j-1}} \leq 0. \end{aligned}$$

Für $u_h \in V_h^p, p > 0$ kann die Bedingung 1) im Allgemeinen nicht erfüllt werden, sie liefert vielmehr Bedingungen der Form

$$\operatorname{sign}(u^+_{j+\frac{1}{2}} - \bar{u}_j) = \operatorname{sign}(\bar{u}_{j+1} - \bar{u}_j). \tag{4.13}$$

Wir halten also folgendes Resultat zusammenfassend fest
**Korollar 4.34** (TV-Stabilität für das DG-Verfahren)**.**
*Das Discontinuous-Galerkin-Verfahren 4.31 ist Total-Variation verringernd, falls $p = 0$ gilt und der numerische Fluss konsistent und monoton ist. Für $p > 0$ gilt diese Aussage im Allgemeinen nicht.*

Während also im linearen Fall zur Stabilisierung lediglich geeignete Flüsse gewählt werden müssen, benötigen wir im nichtlinearen Fall einen zusätzlichen Mechanismus, der Vorzeichenbedingungen der Form (4.13) sicherstellt. Dies wird durch die Einführung von Projektionen - in diesem Kontext auch Limiter genannt - realisiert.

**Definition 4.35** (Massenerhaltende Limiter)**.**
Eine Projektion $\Lambda_h : V_h^p \to V_h^p$ heißt massenerhaltender Limiter, falls gilt

1) $\|\Lambda_h(u_h) - \overline{u_h}\|_{L^\infty} \leq \|u_h - \overline{u_h}\|_{L^\infty}$,
2) $\overline{\Lambda_h}(u_h) = \overline{u_h}$.

Dabei ist $\overline{v_h} \in V_h^0$ definiert durch $\overline{v_h}|_e := \frac{1}{|e|}\int_e v_h$.

**Definition 4.36** (Stabilisierte DG-Verfahren für Erhaltungsgleichungen)**.**
$\Lambda_h : V_h^p \to V_h^p$ sei ein massenerhaltender Limiter und $\hat{\sigma}$ ein vektorwertiger numerischer Fluss. Dann heißt $u_h(\cdot,t) \in V_h$ stabilisierte semidiskrete Discontinuous Galerkin Approximation mit Limiter $\Lambda_h$ und numerischem Fluss $\hat{\sigma}$ der Erhaltungsgleichung (4.11a), (4.11b), falls $\forall e \in \mathcal{E}^0_{\text{leaf}}, \forall v_h \in V_h$ gilt

$$\begin{aligned} \int_e \partial_t u_h(\cdot,t) v_h - \int_e f(\Lambda_h(u_h(\cdot,t))) \cdot \nabla v_h + \int_{\partial e} \hat{\sigma}_e(\Lambda_h(u_h)) \cdot n v_h &= 0, \\ u_h(\cdot,0) &= \pi_h(u_0). \end{aligned}$$

Bei einer stabilisierten DG-Approximation wird also bei nichtlinearen Auswertungen stets $u_h$ durch die Limitierung $\Lambda_h(u_h)$ ersetzt.

**Beispiel 4.37** (Slope-Limiter in einer Raumdimension)**.**
Wir betrachten wieder die eindimensionale Situation mit uniformer Zerlegung durch Entitäten $e = I_j$. Dann gilt für $v_h \in V_h^1$ die Darstellung

$$v_h|_{I_j}(x) = \bar{v}_j + (x - x_j)\partial_x v_j.$$

Wir definieren nun den sogenannten MinMod-Limiter $\Lambda_h^{\mathrm{m}}$ für $v_h \in V_h^1$ durch

$$\Lambda_h^{\mathrm{m}}(v_h)|_{I_j} := \bar{v}_j + (x - x_j)\; m(\partial_x v_j, \frac{\bar{v}_{j+1} - \bar{v}_j}{h/2}, \frac{\bar{v}_j - \bar{v}_{j-1}}{h/2}).$$

Dabei ist die MinMod-Funktion $m$ definiert durch

$$m(a_1, a_2, a_3) := \begin{cases} s \min\limits_{n=1,2,3} |a_n|, & \text{falls } s = \mathrm{sign}(a_1) = \mathrm{sign}(a_2) = \mathrm{sign}(a_3), \\ 0, & \text{sonst.} \end{cases}$$

Setzen wir $\tilde{v}_h = \Lambda_h^{\mathrm{m}}(v_h)$, so erhalten wir

$$\tilde{v}^-_{j+\frac{1}{2}}(x) = \bar{v}_j + m(v^-_{j+\frac{1}{2}} - \bar{v}_j, \bar{v}_j - \bar{v}_{j-1}, \bar{v}_{j+1} - \bar{v}_j)(x - x_j), \tag{4.14}$$

$$\tilde{v}^+_{j-\frac{1}{2}}(x) = \bar{v}_j - m(\bar{v}_j - v^+_{j-\frac{1}{2}}, \bar{v}_j - \bar{v}_{j-1}, \bar{v}_{j+1} - \bar{v}_j)(x - x_j). \tag{4.15}$$

Insbesondere erfüllt diese Limitierung die Vorzeichenbedingungen, die wir zur TV-Stabilität benötigen.

Für Funktionen $v_h \in V_h^p, p > 1$ kann diese Limitierung verallgemeinert werden. Sei dazu $v_h^1$ die $L^2$-Projektion von $v_h$ in $V_h^1$. Dann berechnen wir $\tilde{v}_h := \Lambda_h^{\mathrm{m}}(v_h)$ wie folgt:

1) Berechne $\tilde{v}^-_{j+\frac{1}{2}}$ und $\tilde{v}^+_{j-\frac{1}{2}}$ nach (4.14) und 4.15,

2) Falls $\tilde{v}^-_{j+\frac{1}{2}} = v^-_{j+\frac{1}{2}}$ und $\tilde{v}^+_{j-\frac{1}{2}} = v^+_{j-\frac{1}{2}}$, so setze $\tilde{v}_h|_{I_j} := v_h|_{I_j}$,

3) sonst setze $\tilde{v}_h|_{I_j} = \Lambda_h^{\mathrm{m}}(v_h^1|_{I_j})$.

Wir erhalten somit das folgende Stabilitätsresultat.

**Lemma 4.38** (TV-Stabilität mit MinMod-Limiter in 1D)**.**
*Wir betrachten das stabilisierte DG-Verfahren 4.36 in einer Raumdimension auf uniformen Gittern mit konsistenter und monotoner numerische Flussfunktion und $\Lambda_h = \Lambda_h^m$. Dann gilt unter Voraussetzung der CFL-Bedingung*

$$\frac{\Delta t}{h}\Big(L_{g,1} + L_{g,2}\Big) \leq \frac{1}{2},$$

*dass ein Euler-Schritt TV-Stabil ist, d.h. für $w_h = u_h + \Delta t L_h(\Lambda_h^m(u_h))$ gilt*

$$|\bar{w}_h|_{TV} \leq |\bar{u}_h|_{TV}.$$

*Dabei bezeichnen $L_{g,1}, L_{g,2}$ die Lipschitz-Konstanten von $g$ bezüglich des ersten, bzw. zweiten Arguments.*

Zu stabilisierten DG-Verfahren für Erhaltungsgleichungen im Sinne von Definition 4.36 gibt es bislang noch keine rigorosen A-priori-Fehlerabschätzungen für $V_h^p$ mit $p > 0$. Im folgenden wollen wir ein rigoroses A-posteriori-Resultat vorstellen, dass eine adaptive Wahl der lokalen Gitterweite und des lokalen Limiters erlaubt (siehe [10]). Wir führen dazu zunächst den Begriff der Entropielösung ein.

**Definition 4.39** (Schwache Lösung).
Seien $u_0 \in L^\infty(\mathbb{R}^d)$ und $f \in C^{0,1}(\mathbb{R};\mathbb{R}^d)$ gegeben. Dann heißt $u \in L^1_{loc}(\mathbb{R}^d \times \mathbb{R}^+)$ schwache Lösung von (4.11a), (4.11b), falls gilt:

$$\iint_{\mathbb{R}^d\times\mathbb{R}^+} u\partial_t\varphi + f(u)\cdot\nabla\varphi + \int_{\mathbb{R}^d} u_0\varphi(\cdot,0) = 0 \qquad \forall\varphi \in C_0^\infty(\mathbb{R}^d\times[0,\infty)).$$

**Definition 4.40** (Entropielösung).
Seien $f \in C^1(\mathbb{R};\mathbb{R}^d), u_0 \in L^\infty(\mathbb{R}^d)$ gegeben. Wir nennen $(S, F_S)$ Entropie-Enptropiefluss-Paar zu der Erhaltungsgleichung (4.11a), falls $S \in C^1(\mathbb{R})$ konvex ist und für $F_S \in C^1(\mathbb{R};\mathbb{R}^d)$ gilt

$$F_S' = S'f'.$$

Eine Funktion $u \in L^\infty(\mathbb{R}^d \times \mathbb{R}^+)$ heißt Entropielösung von (4.11a), (4.11b), falls für alle Entropie-Enptropiefluss-Paare $(S, F_S)$ und alle Testfunktionen $\Phi \in C_0^1(\mathbb{R}^d \times \mathbb{R}^+;[0,\infty))$ gilt

$$\int_{\mathbb{R}^d}\int_{\mathbb{R}^+} (S(u)\partial_t\Phi + F_S(u)\cdot\nabla\Phi) + \int_{\mathbb{R}^d} S(u_0)\Phi(\cdot,0) \geq 0. \tag{4.16}$$

**Definition 4.41** (Funktionen beschränkter Variation(BV)).
Für Funktionen $u \in L^1(\mathbb{R}^d)$ definieren wir die Halbnorm

$$|u|_{BV(\mathbb{R}^d)} := \sup_{\varphi\in C_0^\infty(\mathbb{R}^d;\mathbb{R}^d),\|\varphi\|_\infty\leq 1} \int_{\mathbb{R}^d} u\nabla\cdot\varphi$$

und den Raum der Funktionen beschränkter Variation BV durch:

$$BV(\mathbb{R}^d) := \left\{u \in L^1(\mathbb{R}^d)| \quad \|u\|_{BV(\mathbb{R}^d)} := \|u\|_{L^1(\mathbb{R}^d)} + |u|_{BV(\mathbb{R}^d)} < \infty\right\}$$

**Theorem 4.42** (Existenz, Eindeutigkeit und Regularität).
*Es existiert genau eine Entropielösung $u$ von* (4.11a), (4.11b). *Ist $u_0 \in BV(\mathbb{R}^d)$, so gilt $u \in BV(\mathbb{R}^d\times\mathbb{R}^+)$ und es existiert eine Konstante $C > 0$ mit*

$$\|u\|_{BV(\mathbb{R}^d\times\mathbb{R}^+)} \leq C\|u_0\|_{BV(\mathbb{R}^d)}.$$

*Beweis:* Existenz: Konvergenz von Finite-Volumen-Verfahren.
Eindeutigkeit: $L^1$-Kontraktion, Kruzkov-Techniken!
$BV$-Regularität: Uniforme Abschätzungen für Viskositätslimes.
Für einen detaillierten Beweis siehe z.B. Málek, Nečas, Rokyta, Růžička [16]. □

**Annahme 4.43** (Numerische Flüsse).
Wir nehmen an, dass für die numerische Flussfunktion gilt

$$\hat{\sigma}_e(u_h)(\cdot,t)\cdot n = g_n(u_h^-(\cdot,t), u_h^+(\cdot,t)).$$

Dabei sei $n$ die äußere Normale an $e$ und es gelte $g_n \in C^1(\mathbb{R}\times\mathbb{R};\mathbb{R})$, so dass für alle $u, v, u', v' \in$

$[A, B]$, $|A|, |B|$ hinreichend groß, die folgenden Bedingungen gelten:

$$\begin{aligned} \partial_u g_n(u,v) \geq 0, \qquad \partial_v g_n(u,v) \leq 0, \qquad & \text{(Monotonie)} \\ g_n(u,v) = -g_{-n}(v,u), \qquad & \text{(Erhaltung)} \\ g_n(u,u) = f(u) \cdot n, \qquad & \text{(Konsistenz)} \\ |g_n(u,v) - g_n(u',v')| \leq L(|u-u'| + |v-v'|). \qquad & \text{(Lipschitz-Stetigkeit)} \end{aligned}$$

Wir wollen nun auf der Basis von Definition 4.36 ein semidiskretes Discontinuous-Galerkin-Verfahren auf adaptiven Gittern definieren. Dazu führen wir zunächst eine Zerlegung des Zeitintervalls $[0, T]$ wie folgt ein

$$\{0 = t^0, \ldots, t^N = T\}.$$

Weiter setzen wir die lokale Zeitschrittweite $\Delta t^n := t^{n+1} - t^n$. Mit jedem Zeitintervall $(t^n, t^{n+1}]$ assoziieren wir ein Gitter $\mathcal{T}_h^n$ mit zugehörigem diskreten Funktionenraum $V_{h,n}^p$ definiert durch

$$V_{h,n}^p := \{v_h \in BV(\mathbb{R}^d) \mid v_h|_e \in \mathbb{P}_p(e) \text{ für alle } e \in \mathcal{E}_{\text{leaf}}^{n,0}\}.$$

Um einen Gitterwechsel $\mathcal{T}_h^{n-1} \to \mathcal{T}_h^n$ zu berücksichtigen, verallgemeinern wir in der folgenden Definition die massenerhaltenden Limiter.

**Definition 4.44** (Massenerhaltende Limiter auf adaptiven Gittern)**.**
$\Lambda_h^{n,t} : V_{h,n}^p \to V_{h,n}^p$ sei ein massenerhaltender Limiter nach Definition 4.35, der auf dem Intervall $[t^n, t^{n+1}]$ stetig von $t$ abhängt. Darüber hinaus sei $\Lambda_h^{n,t^n} : V_{h,n-1}^p \to V_{h,n}^p$ eine Projektion auf das neue Gitter mit der Eigenschaft

1) $\overline{\Lambda_h^{n,t^n}(u_h(\cdot, t^n))} = \overline{u_h(\cdot, t^n)}$,

die folgende Kompatibilitätsbedingung erfüllt

2) $\|(\Lambda_h^{n,t^n}(u_h) - \overline{u_h})(\cdot, t)\|_{L^\infty} \leq \|(\Lambda_h^{n-1,t^n}(u_h) - \overline{u_h})(\cdot, t)\|_{L^\infty}$.

Im Folgenden werden wir als Notation auch

$$\tilde{u}_h(\cdot, t) := \Lambda_h^{n,t} u_h(\cdot, t)$$

verwenden.

**Definition 4.45** (Semi-diskretes DG-Verfahren auf adaptiven Gittern)**.**
Seien $\mathcal{T}_h^n$ eine Folge von Gittern und $\Lambda_h^{n,t}$ massenerhaltende Limiter auf diesen Gittern. Weiter sei $g$ ein numerischer Fluss, der die Annahme 4.43 erfüllt. Dann heißt $u_h$ semidiskrete DG-Approximation von (4.11a)-(4.11b), falls gilt

$$u_h^{-1} := \Pi_{V_{h,0}^p}(u_0)$$

und für $n = 0, \ldots, N-1$ ist $u_h^n|_{[t^n, t^{n+1}]} \in C^1(t^n, t^{n+1}; V_{h,n}^p)$ definiert durch

$$u_h^n(t^n) := \Lambda_h^{n,t^n}(u_h^{n-1}(t^n)), \tag{4.17}$$

$$\int_e \partial_t u_h^n(\cdot, t) v_h - \int_e f(\tilde{u}_h^n(\cdot, t)) + \int_{\partial e} g_n(\tilde{u}_h^{n,-}(\cdot, t), \tilde{u}_h^{n,+}(\cdot, t)) = 0, \tag{4.18}$$

$$\text{für alle } e \in \mathcal{E}_{\text{leaf}}^{n,0}, v_h \in V_h^{n,p}, t \in (t^n, t^{n+1}).$$

Wir definieren schließlich $u_h \in L^\infty(0,T;V_{h,n}^p)$ durch

$$u_h(0) := u_h^{-1} \text{ und } u_h|_{(t^n,t^{n+1}]} := u_h^n|_{(t^n,t^{n+1}]}.$$

Wir werden nun eine A-posteriori-Fehlerabschätzung für den Fehler

$$\|(u-u_h)(\cdot,T)\|_{L^1(K)}$$

auf einem Teilgebiet $K \subset \mathbb{R}^d$ herleiten. Dazu verwenden wir zunächst die Dreiecksungleichung

$$\|(u-u_h)(\cdot,T)\|_{L^1(K)} \le \|(u-\tilde{u}_h)(\cdot,T)\|_{L^1(K)} + \|(\tilde{u}_h-u_h)(\cdot,T)\|_{L^1(K)}.$$

Da der zweite Term auf der rechten Seite bereits eine A-posteriori-Größe darstellt, werden wir uns nur noch mit einer Abschätzung von

$$\|(u-\tilde{u}_h)(\cdot,T)\|_{L^1(K)}$$

beschäftigen. Dabei müssen wir berücksichtigen, dass $\tilde{u}_h$ in den diskreten Zeitpunkten $t^n, n = 0,\dots,N-1$ unstetig sein kann.

Der Beweis der A-posteriori-Fehlerabschätzung beruht auf der Variablenverdopplungstechnik von Kruzkov. Dazu definieren wir zunächst das Entropieresiduum $R_S$.

**Definition 4.46** (Entropieresiduum $R_S$)**.**
Sei $\tilde{u} \in L^\infty(\mathbb{R}^d \times \mathbb{R}^+)$. Dann definieren wir das Entropieresiduum $R_S$ zu dem Entropie-Entropiefluss-Paar $(S, F_S)$ durch

$$\langle R_S(\tilde{u}), \Phi\rangle := \iint_{\mathbb{R}^d\times\mathbb{R}^+} \Big(S(\tilde{u})\partial_t\Phi + F_S(\tilde{u})\cdot\nabla\Phi\Big) + \int_{R^d} S(u_0)\Phi(\cdot,0). \tag{4.19}$$

Als spezielle zwei-Parameterfamilie von Entropien betrachten wir regularisierte Kruzkov-Entropien.

**Definition 4.47** ($\delta$-regularisierte Kruzkov Entropien)**.**
Sei $\bar{S} \in C^2(\mathbb{R},\mathbb{R}^+)$ definiert durch

$$\bar{S}(v) = \begin{cases} (6v^2 - v^4)/8, & \text{falls } |v| \le 1 \\ 3/8 & \text{sonst.} \end{cases}$$

Für $\delta > 0, \kappa, v \in \mathbb{R}$ definieren wir die Entropien $S_{\delta,\kappa} : \mathbb{R} \to \mathbb{R}^+$ durch

$$S_{\delta,\kappa}(v) := S_\delta(v-\kappa) := \delta\bar{S}\Big(\frac{v-\kappa}{\delta}\Big).$$

Weiter sei der zugehörige Entropiefluss $F_{\delta,\kappa} : \mathbb{R} \to \mathbb{R}$ definiert durch

$$F_{\delta,\kappa}(v) := F_\delta(v,\kappa) = \int_\kappa^v f'(w)S'_\delta(w-\kappa)\mathrm{d}w.$$

**Bemerkung 4.48.**
Die $\delta$-regularisierte Kruzkov Entropien sind zweimal stetig differenzierbar. Für $\delta \to 0$ konvergieren sie gegen die klassischen Kruzkov-Entropien der Form $S(v-\kappa) = |v-\kappa|$. Genauer gilt

$$|v-\kappa| - 3/8\delta\chi_{\mathrm{supp}(v-\kappa)} \le S_\delta(v-\kappa) \le |v-\kappa|.$$

Wir starten den Beweis mit einer fundamentalen Fehlerabschätzung, basierend auf einer Annahme für das Entropieresiduum für $\delta$-regularisierte Kruzkov Entropien.

**Theorem 4.49** (Fundamentale Fehlerabschätzung)**.**
*Sei $u_0 \in BV(\mathbb{R}^d)$ und sei $u$ die Entropielösung von (4.11a), (4.11b) mit $f \in C^2(\mathbb{R})$. Sei $v \in L^\infty_{loc}([0,\infty), L^1_{loc}(\mathbb{R}^d))$ rechts stetig bezüglich $t$. Sei $S(v) = S_{\delta,\kappa}(v)$ eine $\delta-$regularisierte Kruzkov Entropie mit Entropiefluss $F_S(v) = F_{\delta,\kappa}(v)$. Wir nehmen an, dass es Maße $\mu^t_{v,\delta}, \mu^x_{v,\delta}, \mu_{v,\delta} \in M(\mathbb{R}^d \times \mathbb{R}^+)$ gibt, so dass das Entropieresiduum $R_S(v)$ $\forall \kappa \in \mathbb{R}, \delta > 0$ und alle $\Phi \in C^1_0(\mathbb{R}^d \times (0,T)), \Phi \geq 0$ abgeschätzt werden kann durch*

$$\langle R_S(v), \Phi\rangle \geq -\Big( \left\langle |\partial_t \Phi| \, , \mu^t_{v,\delta} \right\rangle + \left\langle |\nabla \Phi| \, , \mu^x_{v,\delta} \right\rangle + \langle |\Phi|, \mu_{v,\delta}\rangle \Big). \tag{4.20}$$

*Dabei habe das Maß $\mu^t_{v,\delta}$ eine Dichte $\alpha^t_{v,\delta} \in L^\infty_{loc}(0,\infty; L^1_{loc}(\mathbb{R}^d))$. Seien $R \geq 0, x_0 \in \mathbb{R}^d$. Wir definieren das Abhängigkeitsgebiet $D_h$ durch*

$$D_h := \bigcup_{0\leq t\leq T} B_{R+L_f(T-t)+h}(x_0) \times \{t\}\,.$$

*Dann existiert ein $h \geq 0$ und Konstanten $C_0, \ldots, C_4$, so dass gilt:*

$$\begin{aligned} \|(u-v)(\cdot,T)\|_{L^1(B_R(x_0))} \quad \leq \quad & \|(u-v)(\cdot,0)\|_{L^1(B_{R+L_f T}(x_0))} + C_0\delta \\ & + C_1 \mu_{v,\delta}(D_{2h}) + C_2 \sqrt{\mu^x_{v,\delta}(D_h)} \\ & + C_3 E^t_{v,\delta} + C_4 \sqrt{E^t_{v,\delta}}. \end{aligned} \tag{4.21}$$

*Dabei ist $E^t_{v,\delta}$ definiert durch*

$$E^t_{v,\delta} := \sup_{0\leq t\leq T+h} \int_{B_{R+L_f(T-t)+h}(x_0)} \alpha^t_{v,\delta}(t,x)\; dx.$$

*Beweis:* Der Beweis folgt den Ideen der fundamentalen Fehlerabschätzung (Theorem 2.2.16) aus der Vorlesung *Numerik partieller Differentialgleichungen II*.

Zunächst definieren wir

$$\Phi_r(x,t,y,s) := \psi(x,t)\omega_r(t-s, x-y),$$

wobei $\psi$ und $\omega_r$ noch zu wählende Testfunktionen sind.

Wählen wir nun in der Residuumsabschätzung (4.20)

$$\kappa = u(y,s) \qquad \text{und} \qquad \Phi(x,t) = \Phi_r(x,t,y,s),$$

so erhalten wir nach Integration über $y, s$ aus (4.20)

$$\begin{aligned} -\iint_{\mathbb{R}^d\times\mathbb{R}^+} R_S(v) \quad = \quad & -\iint_{(\mathbb{R}^d\times\mathbb{R}^+)^2} S_\delta(v(x,t)-u(y,s))\partial_t\Phi_r(x,t,y,s) \\ & + F_\delta(v(x,t),u(y,s))\cdot\nabla_x\Phi_r(x,t,y,s)dx\; dt\; dy\; ds \\ \leq \quad & T_1^{r,\delta} \end{aligned}$$

mit

$$T_1^{r,\delta} := \iint_{\mathbb{R}^d\times\mathbb{R}^+} \left\langle |\partial_t\Phi_r| \, , \mu^t_{v,\delta} \right\rangle + \left\langle |\nabla\Phi_r| \, , \mu^x_{v,\delta} \right\rangle + \langle |\Phi_r|, \mu_{v,\delta}\rangle \, dy\; ds.$$

Mit $\partial_t \omega_r = -\partial_s \omega_r$ und $\nabla_x \omega_r = -\nabla_y \omega_r$ erhalten wir weiter

$$\begin{aligned}
&S_\delta(v(x,t)-u(y,s))\partial_t \Phi_r(x,t,y,s) + F_\delta(v(x,t),u(y,s))\cdot\nabla_x\Phi_r(x,t,y,s)\\
&\quad= \Big(S_\delta(v(x,t)-u(y,s))\partial_t\psi(x,t)\\
&\qquad +F_\delta(v(x,t),u(y,s))\cdot\nabla_x\psi(x,t)\Big)\omega_r(t-s,x-y)\\
&\qquad +\Big(S_\delta(v(x,t)-u(y,s))\partial_t\omega_r(t-s,x-y)\\
&\qquad +F_\delta(v(x,t),u(y,s))\cdot\nabla_x\omega_r(t-s,x-y)\Big)\psi(x,t)\\
&\quad= \Big(S_\delta(v(x,t)-u(y,s))\partial_t\psi(x,t)\\
&\qquad +F_\delta(v(x,t),u(y,s))\cdot\nabla_x\psi(x,t)\Big)\omega_r(t-s,x-y)\\
&\qquad -\Big(S_\delta(u(y,s)-v(x,t))\partial_s\omega_r(t-s,x-y)\\
&\qquad +F_\delta(u(y,s),v(x,t))\cdot\nabla_y\omega_r(t-s,x-y)\Big)\psi(x,t)\\
&\qquad -\Big(F_\delta(v(x,t),u(y,s))-F_\delta(u(y,s),v(x,t))\Big)\\
&\qquad\qquad \cdot\nabla_y\omega_r(t-s,x-y)\psi(x,t)
\end{aligned}$$

Damit folgt insgesamt

$$\begin{aligned}
&-\iint_{(\mathbb{R}^d\times\mathbb{R}^+)^2}\Big(S_\delta(v(x,t)-u(y,s))\partial_t\psi(x,t)\\
&\qquad +F_\delta(v(x,t),u(y,s))\cdot\nabla_x\psi(x,t)\Big)\omega_r(t-s,x-y)\\
&\quad\le T_1^{r,\delta}+T_2^{r,\delta}
\end{aligned}\tag{4.22}$$

mit

$$\begin{aligned}
T_2^{r,\delta} :=\ & -\iint_{(\mathbb{R}^d\times\mathbb{R}^+)^2}\Big(F_\delta(v(x,t),u(y,s))-F_\delta(u(y,s),v(x,t))\Big)\\
&\cdot\nabla_y\omega_r(t-s,x-y)\psi(x,t).
\end{aligned}$$

Wir wollen nun die rechte Seite von Gleichung (4.22) mit dem Fehler in Verbindung bringen. Dazu stellen wir zunächst fest, dass gilt

$$|F_\delta(v,w)|\le L_f S_\delta(v,w).$$

Dies sehen wir z.B. für $v\ge w$ wie folgt ein

$$\begin{aligned}
|F_\delta(v,w)| &= \Big|\int_w^v f'(v)S_\delta'(s-w)\,ds\Big|\\
&\le L_f\int_w^v S_\delta'(s-w)\,ds = L_f S_\delta(v,w).
\end{aligned}$$

Analoges folgt für $v\le w$.

Im nächsten Schritt wollen wir die Testfunktion $\psi$ so wählen, dass gilt

$$\begin{aligned}
&S_\delta(v(x,t)-u(y,s))\partial_t\psi(t,x)+F_\delta(v(x,t),u(y,s))\cdot\nabla_x\psi(t,x)\\
&\quad\le S_\delta(v(x,t)-u(y,s)).
\end{aligned}$$

Dazu wählen wir die Testfunktion $\psi$ wie folgt:

Sei $\varepsilon > 0$. Wir definieren $Y_\varepsilon(t)$ so dass gilt $Y_\varepsilon(-\infty) = 0$ und $Y'_\varepsilon(t) = \frac{1}{\varepsilon} Y'(\frac{t}{\varepsilon})$, mit $Y' \in C_0^\infty((0,1)),\ Y' \geq 0$ und $\int Y' = 1$. Damit definieren wir als Abschneidefunktion in der Zeit

$$X_\varepsilon(t) = Y_\varepsilon(t) - Y_\varepsilon(t-T)\ \in C_0^\infty((0, T+\varepsilon)),\ X_\varepsilon \geq 0$$

und als Abschneidefunktion im Ort

$$Z_{h,\theta}(x,t) = 1 - Y_\theta\big(|x-x_0| - (R + L_f(T-t) + h\big) \geq 0.$$

Die Abschneidefunktion $Z_{h,\theta}$ erfüllt

$$\begin{aligned} \partial_t Z_{h,\theta}(x,t) &= -L_f Y'_\theta\big(|x-x_0| - (R + L_f(T-t) + h\big) \\ \nabla_x Z_{h,\theta}(x,t) &= -Y'_\theta\big(|x-x_0| - (R + L_f(T-t) + h\big) \frac{x-x_0}{|x-x_0|}. \end{aligned}$$

Damit ist schließlich $\psi$ gegeben als

$$\psi(x,t) = X_\varepsilon(t) Z_{h,\theta}(x,t)\,.$$

Mit dieser Wahl von $\psi$ und der Lipschitz-Stetigkeit von $f$ folgt

$$\begin{aligned} &S_\delta(v(x,t) - u(y,z))\partial_t\psi(x,t) + F_\delta(v(x,t)\,,\, u(y,s)) \cdot \nabla_x \psi(x,t) \\ &\quad = S_\delta(v(x,t) - u(y,s)) X'_\varepsilon(t) Z_{h,\theta}(t,x) \\ &\qquad - X_\varepsilon(t) Y'_\varepsilon(\dots)\left(L_f S_\delta(v(x,t) - u(y,s)) + F_\delta(v(x,t), u(y,s)) \cdot \tfrac{x-x_0}{|x-x_0|}\right) \\ &\quad \leq S_\delta(v(x,t) - u(s,y)) X'_\varepsilon(t) Z_{h,\theta}(t,x)\,. \end{aligned}$$

Also folgt aus (4.22)

$$\begin{aligned} -\iint\limits_{(\mathbb{R}^d \times R^+)^2} & S_\delta(v(x,t) - u(s,y)) X'_\varepsilon(t) Z_{h,\theta}(x,t) \\ & \omega_r(t-s, x-y)\,ds\,dt\,dy\,dx \leq T_1^{r,\delta} + T_2^{r,\delta}\,. \end{aligned}$$

Mit

$$\begin{aligned} -S_\delta(v(x,t) - u(x,t)) &= -S_\delta(v(x,t) - u(x,t)) + S_\delta(v(x,t) - u(y,t)) \\ &\quad -S_\delta(v(x,t) - u(y,t)) + S_\delta(v(x,t) - u(y,s)) \\ &\quad -S_\delta(v(x,t) - u(y,s)) \\ &\leq |u(x,t) - u(y,t)| + |u(y,t) - u(y,s)| \\ &\quad -S_\delta(v(x,t) - u(y,s)) \end{aligned}$$

folgt weiter

$$\begin{aligned} -\iint\limits_{(\mathbb{R}^d \times R^+)^2} & S_\delta(v(x,t) - u(x,t)) X'_\varepsilon(t) Z_{h,\theta}(x,t) \\ & \omega_r(t-s, x-y)\,ds\,dt\,dy\,dx \leq (T_1 + T_2 + T_3 + T_4)^{r,\delta,\varepsilon,h}\,. \end{aligned}$$

mit

$$\begin{aligned} T_3^{r,\delta,\varepsilon,h} &= \iint\limits_{(\mathbb{R}^d\times R^+)^2} |u(x,t)-u(y,t)|X'_\varepsilon(t)Z_{h,\theta}(x,t)\omega_r(t-s,x-y)dsdtdydx \\ T_4^{r,\delta,\varepsilon,h} &= \iint\limits_{(\mathbb{R}^d\times R^+)^2} |u(y,t)-u(y,s)|X'_\varepsilon(t)Z_{h,\theta}(x,t)\omega_r(t-s,x-y)dsdtdydx \end{aligned}$$

Jetzt sind wir in der Lage den Grenzwert $\varepsilon \to 0$ zu untersuchen. Dazu betrachten wir zunächst die linke Seite und erhalten mit den Eigenschaften von $S_\delta$ und der Definition von $X_\varepsilon$

$$\begin{aligned} -S_\delta(v(x,t)-u(x,t))X'_\varepsilon(t) &= -S_\delta(v(x,t)-u(x,t))Y'_\varepsilon(t) \\ &\quad +S_\delta(v(x,t)-u(x,t))Y'_\varepsilon(t-T) \\ &\geq -|v(x,t)-u(x,t)|Y'_\varepsilon(t) \\ &\quad +|v(x,t)-u(x,t)|Y'_\varepsilon(t-T) \\ &\quad +3/8\ \delta\ \chi_{\mathrm{supp}(v-u)}Y'_\varepsilon(t-T). \end{aligned}$$

Im Grenzübergang $\varepsilon \to 0$ führt $Y'_\varepsilon(t)$ zu einer Punktauswertung in 0, während $Y'_\varepsilon(t-T)$ zu einer Punktauswertung in $T$ führt. Setzen wir voraus, dass $\omega_r$ so gewählt wird, dass gilt

$$\iint_{\mathbb{R}^d\times\mathbb{R}^+} \omega_r(t-s,x-y)ds\ dy = 1,$$

so erhalten wir

$$\begin{aligned} -\limsup_{\varepsilon\to 0} \iint\limits_{(\mathbb{R}^d\times R^+)^2} & S_\delta(v(x,t)-u(x,t))X'_\varepsilon(t)Z_{h,\theta}(x,t) \\ & \omega_r(t-s,x-y)dsdtdydx \\ \geq & -\int_{B_{R+h+L_fT}(x_0)} |v(x,0)-u(x,0)|dx \\ & +\int_{B_{R+h}(x_0)} |v(x,T)-u(x,T)|dx \\ & +\int_{B_{R+h}(x_0)} 3/8\ \delta\ \chi_{\mathrm{supp}(v-u)}dx. \end{aligned}$$

Es steht nun noch aus diesen Grenzübergang für die Terme der rechten Seite zu betrachten. Dazu wählen wir jetzt die Testfunktion $\omega_r(t-s,x-y)$, so dass t,s sowie x,y höchstens den Abstand $r_1, r_2$ voneinander haben.
Seien dazu $\tilde{\omega}_1 \in C_0^\infty(\mathbb{R},\mathbb{R})$ und $\bar{\omega}_1 \in C_0^\infty(\mathbb{R}^d,\mathbb{R})$ gegeben mit

$$\begin{aligned} \mathrm{supp}\,\tilde{\omega}_1 \subset [-1,0], \quad \tilde{\omega}_1 \geq 0, \quad \int_{\mathbb{R}} \tilde{\omega}_1(t)dt = 1 \\ \mathrm{supp}\,\bar{\omega}_1 \subset B_1(0) \subset \mathbb{R}^d, \quad \bar{\omega}_1 \geq 0, \quad \int_{\mathbb{R}^d} \bar{\omega}_1(x)dx = 1 \end{aligned}$$

Für $r > 0$ definieren wir nun

$$\tilde{\omega}_r(t) = \frac{1}{r}\tilde{\omega}_1(\frac{t}{r}), \quad \bar{\omega}_r(x) = \frac{1}{r^d}\bar{\omega}_1(\frac{x}{r}).$$

und schließlich

$$\omega_r(t-s,x-y) = \tilde{\omega}_{r_1}(t-s)\bar{\omega}_{r_2}(x-y).$$

Betrachten wir nun $T_3^{r,\delta,\varepsilon,h}, T_4^{r,\delta,\varepsilon,h}$, so erhalten wir die Abschätzungen

$$\limsup_{\varepsilon\to 0}\left(T_3^{r,\delta,\varepsilon,h} + T_4^{r,\delta,\varepsilon,h}\right) \le C\|u_0\|_{BV(\mathbb{R}^d)}(r_1 + L_f r_2).$$

Für den Term $T_1^{r,\delta,\varepsilon,h}$ erhalten wir mit $\theta = Cr_2$

$$\limsup_{\varepsilon\to 0} T_1^{r,\delta,\varepsilon,h} \le C\Big(\mu_{v,\delta}(D_h) + (1+\frac{(L_f+1)T}{r_1})E^t_{v,\delta} + \frac{1}{r_2}\mu^x_{v,\delta}(D_h)\Big).$$

Schließlich erhalten wir für $T_2^{r,\delta,\varepsilon,h}$ durch partielle Integration

$$\begin{aligned} T_2^{r,\delta} &= \iint_{(\mathbb{R}^d\times\mathbb{R}^+)^2} \nabla_y\Big(F_\delta(v(x,t),u(y,s)) - F_\delta(u(y,s),v(x,t))\Big) \\ &\qquad \cdot\omega_r(t-s,x-y)\psi(x,t). \end{aligned}$$

Mit der Definition von $F_\delta$ gilt weiter

$$\partial_u(F_\delta(v,u) - F_\delta(u,v)) = \int_v^u S''(w-v)(f'(w)-f'(u))\mathrm{d}w.$$

Also folgt mit der Definition von $S_\delta$

$$\begin{aligned} |\partial_u(F_\delta(v,u) - F_\delta(u,v))| &\le \|f''\|_{L^\infty(\mathbb{R})}\int_v^u \frac{1}{\delta}\bar{S}''(\frac{w-v}{\delta})|w-u|\mathrm{d}w \\ &\le \|f''\|_{L^\infty(\mathbb{R})}\sup_{|w|\le 1}\bar{S}''(w)\frac{1}{2\delta}\delta^2. \end{aligned}$$

Mit $k_1 := \frac{1}{2}\|f''\|_{L^\infty(\mathbb{R})}\sup_{|w|\le 1}\bar{U}''(w)$ folgt also die Abschätzung

$$\limsup_{\varepsilon\to 0} T_2^{r,\delta,\varepsilon,h} \le k_1\|u_0\|_{BV(\mathbb{R}^d)}\delta.$$

Setzen wir jetzt

$$\begin{aligned} C_0 &:= \int_{B_{R+h}(x_0)} 3/8\ \chi_{\mathrm{supp}(v-u)}dx + k_1\|u_0\|_{BV(\mathbb{R}^d)}, \\ C_1 &:= C, \\ C_2 &:= \sqrt{L_f\|u_0\|_{BV(\mathbb{R}^d)}}, \\ C_3 &:= C, \\ C_4 &:= \sqrt{\frac{\|u_0\|_{BV(\mathbb{R}^d)}}{(L_f+1)T}} \end{aligned}$$

und wählen

$$r_1 := \sqrt{\frac{\mu^t_{v,\delta}(D_h)}{C_4}}$$

und

$$r_2 := \sqrt{\frac{\mu^x_{v,\delta}(D_h)}{C_2}},$$

so folgt die Behauptung. □

Um eine A-posteriori-Fehlerabschätzung für das DG-Verfahren zu beweisen, wollen wir die Fundamentale Fehlerabschätzung anwenden mit $v = \tilde{u}_h$. Dazu benötigen wir eine Abschätzung für das Entropieresiduum

$$\langle R_S(\tilde{u}_h), \varphi \rangle$$

In einem ersten Schritt schätzen wir dazu die Entropie auf einer Entität ab. Wie werden dazu folgende Notation verwenden:

**Notation.**
Für eine Entität $e \in \mathcal{E}^{n,0}_{\text{leaf}}$ bezeichnen wir mit $u_e$ die Einschränkung von $u_h$ auf $e \times (t^n, t^n + 1]$ mit $u_{l(e)}$ die Einschränkung von $u_h$ auf $l(e) \times (t^n, t^n + 1]$. Dabei bezeichnet $l(e) \in \mathcal{E}^{n,0}_{\text{leaf}}$ die Nachbarentität zu $e$ über den Rand von $e$.

Für eine kontinuierliche Funktion $\varphi$ sei $\overline{\varphi_e}$ der Mittelwert von $\varphi$ auf $e$ und $\overline{\varphi_h}$ die stückweise konstante Funktion definiert durch $\overline{\varphi_h}|_e := \overline{\varphi_e}$.

**Lemma 4.50** (Zell-Entropieungleichung)**.**
*Sei $(S, F_S)$ ein Entropie-Entropiefluss-Paar. Dann gilt für die limitierte Lösung des DG-Verfahrens $\widetilde{u}_h$ aus Definition 4.45 die folgende Zell-Entropieungleichung für alle Entitäten $e \in \mathcal{E}^{n,0}_{leaf}$*

$$I_e := I_{1,e} + I_{2,e} + I_{3,e} + I_{4,e} - D_e \leq 0, \tag{4.23}$$

*wobei für $\varphi \in C^1_0(\mathbb{R}^d \times \mathbb{R}^+, \mathbb{R}^+)$ die Terme wie folgt definiert sind*

$$\begin{aligned}
I_{1,e} &= \int_e (\partial_t S(\widetilde{u}_e) + \nabla \cdot F_S(\widetilde{u}_e))\ \overline{\varphi_e}, \\
I_{2,e} &= \int_{\partial e} \Big( G_n(\widetilde{u}_e, \widetilde{u}_{l(e)}) - F_S(\widetilde{u}_e) \cdot n \Big) \overline{\varphi_e}, \\
I_{3,e} &= \int_e (\partial_t \widetilde{u}_e + \nabla \cdot f(\widetilde{u}_e))(S'(\overline{\widetilde{u}_e}) - S'(\widetilde{u}_e))\overline{\varphi_e}, \\
I_{4,e} &= \int_{\partial e} \Big( g_n(\widetilde{u}_e, \widetilde{u}_{l(e)}) - f(\widetilde{u}_e) \cdot n \Big)(S'(\overline{\widetilde{u}_e}) - S'(\widetilde{u}_e))\overline{\varphi_e} \Big), \\
D_e &= \int_{\partial e} \Big( \int_{\widetilde{u}_e}^{\widetilde{u}_{l(e)}} \partial_w g_n(\widetilde{u}_e, w) \int_w^{\widetilde{u}_e} S''(s)\, ds dw \Big) \overline{\varphi_e}.
\end{aligned}$$

*Dabei ist $G_n(\alpha, \beta) := \int_\alpha^\beta \partial_s g_n(\alpha, s) S'(s) ds + F_S(\alpha)$ ein diskreter Entropiefluss, der konsistent mit $F_S$ ist, d.h. es gilt*

$$\partial_\beta G_n(\alpha, \beta) = \partial_\beta g_n(\alpha, \beta)\ S'(\beta).$$

*Beweis:* Sei $p \geq 0$. Wir betrachten die lokale Form des DG-Verfahrens (4.18) und wählen als Testfunktion $v_h = S'(\overline{\widetilde{u}_h})\overline{\varphi_h}$. Wir erhalten

$$\int_e \partial_t u_e S'(\overline{\widetilde{u}_e})\overline{\varphi_e} + \int_{\partial e} g_n(\widetilde{u}_e, \widetilde{u}_{l(e)}) S'(\overline{\widetilde{u}_e})\overline{\varphi_e} = 0.$$

Da die Limiter massenerhaltend sind, gilt für stückweise konstante Testfunktionen $v_h \in V^0_{h,n}$ die Identität $(\partial_t u_e, v_e)_e = (\partial_t \widetilde{u}_e, v_e)_e$, Daher erhalten wir

$$\int_e \partial_t \widetilde{u}_e S'(\overline{\widetilde{u}_e})\overline{\varphi_e} + \int_{\partial e} g_n(\widetilde{u}_e, \widetilde{u}_{l(e)}) S'(\overline{\widetilde{u}_e})\overline{\varphi_e} = 0.$$

Durch Addition und Subtraktion von Termen folgt weiter

$$\begin{aligned} 0 &= (\partial_t \widetilde{u}_e + \nabla \cdot f(\widetilde{u}_e), S'(\widetilde{u}_e)\overline{\varphi_e})_e \\ &\quad + \Big(g_n(\widetilde{u}_e, \widetilde{u}_{l(e)}) - f(\widetilde{u}_e)\cdot n, S'(\widetilde{u}_e)\overline{\varphi_e}\Big)_{\partial e} \\ &\quad + (\partial_t \widetilde{u}_e + \nabla \cdot f(\widetilde{u}_e), (S'(\overline{\widetilde{u}_e}) - S'(\widetilde{u}_e))\overline{\varphi_e})_e \\ &\quad + \Big(g_n(\widetilde{u}_e, \widetilde{u}_{l(e)}) - f(\widetilde{u}_e)\cdot n, (S'(\overline{\widetilde{u}_e}) - S'(\widetilde{u}_e))\overline{\varphi_e}\Big)_{\partial e}. \end{aligned}$$

Das Lemma folgt nun mit der folgenden Identität

$$\begin{aligned} &(g_n(\widetilde{u}_e, \widetilde{u}_{l(e)}) - f(\widetilde{u}_e)\cdot n)S'(\widetilde{u}_e) = (g_n(\widetilde{u}_e, \widetilde{u}_{l(e)}) - g_n(\widetilde{u}_e, \widetilde{u}_e))S'(\widetilde{u}_e) \\ &= \int_{\widetilde{u}_e}^{\widetilde{u}_{l(e)}} \partial_w g_n(\widetilde{u}_e, w)S'(w)dw + \int_{\widetilde{u}_e}^{\widetilde{u}_{l(e)}} \partial_w g_n(\widetilde{u}_e, w)\big(S'(\widetilde{u}_e) - S'(w)\big)\mathrm{dw} \\ &= G_n(\widetilde{u}_e, \widetilde{u}_{l(e)}) - G_n(\widetilde{u}_e, \widetilde{u}_e) + \int_{\widetilde{u}_e}^{\widetilde{u}_{l(e)}} \partial_w g_n(\widetilde{u}_e, w)\int_w^{\widetilde{u}_e} S''(s)\mathrm{dsdw}\,. \end{aligned}$$

□

**Bemerkung 4.51.**

1) Aufgrund der Monotonie des numerischen Flusses und der Konvexität der Entropie folgt, dass der Dissipationsterm $D_e$ in der Zell-Entropieungleichung (4.23) positiv ist.
2) Ist $p = 0$, so gilt $I_{3,e}, I_{4,e} = 0$.
3) Für $p > 0$ ist die Ungleichung (4.23) keine Zell-Entropieungleichung im klassischen Sinne.

**Lemma 4.52** (Abschätzung des Entropieresiduums für $\tilde{u}_h$)**.**
*Seien $u_h$, $\widetilde{u}_h$ durch Definition 4.45 gegeben. Dann gilt für alle $\varphi \in C_0^1(\mathbb{R}^d \times (0,T), \mathbb{R}^+)$ die folgende Abschätzung des Entropieresiduums*

$$\langle R_S(\tilde{u}_h), \varphi\rangle \geq T_1 + T_2 + T_3 + T_4 + T_5 + T_6, \tag{4.24}$$

*mit*

$$\begin{aligned} T_1 &:= \iint\limits_{\mathbb{R}^d\times\mathbb{R}^+} \Big(\partial_t S(\widetilde{u}_h) + \nabla\cdot F_S(\widetilde{u}_h)\Big)\Big(\overline{\varphi_h} - \varphi\Big), \\ T_2 &:= \sum_{n=0}^{N-1} \int_{t^n}^{t^{n+1}} \sum_{e\in\mathcal{E}_{leaf}^{n,0}} \Big(G_n(\widetilde{u}_e, \widetilde{u}_{l(e)}) - F_S(\widetilde{u}_e)\cdot n, \overline{\varphi_e} - \varphi\Big)_{\partial e}, \\ T_3 &:= \sum_{n=0}^{N-1} \int_{t^n}^{t^{n+1}} \sum_{e\in\mathcal{E}_{leaf}^{n,0}} \Big(\partial_t \widetilde{u}_e + \nabla\cdot f(\widetilde{u}_e), (S'(\overline{\widetilde{u}_e}) - S'(\widetilde{u}_e))\overline{\varphi_e}\Big)_e, \\ T_4 &:= \sum_{n=0}^{N-1} \int_{t^n}^{t^{n+1}} \sum_{e\in\mathcal{E}_{leaf}^{n,0}} \Big(g_n(\widetilde{u}_e, \widetilde{u}_{l(e)}) - f(\widetilde{u}_e)\cdot n, (S'(\overline{\widetilde{u}_e}) - S'(\widetilde{u}_e))\overline{\varphi_e}\Big)_{\partial e}, \\ T_5 &:= \sum_{n=0}^{N-1} \sum_{e\in\mathcal{E}_{leaf}^{n,0}} \Big(\widetilde{u}_e(t^n) - \widetilde{u}_e^{n-1}(t^n), \int_0^1 S'(v_e(\theta))d\theta\ (\overline{\varphi_e}(t^n) - \varphi(t^n))\Big)_e, \\ T_6 &:= \sum_{n=0}^{N-1} \sum_{e\in\mathcal{E}_{leaf}^{n,0}} \Big(\widetilde{u}_e(t^n) - \widetilde{u}_e^{n-1}(t^n), (S'(\overline{\widetilde{u}_j^{n-1}(t^n)}) - \int_0^1 S'(v_e(\theta))d\theta)\ \overline{\varphi_e}(t^n)\Big)_e. \end{aligned}$$

*Dabei haben wir zur Definition von $T_5$ und $T_6$ die folgende Notation verwendet*

$$v_e(\theta) := \widetilde{u}_e^{n-1}(t^n) + \theta(\widetilde{u}_e(t^n) - \widetilde{u}_e^{n-1}(t^n)).$$

Der Term $T_1$ ist ein Elementresiduum, $T_2$ ein Sprungresiduum und $T_3$, $T_4$ können als Stabilitätsfehlerbeiträge angesehen werden, die nur bei einer Approximation mit höherer Ordnung entstehen. Die Terme $T_5$ und $T_6$ berücksichtigen den Projektionsfehler bei Gitteradaption.

*Beweis:* Zunächst summieren wir die Zellentropieungleichungen (4.23) über alle Entitäten $e \in \mathcal{E}_{\text{leaf}}^{0,n}$ und erhalten nach Integration über die Zeit

$$I_h \quad := \quad \sum_{n=0}^{N-1} \int_{t^n}^{t^{n+1}} \sum_{e \in \mathcal{E}_{\text{leaf}}^{n,0}} \Big( I_{1,e} + I_{2,e} + I_{3,e} + I_{4,e} \Big) \leq 0.$$

Nun wenden wir uns dem Entropieresiduum zu. Mit lokaler partieller Integration in der Zeit im Ort erhalten wir

$$\begin{aligned}
&\langle R_S(\widetilde{u}_h), \varphi \rangle = \\
&\quad - \sum_{n=0}^{N-1} \int_{t^n}^{t^{n+1}} \sum_{e \in \mathcal{E}_{\text{leaf}}^{n,0}} \Big[ \int_e \partial_t S(\widetilde{u}_e)\varphi + \nabla \cdot F_S(\widetilde{u}_e)\varphi + \int_{\partial e} F(\widetilde{u}_e) \cdot n\varphi \Big] \\
&\quad + \sum_{n=0}^{N-1} \sum_{e \in \mathcal{E}_{\text{leaf}}^{n,0}} \int_e \left( S(\widetilde{u}_e)(t^{n+1})\varphi(t^{n+1}) - S(\widetilde{u}_e)(t^n))\varphi(t^n) \right).
\end{aligned}$$

Aufgrund der Erhaltungseigenschaft der numerischen Flussfunktionen und der Stetigkeit von $\varphi$ gilt

$$\sum_{n=0}^{N-1} \int_{t^n}^{t^{n+1}} \sum_{e \in \mathcal{E}_{\text{leaf}}^{n,0}} \int_{\partial e} G_n(\widetilde{u}_e, \widetilde{u}_{l(e)})\varphi = 0.$$

Also erhalten wir durch Umordnung der Summanden

$$\begin{aligned}
&\langle R_S(\widetilde{u}_h), \varphi \rangle = \\
&\quad - \sum_{n=0}^{N-1} \int_{t^n}^{t^{n+1}} \sum_{e \in \mathcal{E}_{\text{leaf}}^{n,0}} \Big( \partial_t S(\widetilde{u}_e) + \nabla \cdot F_S(\widetilde{u}_e), \varphi \Big)_e \\
&\quad + \sum_{n=0}^{N-1} \int_{t^n}^{t^{n+1}} \sum_{e \in \mathcal{E}_{\text{leaf}}^{n,0}} \Big( G_n(\widetilde{u}_e, \widetilde{u}_{l(e)}) - F(\widetilde{u}_e) \cdot n, \varphi \Big)_{\partial e} \\
&\quad - \sum_{n=0}^{N-1} \sum_{e \in \mathcal{E}_{\text{leaf}}^{n,0}} \Big( S(\widetilde{u}_e)(t^n) - S(\widetilde{u}_e^{n-1})(t^n), \varphi(t^n) \Big)_e .
\end{aligned} \tag{4.25}$$

Im letzten Term haben wir bei der Umordnung die Integration im Ort als Summe über Integrale ueber $e \in \mathcal{E}_{\text{leaf}}^{0,n}$ geschrieben, obwohl $\widetilde{u}_h^{n-1}(t^n) \in V_{h,n-1}^p$ liegt. Demnach bezeichnen wir mit $\widetilde{u}_e^{n-1}$ die Einschränkung von $\widetilde{u}_h^{n-1}$ auf $e$, wobei in diesem Fall $\widetilde{u}_e^{n-1}$ Unstetigkeiten enthalten kann.

Aufgrund der Massenerhaltung der Limiter auf adaptiven Gittern gilt weiter

$$\sum_{n=0}^{N-1}\sum_{e\in\mathcal{E}_{\text{leaf}}^{n,0}}\Big(\widetilde{u}_e(t^n)-\widetilde{u}_e^{n-1}(t^n),S'(\overline{\widetilde{u}_e^{n-1}(t^n)})\overline{\varphi_e}(t^n)\Big)_e=0.$$

Mit der Definition von $v_e(\theta)$ können wir nun den letzten Summanden aus Gleichung (4.25) wie folgt umschreiben

$$\begin{aligned}
&-\sum_{n=0}^{N-1}\sum_{e\in\mathcal{E}_{\text{leaf}}^{n,0}}\Big(S(\widetilde{u}_e)(t^n)-S(\widetilde{u}_e^{n-1})(t^n),\varphi(t^n)\Big)_e\\
&=\ -\sum_{n=0}^{N-1}\sum_{e\in\mathcal{E}_{\text{leaf}}^{n,0}}\Big(S(\widetilde{u}_e)(t^n)-S(\widetilde{u}_e^{n-1})(t^n),\varphi(t^n)\Big)_e\\
&\quad+\sum_{n=0}^{N-1}\sum_{e\in\mathcal{E}_{\text{leaf}}^{n,0}}\Big((\widetilde{u}_e-\widetilde{u}_e^{n-1})(t^n),S'(\overline{\widetilde{u}_e^{n-1}(t^n)})\overline{\varphi_e}(t^n)\Big)_e\\
&=\ \sum_{n=0}^{N-1}\sum_{e\in\mathcal{E}_{\text{leaf}}^{n,0}}\Big((\widetilde{u}_e-\widetilde{u}_e^{n-1})(t^n),S'(\overline{\widetilde{u}_e^{n-1}(t^n)})\overline{\varphi_e}(t^n)-\int_0^1 S'(v_e(\theta))\mathrm{d}\theta\varphi(t^n)\Big)_e\\
&=\ \sum_{n=0}^{N-1}\sum_{e\in\mathcal{E}_{\text{leaf}}^{n,0}}\Big((\widetilde{u}_e-\widetilde{u}_e^{n-1})(t^n),\int_0^1 S'(v_e(\theta))\mathrm{d}\theta\ (\overline{\varphi_e}(t^n)-\varphi(t^n))\Big)_e\\
&\quad+\sum_{n=0}^{N-1}\sum_{e\in\mathcal{E}_{\text{leaf}}^{n,0}}\Big((\widetilde{u}_e-\widetilde{u}_e^{n-1})(t^n),(S'(\overline{\widetilde{u}_e^{n-1}(t^n)})-\int_0^1 S'(v_e(\theta))\mathrm{d}\theta)\ \overline{\varphi_e}(t^n)\Big)_e.
\end{aligned}$$

Schließlich erhalten wir durch einen Vergleich des Entropieresiduums mit $I_h$ das Resultat des Lemmas wie folgt

$$\langle R_S(u_h),\varphi\rangle\ \geq\ \langle R_S(u_h),\varphi\rangle+I_h=T_1+T_2+T_3+T_4+T_5+T_6.$$

□

Wir wollen nun das Entropieresiduum mit Hilfe des gerade bewiesenen Lemmas weiter abschätzen, um die fundamentale Fehlerabschätzung anwenden zu können. Dazu betrachten wir die Terme $T_i$ aus (4.24).

**Lemma 4.53** (Abschätzung des Entropieresiduums)**.**
*Für die Schranke des Entropieresiduums aus Lemma 4.52 gilt die folgende Abschätzung*

$$\begin{aligned}
|T_1+T_2+T_5|\ &\leq\ \|S'\|_{L^\infty}\|\nabla\varphi\|_{L^\infty}\sum_{n=0}^{N-1}\sum_{e\in\mathcal{E}_{leaf}^{n,0}}\eta_{1,e},\\
|T_3+T_4+T_6|\ &\leq\ \|S''\|_{L^\infty}\|\varphi\|_{L^\infty}\sum_{n=0}^{N-1}\sum_{e\in\mathcal{E}_{leaf}^{n,0}}\eta_{2,e}.
\end{aligned}$$

*Dabei sind die Fehlerindikatoren $\eta_{1,e},\eta_{2,e}$ für $e\in\mathcal{E}_{leaf}^{0,n}$ wie folgt definiert*

$$\begin{aligned}
\eta_{1,e} &:= h_e R_{E,e} + \frac{1}{2} h_{e,l(e)} R_{S,e} + h_e R_{P,e}, \\
\eta_{2,e} &:= \|\overline{\widetilde{u}_e} - \widetilde{u}_e\|_{L^\infty((t^n,t^{n+1})\times e)} R_{E,e} \\
&\quad + \frac{1}{2} \max_{k\in\{e,l(e)\}} \|\overline{\widetilde{u}_k} - \widetilde{u}_k\|_{L^\infty((t^n,t^{n+1})\times \bar{e}\cap\overline{l(e)})} R_{S,e} \\
&\quad + \|\overline{\widetilde{u}_e^{n-1}}(t^n) - \widetilde{u}_e^{n-1}(t^n)\|_{L^\infty(e)} R_{P,e}.
\end{aligned}$$

*Dabei verwenden wir folgende Notation für die Element-, Sprung- und Projektionsresiduen*

$$\begin{aligned}
R_{E,e} &:= \int_{t^n}^{t^{n+1}} \int_e \left| \partial_t \widetilde{u}_e + \nabla \cdot f(\widetilde{u}_e) \right|, \\
R_{S,e} &:= \int_{t^n}^{t^{n+1}} \sum_{l(e)} \int_{e^1} Q_n(\widetilde{u}_e, \widetilde{u}_{l(e)}) |\widetilde{u}_e - \widetilde{u}_{l(e)}|, \\
R_{P,e} &:= \int_e |\widetilde{u}_e^n(t^{n+1}) - \widetilde{u}_e^{n+1}(t^{n+1})|,
\end{aligned}$$

$$\begin{aligned}
h_{e,l(e)} &:= \max_{l(e)} \operatorname{diam}(e \cup l(e)) \\
\textit{und} \quad Q_n(u,v) &:= \frac{2g_n(u,v) - g_n(u,u) - g_n(v,v)}{u - v}.
\end{aligned}$$

*Beweis:* Ziel der weiteren Abschätzung der Terme $T_i$ ist es zusätzliche $h$-Potenzen aus Differenzen der Testfunktion zu gewinnen. Für jede $h$-Potenz müssen wir jedoch mit höheren Ableitungen von $\varphi$ oder $S$ bezahlen. Um die fundamentale Fehlerabschätzung anwenden zu können, werden wir uns auf erste Ableitungen von $\varphi$ und zweite Ableitungen von $S$ einschränken.

Abschätzung von $T_1$:
Um diesen Term abzuschätzen, benötigen wir lokale Abschätzungen für die Differenz der Testfunktionen. Da $\Pi_{V_h^0}$ exakt auf Polynomen vom Grad $p = 0$ ist, erhalten wir $|(\Pi_{V_h^0}(\varphi)(x) - \varphi(x))|_e| \leq h_e \|\nabla\varphi\|_{L^\infty(e)}$, was schließlich zum Abschätzung von $T_1$ führt.

Abschätzung von $T_2$:
Durch Umsortierung der Summation im Ort erhalten wir

$$\begin{aligned}
T_2 = \int_0^T \sum_{\substack{e^1 \in \Gamma \\ e^1 = \bar{e}\cap\overline{l(e)}}} \int_{e^1} & (G_n(\widetilde{u}_e, \widetilde{u}_{l(e)}) - G_n(\widetilde{u}_e, \widetilde{u}_e))(\overline{\varphi_e} - \varphi) \\
& - (G_n(\widetilde{u}_e, \widetilde{u}_{l(e)}) - G_n(\widetilde{u}_{l(e)}, \widetilde{u}_{l(e)}))(\overline{\varphi_{l(e)}} - \varphi).
\end{aligned}$$

Dies führt mit der Monotonie der numerischen Flussfunktionen $g_n$ zu folgender Abschätzung

$$\begin{aligned}|T_2| &\leq \|S'\|_{L^\infty}\|\nabla\varphi\|_{L^\infty}\sum_{n=0}^{N-1}\int_{t^n}^{t^{n+1}}\sum_{\substack{e^1\in\Gamma\\ e^1=\bar{e}\cap\overline{l(e)}}} h_{e,l(e)}\int_{e^1} Q_n(\widetilde{u}_e,\widetilde{u}_{l(e)})|\widetilde{u}_e-\widetilde{u}_{l(e)}|\\ &\leq \|S'\|_{L^\infty}\|\nabla\varphi\|_{L^\infty}\sum_{n=0}^{N-1}\sum_{e\in\mathcal{E}^{n,0}_{\text{leaf}}}\frac{1}{2}\int_{t^n}^{t^{n+1}}\sum_{l(e)} h_{e,l(e)}\int_{e^1} Q_n(\widetilde{u}_e,\widetilde{u}_{l(e)})|\widetilde{u}_e-\widetilde{u}_{l(e)}|.\end{aligned}$$

Analog zu den Abschätzungen für $T_1, T_2$ erhalten wir folgende Abschätzungen für $T_3$ und $T_4$:

$$\begin{aligned}|T_3| &\leq \|S''\|_{L^\infty}\|\varphi\|_{L^\infty}\sum_{n=0}^{N-1}\sum_{e\in\mathcal{E}^{n,0}_{\text{leaf}}}\int_{t^n}^{t^{n+1}}\|\overline{\widetilde{u}_e}-\widetilde{u}_e\|_{L^\infty(T_e)}\int_e|\partial_t\widetilde{u}_h+\nabla\cdot f(\widetilde{u}_h)|,\\ |T_4| &\leq \|S''\|_{L^\infty}\|\varphi\|_{L^\infty}\sum_{n=0}^{N-1}\sum_{e\in\mathcal{E}^{n,0}_{\text{leaf}}}\frac{1}{2}\int_{t^n}^{t^{n+1}}\sum_{l(e)}\max_{k\in\{e,l(e)\}}\|\overline{\widetilde{u}^n_k}-\widetilde{u}^n_k\|_{L^\infty(S_{jl})}\\ &\qquad\int_{e^1} Q_n(\widetilde{u}_e,\widetilde{u}_{l(e)})|\widetilde{u}_e-\widetilde{u}_{l(e)}|.\end{aligned}$$

Abschätzung von $T_5$ und $T_6$:

$$\begin{aligned}|T_5| &\leq \|S'\|_{L^\infty}\|\nabla\varphi(t^n)\|_{L^\infty(T_e)}\sum_{n=0}^{N-1}\sum_{e\in\mathcal{E}^{n,0}_{\text{leaf}}} h_e\int_e|\widetilde{u}_e(t^n)-\widetilde{u}_e^{n-1}(t^n)|,\\ |T_6| &\leq \|S''\|_{L^\infty}\|\varphi(t^{n+1})\|_{L^\infty(T_e)}\\ &\quad\sum_{n=0}^{N-1}\sum_{e\in\mathcal{E}^{n,0}_{\text{leaf}}}\|\overline{\widetilde{u}_e^{n-1}}(t^n)-\widetilde{u}_e^{n-1}(t^n)\|_{L^\infty(T_e)}\int_e|\widetilde{u}_e(t^n)-\widetilde{u}_e^{n-1}(t^n)|.\end{aligned}$$

Für die letzte Abschätzung haben wir verwendet, dass gilt

$$\begin{aligned}&|S'(\overline{\widetilde{u}_e^{n-1}(t^n)})-\int_0^1 S'(v_e(\theta))\mathrm{d}\theta|\\ &\qquad\leq\|S''\|_{L^\infty}\int_0^1|\overline{\widetilde{u}_e^{n-1}(t^n)}-\widetilde{u}_e^{n-1}(t^n)+\theta(\widetilde{u}_e(t^n)-\widetilde{u}_e^{n-1}(t^n))|\mathrm{d}\theta.\end{aligned}$$

Dies liefert die Abschätzung für $T_6$, da der Integrand monoton fallend in $\theta$ ist. □

**Theorem 4.54** (A-posteriori-Fehlerabschätzung)**.**
*Sei $u_h$ die Lösung der semidiskreten DG-Methode auf adaptiven Gittern* (4.17)-(4.18) *und $u$ die Entropielösung von* (4.11a), (4.11b) *mit $f\in C^2(\mathbb{R})$. Dann gilt die folgende A-posteriori-Fehlerabschätzung*

$$\|(u-u_h)(T)\|_{L^1(B_R(x_0))} \leq \|(\widetilde{u}_h-u_h)(T)\|_{L^1(B_R(x_0))}+\eta_h$$

*mit*

$$\eta_h := \eta_0+\sqrt{K_1\eta_1}+\sqrt{K_2\eta_2}.$$

*Dabei sind die globalen Indikatoren gegeben durch*

$$\eta_0 := \sum_{\substack{e \in \mathcal{E}_{leaf}^{0,0} \\ e \subset B_{R+L_f T}(x_0)}} \eta_{0,e}, \qquad \eta_i := \sum_{n=0}^{N-1} \sum_{\substack{e \in \mathcal{E}_{leaf}^{n,0} \\ e \subset D_{2h}}} \eta_{i,e}, i = 1, 2.$$

*Die lokalen Indikatoren $\eta_{1,e}$, $\eta_{2,e}$ sind in Lemma 4.53 definiert und $\eta_{0,e}$ ist gegeben als*

$$\eta_{0,e} := \int_e |u_0 - \widetilde{u}_e(0)|$$

*$K_1, K_2$ sind Konstanten, die im Beweis genauer angegeben werden.*

*Beweis:* Die Lemmata 4.52 und 4.53 zeigen, dass $\widetilde{u}_h$ die Voraussetzungen für die fundamentale Fehlerabschätzung 4.49 erfüllt mit $\mu_{v,\delta}^t = 0$ und $\mu_{v,\delta}, \mu_{v,\delta}^x$ definiert durch

$$\begin{aligned} \langle \mu_{v,\delta}, \Phi \rangle &:= \|S_\delta''\|_{L^\infty} \sum_{n=0}^{N-1} \int_0^T \sum_{e \in \mathcal{E}_{\text{leaf}}^{n,0}} \int_e \frac{1}{\Delta t \ |e|} \eta_{2,e} \Phi, \\ \langle \mu_{v,\delta}^x, \Phi \rangle &:= \sum_{n=0}^{N-1} \int_0^T \sum_{e \in \mathcal{E}_{\text{leaf}}^{n,0}} \int_e \frac{1}{\Delta t \ |e|} \ \eta_{1,e} \Phi. \end{aligned}$$

Mit der Definition von $S_\delta$ gilt weiter $\|S''\|_{L^\infty} \leq K_S \frac{1}{\delta}$. Die A-posteriori-Fehlerabschätzung folgt nun mit der fundamentalen Fehlerabschätzung 4.49 mit $K_1 := C_2^2$, $K_2 := 4 K_S C_0 C_1$ und folgender optimaler Wahl von $\delta$

$$\delta := \sqrt{\sum_{n=0}^{N-1} \sum_{e \in \mathcal{E}_{\text{leaf}}^{n,0}} \eta_{2,j}^n} \ \sqrt{\frac{K_S \ C_1}{C_0}}.$$

□

**Bemerkung 4.55** (Verwendung des Fehlerschätzers und adaptive Verfahren)**.**

1) Zur Zeitdiskretisierung des limitierten DG-Verfahrens eignen sich besonders die expliziten SSP-Runge-Kutta-Verfahren aus Abschnitt 4.2.1. Eine A-posteriori-Fehlerabschätzung für den voll-diskreten Fall steht zur Zeit noch aus. Um dennoch den Fehlerschätzer für die Adaptivität nutzen zu können, kann die Natural Continuous Extension der SSP-RK-Verfahren verwendet werden. Dies ermöglicht insbesondere eine Auswertung von $\partial_t \tilde{u}_h$ in den Fehlerindikatoren.

2) Während die Element-, Sprung- und Projektionsresiduen ($R_{E,e}$, $R_{S,e}$, $R_{P,e}$) zur adaptiven Steuerung der lokalen Gitterweite herangezogen werden können ($h$-Adaption), eignen sich die Oszillationsindikatoren $\|\widetilde{\widetilde{u}_e} - \widetilde{u}_e\|_{L^\infty}$ vor allem zur Konstruktion geeigneter massenerhaltender Limiter ($p$-Adaption). Die $p$-Adaption wird dabei so gesteuert, dass das Verfahren einerseits stabil wird, andererseits aber seine optimale Konvergenz behält.

3) Für eine detaillierte Darstellung eines $hp$-adaptiven Algorithmus und numerische Resultate verweisen wir auf die Originalarbeit [10].

### 4.3.3 LDG-Verfahren für die Wärmeleitungsgleichung

In diesem Abschnitt wollen wir uns mit Discontinuous-Galerkin-Verfahren für die lineare Wärmeleitungsgleichung beschäftigen. Insbesondere werden wir dabei wieder auf A-posteriori-Fehlerabschätzungen eingehen, wobei wir dem Zugang von Georgoulis und Lakkis [13] folgen.

**Definition 4.56** (Wärmeleitungsgleichung)**.**
Seien $\Omega \subset \mathbb{R}^d$ ein polygonal berandetes Gebiet und $T > 0$ gegeben. Seien weiter

$$f \in L^\infty(0,T;L^2(\Omega)) \quad \text{und} \quad u_0 \in H^1(\Omega)$$

gegeben. Dann ist $u \in L^2(0,T;H^1(\Omega))$ mit $\partial_t u \in L^\infty(0,T;H^-1(\Omega)$ eine Lösung des linearen parabolischen Anfangs-Randwertproblems, falls im schwachem Sinne gilt

$$\partial_t u - \Delta u) = f, \qquad \text{auf } \Omega \times (0,T], \tag{4.26a}$$
$$u(\cdot,0) = u_0 \qquad \text{auf } \Omega \tag{4.26b}$$
$$u(x,t) = 0 \qquad \text{auf } \partial\Omega \times (0,T]. \tag{4.26c}$$

Zur Definition allgemeiner semidiskreter Local Discontinuous-Galerkin-Verfahren gehen wir nun analog zu Kapitel 2 vor und betrachten zunächst das folgende System von Differentialgleichungen erster Ordnung für das Paar $(u,\sigma)$:

$$\begin{aligned} \partial_t u - \nabla \cdot \sigma &= f, \\ \sigma &= a\nabla u, \end{aligned}$$

Durch eine schwache Formulierung analog zu Kapitel 2 und geeignete Wahl von numerischen Flüssen gelangen wir dann mit den Notationen aus Kapitel 2 zu folgender primalen Formulierung des LDG-Verfahrens.

**Definition 4.57** (Primale Formulierung des semidiskreten LDG-Verfahrens)**.**
Sei $\mathcal{T}_h$ ein hierarchisches Gitter auf $\Omega \subset \mathbb{R}^n$ und seien konsistente numerische Flüsse $\hat{u}, \hat{\sigma}$ auf $\mathcal{T}_h$ gegeben und $V(h) := V_h + H^2(\Omega) \cap H_0^1(\Omega)$ wie in Kapitel 2 definiert. Wir definieren die Bilinearform

$$B_{(\hat{u},\hat{\sigma})} : V(h) \times V(h) \longrightarrow \mathbb{R}$$

durch:

$$\begin{aligned} B_{(\hat{u},\hat{\sigma})}(u,\varphi) \quad := \quad & \int_\Omega \nabla_h u \cdot \nabla_h \varphi \\ & + \int_{\Gamma \setminus \partial\Omega} ([\hat{u}(u) - u]\{\nabla_h \varphi \cdot n\} - \{\hat{\sigma}(u, s(u)) \cdot n\}[\varphi]) \\ & + \int_{\Gamma \setminus \partial\Omega} (\{\hat{u}(u) - u\}[\nabla_h \varphi \cdot n] - [\hat{\sigma}(u, s(u)) \cdot n]\{\varphi\}) \\ & + \int_{\partial\Omega} (\hat{u}(u) - u)\nabla_h \varphi \cdot n - \hat{\sigma}(u, s(u)) \cdot n\varphi, \end{aligned}$$

mit $s(u) := \nabla_h u - r([\hat{u} - u]) - l(\{\hat{u} - u\})$ und den Lift-Operatoren $r, l : L^2(\Gamma) \to [H^2(\mathcal{T}_h)]^n$ aus Lemma 2.13.

Dann heißt $u_h \in C^{0,1}(0,T;V_h)$ semidiskrete LDG Approximation der Wärmeleitungsgleichung (4.26) mit den numerischen Flüssen $\hat{u}, \hat{\sigma}$, falls gilt

$$(\partial_t u_h, v_h) + B_{(\hat{u},\hat{\sigma})}(u_h, v_h) = (f, v_h), \qquad \forall v_h \in V_h, t \in [0,T], \tag{4.27a}$$

$$u_h(\cdot, 0) = \Pi_{V_h}(u_0). \tag{4.27b}$$

Die Approximation $\sigma_h \in C^{0,1}(0,T;\Sigma_h)$ des Flusses $\sigma$ ist gegeben durch

$$\sigma_h = s(u_h). \tag{4.28}$$

In diesem Kapitel werden wir für die Bilinearform auch abkürzend $B(u,v)$ anstelle von $B_{(\hat{u},\hat{\sigma})}(u,v)$ schreiben.

Die Konsistenz der numerischen Flussfunktionen impliziert die Konsistenz der Bilinearform $B = B_{(\hat{u},\hat{\sigma})}$, so dass für die exakte Lösung $u$ der Wärmeleitungsgleichung gilt

$$(\partial_t u, v) + B(u,v) = (f,v), \qquad \forall v \in H_0^1(\Omega), t \in [0,T], \tag{4.29a}$$

$$u(\cdot, 0) = (u_0). \tag{4.29b}$$

Wir wollen nun eine A-posteriori Fehlerabschätzung für den Fehler

$$\|u - u_h\|_{L^2(0,T;V(h))} := \Big( \int_0^T |\!|\!| u(\cdot,t) - u_h(\cdot,t) |\!|\!|^2 \Big)^{1/2}$$

herleiten. Dabei bezeichne $|\!|\!| \cdot |\!|\!|$ die DG-Energienorm aus Definition 2.19. Bei der Fehlerabschätzung werden wir den Fehler in einen elliptischen Anteil und einen parabolischen Anteil zerlegen. Dazu benötigen wir die elliptische Rekonstruktion von $u_h$ und den diskreten DG-Operator $A_h$.

**Definition 4.58** (Elliptische Rekonstruktion und diskreter DG-Operator)**.**
Sei $t \in [0,T]$ gegeben. Dann ist der diskrete DG-Operator $A_h : V_h \to V_h$ für alle $v_h \in V_h$ definiert durch

$$(A_h v_h, \varphi_h) = B(v_h, \varphi_h), \qquad \forall \varphi_h \in V_h.$$

Weiter definieren wir den Rekonstruktionsoperator $\mathcal{R}^t : V_h \to H_0^1(\Omega)$ durch

$$B(\mathcal{R}^t v_h, \varphi) = (A_h v_h - \Pi_{V_h} f(\cdot,t) + f(\cdot,t), \varphi), \qquad \forall \varphi \in H_0^1(\Omega). \tag{4.30}$$

Dabei bezeichne $\Pi_{V_h} : L^2(\Omega) \to V_h$ die $L^2$-Projektion.
Für $u_h \in C^{0,1}(0,T;V_h)$ bezeichnen wir mit $\mathcal{R}u_h$ die elliptische Rekonstruktion von $u_h$ definiert durch

$$(\mathcal{R}u_h)(\cdot,t) := \mathcal{R}^t u_h(\cdot,t).$$

Die Haupteigenschaft der Rekonstruktion $\mathcal{R}u_h$ ist, dass die DG-Approximation $u_h$ gleichzeitig eine DG-Approximation des stationären Randwertproblems (4.30) ist, da für $\varphi_h \in V_h$ gilt

$$(A_h u_h(\cdot,t) - \Pi_{V_h} f(\cdot,t) + f(\cdot,t), \varphi_h) = (A_h u_h(\cdot,t), \varphi_h) = B(u_h(\cdot,t), \varphi_h).$$

**Definition 4.59** (Elliptischer und parabolischer Fehler)**.**
Wir zerlegen den Fehler

$$e := u_h - u$$

in den elliptischen Anteil

$$\varepsilon := \mathcal{R}u_h - u_h$$

und den parabolischen Fehleranteil

$$\rho := \mathcal{R}u_h - u.$$

Dabei bezeichnet $\mathcal{R}u_h \in H_0^1(\Omega)$ die elliptische Rekonstruktion von $u_h$. Es gilt

$$e = \rho - \varepsilon.$$

**Lemma 4.60** (Fehleridentität)**.**
*Sei $u$ die schwache Lösung der Wärmeleitungsgleichung (4.26) und $u_h$ die semidiskrete LDG-Approximation aus Definition 4.57. Dann gilt mit der Notation aus Definition 4.59*

$$\langle \partial_t e, \varphi\rangle + B(\rho, \varphi) = 0, \qquad \text{für alle } \varphi \in H_0^1(\Omega). \tag{4.31}$$

*Beweis:* Für $\varphi \in H_0^1(\Omega)$ gilt

$$\begin{aligned}
\langle \partial_t e, \varphi\rangle + B(\rho, \varphi) &= \langle \partial_t(u_h - u), \varphi\rangle + B(\mathcal{R}u_h - u, \varphi) \\
&= \langle \partial_t u_h, \varphi\rangle + B(\mathcal{R}u_h, \varphi) - (f, \varphi) \\
&= \langle \partial_t u_h, \varphi\rangle + (A_h u_h - \Pi_{V_h} f + f, \varphi) - (f, \varphi) \\
&= \langle \partial_t u_h, \Pi_{V_h}\varphi\rangle + (A_h u_h - \Pi_{V_h} f, \Pi_{V_h}\varphi)) \\
&= \langle \partial_t u_h, \Pi_{V_h}\varphi\rangle + B(u_h, \Pi_{V_h}\varphi) - (f, \Pi_{V_h}\varphi) \\
&= 0.
\end{aligned}$$

□

**Definition 4.61** (Zerlegung in konformen und nicht-konformen Anteil)**.**
Für die folgende Analysis nehmen wir an, dass die DG-Approximation $u_h(\cdot, t) \in V_h$ in einen konformen und nicht-konformen Anteil zerlegt werden kann, d.h. es existieren $u_c(\cdot, t) \in H_0^1(\Omega) \cap V_h$ und $u_d(\cdot, t) \in V_h$ mit

$$u_h(\cdot, t) = u_c(\cdot, t) + u_d(\cdot, t).$$

Wir definieren damit die konformen Fehleranteile

$$e_c := e - u_d = u_c - u \in H_0^1(\Omega), \qquad \varepsilon_c := \varepsilon + u_d = \mathcal{R}u_h - u_c \in H_0^1(\Omega).$$

**Theorem 4.62** (Abstrakte A-posteriori-Fehlerabschätzung)**.**
*Sei $u$ die schwache Lösung der Wärmeleitungsgleichung (4.26) und $u_h$ die semidiskrete LDG-Approximation aus Definition 4.57. Wir nehmen an, dass die Bilinearform $B = B_{(\hat{u},\hat{\sigma})}$ stetig und stabil bezüglich der DG-Norm $|||\cdot|||$ sei und dass eine Zerlegung nach Definition 4.61 existiere. Dann gilt die folgende A-posteriori-Fehlerabschätzung*

$$\begin{aligned}
\|u - u_h\|_{L^2(0,T;V(h))} \leq\ & \sqrt{3}\|\mathcal{R}u_h - u_h\|_{L^2(0,T;V(h))} \\
& + \sqrt{3}\|u_d\|_{L^2(0,T;V(h))} \\
& + \sqrt{3}\|\partial_t u_d/\alpha\|_{L^2(0,T;H^{-1}(\Omega))} \\
& + \|u_0 - u_c(0)\|_{L^2(\Omega)}.
\end{aligned} \tag{4.32}$$

*Beweis:* Wir starten von der Fehleridentität in Lemma 4.60 und wählen als Testfunktion $\varphi = e_c$. Dann folgt mit Definition 4.61

$$\begin{aligned}\langle \partial_t e_c, e_c\rangle + B(\rho,\rho) &= \langle \partial_t(e_c+u_d), e_c\rangle + B(\rho, e_c)\\ &\quad -\langle \partial_t u_d, e_c\rangle + B(\rho, \rho - e_c)\\ &= -\langle \partial_t u_d, e_c\rangle + B(\rho,\varepsilon_c).\end{aligned}$$

Mit der Stabilität und Stetigkeit von $B$ folgt dann

$$\frac{1}{2}\frac{d}{dt}\|e_c\|^2 + |\!|\!|\rho|\!|\!|^2 \le |\!|\!|\rho|\!|\!|\ |\!|\!|\varepsilon_c|\!|\!| + \|\partial u_d\|_{H^{-1}(\Omega)}\|\nabla e_c\|.$$

Mit $e_c = \rho - \varepsilon_c$ folgt

$$\|\nabla e_c\| \le |\!|\!|e_c|\!|\!| = |\!|\!|\rho - \varepsilon_c|\!|\!| \le |\!|\!|\rho|\!|\!| + |\!|\!|\varepsilon_c|\!|\!|.$$

und somit erhalten wir mit $I_1 := |\!|\!|\varepsilon_c|\!|\!|$ und $I_2 := \|\partial u_d\|_{H^{-1}(\Omega)}$

$$\frac{1}{2}\frac{d}{dt}\|e_c\|^2 + |\!|\!|\rho|\!|\!|^2 \le |\!|\!|\rho|\!|\!|\Big(I_1 + I_2\Big) + I_1 I_2.$$

Mit der Youngschen Ungleichung folgt

$$\frac{d}{dt}\|e_c\|^2 + |\!|\!|\rho|\!|\!|^2 \le 3\Big(I_1^2 + I_2^2\Big).$$

Integration über $[0,T]$ ergibt nun

$$\|e_c(T)\|^2 + \int_0^T |\!|\!|\rho|\!|\!|^2 \le \|e_c(0)\|^2 + 3\int_0^T \Big(I_1^2 + I_2^2\Big).$$

Mit $\varepsilon_c = (\mathcal{R}u_h - u_h) + u_d$ erhalten wir nun das Theorem durch Wurzel-ziehen und Dreiecksungleichung. □

Die Fehlerbeiträge $\|u_d\|_{L^2(0,T;V(h))}$ und $\|\partial_t u_d/\alpha\|_{L^2(0,T;H^{-1}(\Omega))}$ können bei geeigneter Wahl der Zerlegung $u_h = u_c + u_d$ durch Sprungindikatoren von $u_h$ abgeschätzt werden. Genauer gilt folgendes Lemma.

**Lemma 4.63** (Abschätzung des nicht-konformen Anteils)**.**
*Sei $\mathcal{T}_h$ eine simpliziales Gitter in zwei oder drei Raumdimensionen. Dann existiert für jedes $v_h \in V_h$ eine konforme Approximation $v_c \in V_h \cap H_0^1(\Omega)$, so dass gilt*

$$\|v_h - v_c\|_{L^2(\Omega)} \le C_1 \|\sqrt{h}[v_h]\|_{L^2(\Gamma)}, \tag{4.33}$$

*und*

$$\|\nabla_h(v_h - v_c)\|_{L^2(\Omega)} \le C_2 \|\sqrt{h^{-1}}[v_h]\|_{L^2(\Gamma)}. \tag{4.34}$$

*Dabei hängen die Konstanten $C_1, C_2 > 0$ von der Regularität des Gitters ab*

*Beweis:* Für den Beweis verweisen wir auf [15, Theorem 2.2 und 2.3]. □

**Bemerkung 4.64.**

1) Die Fehlerabschätzung in Theorem 4.62 nennen wir abstrakt, da auf der rechten Seite noch die elliptische Rekonstruktion $\mathcal{R}u_h$ auftaucht. Da $u_h(\cdot,t)$ die Discontinuous-Galerkin-Approximation von $\mathcal{R}u_h(\cdot,t)$ für das elliptische Problem (4.30) ist, erhalten wir eine brauchbare A-posteriori-Fehlerschranke, sobald wir eine A-posteriori-Fehlerabschätzung für das elliptische Problem kennen (vgl. Kapitel 2).

2) Für weitere Details und eine A-posteriori-Fehlerabschätzung für voll diskrete DG-Approximationen verweisen wir auf die Originalarbeit [13].

### 4.3.4 LDG-Verfahren für allgemeinere Evolutionsgleichungen

Wir betrachten nun als verallgemeinertes parabolische Problem eine nichtlineare Advektions-Diffunsionsgleichung der Form

$$\partial_t u + \nabla \cdot (f(u) - d(u)\nabla\sigma(u)) - S(u) = 0.$$

Auch hier schreiben wir zunächst die Differentialgleichung zweiter Ordnung als System erster Ordnung. Dabei haben wir jedoch mehrere Möglichkeiten, von denen wir zwei exemplarisch aufzeigen wollen:

**Ansatz 1:**

$$\begin{aligned} u_1 &= -\nabla\sigma(u) \\ u_2 &= S(u) - \nabla \cdot (f(u) + d(u)u_1) \\ \partial_t u &= u_2 \end{aligned}$$

**Ansatz 2:**

$$\begin{aligned} u_1 &= \sigma(u) \\ u_2 &= -d(u)\nabla u_1 \\ u_3 &= S(u) - \nabla \cdot (f(u) + u_2) \\ \partial_t u &= u_3 \end{aligned}$$

In beiden Fällen besteht die Formulierung aus einer Folge von Gleichungen desselben Typs

$$\begin{aligned} u_k = S_k(u_{k-1},\dots,u_0) - \nabla \cdot f_k(u_{k-1},\dots,u_0) \\ - \sum_{l=0}^{k-1} a_{kl}(u_{k-1},\dots,u_0)\nabla u_l \end{aligned} \tag{H}$$

für $k = 1,\dots K, u_0 \equiv u$. Hinzu kommt eine Evolutionsgleichung vom Typ

$$\partial_t u = F(u_K,\dots,u_0).$$

Für **Ansatz 1** erhalten wir beispielsweise ($K = 2$):

$$\begin{aligned} f_1(u_0) &= \sigma(u_0)Id, & S_1 &= a_0 = 0, \\ f_2(u_1,u_0) &= f(u_0) + d(u_0)u_1, & S_2(u_1,u_0) &= S(u_0), a_{20} = a_{21} = 0, \\ F(u_2,u_1,u_0) &= u_2. \end{aligned}$$

**Bemerkung 4.65** (Weitere Zerlegungen)**.**
Es ist noch eine andere Zerlegung sinnvoll,

$$\begin{aligned} v_0 &= u_0, & v_1 &= -\nabla \cdot f(u_0) \\ w_0 &= u_0, & w_1 &= \nabla\sigma(w_0), w_2 = \nabla(d(w_0)w_1) \\ \partial_t u &= S(u) + v_1 + w_2 \end{aligned}$$

In diesem Fall haben wir den konvektiven und diffusiven Anteil getrennt. Die Formulierung erlaubt es dann $v_1$ in der Zeit explizit zu diskretisieren, und $w_2$ implizit (IMEX-Verfahren), was zu günstigen Stabilitätseigenschaften führt.

**Definition 4.66** (Allgemeine Formulierung für LDG-Verfahren)**.**

Wir betrachten eine Evolutionsgleichung der Form

$$\partial_t u + L[u] = 0$$

mit einem Ortsoperator $L$. Wir setzen $u_0 := u$ und nehmen an, dass $L$ wie folgt zerlegt werden kann:

$$\begin{aligned} (u_i, \dots, u_0) &:= L_i[u_{i-1}, \dots, u_0], \quad i = 1, \dots, K, \\ L_i &: (u_{i-1}, \dots, u_0) \longrightarrow (u_i, \dots, u_0), \end{aligned}$$

so dass

$$L[u_0] = -F(u_K, \dots, u_0) = -F(L_K[L_{K-1}[\dots L_1[u_0]\dots]]).$$

In dieser Form gilt für die Lösung $u$

$$\partial_t u = F(u_K, \dots, u_1, u_0).$$

Die Operatoren $L_i$ seien dabei vom Typ (H).

**Beispiel 4.67** (Gleichung dritter Ordnung)**.**
Wir betrachten als vereinfachtes Modell der Navier-Stokes-Korteweg-Gleichung folgende partielle Differentialgleichung dritter Ordnung.

$$\partial_t u + \partial_x f(u) = \varepsilon \partial_x^2 u + \eta \partial_x^3 u.$$

Dann ist eine allgemeine Formulierung im Sinne von Definition 4.66 wie folgt gegeben

$$\begin{aligned} u_0 &= u, \\ (u_1, u_0) &= L_1[u_0] := (\partial_x u_0, u_0), \\ (u_2, u_1, u_0) &= L_2[u_1, u_0] := (\partial_x u_1, u_1, u_0), \\ F(u_2, u_1, u_0) &:= \partial_x(-f(u_0) + \varepsilon u_1 + \eta u_2), \\ \partial_t u &= F(u_2, u_1, u_0). \end{aligned}$$

Die Gleichungen sind jetzt in der Form von Def. 4.66 und alle $L_k$ sind vom Typ (H).

**Beispiel 4.68** (System zweiter Ordnung)**.**
Wie betrachten folgendes System für $(u, p)$:

$$\begin{aligned} \partial_t u + \nabla \cdot f(u) &= \nabla p, \\ \Delta p &= S(u). \end{aligned}$$

Dann erhalten wir mit Elementaroperatoren vom Type (H) und inversem Laplaceoperator die Form aus Definition 4.66 wie folgt:

$$\begin{aligned} L_1[u_0] &:= (\Delta^{-1} S(u_0), u_0), \\ L_2[u_1, u_0] &:= (-\nabla \cdot (f(u_0) + u_1\ I), u_1, u_0) \\ F(u_2, u_1, u_0) &:= u_2, \\ \partial_t u &= F(u_2, u_1, u_0). \end{aligned}$$

Dabei muss zur Berechnung von $u_1$ ein elliptisches Problem gelöst werden.

**Lemma 4.69** (Variationsformulierung für (H))**.**
*Sei $\mathcal{T}_h$ ein Gitter auf $\Omega$ und $\varphi \in H^2(\mathcal{T}_h)$ eine Testfunktion. Dann lautet die schwache Formulierung von* (H)

$$\begin{aligned} \sum_{e \in \mathcal{E}^0_{leaf}} \int_e u_k \varphi = \sum_{e \in \mathcal{E}^0_{leaf}} \int_e \left( S_k - \sum_{l=0}^{k-1} a_{kl} \nabla u_l \right) \varphi + f_k \nabla \varphi \\ - \sum_{e \in \mathcal{E}^0_{leaf}} \int_{\partial e} \left( \tilde{f}_k \cdot n \varphi + \sum_{l=0}^{k-1} \widetilde{a_{kl}\varphi}[u_l] \right). \end{aligned}$$

*Dabei seien $\tilde{f}_k$ und $\widetilde{a_{kl}\varphi}$ geeignete numerische Flussfunktion.*

**Bemerkung.**
Der Term $a_{kl}\nabla u_l$ ist nicht in Divergenzform und für stückweise stetige Funktionen ist die Wohldefiniertheit im allgemeinen ein schwieriges Problem. Man kann zeigen, dass der Term als Maß auf dem Rändern der Entitäten definiert werden kann. Dies geht jedoch nur, falls der Term linear ist. Die Behandlung von $d(u)\nabla\sigma(u)$ ist dagegen problematisch, daher der formale Trick in Ansatz 2 oben. Die Variationsformulierung beinhaltet einen Elementanteil und einen "Maßanteil" auf den Elementrändern.

**Einschub**: (Wahl von $\widetilde{a\varphi}$)

Wir betrachten die Burgers-Gleichung und schreiben diese in nicht-konservativer Form:

$$\begin{aligned} \partial_t u + \partial_x (\frac{1}{2} u^2) = 0, \\ \Longrightarrow \partial_t u + u \partial_x u = 0, \end{aligned}$$

Also erhalten wir einen Anteil der Form $a(u)\partial_x u$ mit $a(u) = u$. Mit $\varphi = 1$ und $u_h|_e = u_e$ konstant folgt dann aus der Variationsformulierung von (H)

$$\begin{aligned} \int_e \partial_t u + \int_e u \partial_x u + \int_{\partial e} \tilde{u}[u] = 0, \\ \overset{(\partial_x u = 0)}{\Longrightarrow} \int_t \partial_t u + \int_{\partial e} \tilde{u}[u] = 0, \end{aligned}$$

Wählt man $\tilde{u}$ einfach als Mittelwert, so folgt

$$\int_e \partial_t u + \int_{\partial e} \frac{1}{2}[u^2] = 0.$$

Dies ist eine zentrale Differenzen Diskretisierung, die instabil ist. Dies zeigt, dass in diesem Fall $\tilde{a}$ geschickt gewählt werden muss, um Upwinding/Stabilität zu garantieren. Ist $a\nabla u$ im Vergleich zu

$\nabla \cdot f(u)$ weniger dominant, so reicht hingegen häufig eine einfache Mittelung der Form $\tilde{a} = \frac{1}{2}(a_l + a_r)$ aus.

**Definition 4.70** (LDG-Diskretisierung von (H))**.**

Zur Diskretisierung der Variationsformulierung von (H) wählen wir

$$u_{k,h} \in V_{k,h}, \qquad \text{mit } u_{k,h}|_e \in V_{k,h}(e),$$

mit $V_{k,h}(e) = \text{span}\left\{\varphi^e_{k,1}, \ldots, \varphi^e_{k,r^e_k}\right\}, e \in \mathcal{E}^0_{\text{leaf}}$.

Sei $U^e_k = (U^e_{k,j})_{j=1}^{r^e_k}$ der Koeffizientenvektor auf der Entität $e$. Dann betrachten wir die diskrete Variationsformulierung für alle $e \in \mathcal{E}^0_{\text{leaf}}$, $i = 1, \ldots, r^e_k$:

$$\sum_{j=1}^{r^e_k} U^e_{k,j} \int_e \varphi^e_{k,j}\varphi^e_{k,i} = \int_e \left(S_k - \sum_{l=0}^{k-1} a_{k,l}\nabla u_l\right)\varphi^e_{k,i} + f_k \cdot \nabla\varphi^e_{k,i} - \int_{\partial e} \tilde{f}_k \cdot n\ \varphi^e_{k,i} + \sum_{l=0}^{k-1} \widetilde{a_{k,l}\varphi^e_{k,i}}[u_l].$$

Definieren wir die Elementmassenmatrix $M^e_k$ durch $(M^e_k)_{ij} := \int_e \varphi^e_{k,j}\varphi^e_{k,i}$, so folgt in Matrix-Vektorschreibweise

$$U^e_k = (M^e_k)^{-1}(E^{e,0}_k + E^{e,1}_k).$$

Dabei sammelt $E^{e,0}_k$ die Anteile von Integralen über $e$ und $E^{e,1}_k$ die Anteile von Integralen über $\partial e$, d.h.

$$\begin{aligned}(E^{e,0}_k)_i &:= \int_e \left(S_k - \sum_{l=0}^{k-1} a_{k,l}\nabla u_l\right)\varphi^e_{k,i} + f_k \cdot \nabla\varphi^e_{k,i},\\ (E^{e,1}_k)_i &:= -\int_{\partial e} \tilde{f}_k \cdot n\ \varphi^e_{k,i} + \sum_{l=0}^{k-1} \widetilde{a_{k,l}\varphi^e_{k,i}}[u_l].\end{aligned}$$

Im Gegensatz zum elliptischen stellen wir hier keine globalen Matrizen auf, sondern berechnen elementweise sukzessive für $k = 1, \ldots, K$ die Koeffizientenvektoren $U^e_k$. Sind die Basen $\varphi^e_{k,1}, \ldots, \varphi^e_{k,r^e_k}$ orthonormiert, so ist die Massenmatrix die Identität und somit trivial zu invertieren.

**Stabilisierung**: Wie in Abschnitt 4.3.1 und 4.3.2 bereits gezeigt, benötigen wir bei DG-Verfahren höherer Ordnung aus Stabilitätsgründen eine zusätzliche Limitierung der Gradienten (minmod, ...). Wir definieren daher in Anlehnung an Abschnitt 4.3.2. ein allgemeines semidiskretes LDG-Verfahren mit Stabilisierung wie folgt.

**Definition 4.71** (Allgemeines semidiskretes LDG-Verfahren mit Stabilisierung)**.**
Sei $t^0 = 0 < t^1 < \ldots < t^N = T$, eine Zerlegung von $[0, T]$. Auf $(t^n, t^{n+1})$ seien DG-Räume definiert durch

$$V_{h,n} = V^p_{h,n} = \left\{v_h \in L^\infty(\Omega) \mid v_h|_e \in P_p(e)\ \forall e \in \mathcal{E}^{0,n}_{\text{leaf}}\right\},$$

wobei $\mathcal{T}^n_h$ Gitter auf $\Omega$ für das Zeitintervall $(t^n, t^{n+1})$ seien.

Seien $\Lambda^{n,t}_h : V_{h,n} \longrightarrow V_{h,n}, t \in (t^n, t^{n+1})$ und $\Lambda^{n,t^n}_h : V_{h,n-1} \to V_{h,n}$ massenerhaltende Limiter auf

adaptiven Gittern im Sinne von Definition 4.44.

Dann definieren wir eine Approximation $u_h^n$ von $u$ auf $[t^n, t^{n+1})$ durch

$$u_h^n(t^n) = \Lambda_h^{n,t^n}[u_h^{n-1}(t^n)],$$
$$\frac{d}{dt}(u_h^n, \varphi) = -\left(L[\Lambda_h^{n,t}(u_h^n(t))], \varphi_h\right), \ \forall \varphi \in V_{h,n}.$$

Dabei wird $-\left(L[\Lambda_h^{n,t}(u_h^n(t))], \varphi_h\right)$ mit Hilfe der Zerlegung aus Definition 4.66 und der LDG-Diskretisierung nach Definition 4.70 berechnet.

**Bemerkung 4.72.**

1) $\Lambda_h^{n,t^n}$ beschreibt die Prolongation/Restriktion zwischen den Gittern $\mathcal{T}_{h,n-1}$ und $\mathcal{T}_{h,n}$, d.h. $t^n$ sind die Zeitpunkte, zu denen das Gitter adaptiert wird. Durch $\Lambda_h^{n,t}$ wird eine nötige Limitierung der Gradienten beschrieben (siehe Abschnitt 4.3.1 und 4.3.2).

2) Zur Stabilisierung müssen die Integrale mit ausreichender Ordnung berechnet werden (Formel von Cockburn-Shu).
Für $p = 1$ muss beispielsweise $\int_e$ mit einer Quadratur, welche exakt ist für Polynome 1. Grades ist berechnet werden, während $\int_{\partial e}$ sogar von zweiter Ordnung approximiert werden muss (z.B. in 2D durch 2 Punkt Gauß-Formeln).

## 4.4 Implementierung von LDG-Verfahren für Evolutionsgleichungen in Dune-Fem

**Ziel.**
Implementieren von diskreten zusammengesetzten Operator $L_h : V_h \longrightarrow V_h$ mit Zerlegungen der Form

$$L_h[u_h] = L_h^K[L_h^{K-1}[\dots L_h^1[u_h]\dots].$$

Dabei seien $L_h^i$ vorgegebene "einfache" Operatoren, etwa definiert durch LDG-Diskretisierungen von (H) im Sinne von Definition 4.70.

### Abstraktes Konzept für diskrete Orts-Operatoren in Dune-Fem

Seien $V$, $W$ beliebige Funktionenräume über Körpern $\mathbb{K}_V$, $\mathbb{K}_W$. Ein Operator $L : V \to W$ ist dann eine Abbildung von $V$ nach $W$. Wir formulieren nun das diskrete Analogon dazu:

**Definition 4.73** (Diskrete Operatoren)**.**
Ein diskreter Operator $L_h$ ist ein Operator von einem diskreten Funktionenraum in einen anderen, d.h. $L_h : V_h \to \tilde{V}_h$.
Da einem diskreten Funktionenraum nach Kapitel 3 eine Gitter zugrunde liegt, nehmen wir an, dass $L_h$ wie folgt zerlegt werden kann in einen globalen Operator $L_{\text{pre}}$, einer Menge von lokalen Operatoren $\mathcal{L}_e$, die nur auf lokalen diskreten Funktionen auf Entitäten $e$ der Kodimension Null operieren, und einem globalen Operator $L_{\text{post}}$:

$$L_h = L_{\text{post}} \circ \text{diag}\{L_e, e \in \mathcal{E}_{\text{leaf}}^0\} \circ L_{\text{pre}}.$$

Dabei sind die Operatoren wie folgt definiert

$$\begin{aligned} L_{\text{pre}} &: V_h \to \{V_e, e \in \mathcal{E}^0_{\text{leaf}}\}, \\ L_e &: V_e \to \tilde{V}_e \text{ für alle } e \in \mathcal{E}^0_{\text{leaf}}, \text{ und} \\ L_{\text{post}} &: \{\tilde{V}_e, e \in \mathcal{E}^0_{\text{leaf}}\} \to \tilde{V}_h. \end{aligned}$$

Die Zerlegung der diskreten Operatoren ist notwendig, um eine effiziente lokale Verknüpfung mit nur einem Gitterdurchlauf realisieren zu können, d.h. wir haben

$$L^2_h \circ L^1_h = L^2_{\text{post}} \circ \text{diag}\{L^2_e \circ L^1_e, e \in \mathcal{E}^0_{\text{leaf}}\} \circ L^1_{\text{pre}},$$

falls $L^2_{\text{pre}} L^1_{\text{post}} = Id$ gilt.

Neben diskreten Operatoren, werden wir nun auch Projektionsoperatoren und inverse Operatoren einführen.

**Definition 4.74** (Projektions-Operatoren)**.**
Seien $V_h$ ein diskreter Funktionenraum, $\mathcal{B}_{\mathcal{E}^0_{\text{leaf}}}$ eine Basis von $V_h$ und $\mathcal{B}^*_{\mathcal{E}^0_{\text{leaf}}}$ eine Menge von linearen Funktionalen auf dem kontinuierlichen Funktionenraum $V$ mit $|\mathcal{B}^*_{\mathcal{E}^0_{\text{leaf}}}| = |\mathcal{B}_{\mathcal{E}^0_{\text{leaf}}}|$, so dass für alle $\varphi \in \mathcal{B}_{\mathcal{E}^0_{\text{leaf}}}$ genau ein $\varphi^* \in \mathcal{B}^*_{\mathcal{E}^0_{\text{leaf}}}$ existiert mit

$$\varphi^*(\psi) = \delta_{\varphi\psi} \; \forall \, \psi \in \mathcal{B}_{\mathcal{E}^0_{\text{leaf}}}.$$

Dann definiert diese Menge von Funktionalen eine Projektion $\Pi_{\mathcal{E}^0_{\text{leaf}}} : V \to V_{\mathcal{E}^0_{\text{leaf}}}$ durch

$$\big(\Pi_{\mathcal{E}^0_{\text{leaf}}}(v)\big)(\boldsymbol{x}) := \sum_{\varphi \in \mathcal{B}_{\mathcal{E}^0_{\text{leaf}}}} \varphi^*(v)\, \varphi(\boldsymbol{x}) \qquad \text{für alle } v \in V.$$

Standard $L^2$-Projektionen, aber auch Lagrange-Interpolationen können in diesem Framework realisiert werden.

Mit der Definition von diskreten Operatoren lassen sich sehr allgemeine Gitter-basierte Diskretisierungsverfahren schreiben in der Form

$$L_h(v_h) = .$$

Da wir an der Lösung $v_{\mathcal{E}^0_{\text{leaf}}}$ solcher Diskretisierungen interessiert sind, benötigen wir numerische Lösungsverfahren, die den diskreten Operator $L_h$ invertieren. Formal können wir dies wie folgt schreiben: $v_{\mathcal{E}^0_{\text{leaf}}} = {L_h}^{-1}(f_{\mathcal{E}^0_{\text{leaf}}})$. Ein numerisches Lösungsverfahren kann also als diskreter inverser Operator aufgefasst werden.

**Definition 4.75** (Inverse Operatoren)**.**
Sei $L_h : V_h \to \tilde{V}_h$ ein diskreter Operator. Dann definieren wir einen inversen Operator $S$ durch

$$S_{L_h} : \tilde{V}_h \to V_h.$$

Ein inverser Operator wird also mit einem diskreten Operator initialisiert und bildet von $\tilde{V}_h$ nach $V_h$ ab. Iterative oder direkte numerische Lösungsverfahren von linearen oder nichtlinearen Gleichungssystemen können in dem Konzept als inverse Operatoren realisiert werden.

Wenden wir uns nun den speziellen Gegebenheiten von zusammengesetzten Operatoren zu, wie sie in Abschnitt 4.3.4 eingeführt wurden.

Seien dazu $V, W$ Funktionenräume und $L : V \to W$ ein Ortsoperator. Wir nehmen an, dass wir Funktionenräume $V_s, W_s, s = 1, \ldots, S$ haben mit $V_0, W_0 := V$, $V_s := W_s \times V_{s-1}, s = 1, \ldots, S$ und $W_S := W$, und einfache Operatoren (im folgenden Passes genannt)

$$L_s : V_{s-1} \to V_s, \quad s = 1, \ldots, S, \quad \text{und} \quad \Pi_S : W_S \times \cdots \times W_0 \to V_S,$$

so dass gilt

$$L = \Pi_S \circ L_S \circ \cdots \circ L_1.$$

Jetzt verwenden wir das Konzept diskreter Operatoren, um diskrete zusammengesetzte Operatoren wie folgt zu definieren.

**Definition 4.76** (Diskrete zusammengesetzte Operatoren und Passes)**.**
Sei $L : V \to W$ ein zusammengesetzter Operator $L_s, s = 1, \ldots, S$. Wenn jeder Pass $L_s : V_{s-1} \to V_s$ durch einen diskreten Pass $L_{s,h} : V_{s-1,h} \to V_{s,h}$ approximiert wird, so definieren wir den zusammengesetzen diskreten Operator $L_h$ durch

$$L_h = \Pi_S \circ L_{S,h} \circ \cdots \circ L_{1,h}.$$

### Klassen in DUNE-FEM zur Realisierung des abstrakten Konzepts

- `Mapping< DFieldType, RFieldType, DType, RType >`: Diese Klasse repräsentiert allgemeine Abbildungen von einem Vektorraum des Typs `DType` über einem Körper vom Typ `DFieldType` in einem Vektorraum des Typs `RType` über einem Körper vom Typ `RFieldType`.

  **Wichtige Methoden (Auswahl):**

  ```
  operator () (const DomainType & arg, RangeType dest);
  ```

  Des Weiteren: `operator +, +=, * (RangeFieldType)` .

- Abgeleitete Klassen sind
  - `Function` (`DType` ist $(\texttt{DFieldType})^n$, RType ist $(\texttt{RFieldType})^m$),
  - `Operator` (`DType`, `RType` sind Functions).

- `DiscreteOperator< LocalOperator, DFDomainType, DFRangeType >`: Beschreibt einen diskreten Operatoren im Sinne von Definition 4.73 der Form

  $$L_h = L_{\text{post}} \left(L_e \mid e \in \mathcal{T}_h\right) \cdot L_{\text{pre}}.$$

  Die Operatoren $(L_e \mid e \in \mathcal{T}_h)$ auf den Entitäten sind dabei vom Typ `LocalOperator`. `DFDomainType`, `DFRangeType` sind die Typen der diskreten Funktionen im Urbild und Bildraum des Operators.
  **Beispiel** (Finite-Volumen-Verfahren)**.**

  | | |
  |---|---|
  | $L_{\text{pre}}$: | $upd = 0$ (Initialisiert den Updatevektor), |
  | $L_e$: | Berechne den Beitrag des Updatevektors auf Entität e: $upd[i(e)]$, |
  | $L_{\text{post}}$: | $dest = arg + \Delta t * upd$ (Operation auf dem gesamten DOF-Vektor). |

**Vorteil**: Verknüpfung wie + können auf den lokalen Operatoren durchgeführt werden, falls gilt $L^2_{\text{pre}} \circ L^1_{\text{post}} = Id$:

$$L^2_h + L^1_h = L^2_{\text{post}}(L^1_e + L^2_e) \circ L^1_{\text{pre}}.$$

Dadurch werden Gitterdurchläufe reduziert.
**Bemerkung** (Auswertung von Verknüpfungen)**.**
Definiert man $L_h$ durch $L_h = L^2_h + L^1_h$, so ist $L_h$ vom allgemeinen Typ Mapping. Setzt man also anschließend $L_h \mathrel{+}= L^3_h$, so operiert $L_h$ auf globaler Ebene und nicht lokal. Es werden also zwei Gitterdurchläufe angestoßen. Definiert man hingegen $L_h = L^3_h + L^2_h + L^1_h$, so wird lokal verknüpft und nur eine Gitterdurchlauf angestoßen.

- `InverseOperator< DiscreteFunctionType, OperatorType >`: Basisklasse zur Implementierung eines numerischen Lösers zur Invertierung eines diskreten Operators vom Typ `OperatorType`. Eine Implementierung eines CG-Verfahrens für lineare Operatoren wird z.B. durch die Klassen `OEMCGOp` oder `CGInverseOp` realisiert.

- `Pass< DiscreteModel, PreviousPass, passId>`: Dies ist die Basisklasse zur Implementierung eines Passes im Sinne von Definiton 4.76.

  Die Implementierung der `Pass`-Klasse soll einerseits das abstrakte Konzept aus Definition 4.76 realisieren, aber andererseits soll ein `Pass` auch ein DUNE-Operator sein. Daher wurde bei der Implementierung nicht die Variante

  $$L^{i+1}(U^i, \dots, U^0) \longrightarrow (U^{i+1}, \dots, U^0)$$

  realisiert, sondern die Variante

  $$L^{i+1} : U^0 \longrightarrow U^{i+1},$$

  und $(U^i, \dots, U^1)$ sind Parameter, d.h.

  $$U^{i+1} = L^{i+1}(U^0; U^i, \dots, U^1).$$

  Damit ist $L^{i+1}$ ein DUNE-Operator.

  Die Template-Argumente sind

  `DiscreteModel`**:** Interface zur Problembeschreibung, etwa Quellterm, Koeffizientenfunktionen, numerischer Fluss (vgl. Typ (H)).

  `PreviousPass`**:** Definiert den Typ von $L_i$. `PreviousPass` ist von `Pass` abgeleitet oder von Typ `StartPass`. Im Konstruktor übergibt man eine Referenz auf eine Instanz von `PreviousPass` und von `DiscreteModel`.

  **Wichtige Methoden (Auswahl):**

  ```
  operator() (arg, dest) {
     previousPass(arg, dest);
     compute(dest, previousPass.parameters());
  }
  ```

  Rekursiver Aufruf zur Berechnung des Ergebnisses `dest`.

  `parameters()`
  Liefert einen Tupel mit $(U^i, \dots, U^1, U^0)$ indem es `Pair<`$U^{i+1}$`, previousPass.parameters()>`

zurück gibt. Für `StartPass` gibt `parameters()` `Pair<Arg, Nil>` zurück.

`compute()` ist eine virtuelle Methode zur Durchführung der eigentlichen Berechnung, z.B.

(DG-Pass) $V \approx S(U) - \nabla \cdot f(u)$, oder
(Laplace-Pass) $V \approx \Delta^{-1} f(U)$.

`StartPass` hat eine leere `compute()` Methode.

**Vorhandene Implementierungen**

In DUNE-FEM sind als Passes bereits `LocalDGPass` und `LocalDGElliptPass` implementiert. `LocalDGPass` realisiert eine LDG-Diskretisierung für Gleichungen vom Typ (H), wie in Abschnitt 4.3.4 dargestellt, während `LocalDGElliptPass` eine Pass der Form $u = \Delta^{-1} f$ mit Hilfe einer LDG-Diskretisierung realisiert.

Die Klassen `CreatePass` und `CreatePassTree` erlauben eine einfache Implementierung von Passes im Kontext allgemeiner LDG-Diskretisierungen, wie das folgende Beispiel demonstriert:

```
1   // diffusion pass
2   typedef CreatePass< DiscreteModel1Type, LocalDGPass > Pass1Type;
3   Pass1Type pass1( discreteModel1_, space1_ );
4
5   // advection pass
6   typedef CreatePass< DiscreteModel2Type, LocalDGPass > Pass2Type;
7   Pass2Type pass2( discreteModel2_, space2_ );
8
9   // create pass tree and return pointer to resulting
10  // operator satisfying the SpaceOperatorInterface.
11  SpaceOperatorInterface< DestinationType >* passTree
12    = CreatePassTree::create( pass1 , pass2 );
```

# KAPITEL 5

# Parallelisierung

In diesem Kapitel werden wir auf Grundzüge der parallelen Programmierung im Zusammenhang mit Gitter-basierten Diskretisierungsverfahren eingehen. Detailliertere Darstellungen sind beispielsweise dem Skript von Bastian [1] zu entnehmen, dem wir hier auch partiell folgen.

**Definition** (Klassifizierung von Rechnern nach Flynn (1972))**.**
Nach Flynn (1972) können Rechner wie folgt klassifiziert werden:

1) SISD – single instruction single data. Dies sind sequentielle Rechner.
2) SIMD – single instruction multiple data. Diese Rechner, auch Feldrechner oder Vektorrechner genannt, verfügen über ein Instruktionswerk und mehrere unabhängige Rechenwerke von denen jedes mit einem eigenen Speicher verbunden ist. Die Rechenwerke werden taktsynchron vom Instruktionswerk angesteuert und führen die selbe Operation auf unterschiedlichen Daten aus. Solche Rechner können sehr effizient für Finite-Differenzen-Verfahren sein, sind aber eher ungeeignet für Verfahren auf unstrukturierten Gittern.
3) MISD – multiple instruction single data. Diese Klasse ist leer.
4) MIMD – multiple instruction multiple data. Dies entspricht einer Kollektion eigenständiger Rechner, jeder mit einem Instruktions– und einem Rechenwerk ausgestattet.

Im folgenden werden wir uns nur mit MIMD-Rechnern beschäftigen und zwei Typen unterscheiden: *Shared-Memory-Rechner* und *Distributed-Memory-Rechner*. Zuvor wollen wir jedoch definieren, was wir unter *Speedup* und *Effizienz* eines parallelen Algorithmus verstehen.

**Definition** (Speedup und Effizienz)**.**
Sei $T_{\text{ser}}(N)$ dazu die Laufzeit eines seriellen Codes. Es gelte

$$T_{\text{ser}}(N) = \underbrace{T^1_{\text{ser}}(N)}_{\text{perfekt parallelisierbar}} + \underbrace{T^2_{\text{ser}}(N).}_{\text{nur seriell ausführbar}} .$$

Dann gilt für die parallele Laufzeit auf $p$ Prozessoren

$$T_{\text{par}}(N,p) = \frac{T^1_{\text{ser}}(N)}{p} + T^2_{\text{ser}}(N) + T_{\text{overhead}}(N,p).$$

Dabei bezeichnet $T_{\text{overhead}}(N,p)$ den Laufzeitanteil, der beispielsweise durch zusätzliche Kommunikation zwischen den Prozessoren entsteht. Wir definieren nun als Maß für die Parallelisierbarkeit

eines Codes:

$$\text{Speedup: } S_p(N) = \frac{T_{\text{ser}}(N)}{T_{\text{par}}(N,p)}, \qquad \text{optimal ist } S_p(N) = p.$$

$$\text{Effizienz: } E_p(N) = \frac{S_p(N)}{p}, \qquad \text{optimal ist } E_p(N) = 1.$$

Abbildung 5.1 veranschaulicht den Speedup für festes $N$.

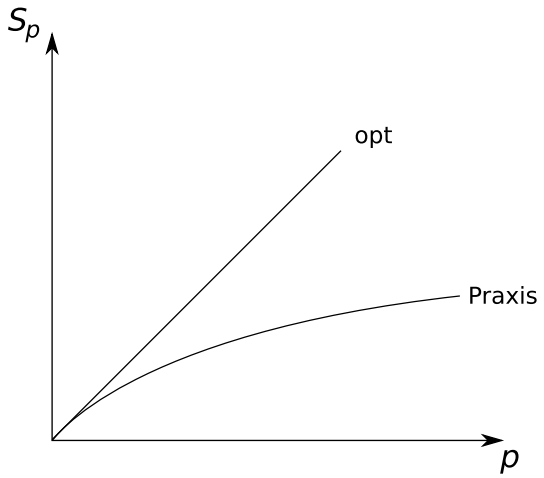



Abbildung 5.1: Speedup bei festem $N$

## 5.1 Shared-Memory-Rechner

Beim *Shared-Memory-Rechner* greifen mehrere Prozesse auf einen gemeinsamen Speicher zu. Parallelisierung von Code ist einfach. Beispielsweise zerlegt man eine Schleife

```
for( i = 0; i < N; ++i )
```

bei $p$ Prozessoren einfach in $p$ Teile. D.h. auf Prozessor $q$ berechnet man

```
for( i = q(N/p); i < (q+1)(N/p); ++i ).
```

**Vorteil:** Jeder Prozess kann auf sämtliche Daten zugreifen. Ein serielles Programm kann daher ohne größere Schwierigkeiten parallelisiert werden. Dies führt in der Regel bei kleinen Prozessorzahlen schnell zu einer ersten Leistungssteigerung.

### Zugriffskonflikte

Gemeinsamer schreibender Zugriff auf gleiche Speicherstelle.
Zugriffskonflikte wachsen bei größerer Prozessoranzahl stark an. Eine gute Skalierbarkeit ist folglich schwer zu gewährleisten. Zur Verringerung von Zugriffskonflikten werden leistungsfähige Zugriffsverwaltungen notwendig, die solche Systeme sehr teuer machen.

**Beispiel** (Berechnung von numerischen Flüssen)**.**
Die Entität $e$ gehöre zu Prozessor $p_1$, die Entität $\tilde{e}$ zu Prozessor $p_2$. Prozessor $p_1$ berechnet den Fluss $g_{e\tilde{e}}$ und setzt

$$\begin{aligned} upd_e &\mathrel{+}= g_{e,\tilde{e}} \\ upd_{\tilde{e}} &\mathrel{-}= g_{e,\tilde{e}} \end{aligned}$$

Parallel dazu berechnet $p_2$ den Fluss $g_{\tilde{e},\hat{e}}$ zwischen $\tilde{e}, \hat{e}$ und führt folgende Anweisung aus

$$upd_{\tilde{e}} \mathrel{+}= g_{\tilde{e},\hat{e}}.$$

Auf die Speicherstelle von $upd_{\tilde{e}}$ kann also gleichzeitig von zwei Prozessen schreibend zugegriffen werden.

**Lösung**: Entweder berechnet man den Fluss $g_{e,\tilde{e}}$ doppelt, einmal auf $p_1$ und einmal auf $p_2$ und greift jeweils nur von definierten Prozessoren auf die Speicherstellen zu, oder man arbeite mit *Locks*:

| $p_1$ | $p_2$ |
|---|---|
| hole Lock für $upd_{\tilde{e}}$ | hole Lock für $upd_{\tilde{e}}$ - fehlgeschlagen |
| $upd_{\tilde{e}} \mathrel{+}= g_{e,\tilde{e}}$ | warten, bis Lock wieder weg ist |
| schreibe $upd_{\tilde{e}}$ | |
| Freigabe von $upd_{\tilde{e}}$ | |
| | hole Lock |
| | $upd_{\tilde{e}} \mathrel{+}= g_{\tilde{e},\hat{e}}$ |
| | schreibe $upd_{\tilde{e}}$. |
| | Freigabe von $upd_{\tilde{e}}$ |

Beide Versionen führen zu parallelem Overhead.

### Re-entrancy

Die Methoden müssen so programmiert sein, dass sie aufgerufen werden können, während sie bereits auf einem anderen Prozess laufen. Insbesondere dürfen also keine statischen oder globalen Variablen verwendet werden.
**Beispiel** (findToken, findNextToken "END")**.**


Die Position des zuletzt gefundenen Tokens wird in einer statischer Variable gespeichert. Verwenden zwei Prozessoren die Methode gleichzeitig, so verwenden sie denselben Speicher, folglich ist die Methode nicht *Thread safe*.

Aufgrund der Nachteile von Shared-Memory-Programmierung bei großen Prozessorzahlen, werden wir uns im folgenden nur noch mit Shared-Memory-Programmierung beschäftigen.

## 5.2 Distributed-Memory-Rechner

Solche Rechner bestehen aus Speicherteilen, auf welche im gesamten Programmablauf stets nur ein Prozess zugreifen kann. Werden Daten eines anderen Prozesses benötigt, so kann dies nur über explizite Kommunikation zwischen den Prozessoren erfolgen. Daher gibt es keine Speicherzugriffsprobleme und alle Methoden sind automatisch *Thread safe*. Im Vergleich zu Shared-Memory-Rechnern,

sind solche Systeme relativ preiswert. Ein weiterer Vorteil ist eine fast unbeschränkte Skalierbarkeit.

**Nachteil:** Synchronisation aufgrund des expliziten Datenaustauschs. Dadurch müssen alle Prozesse auf den "Langsamsten" warten. Das Ziel einer sogenannten "Lastverteilung" muss es also sein, den Aufwand auf alle Prozessoren möglichst gleich zu verteilen. Ein typischer Ansatz dazu ist die Gebietszerlegungsmethode. Dabei wird das zugrundeliegende Gitter partitioniert und möglichst "gleich große" Teilgebiete den jeweiligen Prozessen zugeordnet. Wir werden später noch detaillierter auf diesen Zugang eingehen.

Moderne Rechner bestehen heute häufig aus einer Kombination von Shared- und Distributed-Memory-Rechnern, bei denen auf einen Speicherbereich nicht nur ein Prozessor, sondern eine kleine Anzahl von Prozessoren (2-64) zugreifen können.

Als Programmiermodell auf Distributed-Memory-Rechnern werden wir nun Kommunikation durch Nachrichtenaustausch (message passing) betrachten.

## 5.3 Message Passing und MPI

In diesem Abschnitt behandeln wir Syntax und Semantik der grundlegenden Funktionen für den Nachrichtenaustausch (message passing). Beim Nachrichtenaustausch wird zwischen blockierender Kommunikation (synchrone Kommunikation) und nichtblockierender (asynchroner) Kommunikation unterschieden. Die Basis der Kommunikation bilden Funktionen zum verschicken (send) und empfangen (receive) von Nachrichten (messages).

### Synchrone Kommunikation

Wir betrachten zunächst Befehle für den synchronen Punkt-zu-Punkt Nachrichtenaustausch. Dazu stehen die beiden Befehle

- `send(dest - process, expr_1, ... , expr_n)`
- `recv(src - process, var_1, ... , var_n)`

zur Verfügung.
Der Befehl `send` sendet eine Nachricht an den Prozess `dest - process`, die die Werte der Ausdrücke `expr_1` bis `expr_n` enthält. Der Prozess `dest - process` empfängt diese Nachricht mit dem entsprechenden `recv`-Befehl und speichert die Werte der Ausdrücke in den Variablen `var_1` bis `var_n` ab. Die Variablen müssen von passendem Typ sein.
Sowohl `send` als auch `recv` sind blockierend, d.h. werden erst beendet, wenn die Kommunikation stattgefunden hat. Die beteiligten Prozesse werden dadurch synchronisiert. Sende– und Empfangsprozess müssen ein passendes send/recv-Paar ausführen, sonst entsteht eine Verklemmung (Deadlock).

Blockierende Punkt-zu-Punkt-Kommunikation ist nicht ausreichend um alle Aufgaben zu lösen. In einigen Situationen kann ein Prozess nicht wissen, welcher von mehreren möglichen Partnerprozessen als nächstes zu einem Datenaustausch bereit ist. Ein Deadlock tritt z.B. bei dem Versuch auf, Daten zwischen zwei Prozessen mit blockierenden `send`-, bzw. `recv`-Anweisungen auszutauschen.

Eine Möglichkeit dieses Problem zu lösen besteht in der Bereitstellung zusätzlicher Funktionen, die es ermöglichen festzustellen, ob ein Partnerprozess bereit ist, eine Nachricht zu senden oder zu empfangen, ohne selbst zu blockieren:

- `int sprobe(dest - process)`,
- `int rprobe(src - process)`.

Die Funktionen liefern `1 = true`, falls ein entsprechender Aufruf von `send` oder `recv` erfolgreich wäre.

#### Asynchrone Kommunikation

Für asynchrone Kommunikation stehen die beiden Befehle

- `asend(dest - process, expr_1, ... , expr_n)`
- `arecv(src - process, var_1, ... , var_n)`

zur Verfügung. Sie haben die gleiche Semantik wie `send`, `recv`, sind aber nicht blockierend.

#### Globale Kommunikation

- Gather-Scatter-Operationen: Sammeln oder Verteilen von Daten durch einen ausgezeichneten Prozess.
- Broadcast: Datenübertragung von einem (oder allen Prozessen) an alle anderen.
- Reduce- and All-Reduce-Operationen: Bildung von Ergebnissen über Daten aller Prozesse. Das Ergebnis steht dem Root-Prozess (reduce) oder allen (all-reduce) zur Verfügung.
- Barrier: Synchronisation aller Prozesse.

#### Virtuelle Kanäle

Die bisher vorgestellten Kommunikationsfunktionen werden als verbindungslos bezeichnet. Alternativ dazu kann man sich vorstellen, dass die Kommunikation über sogenannte "Kanäle" erfolgt. Diese Kommunikationskanäle (communication channels) können als gemeinsame Objekte verstanden werden. Bevor ein Kanal benutzt werden kann, muss er aufgebaut werden. Dies kann statisch (zur Ladezeit) oder dynamisch, während des Programmablaufes geschehen. Sende– und Empfangsbefehle operieren dann auf Kanälen:

- `send(channel, expr_1, ... , expr_n)`,
- `recv(channel, var_1, ... , var_n)`.

### 5.3.1 Der MPI Standard

Das Message-Passing-Interface (MPI) ist eine portable Bibliothek von Funktionen zum Nachrichtenaustausch zwischen Prozessen. MPI wurde 1993-94 von einem internationalen Gremium entwickelt und ist heute auf praktisch allen Plattformen verfügbar. Freie Implementierungen für LINUX-Cluster (und weitere Rechner) sind MPICH2, Open MPI und LAM3.

MPI hat die folgenden Merkmale:

- Bibliothek zum Binden mit C-, C++- und FORTRAN-Programmen (keine Spracherweiterung).
- Große Auswahl an Punkt-zu-Punkt Kommunikationsfunktionen.
- Globale Kommunikation.
- Datenkonversion für heterogene Systeme.
- Teilmengenbildung und Topologien.

MPI besteht aus über 125 Funktionen, die auf über 200 Seiten im Standard beschrieben werden. Hier wollen wir nur auf eine grundlegende Auswahl der Funktionalität eingehen.
**Beispiel** (Ein einfaches Beispiel)**.**

MPI ist eine Bibliothek, die zu einem Standard-C++(C/FORTRAN) Programm hinzugebunden wird. Ein erstes Beispiel in C++ zeigt folgender Code:

```
1  #include <stdlib.h>
2  #include <stdio.h>
3  #include "mpi.h"
4
5  int main( int argc, char *argv[] )
6  {
7     int my_rank;
8     int P;
9     int dest;
10    int source;
11    int tag=50;
12    char message[100];
13
14    MPI_Status status;
15    MPI_Init( &argc, &argv );
16    MPI_Comm_size( MPI_COMM_WORLD, &P );
17    MPI_Comm_rank( MPI_COMM_WORLD, &my_rank );
18    if( my_rank != 0 )
19    {
20       sprintf( message, "I␣am␣process␣%d\n", my_rank );
21       dest = 0;
22       MPI_Send( message, strlen(message) + 1, MPI_CHAR,
23         dest, tag, MPI_COMM_WORLD );
24    }
25    else
26    {
27       puts( "I␣am␣process␣0\n" );
28       for( source = 1; source < P; source++ )
29       {
30          MPI_Recv( message, 100, MPI_CHAR, source, tag,
31            MPI_COMM_WORLD, &status );
32          puts( message );
33       }
34    }
35    MPI_Finalize();
```

```
36     return 0;
37 }
```

Dieses Beispielprogramm ist im SPMD-Stil geschrieben. Dies ist von MPI nicht zwingend vorgeschrieben, es erleichtert nur das Starten des parallelen Programms etwas. Übersetzen, binden und ausführen des Programms unterscheiden sich von Implementierung zu Implementierung. Viele Implementierungen enthalten eine Reihe von Shell-Skripten, die den Installationsort der Bibliotheken verbergen. So benötigt man etwa in MPICH die Kommandos

```
mpicc -o hello hello.c
mpirun -machinefile machines -np 8 hello
```

um das Programm zu übersetzen und acht Prozesse zu starten. Dabei werden die Namen der zu benutzenden Rechner aus der Datei machines genommen.

Das Programm selbst erklärt sich von alleine. Die MPI-Funktionen und -Macros werden durch die Datei mpi.h zur Verfügung gestellt. Jedes MPI-Programm beginnt mit `MPI_Init` und endet mit `MPI_Finalize`. Die Funktion `MPI_Comm_size` liefert die Anzahl der beteiligten Prozesse P und MPI_Comm_rank liefert die Nummer des Prozesses, in MPI „rank“ genannt. Die Nummer eines Prozesses ist zwischen 0 und $p-1$. Schließlich werden noch die Funktionen `MPI_Send` und `MPI_Recv` zum senden/empfangen von Nachrichten verwendet. Dabei handelt es sich um blockierende Kommunikationsfunktionen. Die genaue Bedeutung der Parameter werden im folgenden bei der Auflistung der wichtigsten Funktionen besprechen.

### Die wichtigsten Funktionen in MPI

- `MPI_Init(&argc,&argv)` ist die Initialisierungsroutine. Sie muss vor dem Benutzen von MPI-Befehlen ausgeführt werden. Die Argumente lauten argc und argv und sind Pointer auf Parameter der Hauptfunktion.
- `MPI_Finalize()` beendet MPI und räumt sämtlichen hinterlassenen Müll von MPI weg.
- `MPI_Abort()` sorgt für einen sauberen Abbruch im Fehlerfall.
- `MPI_Comm_Size(MPI_Comm comm, int size)` gibt die Anzahl der beteiligten Prozesse durch das zweite Argument (size) zurück. Das erste Argument ist ein Kommunikator. Letzten Endes ist das eine Anzahl von Prozessen, die Nachrichten austauschen. Für einfache Programme wird jedoch nur
- `MPI_Comm_World` benötigt. Dies beinhaltet alle Prozesse, die laufen, wenn die Ausführung beginnt (siehe auch Kommunikationsarten).
- `MPI_Comm_Rank(MPI_Comm comm, int rank)` gibt den Rang (Prozessidentifikation) zurück.

  Zentrale Send-/Receive-Funktionen:

- `MPI_Send(start, count, datatype, dest, tag, comm)` Standardfunktion zum Senden von Nachrichten. Start bestimmt die Speicheradresse, count die Länge und Datatype den Datentyp (hier immer MPI-Datentypen, keine C- oder Fortran-Datentypen). Tags sind zur Bestimmung von Kontexten, z.B. wenn zwei Nachrichten zwei floats senden, die einmal verarbeitet und einmal gedruckt werden sollen. Dest (Send) und Source (Receive) sind die Ränge der sendenden

oder empfangenden Prozesse. `MPI_ANY_SOURCE` ist eine Wildcard für source, wenn man von irgendeinem Prozess statt eines speziellen empfangen will.

- `MPI_Recv(start, count, datatype, source, tag, comm, status)` Standardfunktion zum Empfang von Nachrichten
- `MPI_Bcast(start, count, datatype, source, tag, comm, status)` sendet von einem an alle Knoten.
- `MPI_Reduce(start, result, count, datatype, operation, root, comm)` sammelt Daten von allen Knoten mittels einer bestimmten Operation (`MPI_MAX, MPI_MIN, MPI_SUM, MPI_PROD, MPI_LAND, MPI_BAND, MPI_LOR, MPI_BOR, MPI_LXOR, MPI_BXOR, MPI_MAXLOC, MPI_MINLOC`; die letzten zwei suchen auch noch den Ort).

### Datentypen in MPI

| MPI-Datentyp | C-Datentyp |
|---|---|
| `MPI_SHORT` | `signed short int` |
| `MPI_INT` | `signed int` |
| `MPI_CHAR` | `signed char` |
| `MPI_LONG` | `signed long int` |
| `MPI_UNSIGNED_CHAR` | `unsigned char` |
| `MPI_UNSIGNED_SHORT` | `unsigend short` |
| `MPI_UNSIGNED` | `unsigend int` |
| `MPI_UNSIGNED_LONG` | `unsigned long int` |
| `MPI_FLOAT` | `float` |
| `MPI_DOUBLE` | `double` |
| `MPI_LONG_DOUBLE` | `long double` |

Weiterhin kann man Arrays `MPI_TYPE_vector(count, blocklength, stride, oldtype, &newtype)` und Verbünde `MPI_Type_structure(count, array of length, array of location, array of types, &newtype)` definieren. Hierbei steht stride für den Abstand der Elemente.

### Weitere Funktionen

- `MPI_Barrier(comm)` blockiert Abarbeitung, bis alle Prozesse in `comm` Barrier aufgerufen haben.
- t=`MPI_WTime()` Funktion zur Zeitmessung.
- `MPI_Comm_create` oder `MPI_Comm_split` kreiert einen neuen Kommunikator oder teilt ihn in mehrere.
- `MPI_Probe(source, tag, comm, status)` ist ein blockierender Empfangstest.

## 5.4 Parallele Kommunikation in DUNE

Die Softwarebibliothek DUNE bietet eine Schnittstelle zur Parallelisierung Gitter-basierter Diskretisierungsverfahren an, die auf MPI basiert. Im Folgenden werden wir auf das Parallelisierungskonzept zunächst abstrakt eingehen und anschließend die wesentlichen Klassen und Methoden erläutern.

### 5.4.1 Abstraktes Parallelisierungskonzept in DUNE

Die Programmierung in DUNE folgt dem "single programm multiple data" (SPMD) Programmiermuster und setzt eine geeignete Gebietszerlegung des Rechengitters voraus. Die Gebietszerlegung erfolgt in zwei Schritten. Zunächst werden Kodimension-0-Entitäten den vorhandenen Prozessen eindeutig zugeordnet (master decomposition). In einem zweiten Schritt wird die Verteilung der übrigen Entitäten von dieser *master decomposition* abgeleitet (extended decomposition). Wir nehmen im Folgenden an, dass $K \geq 1$ Prozesse vorhanden sind und jeder Prozess durch seine Nummer $p \in P := \{0, \ldots, K-1\}$ identifiziert wird.

**Definition 5.1** (Master und Extended Decomposition)**.**
Sei ein hierarchisches Gitter $\mathcal{T}_h$ gegeben. Die "Master Decomposition" ist definiert durch die Relation

$$\mathcal{D}^0 \subset \mathcal{E}^0 \times P.$$

Eine Entität $e \in \mathcal{E}^0$ gehört folglich zum Prozess $p$, falls gilt $(e, p) \in \mathcal{D}^0$.

Die Menge aller Entität $e \in \mathcal{E}^0$, die zu Prozessor $p$ gehören bezeichnen wir mit $\mathcal{E}^0_p$.

Für Entitäten höherer Kodimension definieren wir die "Extended Decomposition" durch die Relation

$$\mathcal{D} \subset \mathcal{E} \times P$$

durch die Zuordnung zur "Master Decomposition" mit Hilfe der Subentitätenrelation. Für alle $e \in \mathcal{E}^0$, $e^c \in \mathcal{E}^c, c > 0$ mit $e^c \in C(e)$ gelte also für alle Prozesse $p$:

$$(e^c, p) \in \mathcal{D} \Longleftrightarrow (e, p) \in \mathcal{D}^0.$$

Dies bedeutet, dass ein Prozess für jedes seiner Entitäten auch alle Subentitäten kennt. Die Menge der Entitäten auf Prozess $p$ bezeichnen wir mit $\mathcal{E}_p$.

Die Elemente eines Prozesses werden in unterschiedliche Klassen eingeteilt.

**Definition 5.2** (Partition Type)**.**
Die Abbildung

$$t^0 : \mathcal{D}^0 \to \{i, o, g\}$$

weist jedem Paar $(e, p) \in \mathcal{D}^0$ eindeutig einen "Partition Type" zu. Die Typen sind *interior (i)*, *overlap (o)* und *ghost (g)*.

Für Entitäten höherer Kodimension gibt es zwei weitere Typen, *border (b)* und *front (f)*. *border* Entitäten bilden den Rand zwischen *interior* und *overlap* Entitäten der Kodimension 0 und *front* Entitäten bilden den Rand zwischen *interior* oder *overlap* und *ghost* Entitäten der Kodimension 0. Die Abbildung

$$t^c : \mathcal{D}^c \to \{i, o, g, b, f\}$$

realisiert die eindeutige Zuordnung für $c > 0$.

Die Zerlegung in "Partition Types" ist in Abb. 5.2 veranschaulicht.
**Bemerkung 5.3** (Eindeutigkeit von Partition Types)**.**
Eine Entität $e \in \mathcal{E}^0$ hat den Partition Type *interior* in genau einem Prozess. Die *interior* Entitäten der Kodimension 0 bilden also eine nicht-überlappende Zerlegung von $\mathcal{E}^0$. Im Gegensatz dazu existieren *overlap* oder *ghost* Entitäten auf mehreren Prozessen.

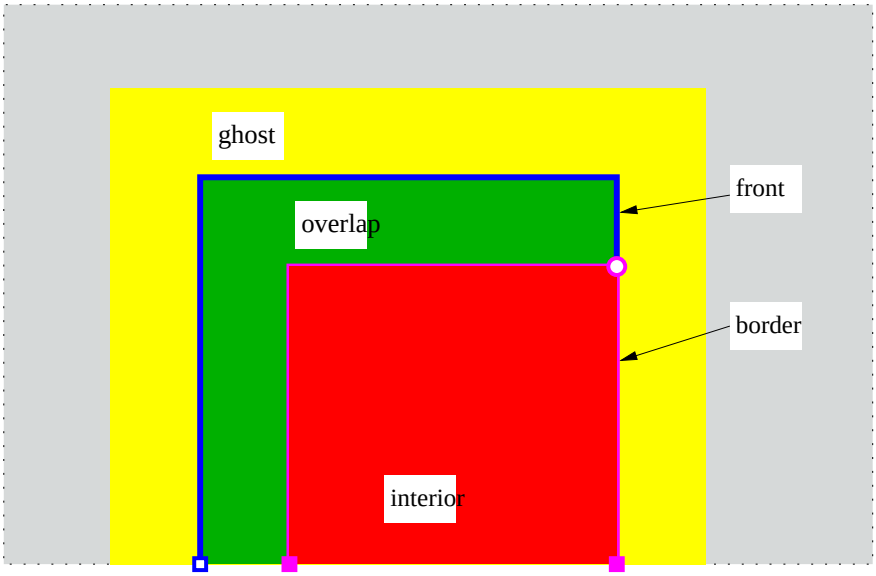



Abbildung 5.2: Gebietszerlegung mit Zuordnung von "Partition Types".

Zum Datenaustausch werden in DUNE Kommunikations-Schnittstellen genutzt, die wir abstrakt wie folgt definieren.

**Definition 5.4** (Communication Interface)**.**
Seien die Mengen $\Sigma_p \subset \mathcal{E}_p$ der "Source"-Entitäten und $\Delta_p \subset \mathcal{E}_p$ der "Destination" Entitäten für alle $p \in P$ gegeben. Dann heißt ein Paar

$$(\Sigma_p, \Delta_q)$$

Communication Interface. Wird eine Kommunikation mit dem Communication Interface $(\Sigma_p, \Delta_q)$ aufgerufen, so werden alle Daten, die mit einer Entität $e \in \Sigma_p \cap \Delta_q$ assoziiert sind von Prozess $p$ nach Prozess $q$ verschickt (forward communication) oder umgekehrt (backward communication).

In Abb. 5.3 wird die Kommunikation an zwei Beispielen illustriert.

In DUNE sind spezielle Kommunikations-Schnittstellen vordefiniert, diese sind

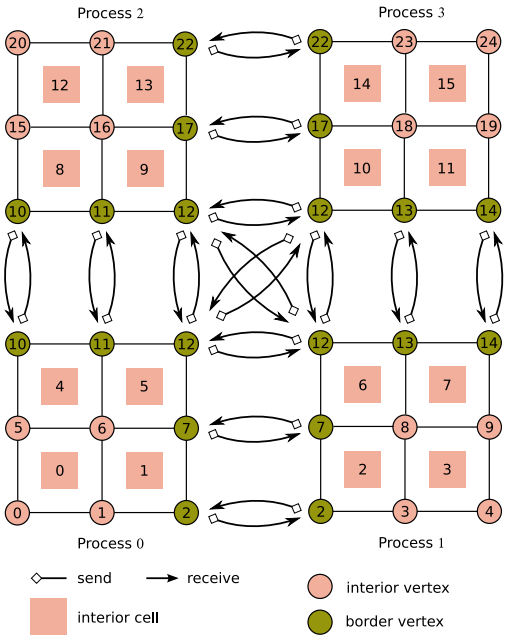
| | |
|---|---|
| `InteriorBorder_InteriorBorder_Interface` | sende/empfange interior und border Entitäten |
| `InteriorBorder_All_Interface` | sende interior und border, empfange alle |
| `Overlap_OverlapFront_Interface` | sende overlap, empfange overlap und front |
| `Overlap_All_Interface` | sende overlap, empfange alle Entitäten |
| `All_All_Interface` | sende/empfange alle Entitäten |

### 5.4.2 Klassen und Methoden zur Parallelisierung in DUNE

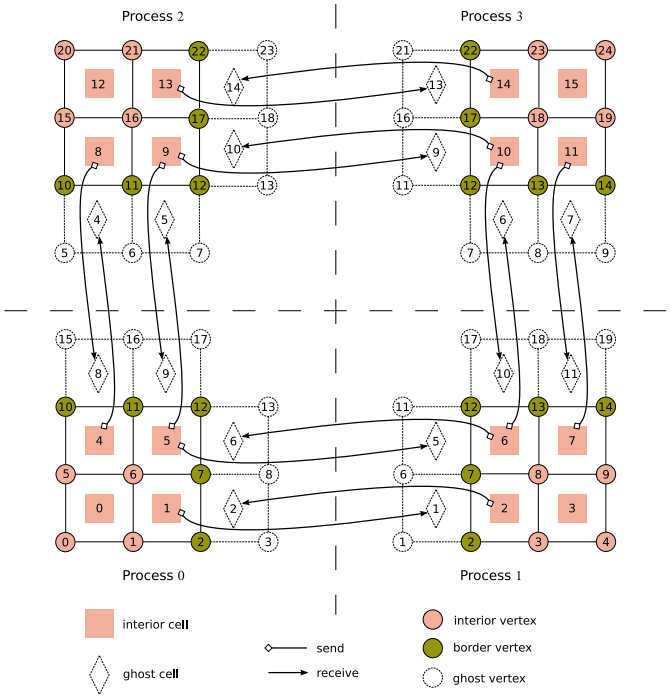
Im folgenden stellen wir die Klassenkonzepte zur Parallelisierung in DUNE vor.

#### Die Klasse `MPIHelper`

Kapselt die Initialisierung und grundlegende Funktionalität von MPI (Implementierung als *Singleton*).




(a) Nicht-überlappende Gebietszerlegung mit Kommunikation




(b) Gebietszerlegung mit dem Ghost-Zellen Ansatz und Kommunikation.

Abbildung 5.3: Partition Types und Datenaustausch für eine border–border (a) und eine interior–ghost (b) Kommunikation

**Methoden:**

- `static MPIHelper &instance (int &argc, char **&argv)`: Ruft `MPI_Init` auf und gibt eine Instanz vom Typ MPIHelper zurück (als Singleton). `MPI_Finalize` wird automatisch am Programmende aufgerufen.
- `static MPICommunicator getCommunicator ()`: Gibt den Default-Kommunikator `MPI_COMM_WORLD` zurück.
- `int rank () const`: Gibt den Rang des Prozesses zurück.
- `int size () const`: Gibt die Anzahl der Prozesses zurück.

Der Rumpf eines parallelen Programms sieht mit DUNE demnach wie folgt aus:

```
1  #include <dune/common/mpihelper.hh>
2
3  // main routine
4  int main( int argc, char **argv )
5  {
6    // Initialize MPI by calling MPI_Init, finalize is done automatically
7    // on exit. An instance of the MPIHelper class is returned.
8    MPIHelper& mpiHelper = MPIHelper::instance( argc, argv );
9
10   // get rank from helper
11   int rank = mpiHelper.rank();
12   // get size from helper
```

```
13    int size = mpiHelper.size();
14    //...
15
16    return 0;
17  }
```

### Methoden zur parallelen Kommunikation in der `Grid`-Klasse

**Zentrale Methoden:**

- `bool loadBalance ()`: Re-balanziert die Last der Gebietszerlegung für das parallele Gitter. Gibt `true` zurück, falls das Gitter verändert wurde.
- `bool loadBalance (DataHandle &data)`: Re-balanziert die Last der Gebietszerlegung für das parallele Gitter und damit assoziierte Daten. Gibt `true` zurück, falls das Gitter verändert wurde. Die Klasse `DataHandle` werden wir weiter unten noch genauer betrachten.
- `int overlapSize (int codim) const`: Gibt die Größe der Overlap-Region für eine gegebene Kodimension auf dem Blattgitter an.
- `int ghostSize (int codim) const`: Gibt die Größe der Ghost-Zellen-Region für eine gegebene Kodimension auf dem Blattgitter an.
- `void communicate (CommDataHandleType &data, CommInterfaceType comif, CommunicationDirection dir) const`: Kommuniziert Daten, die in der Klasse CommDataHandleType spezifiziert sind über die Kommunikations-Schnittstelle `comif` in Richtung `dir`. Die Klasse CommDataHandle werden wir noch genauer betrachten. `communicate` realisiert also eine Kommunikation im Sinne von Definition 5.4.
- `const Codim < 0 >::CollectiveCommunication &comm () const`: Liefert ein Objekt vom Typ CollectiveCommunication. Diese Klasse werden wir noch genauer betrachten.

Eine Routine zum Anlegen eines Gitters mit Gebietszerlegung könnte wie folgt aussehen. Dabei wird von der Klasse `GridPtr` Gebrauch gemacht, die es erlaubt ein Gitter mit Hilfe eines DGF-Files zu erzeugen. DGF steht für DUNE-*Grid-Format*.

```
1  // create grid from macro grid file
2  template< class GridType >
3  void createNewGrid( int rank )
4  {
5    // set name of DGF-file
6    std::string macroGridName( "MacroGrid_DGF" );
7
8    // initialize parallel grid,
9    // MPICOMM=MPIHelper::getCommunicator() is set as default
10   GridPtr< GridType > gridPtr( macroGridName );
11   GridType& grid = *gridPtr;
12
13   if( rank == 0 )
14   {
15     std::cout << "Created␣Grid␣with␣" << grid.size(0) << "␣elements\n";
16   }
17
```

```
18    // (re-)distribute grid
19    grid.loadBalance();
20  }
```

### Die Klasse `CommDataHandleIF< DataHandle, DataType >`

Diese Klasse definiert den DatenTyp für die Kommunikation und liefert `gather` und `scatter` Methoden zur Kommunikation der Daten.

**Wichtigste Methoden:**

- `bool contains (int dim, int codim) const`: Gibt `true` zurück, falls Daten für Entitäten der Dimension `dim` und Kodimension `codim` kommuniziert werden sollen.
- `bool fixedsize (int dim, int codim) const`: Gibt `true` zurück, falls die Anzahl der Daten für Entitäten der Dimension `dim` und Kodimension `codim` immer gleich ist.
- `size (const EntityType &en) const`: Gibt die Anzahl der Objekte vom Typ `DataType` zurück, die für die Entität `en` kommuniziert werden sollen.
- `gather (MessageBufferImp &buff, const EntityType &en) const`: Packt die Daten zur Entität `en` ein und schreibt sie in den message buffer `buff`.
- `scatter (MessageBufferImp &buff, const EntityType &en, size_t n)`: Liest `n` Objekte, die zur Entität `en` gehören aus dem message buffer `buff` aus.

### Die Klasse `CollectiveCommunication< C >`

Diese Klasse kapselt einige Routinen zur (globalen) Kommunikation.

**Wichtigste Methoden:**

- `int rank () const`: Gibt den `rank` des Prozesses zurück (zwischen 0 und size()-1).
- `int size () const` Gibt die Anzahl der Prozesse zurück.
- `T sum (T &in) const`: Berechnet die Summe des Arguments über alle Prozesse und liefert das Resultat an alle Prozesse. Es wird angenommen, dass Objekte vom Type `T` einen +-Operator hat. template<typename T >
- `int sum (T *inout, int len) const`: Berechnet die Summe jeder Komponente eines Arrays über alle Prozesse und liefert das Resultat an alle Prozesse.
- `T prod (T &in) const`
- `int prod (T *inout, int len) const`
- `T min (T &in) const`
- `int min (T *inout, int len) const`
- `T max (T &in) const`
- `int max (T *inout, int len) const`
- `int barrier () const`: Setzt eine Barriere.

- `int broadcast (T *inout, int len, int root) const`: Verteilt ein Array von dem Prozess mit Rank `root` an alle Prozesse.
- `int gather (T *in, T *out, int len, int root) const`: Jeder Prozess sendet ein Array der Länge `len` zum `root` Prozess (`root` inklusive). Im `root` Prozess werden diese Arrays in der Reihenfolge der Ranks in `out` gespeichert.
- `int allreduce (Type *inout, int len) const`: Berechnet etwas über alle Prozesse hinweg für jede Komponente eines Arrays und liefert das Resultat an alle Prozesse. Die Rechenoperation wird über ein Template-Argument definiert.

## 5.5 Parallele Algorithmen auf verteilten Gittern

In diesem letzten Abschnitt geben wir einen Ausblick zu parallelen Algorithmen auf verteilten Gittern. Dazu sehen wir uns zunächst iterative Verfahren zur Lösung linearer Gleichungssysteme an. Im folgenden setzen wir eine nicht-überlappende Gebietszerlegung voraus, wie sie in Abb. 5.3 (a) dargestellt ist. Jedem Knoten sei im Sinne von Finite-Elemente-Verfahren eine Komponente eines Vektors zugeordnet.

### 5.5.1 Vektortypen und grundlegende Operationen

Bei einer nicht-überlappenden Gebietszerlegung, kommen die *interior*-Knoten genau in einem Prozess vor, während die *border*-Knoten mehreren Prozessen zugeordnet sein können.

Je nachdem, ob ein Datum in einem *border*-Knoten dem vollen Wert eines globalen Vektors entspricht, oder nur einem Anteil unterscheiden wir folgende Typen von Vektoren:

**Typ-I-Vektor $v$:** Jeder Eintrag entspricht dem globalen Wert.

**Typ-II-Vektor $v$:** Die Summe der lokalen Einträge entspricht dem globalen Wert.

Bei Finite-Elemente-Verfahren treten Typ-II-Vektoren natürlich auf, da beispielsweise der Rechte-Seite-Vektor $f$ durch die Integration über Entitäten definiert ist. Analoges gilt für die Steifigkeitsmatrix.

Ein lineares Gleichungssystem, dass aus einem Finite Elemente Verfahren kommt, könnte man also in der Form

$$Au = f$$

schreiben. Zur Aufstellung der rechten Seite und der Steifigkeitsmatrix, ist dann keine Kommunikation nötig.

### 5.5.2 Rechenoperationen mit Typ-I- und Typ-II-Vektoren/Matrizen

**Addition:**
Die Addition von Vektoren des gleichen Typs erfordert keine Kommunikation. Bei der Umwandlung von Typ II zu Typ I müssen mit einer Kommunikation alle Einträge von *border*-Knoten aufaddiert werden und das Ergebnis an alle Prozesse verteilt werden. Eine Umwandlung von Typ I in Typ II ist nicht eindeutig.

**Skalarprodukt:**
Das Skalarprodukt von unterschiedlichen Vektortypen erfordert bzgl. der Kommunikation lediglich die Summation einer reellen Zahl:

$$(v, w) = \sum_{p=0}^{K-1} (v_p, w_p).$$

Ein Skalarprodukt von Vektoren gleichen Typs, erfordern zunächst eine Typumwandlung mit Kommunikation.

**Matrix-Vektor Multiplikation:**
Multipliziert man eine Typ II Matrix mit einem Typ I Vektor, so erhält man einen Typ II Vektor, d.h.

$$Au = v.$$

Dazu ist keine Kommunikation nötig, sondern lediglich lokale Matrix-Vektor-Multiplikationen. Benötigt man als Resultat einen Typ I Vektor, so ist eine Typumwandlung durch Addition der Einträge der *border*-Knoten notwendig,

$$v = R(v).$$

### 5.5.3 Paralleles CG-Verfahren

**Definition 5.5** (Allgemeines Abstiegsverfahren)**.**
Seien $(\beta_n)_{n\in\mathbb{N}}$ und $(t_n)_{n\in\mathbb{N}}$ gegeben mit $\beta_n \in [0, 2]$ und $t_n \in \mathbb{R}^m$. Dann heißt die Folge $(z_n)_{n\in\mathbb{N}}$, $z_n \in \mathbb{R}^m$ Lösung des Abstiegsverfahrens mit Startwert $z_1 \in \mathbb{R}^m$ wenn gilt:

$$r_1 = b - Az_1$$

und für $n = 1, 2, \ldots$ gilt

$$\alpha_n = \beta_n \frac{(t_n, r_n)}{(At_n, t_n)},$$
$$z_{n+1} = z_n + \alpha_n t_n,$$
$$r_{n+1} = b - Az_{n+1} = r_n - \alpha_n At_n.$$

**Definition 5.6** (Serielles CG-Verfahren)**.**

**Idee.**
Ein A-orthogonales System von Suchrichtungen wird mit Hilfe des Schmidt'schen Orthogonalisierungsverfahrens bzgl. $(\cdot, \cdot)_A$ schrittweise aufgebaut.

Wähle also $\beta_n = 1 \ \forall n = 1, \ldots, m + 1$, $t_1 = r_1$ und

$$t_n = r_n + \gamma_{n-1} t_{n-1}$$

mit $\gamma_{n-1} := -\frac{(Ar_n, t_{n-1})}{(At_{n-1}, t_{n-1})}$ für $n = 2, 3, \ldots$.

Das CG-Verfahren kann algorithmisch wie folgt geschrieben werden:

**Definition 5.7** (Serieller CG-Algorithmus).

$z := z_1$
$r := f - Az$
$s := t := r$
$\sigma := \sigma_{old} := \sigma_1 := (t, r)$
wiederhole
$v := As$
$\alpha := \sigma/(s, v)$
$z := z + \alpha s$
$r = r - \alpha v$
$t = r$
$\sigma := (t, r)$
$\beta := \sigma/\sigma_{old}$
$\sigma_{old} := \sigma$
$s := t + \beta s$
bis
$\sigma/\sigma_0 < TOL$

Wir haben diese Notation bewusst gewählt, da sich das CG-Verfahren nun mit Hilfe der Elementaroperationen für Typ-I- und Typ-II-Vektoren ganz einfach parallelisieren lässt. Dazu wählen wir als Typ I Vektoren $z, s, t$ und als Typ II Vektoren $v, r, f$, sowie die Matrix $A$.

**Definition 5.8** (Paralleler CG-Algorithmus).

$z := z_1$
$r := f - Az$
$s := t := R(r)$
$\sigma := \sigma_{old} := \sigma_1 := (t, r)$
wiederhole
$v := As$
$\alpha := \sigma/(s, v)$
$z := z + \alpha s$
$r = r - \alpha v$
$t = R(r)$
$\sigma := (t, r)$
$\beta := \sigma/\sigma_{old}$
$\sigma_{old} := \sigma$
$s := t + \beta s$
bis
$\sigma/\sigma_0 < TOL$

Bei diesem Algorithmus, findet pro Iteration eine Typumwandlung $R(r)$ mit Kommunikation statt. Darüber hinaus sind zwei Skalarprodukte mit minimaler Kommunikation zu berechnen. Alle anderen Rechenoperationen sind lokal durchführbar.

Andere Iterative Lösungsverfahren (z.B. GMRES, BI-CG-STAB, ...), die lediglich auf Skalarprodukten und Matrix-Vektor-Operationen beruhen, lassen sich auf ähnliche Art parallelisieren.

### 5.5.4 Überlappende Gebietszerlegungsverfahren

### Klassisches Schwarz-Verfahren

Wir betrachten das Laplace-Problem auf einem Gebiet $\Omega$:

$$\begin{aligned} -\Delta u &= f, && \text{in } \Omega, \\ u &= 0, && \text{auf } \partial\Omega. \end{aligned}$$

Nun nehmen wir an, dass $\Omega$ in überlappende Gebiete zerlegt ist, d.h. $\Omega = \Omega_1 \cup \Omega_2$ mit $\Omega_1 \cap \Omega_2 \neq \emptyset$.

Wir definieren

$$\Gamma_1 = \partial\Omega_1 \cap \Omega_2, \quad \Gamma_2 = \partial\Omega_2 \cap \Omega_1.$$

Das alternierende Schwarz-Verfahren bestimmt die kontinuierliche Lösung $u$ in ganz $\Omega$ durch abwechselndes Lösen in den Teilgebieten $\Omega_1$ und $\Omega_2$. Dazu setzen wir im Iterationsschritt $k$:

$$\begin{aligned} u_i^k &\quad \text{Lösung in } \Omega_i, i = 1, 2, \\ u_1^k|_{\Gamma_2} &\quad u_1^k \text{ ausgewertet auf } \Gamma_2, \\ u_2^k|_{\Gamma_2} &\quad u_2^k \text{ ausgewertet auf } \Gamma_1, \end{aligned}$$

Dann lautet die alternierende Schwarz-Iteration bei gegebenem Startwert $u_0$ (in ganz $\Omega$).

**Definition 5.9** (Alternierende Schwarz-Iteration)**.**

Für $k = 0, 1, 2, \ldots$ {
  Löse {
    $-\Delta u_1^{k+1} = f, \quad$ in $\Omega_1$,
    $u_1^{k+1} = u^k|_{\Gamma_1}, \quad$ auf $\Gamma_1$,
    $u_1^{k+1} = 0, \quad$ auf $\partial\Omega_1 \setminus \Gamma_1$,
  }
  Löse {
    $-\Delta u_2^{k+1} = f, \quad$ in $\Omega_2$,
    $u_2^{k+1} = u^k|_{\Gamma_2}, \quad$ auf $\Gamma_2$,
    $u_2^{k+1} = 0, \quad$ auf $\partial\Omega_2 \setminus \Gamma_2$,
  }
  Definiere $u^{k+1}$ durch $u_2^{k+1}$ auf $\Omega_2$ und durch $u_2^{k+1}$ auf $\Omega \setminus \Omega_2$.
}

Man kann zeigen, dass für die entsprechende schwache Form des Verfahrens gilt

$$\|u - u^{k+1}\|_{H_0^1(\Omega)} \leq \rho \|u - u^k\|_{H_0^1(\Omega)}.$$

Dabei hängt $\rho$ von der Form der Teilgebiete ab.

Die alternierende Schwarz-Methode ist der Prototyp einer überlappenden Gebietszerlegungsmethode. Sie wurde von Schwarz bereits 1890 eingeführt, um die Existenz von Lösungen auf komplexen Gebieten zu zeigen. Erst sehr viel später wurden auf dieser Grundlage parallele Implementierungen realisiert.

### Schwarz-Verfahren auf mehreren Teilgebieten

Das alternierende Schwarz-Verfahren lässt sich auch auf eine beliebige Anzahl von Teilgebieten verallgemeinern. Sei dazu wieder $\Omega$ unser Gebiet. Zur Konstruktion von überlappenden Teilgebieten gehen wir in zwei Schritten vor:

**1. Schritt:** Konstruktion einer nicht-überlappenden Zerlegung:
Dazu zerlegen wir $\Omega$ in Teilgebiete $\hat{\Omega}_i, i = 1, \ldots, p$, so dass

$$\bar{\Omega} = \bigcup_{i=1}^{p} \overline{\hat{\Omega}_i}, \quad \hat{\Omega}_i \cap \hat{\Omega}_j = \emptyset, \quad \forall i \neq j.$$

**2. Schritt:** In einem nächsten Schritt erweitern wir jedes Teilgebiet $\Omega_i$, wie folgt

$$\Omega_i := \{x \in \Omega \mid \operatorname{dist}(x, \hat{\Omega}_i) < \beta H\}.$$

Dabei bezeichnet $H$ eine typische Gitterfeinheit und $\beta$ ist ein Skalierungsfaktor, der die Größe des "Overlapps" angibt.

Nun könnten wir die diskrete alternierende Schwarz-Iteration dadurch definieren, dass wir

- alle Teilgebiete in vorgegebener Reihenfolge abarbeiten und
- in jedem Teilgebiet eine neue Lösung unter Beachtung jeweils neuester Randwerte ausrechnen.

Bei Finite Elemente Diskretisierungen mit Lagrange Elementen, kann man die Einträge der DOF-Vektoren den Knoten des Gitters zuordnen. Dadurch lässt sich die Idee der alternierenden Schwarz-Verfahren auch direkt auf der Ebene der resultierenden linearen Gleichungssysteme formulieren. Varianten dieser Idee liefern das Multiplikative, oder das Additive Schwarz-Verfahren. Für weitere Details verweisen wir auf die Literatur (vgl. Skript von Peter Bastian [2] und die Literaturangaben darin).

### 5.5.5 Nichtüberlappende Gebietszerlegungsverfahren

Analog zum Fall überlappender Gebietszerlegungsverfahren betrachten wir das Laplace-Problem auf einem Gebiet $\Omega$:

$$\begin{aligned} -\Delta u &= f, && \text{in } \Omega, \\ u &= 0, && \text{auf } \partial\Omega. \end{aligned}$$

Nun nehmen wir an, dass $\Omega$ in nicht-überlappende Gebiete zerlegt ist, d.h. $\bar{\Omega} = \bar{\Omega}_1 \cup \bar{\Omega}_2$ mit $\Omega_1 \cap \Omega_2 = \emptyset$.
Wir bezeichnen mit $\Gamma := \bar{\Omega}_1 \cap \bar{\Omega}_2$ das Interface zwischen $\Omega_1$ und $\Omega_2$.

**Definition 5.10** (Mehr-Gebiets-Formulierung)**.**
Sei $n$ eine festgelegte Normale an $\Gamma$. Dann ist das Laplace-Problem äquivalent zu

$$\begin{aligned} -\Delta u_1 &= f && \text{in } \Omega_1, \\ u_1 &= 0, && \text{auf } \partial\Omega_1 \cap \partial\Omega, \\ u_1 &= u_2, && \text{auf } \Gamma, \\ \nabla u_1 \cdot n &= \nabla u_2 \cdot n, && \text{auf } \Gamma, \\ u_2 &= 0, && \text{auf } \partial\Omega_2 \cap \partial\Omega, \\ -\Delta u_2 &= f && \text{in } \Omega_2, \end{aligned}$$

wobei $u$ auf $\Omega_1$ durch $u_1$ und auf $\Omega_2$ durch $u_2$ gegeben ist.

Diese Form des Laplace-Problems suggeriert folgendes iteratives Verfahren

**Definition 5.11** (Paralleles Dirichlet-Neumann-Verfahren)**.**
Seien $\theta \in (0, 1]$, $\lambda^0, \mu^0$ gegeben. Dann ist das parallele Dirichlet-Neumann-Verfahren gegeben durch:

Für $k = 0, 1, 2, \ldots$ {
  Löse {
    $-\Delta u_1^{k+1} = f, \quad$ in $\Omega_1$,
    $u_1^{k+1} = \lambda^k, \quad$ auf $\Gamma$,
    $u_1^{k+1} = 0, \quad$ auf $\partial\Omega_1 \cap \partial\Omega$,
  }
  Löse {
    $-\Delta u_2^{k+1} = f, \quad$ in $\Omega_2$,
    $\nabla u_2^{k+1} \cdot n = \mu^k, \quad$ auf $\Gamma_2$,
    $u_2^{k+1} = 0, \quad$ auf $\partial\Omega_2 \cap \partial\Omega$,
  }
  Definiere {
    $\lambda^{k+1} := u_2^{k+1}$ auf $\Gamma$,
    $\mu^{k+1} := \theta \nabla u_1^{k+1} \cdot n + (1 - \theta)\mu^k$ auf $\Gamma$.
  }
}

Dabei ist $\theta$ ein Beschleunigungsparameter.

Alternativ zu der Dirichlet-Neumann-Methode, kann auch eine Neumann-Neumann, oder eine Robin-Robin-Methode definiert werden. Für weitere Details verweisen wir auf die Literatur (siehe z.B. das Buch von Quarteroni [18]).

# Literaturverzeichnis


[1] Vorlesungsskript paralleles höchstleistungsrechnen, heidelberg, 2008. `http://conan.iwr.uni-heidelberg.de/teaching/scripts/pr1.pdf`, .

[2] Vorlesungsskript parallele lösung großer gleichungssysteme, heidelberg, 2009. `http://conan.iwr.uni-heidelberg.de/teaching/parsolve_ss2009/dd.pdf`, .

[3] DUNE-FEM – The FEM Module. `http://dune.mathematik.uni-freiburg.de`, .

[4] DUNE – Distributed and Unified Numerics Environment. `http://dune-project.org/`, .

[5] M. Ainsworth. Robust a posteriori error estimation for nonconforming finite element approximation. *SIAM J. Numer. Anal.*, 42(6):2320–2341, 2005.

[6] D. N. Arnold, F. Brezzi, B. Cockburn, and L. D. Marini. Unified analysis of discontinuous Galerkin methods for elliptic problems. *SIAM J. Numer. Anal.*, 39(5):1749–1779, 2002.

[7] P. Bastian, M. Blatt, A. Dednder, C. Engwer, R. Klöfkorn, M. Ohlberger, and O. Sander. DUNE grid howto. `http://www.dune-project.org/doc/grid-howto/grid-howto.html`.

[8] R. Bustinza, G. N. Gatica, and B. Cockburn. An a posteriori error estimate for the local discontinuous Galerkin method applied to linear and nonlinear diffusion problems. 2005.

[9] B. Cockburn. Discontinuous Galerkin methods. *ZAMM Z. Angew. Math. Mech.*, 83(11):731–754, 2003.

[10] A. Dedner, C. Makridakis, and M. Ohlberger. Error control for a class of Runge-Kutta discontinuous Galerkin methods for nonlinear conservation laws. *SIAM J. Numer. Anal.*, 45(2): 514–538, 2007.

[11] P. Deuflhard and F. Bornemann. *Scientific computing with ordinary differential equations. Transl. from the German by Werner C. Rheinboldt.* Texts in Applied Mathematics. 42. New York, NY: Springer. xix, 485 p., 2002.

[12] W. Dörfler. Orthogonale Fehler-Methoden. `http://www.mathematik.uni-freiburg.de/IAM/Teaching/scripts/paper01.html`.

[13] E. H. Georgoulis and O. Lakkis. A posteriori error control for discontinuous galerkin methods for parabolic problems. Preprint arxiv:0804.4262v1, 2008.

[14] W. Hackbusch. *Iterative Lösung großer schwachbesetzter Gleichungssysteme. (Iterative solution of large sparse systems of equations).2., Überarb. u. erw. Aufl.* Stuttgart: Teubner. 404 S. , 1993.

[15] O. A. Karakashian and F. Pascal. A posteriori error estimates for a discontinuous Galerkin approximation of second-order elliptic problems. *SIAM J. Numer. Anal.*, 41(6):2374–2399, 2003.

[16] J. Málek, J. Nečas, M. Rokyta, and M. Růžička. *Weak and Measure-valued Solutions to Evolutionary PDEs.*, volume 13 of *Applied Mathematics and Mathematical Computation*. Chapman and Hall, London, Weinheim, New York, Tokyo, Melbourne, Madras, 1968.

[17] L. Pareschi and G. Russo. Implicit-explicit runge-kutta schemes and applications to hyperbolic systems with relaxation. *Journal of Scientific computing*, 25(1):129–155, 2005.

[18] Alfio Quarteroni and Alberto Valli. *Domain decomposition methods for partial differential equations*. Numerical Mathematics and Scientific Computation. The Clarendon Press Oxford University Press, New York, 1999. Oxford Science Publications.

[19] B. Riviere and M.F. Wheeler. A posteriori error estimates for a discontinuous Galerkin method applied to elliptic problems. *Comput. Math. Appl.*, 46(1):141–163, 2003.

[20] C.-W. Shu. A survey of strong stability preserving high order time discretizations. In *Collected lectures on the preservation of stability under discretization (Fort Collins, CO, 2001)*, pages 51–65. SIAM, Philadelphia, PA, 2002.

[21] K. Strehmel and R. Weiner. *Numerik gewöhnlicher Differentialgleichungen. (Numerical methods for ordinary differential equations).* Teubner Studienbücher: Mathematik. Stuttgart: Teubner. 462 p. , 1995.

[22] M. Zennaro. Natural continuous extensions of Runge-Kutta methods. *Math. Comput.*, 46: 119–133, 1986.

[23] Qiang Zhang and Chi-Wang Shu. Error estimates to smooth solutions of Runge-Kutta discontinuous Galerkin methods for scalar conservation laws. *SIAM J. Numer. Anal.*, 42(2):641–666, 2004.

# Index