Wissenschaftliches Rechnen und Anwendungen in der Strömungsmechanik

SS 06 gelesen von Dr. Mario Ohlberger und Dr. Andreas Dedner

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 0

Überblick

Aufbau der Vorlesung

1) Einleitung (Standard FEM, Einführung DG)
2) Local Dicontinuous Galerkin Verfahren (LDG) für elliptische PDGLn
3) Programmierkonzepte und abstraktes Gitterinterface
4) Implementierungskonzepte für LDG
5) LDG für parabolische Probleme ($\partial_t u - \Delta u = f$)
6) Implementierung für Evolutionsgleichungen
7) LDG für hyperbolische PDGL ($\partial_t u + \nabla \cdot F(u) = 0$)

DUNE (**D**istributed and **U**nified **N**umerics **E**nvironment)

DUNE is a software framework for the numerical solution of partial differential equations with grid-based methods. It is based on the following main principles:

1) Seperation of data structures and algorithms by abstract interfaces. This provides more functionality with less code and also ensures maintainability and extendability of the framework.
2) Efficient implementation of these interfaces using generic programming techniques. Static polymorphism allows the compiler to do more optimizations, in particular function inlining, which in turn allows the interface to have very small functions (implemented by one or few machine instructions) without a severe performance penalty. In essence the algorithms are parametrized with a particular data structure and the interface is removed at compile time. Thus the resulting code is as efficient as if it would have been written for the special case.
3) Reuse of existing finite element packages with a large body of functionality.

Dieser kurze Auszug über DUNE stammt von der offiziellen Website des Projekts:
`http://www.dune-project.org/` [1].
Über die Arbeit der Forschungsgruppe in Freiburg informiert die Seite:
`http://www.mathematik.uni-freiburg.de/IAM/Research/projectskr/dune/`

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). Beim DG-Verfahren wird der Ansatzraum mit stückweise polynomialen Funktionen gewählt, aber es wird keine Stetigkeit zwischen Gitterzellen gefordert.

Resultat: $V_h \not\subset X$, falls X: kontinuierlicher Lösungsraum,
V_h: diskreter Lösungsraum.

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

Implementiertung: Möglichst allgemeiner Schnittstellen-Ansatz.

1) Ermittlung von Eigenschaften eines Gitters, die zur Diskretisierung von PDGLn 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

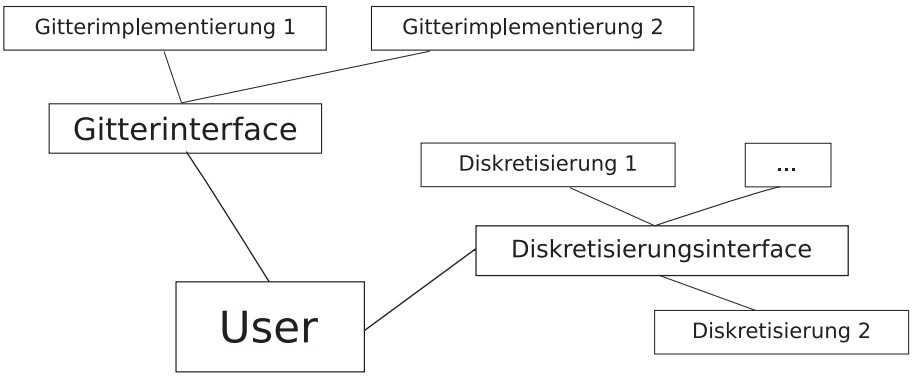


Abbildung 1.1: schematischer Aufbau des Interface

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

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

falls gilt

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

mit

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

FEM-Diskretisierung 1.2

Algorithmus der FEM-Diskretisierung

1. Schritt: Konstruktion eines Gitters $\mathcal{T}_h := \{T_i\}_{i\in I}$ auf Ω (genauere Bedingungen s.u.).

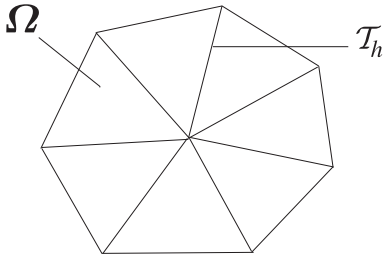


Abbildung 1.2: eine Triangulierung

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

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

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 $\{v_1, \ldots, v_N\}$ Basis von V_h, genügt es, (1.2) mit v_i, $i = 1, \ldots, N$, zu testen:

$$a(u_h, v_i) = b(v_i) \ \forall i \in \{1, \ldots, N\}$$

$$\Longrightarrow \sum_{j=1}^{N} u_j a(v_j, v_i) = b(v_i) \ \forall i \in \{1, \ldots, N\}. \tag{1.3}$$

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.

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

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 LGSe.

→ **Finite Elemente Ansatz basierend auf Referenzelementen**.

1.1 Gitterdefinition

Definition 1.3 (Dreicksgitter in 2D, Referenzsimplex, Subentitäten)
Sei 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) $\overline{T_i} \cap \overline{T_j} = \begin{cases} \overline{T_i}, & i = j, \\ \emptyset, & sonst, \\ e_{ij}, p_k \end{cases}$

mit e_{ij} *: gemeinsame Kante von* T_i *und* T_j,
p_k*: Knoten in* T_i *und* T_j.

Definiton 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 Entitiäten) 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,*

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

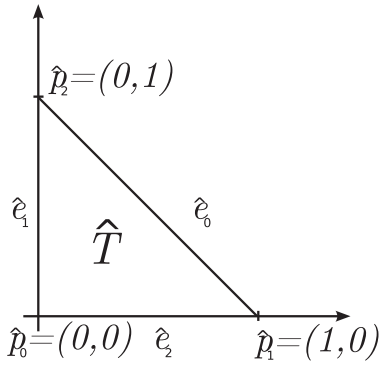


Abbildung 1.3: Referenzsimplex in 2D

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 ${p_i}^T := F_T(\hat{p}_i)$.

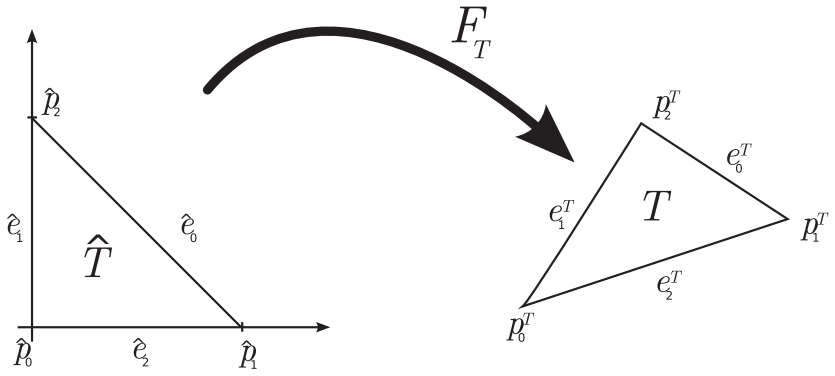


Abbildung 1.4: Referenzabbildung in 2D

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 $c = 0, \ldots, n = 2$ (bzw. codim c Entitäten), sind:

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

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 FE:

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

Benötigen zur Implementierung 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 Notatation nehmen wir an, dass $\mu^c : \mathcal{E}^c \to \{1, \ldots, N_c\}$ 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 lokalen auf globale Nummerierung induziert.

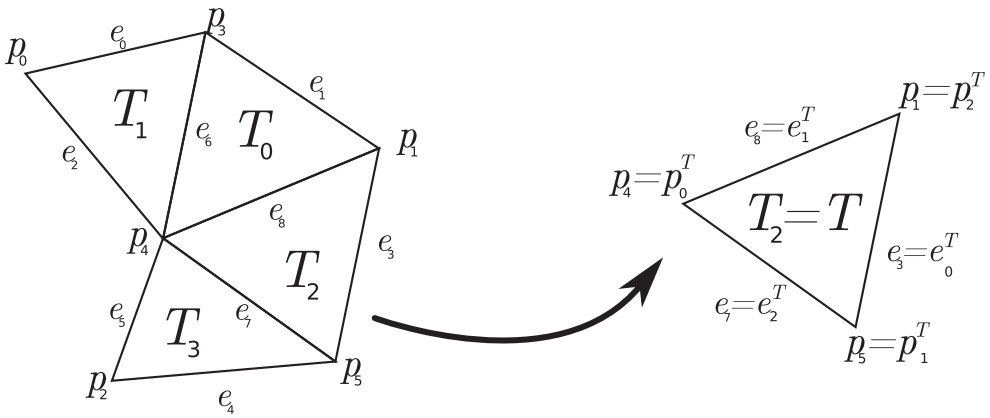


Abbildung 1.5: globale/lokale Nummerierung

1.2 Diskrete Funktionen

Wir betrachten hier stückweise lineare und stetige FE 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 FEM 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 geg. $T \in \mathcal{T}_h$ und $\hat{x} \in \hat{T}$.

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 $\Phi^T = \{{\varphi_0}^T, {\varphi_1}^T, {\varphi_2}^T\}$ 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

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

$\Longrightarrow \Phi = \{\varphi_1, \ldots, \varphi_{N^2}\}$ *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 DOF)
*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*

$$v_h|_T = v_T = \sum_{i=0}^{2} {v_k}^T {\varphi_k}^T$$

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

Allgemein: $\nu^T : \hat{\Phi} \to \mathbb{N}$ mit (für lineare FE) $\nu_T(\hat{\varphi}_k) = \mu^2({p_k}^T)$ beschreibt die 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 wird $\nu^T(k) = \nu^T(\hat{\varphi}_k)$.

1.3 Assemblieren des linearen Gleichungssystems

1.3.1 Rechte Seite

Ziel: Berechnen von $b_i := \int_\Omega f \varphi_i$ für $i = 1, \ldots, N_2$.
Falls f keine "einfache" Funktion ist, benötigt man dafür numerische Quadraturen. Dazu reduzieren wir das Problem auf das Referenzelement.
Sei $g \in C^0(\overline{\Omega})$, dann gilt:

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

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

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* $w_\alpha = |detA_T|\hat{w}_\alpha$*, 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:

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

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

Beweis:

i) nachrechnen; gilt, da $detDf_T$ konstant

ii) Literatur

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.
Optimaler Aufwand zur Berechnung von b ist sicherlich linear.

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

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}

$$10^8 \left\{ \begin{array}{ll} \frac{1}{100} \text{ Sekunde} & \rightsquigarrow \quad 12 \text{ Tage} \\ \text{Speicher (8 byte)} & \rightsquigarrow \quad 80 \text{ Mb} \end{array} \right.$$

Abbildung 1.6: Daten für Aufwand zur Illustration

Es ist:

$$\begin{aligned} b &= \sum_{i=1}^{N_2} \sum_{T \in \mathcal{T}_h} Q_T(f \cdot \varphi_i) \overrightarrow{e}_i \qquad (\overrightarrow{e}_i : \ i-ter \ Einheitsvektor) \\ &= \sum_{T \in \mathcal{T}_h} \sum_{i=1}^{N_2} Q_T(f \cdot \varphi_i) \overrightarrow{e}_i \end{aligned}$$

$$\left[\varphi_i = \left\{ \begin{array}{ll} 0, & p_i \notin \mathcal{E}^2(T), \\ {\varphi_k}^T, & i = \mu^{T,2}(k), \end{array} \right. \right]$$

$$\begin{aligned} b &= \sum_{T \in \mathcal{T}_h} \sum_{k=0}^{2} Q_T(f {\varphi_k}^T) \overrightarrow{e}_{\mu^{T,2}(k)} \\ &\left[= \sum_{T \in \mathcal{T}_h} \sum_{\hat{\varphi} \in \hat{\Phi}} Q_T(f(\hat{\varphi} \circ F_T)) \overrightarrow{e}_{\nu^T(\hat{\varphi})} \right]. \end{aligned}$$

Definieren wir nun die "lokalen rechten Seiten":

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

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

Implementierung:

1) Berechen 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 $d := |detA_T|$, $y_\alpha := F_T(\hat{x}_\alpha)$, $f_\alpha := f(y_\alpha)$.
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)

$$b\left[\nu^T(\hat{\varphi})\right] += d\beta.$$

1.3.2 Assemblieren der Steifigkeitsmatrix

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

Mit der Strategie aus 1.3.1 betrachten wir zunächst lokale Steifigkeitsmatrizen:

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

$\Longrightarrow A = \sum_T {a_{kl}}^T E_{\nu^T(k),\nu^T(l)}$, mit $E_{ij} = (e_{ij}) \in \mathbb{R}^{N_2 \times N_2}$, $e_{ij} = \delta_{ij}$.
Wieder ist der Aufwand $O(|\mathcal{E}^0|)$ statt $O(|\mathcal{E}^0||\mathcal{E}^2|)$.

Zur Berechnung von ${a_{kl}}^T$:

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

(nur falls $\hat{\varphi}_k, \hat{\varphi}_l$ linear, T ist Dreieck), da ${\varphi_k}^T = \hat{\varphi}_k \circ F_T^{-1}$.
Haben Speicheraufwand $O(|\mathcal{E}^0|^2)$, obwohl A dünn besetzt ist, d.h. man speichert im wesentlichen Nullen.

Idee: Compressed Row Storage (CRS)
Sei 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.

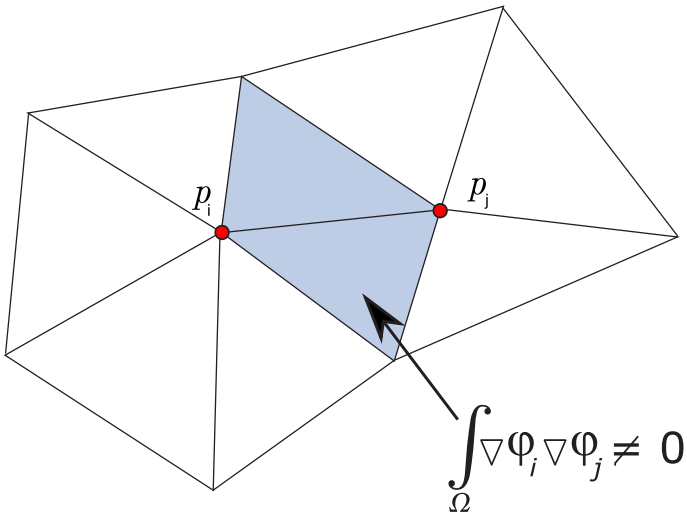


Abbildung 1.7: gemeinsamer Träger von Basisfunktionen liefert Beitrag

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

Idee: Speichere Werte $a_{ij} \neq 0$ und Spaltenposition j. Sei dazu $M > 0$ eine obere Schranke für die Anzahl der codim 1 Entitäten, die einen Knoten gemeinsam haben (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:

$$S_i = (a_{ij_1}, a_{ij_1}, \ldots, a_{ij_{M_i}}, 0, \ldots, 0),$$

wobei a_{ij} die Einträge ungleich Null sind.

$$\begin{aligned} & C_i = (j_1, j_2, \ldots, j_{M_i}, -1, \ldots, -1), \\ \Longrightarrow \quad & a_{ij} = \begin{cases} 0, & C_{ik} \neq j, k = 1, \ldots, M \\ S_{ik}, & C_{ik} = j. \end{cases} \end{aligned}$$

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

1.4 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

Ansatz sind **Krylovraum Methoden**. Aus der Numerik II bekannt ist das cg-Verfahren für positiv definite symmetrische Matrizen. Ist A nicht symmetrisch, existieren Verallgemeinerungen (gmres, bcgstab).
Siehe näheres in [8], [9].
Alternativer Ansatz: Multigrid Löser
Bei den Krylovraum Methoden wird die Matrix A nicht benötigt, sondern nur die Matrix-Vektor-Multiplikation $y = Ax$ für gegebenes $x \in \mathbb{R}^{N_2}$.

Implementierung: $x \in \mathbb{R}^{N_2}$ gegeben, A CRS-Matrix, $y \equiv 0$.

```
for (i = 1,..., C_2) {
  k = 1;
  while (C_ik ≠ -1) {
    y_i = S_ik x_{C_ik};
    k++;
  }
}
```

Aufwand: $O(|\mathcal{E}^2|)$.

Bemerkung: In realistischen Anwendungen benötigt man Vorkonditionierer, um die Anzahl der cg-Iterationen zu reduzieren.

1.5 Verallgemeinerte elliptische PDGLn

1) **Terme niedriger Ordnung:**

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

Annahme: Gleichung ist elliptisch.

FE-Ansatz: $u_h(x) = \sum_j u_j \varphi_j(x)$, dann:

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

Führt auf LGS $AU = b$ mit Matrix $A = S - G + M$,

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

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

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

Die Berechnung der Matrixeinträge und Speicherkonzepte wie ein 1.3 einsetzbar. Die Matrix A ist dünn besetzt, unter Voraussetzungen an a, b, c positiv definit (s. Literatur), aber nicht symmetrisch, da im allgemeinen G nicht symmetrisch ist.

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

2) **Allgemeine Randbedingungen:**

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

$$\begin{aligned} -\nabla\cdot(a\nabla u) + \nabla\cdot(bu) + cu &= f \; in\; \Omega, \\ u &= g_D \; auf \; \partial\Omega_D, \; \text{(Dirichlet)} \\ (a\nabla u - bu)\cdot n &= g_N \; auf \; \partial\Omega_N, \; \text{(Neumann)} \\ (a\nabla u - bu) + \alpha u &= g_R \; auf \; \partial\Omega_R, \; \text{(Robin)}, \end{aligned} \tag{1.4}$$

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

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

$$\Longrightarrow \quad \int_\Omega a\nabla u\nabla\varphi - \int_{\partial\Omega} a\nabla u\cdot n\varphi - \int_\Omega bu\nabla\varphi + \int_{\partial\Omega} bu\cdot n\varphi + \int_\Omega cu\varphi = \int_\Omega f\varphi$$

$$\Longrightarrow \quad \underbrace{\int_\Omega (a\nabla u\nabla\varphi - bu\nabla\varphi + cu\varphi)}_{=:a(u,\varphi)} - \int_{\partial\Omega}(a\nabla u - bu)\cdot n\varphi = \underbrace{\int_\Omega f\varphi}_{=:b(\varphi)}$$

$$\Longrightarrow \quad a(u,\varphi) - \underbrace{\int_{\partial\Omega_D}(a\nabla u - bu)\cdot n\varphi}_{=0,\ \text{da}\ \varphi|_{\partial\Omega_D}=0} - \underbrace{\int_{\partial\Omega_N}(a\nabla u - bu)\cdot n\varphi}_{=\int_{\partial\Omega_N} g_N\varphi} - \underbrace{\int_{\partial\Omega_R}(a\nabla u - bu)\cdot n\varphi}_{=\int_{\partial\Omega_R}(g_R\varphi - \alpha u)\varphi} = b(\varphi).$$

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

(D) 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 durch g_D 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:

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

$$\uparrow$$
$$i$$
$$\Longleftrightarrow u_i = g_D(p_i)$$

Allerdings ist A nicht mehr symmetrisch, daher:

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

$$\uparrow$$
$$i$$

(N) Neumann Ranbedingung:

$$b_i := \int_\Omega f\varphi_i + \int_{\partial\Omega_N} g_N\varphi_i, \qquad \text{für } p_i \in \mathcal{E}^2 \text{ mit } p_i \in \partial\Omega_N.$$

(R) Robin Randbedingung:

$$b_i := \int_\Omega f\varphi_i + \int_{\partial\Omega_R} g_R\varphi_i, \qquad \text{für } p_i \in \mathcal{E}^2 \text{ mit } p_i \in \partial\Omega_R,$$

und

$$M_{ij} := \int_\Omega c\varphi_j\varphi_i + \int_{\partial\Omega_R} \alpha\varphi_i\varphi_j, \qquad \text{für } p_i, p_j \in \mathcal{E}^2 \text{ mit } p_i, p_j \in \partial\Omega_R.$$

1.6 Das Discontinuous-Galerkin Verfahren

Problem:

$$-\Delta u = f \text{ in } \mathbb{R}^2,\ u = 0 \text{ auf } \partial\Omega. \tag{1.5}$$

Ist $f \in L^2(\Omega)$, Ω glattes, konvexes Gebiet, so hat (1.5) genau eine Lösung in $H^2(\Omega) \cap \mathring{H}^1(\Omega)$.

Sei $\mathcal{T}_h$ eine Zerlegung von Ω (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 stückweise H^2-Funktionen.

Die Lösung u von (1.5) ist in $H^2(\mathcal{T}_h)$, aber die Gleichung (1.5) 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 (1.5) 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.

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*

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

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 Ω 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 $S = \overline{T_i} \cap \overline{T_i}^s$.

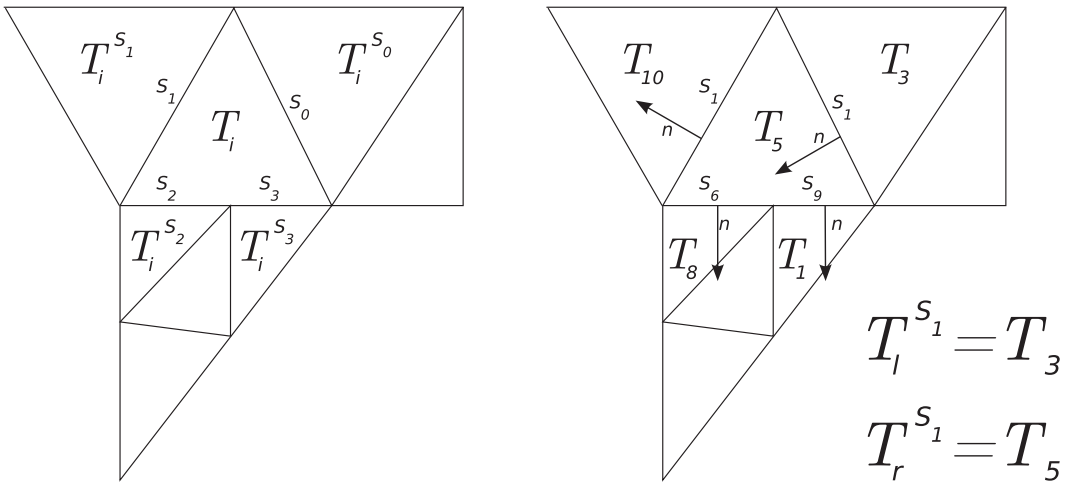


Abbildung 1.8: Beispiel Intersections

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:

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

Wir definieren zu $S \in \mathcal{I}_S$, ${T_S}^l = T$ 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:

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

*Desweitern definieren wir noch den **Sprung** von φ über S durch:*

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

*und das **Mittel** von φ:*

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

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

$$[\varphi](x) := -\varphi^-(x), \qquad \text{und} \qquad \{\varphi\}(x) := \varphi(x).$$

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

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

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

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

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:

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

Beweis:

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

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: $n=-n_S$.

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

Die Behauptung folgt mit Lemma 1.14 ii)

Wir wenden Lemma 1.15 auf die Gleichung $-\Delta u=f$ auf T an.
Sei $\varphi\in H^2(\mathcal{T}_h)$,

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

Ist u Lösung von (1.5), ist $[\nabla u\cdot n_S]=0$, d.h.

$$\int_\Omega f\varphi = \sum_T\int_T\nabla u\cdot\nabla\varphi+\sum_{S\in\mathcal{I}\cup\mathcal{I}_\partial}\int_S(\{\nabla u\cdot n_S\}[\varphi]). \tag{1.6}$$

Ist $u\in H^2(\Omega)\cap\mathring{H}^1(\Omega)$, so ist u auch klassische Lösung von (1.6). Allerdings gilt die Umkehrung nicht. So ist etwa jede stückweise konstante Funktion Lösung von (1.6) mit $f=0$, aber nur $u=0$ ist Lösung von (1.5) mit $f=0$.

Wir benötigen also weitere Terme, die berücksichtigen, dass die Lösung u von (1.5) auch die folgenden Gleichungen erfüllt:

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

Kontinuierliches DG-Problem: Wir betrachten das folgende schwache Problem:

$$B_{\alpha,\beta}(u,\varphi) = I(\varphi) \qquad \text{für alle } \varphi \in H^2(\mathcal{T}_h), \tag{1.7}$$

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

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

wobei h_S eine geeignete lokale Gitterfunktion ist.

Bemerkung: Ist u die Lösung von (1.5), so ist u Lösung von (1.7).
"Die Umkehrung gilt, falls $\beta > 0$" (ohne Beweis).

Spezielle Verfahren sind:

Interior Penalty: $\alpha = -1, \beta$ groß.

Bemerkung: Nur falls $\alpha = -1$ ist $B_{\alpha,\beta}$ symmetrisch.

Babuska, Oder, Baumann: $\alpha = 1, \beta = 0$

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

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

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

Diskeretes DG-Problem: Gesucht: $u_h \in V(\mathcal{T}_h)$ mit $B_{\alpha,\beta}(u_h,\varphi_h) = I(\varphi_h)\ \forall\varphi_h \in V(\mathcal{T}_h)$ mit

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

und für $p_T \geq 1$ für $T \in \mathcal{T}_h$.

Wahl einer Basis von $V(\mathcal{T}_h)$: 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

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

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 $P_{p_{T_i}}(\hat{T})$.

Beispiel:

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

$P_{p_{T_i}}(\hat{T}) \subset P_{p_{T_j}}(\hat{T})$, falls $p_{T_i} \leq p_{T_j}$.
Implementierung: Assemblieren des LGS funktioniert analog zum klassischen Verfahren.
Unterschied: Die Systemmatrix A ist eine schwachbesetzte **Blockmatrix**:

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

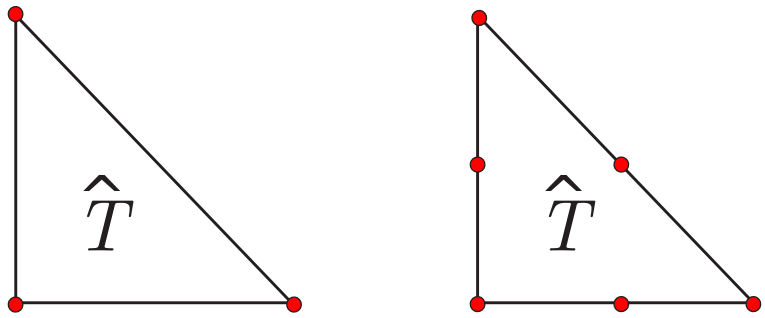


Abbildung 1.9: Stützstellen der Lagrange-Basisfunktionen für $p_{T_i} = 1$ (links) und $p_{T_i} = 2$ (rechts)

mit $A_{ii} \in \mathbb{R}^{d_i \times d_i}, \tilde{A}_{ii} = \left(\int_{T_i} \nabla {\varphi_k}^{T_i} \nabla {\varphi_l}^{T_i}\right)_{1 \le k,l \le d_i}$,
d.h. dies sind die lokalen Steifigkeitsmatrizen $\tilde{A}_{ii} + \overline{A}_{ii}$, etwa:

$$\begin{aligned} \beta h_S^{-1}[u][{\varphi_l}^{T_i}] &= \int_S \beta h_S^{-1} \left(\sum_{k=1}^{d_i} {u_k}^{T_j} {\varphi_k}^{T_j} - \sum_{k=1}^{d_i} {u_k}^{T_i} {\varphi_k}^{T_i} \right) (0 - {\varphi_l}^{T_i}) \\ &= \beta h_S^{-1} \left(\sum_{k=1}^{d_i} {u_k}^{T_j} \underbrace{\int_S {\varphi_k}^{T_j} {\varphi_l}^{T_i}}_{\overline{A}_{ij}} \right) + \sum_{k=1}^{d_i} {u_k}^{T_i} \underbrace{\int_S {\varphi_k}^{T_i} {\varphi_l}^{T_i}}_{\overline{A}_{ii}}, \end{aligned}$$

S: Kante zwischen T_j, T_i. Es ist $A_{ij} \neq 0 \Rightarrow T_j \in \mathcal{I}(T_i)$.

DG vs. CG

Nachteile:

- Viel mehr DOFs als z.B. bei Linearen Ansatzfunktionen: CG : $|\mathcal{E}^2|$, DG : $3|\mathcal{E}^0|$.
- LGS ist i.a. schwerer zu lösen. Etwa für $a \neq -1$ kann das cg-Verfahren nicht eingesetzt werden, d.h. teuerere Verfahren wie bicgstab müssen 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

Inhalt

Literatur

- Arnold, Brezzi, Cockburn, Marini: *Unified analysis of discontinuous Galerikin mehtods for elliptic problems*, [3]
- Bastian et al.: *Towards a unified framework for scientific computing* (.pdf-Version auf der Homepage von Mario Ohlberger), [5]

Lemma 2.1 und Definition (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

$$\begin{aligned} -\nabla \cdot \sigma &= f \ \ in\, \Omega, \\ \sigma &= \nabla u \ \ in\, \Omega, \\ u &= 0 \ \ auf\, \partial\Omega. \end{aligned} \tag{2.1}$$

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, beschränkt und Lipschitzrand, gilt:

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

Dabei ist n_K die äußere Normale an ∂K.

Bemerkung:

1) Man erhält die schwache Formulierung (2.2), indem man die erste Gleichung in (2.1) mit φ und die zweite Gleichung mit ψ 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 nD und LDG-Verfahren

Definition 2.3 (Referenzelement in nD)

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

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

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

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:

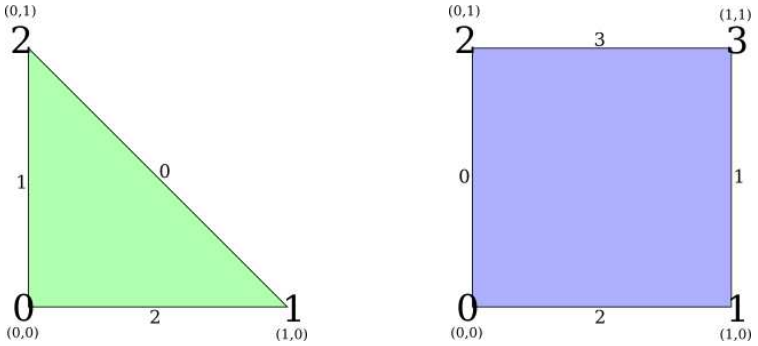


Abbildung 2.1: Dreieck(links), Quadrat (rechts)

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

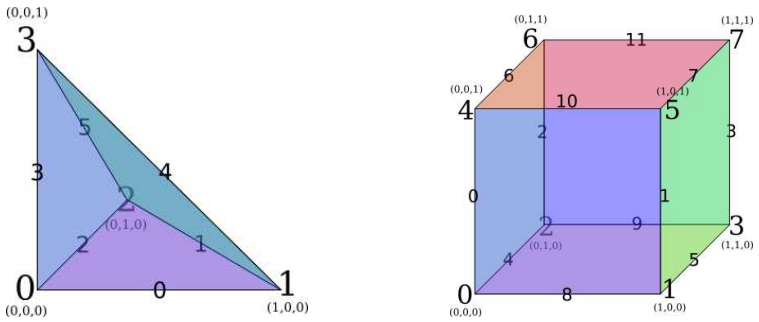


Abbildung 2.2: Tetraeder (links), Hexaeder (rechts)

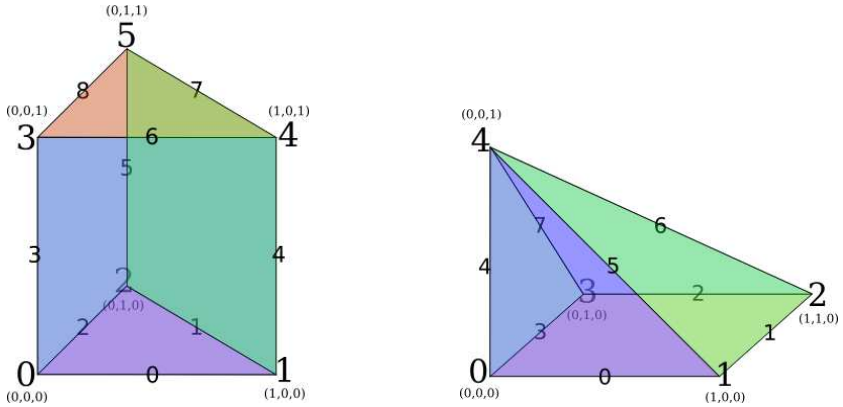


Abbildung 2.3: Prisma (links), Pyramide (rechts)

Definition 2.5 (Hierarchische Gitter in nD)
Sei $\Omega \subset \mathbb{R}^n$ ein Gebiet mit stückweise glattem Rand $\partial\Omega$. Ein ***hierarchisches Gitter*** *$\mathcal{T}_h$ mit $L+1$ 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 $c = 0, \ldots, n$,

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

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 Ω, falls gilt:

1) die Entitäten der Kodimension 0 auf Level 0 definieren eine Partition von Ω:

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

Definition 2.5 (Fortsetzung)

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:

$$F_{l,i} : \hat{e}^0 \to {e_{l,i}}^0 \text{ mit der Eigenschaft}$$
$$F_{l,i}({\hat{e}_j}^c) = e^c_{l,{\mu_{l,i}}^c(j)} \; \forall 0 \leq j \leq k_l, \; 1 \leq c \leq n,$$

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

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

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:

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

Beispiel 2.6

Hier sehen wir die Verfeinerung eines einfachen Makrogitters:

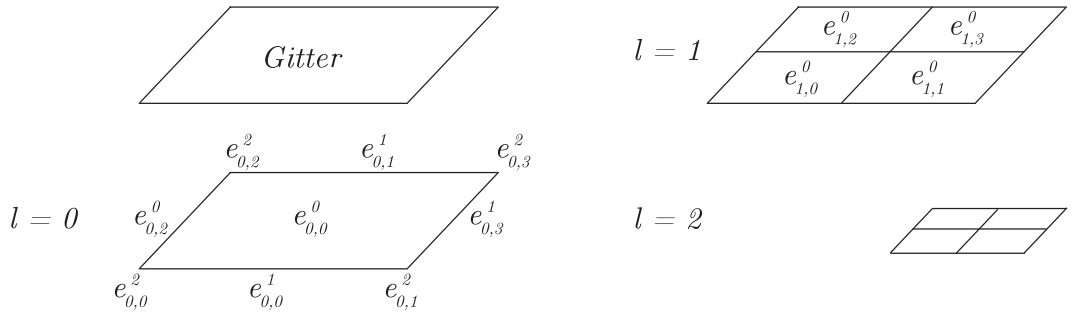


Abbildung 2.4: Ein zweimal verfeinertes Gitter

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

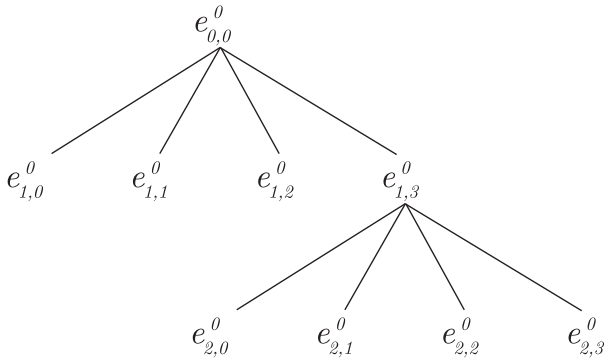


Abbildung 2.5: zugehöriger Entitätenbaum

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:*

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

mit den Eigenschaften:

1) $\mathcal{E}^0_{leaf} = \{e^0_{l,j} \in \mathcal{E}^0_l \mid C(e^0_{l,j}) = \emptyset,\ l = 0, \ldots, L\}$,
2) $\mathcal{E}^c_{leaf} \subset \bigcup_{l=0}^{L} \mathcal{E}^c_l$,
3) $\mathcal{T}_{leaf}$ *ist ein hierarchisches Gitter im Sinne von Definition 2.5 mit $L = 0$. Die Indexmenge zu $\mathcal{E}^c_{leaf}$ bezeichnen wir mit I^c_{leaf} $\forall 0 \leq c \leq n$.*

Beispiel 2.8 (Entitätenbaum aus Bsp. 2.6)

Es gilt: $\mathcal{E}^0_{leaf} = \{e^0_{1,0}, e^0_{1,1}, e^0_{1,2}, e^0_{2,0}, e^0_{2,1}, e^0_{2,2}, e^0_{2,3}\}$. Insbesondere ist $\mathcal{T}_{leaf}$ eine Partition von Ω.

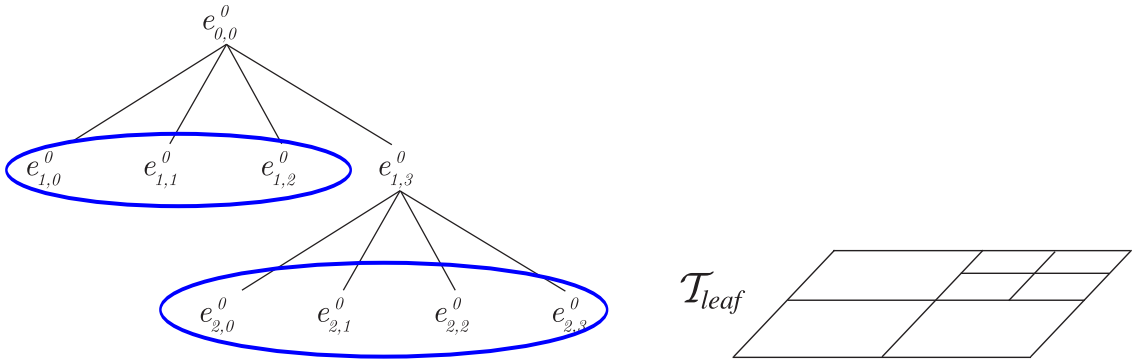


Abbildung 2.6: Entitätenbaum und Darstellung von $\mathcal{T}_{leaf}$ aus Bsp. 2.6

Definition 2.9 (lokale Sobolevräume)

1) Sei $\Omega \in \mathbb{R}^n$ und $\mathcal{T}_h$ ein hierarchisches Gitter auf Ω. Wir definieren die ***lokalen Sobolevräume*** *$H^m(\mathcal{T}_h)$ durch*

$$H^m(\mathcal{T}_h) := \left\{ v \in L^2(\Omega) \mid v|_{e^0} \in H^m(e^0) \forall e^0 \in \mathcal{E}_{leaf} \right\}.$$

Das heißt $H^m(\mathcal{T}_h)$ ist der Raum der stückweise H^m-Funktionen auf dem Blattgitter $\mathcal{T}_{leaf}$.

2) Sei $\Gamma := \bigcup_{e^1 \in \mathcal{E}^1_{leaf}} \overline{e}^1$ die Vereinigung der Kodim 1-Entitäten von $\mathcal{T}_{leaf}$. Wir definieren den Raum der Spuren von Funktionen in $H^1(\mathcal{T}_h)$ als

$$H_S(\Gamma) := \prod_{e^0 \in \mathcal{E}^0_{leaf}} L^2(\partial e^0).$$

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 $x \in \Gamma \backslash \partial\Omega, x \in \overline{e_j}^0, \overline{e_i}^0 \in \mathcal{E}^0_{leaf}, j < i$, so gilt:

$$\begin{aligned} \varphi(x) &= \{\varphi^-(x), \varphi^+(x)\}, \\ \varphi^-(x) &:= \lim_{y \to x, y \in e^0_j} \overline{\varphi}(y), \\ \varphi^+(x) &:= \lim_{y \to x, y \in e^0_i} \overline{\varphi}(y) \qquad (s.\ Skizze\ unten). \end{aligned}$$

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. φ ist einwertig.

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

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

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

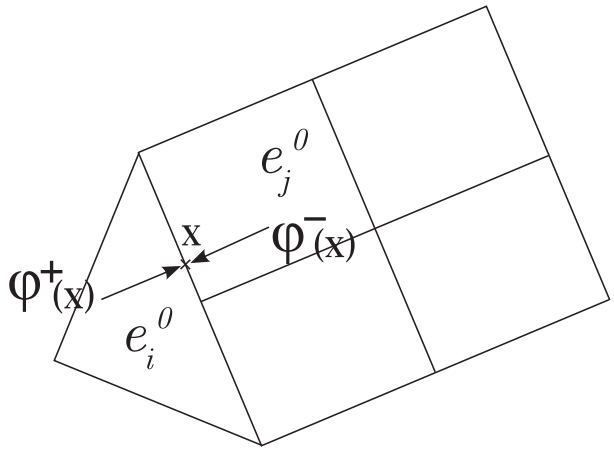


Abbildung 2.7: Skizze zu Definition 2.9: $\varphi^-(x), \varphi^+(x)$

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

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

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

Definition 2.11 (Numerische Flüsse)
Der numerische skalare Fluss

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

und der vektorwertige numerische Fluss

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

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.*

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

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 $e \in \mathcal{E}_{leaf}^0$ gilt:

$$\begin{aligned} \int_e \sigma_h \cdot \nabla \varphi_h &= \int_e f \varphi_h + \int_{\partial e} \hat{\sigma}_e(u_h, \sigma_h) \varphi_h, \ \forall \varphi_h \in V_h^p \\ \int_e \sigma_h \psi_h &= -\int_e u_h \nabla \cdot \psi_h + \int_{\partial e} \hat{u}_e(u_h) n_e \cdot \psi_h, \ \forall \psi_h \in \Sigma_h^q. \end{aligned} \tag{2.3}$$

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

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

durch:

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

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

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

Dann gilt für (u_h, σ_h):

1) u_h löst:

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

2) σ_h ist gegeben durch:

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

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

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

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

$$\int_\Omega \sigma_h \cdot \nabla_h \varphi_h = \int_\Omega f\varphi_h + \int_{\Gamma\setminus\partial\Omega} (\{\hat{\sigma} \cdot n\}[\varphi_h] + [\hat{\sigma} \cdot n]\{\varphi_h\}) + \int_{\partial\Omega} \hat{\sigma} n \varphi_h \tag{2.6}$$

$$\int_\Omega \sigma_h \cdot \psi_h = -\int_\Omega u_h \nabla_h \cdot \psi_h + \int_{\Gamma\setminus\partial\Omega} ([\hat{u}]\{\psi_h \cdot n\} + \{\hat{u}\}[\psi_h \cdot n]) + \int_\Omega \hat{u}\psi_h \cdot n \tag{2.7}$$

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

$$-\int_\Omega \nabla_h \cdot \psi_h \varphi_h = \int_\Omega \psi_h \cdot \nabla_h \varphi_h - \int_{\Gamma\setminus\partial\Omega} (\{\psi_h \cdot n\}[\varphi_h] + [\psi_h \cdot n]\{\varphi_h\}) - \int_{\partial\Omega} \psi_h \cdot n\varphi_h$$

Wähle $\varphi_h = u_h$, dann folgt aus (2.7):

$$\int_\Omega \sigma_h \cdot \psi_h = \int_\Omega \nabla_h u_h \cdot \psi_h + \int_{\Gamma\setminus\partial\Omega} ([\hat{u} - u_h]\{\psi_h \cdot n\} + \{\hat{u} - u_h\}[\psi_h \cdot n]) + \int_{\partial\Omega} (\hat{u} - u_h)\psi_h \cdot n \tag{2.8}$$

Mit der Definition der Liftoperatoren r, l folgt dannL $r_h = s(u_h)$.
Mit (2.8) und (2.6) folgt dann für $\psi_h = \nabla_h \varphi_h$:

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

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:

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

mit u_h 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:

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

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

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

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

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

da $u \in H^2 \cap H_0^1$ Lösung.

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 gelteL

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

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

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

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

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

$$[v] = [\nabla v \cdot n] = 0, \text{ und}$$
$$[v] = \{\nabla v \cdot n\} = \nabla v \cdot n.$$

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

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

Beispiel 2.17 (Wahl der Flüsse und resultierende DG-Verfahren)
Wir definieren Abbildungen

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

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

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

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

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

Skizze:

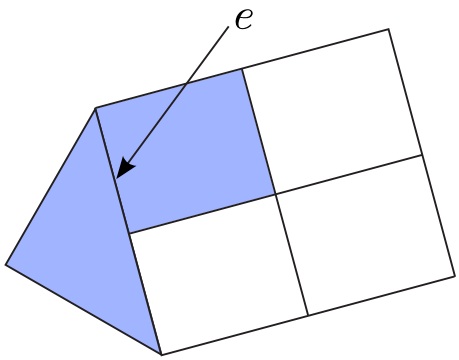


Abbildung 2.8: $supp(r_e(\varphi))$

Wir definieren die Verfahren $LDG_1 - LDG_9$:

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

2)

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

3)

$$\begin{aligned}< u_h,\varphi_h >_\Omega &:= \int_\Omega u_h\cdot\varphi_h, \\ < u_h,\varphi_h >_\Gamma &:= \int_\Gamma u_h\varphi_h.\end{aligned}$$

Weiter setzen wir:

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

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

Nr	$B(\hat{u},\hat{\sigma})$
1	$B_-(w,v) + < r([w]), r([v]) >_\Omega$
2	$B_-(w,v) + < r([w]), r([v]) >_\Omega + \alpha^r(w,v)$
3	$B_+(w,v) + < \beta[w], [\nabla_h v\cdot n] >_{\Gamma\setminus\partial\Omega} + < \beta[v], [\nabla_h w\cdot n] >_{\Gamma\setminus\partial\Omega}$ $+ < r([w]) + l(\beta[w]), r([v]) + l(\beta[v]) >_\Omega + \alpha^j(w,v)$
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	$< \nabla_h w, \nabla_h v >_\Omega + \alpha^j(w,v)$
9	$< \nabla_h w, \nabla_h v >_\Omega + \alpha^r(w,v)$

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:*

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

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

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

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

b) Die Bilinearform erfüllt die Stabilitätsbedingung, falls gilt:

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

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

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

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

zu 1) Es ist

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

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

zu 2) Mit Skalierungsargumenten zeigt man, dass für alle $w \in H^2(e^0)$, $e^0 \in \mathcal{E}_{leaf}^0$, und $e1 \subset \partial e^0, e^1 \in \mathcal{E}_{leaf}^1$, gilt:

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

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

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

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

$$< \{\nabla_h w \cdot n\}, [v] >_{\Gamma \setminus \partial\Omega} + < \nabla_h w \cdot n, v >_{\partial\Omega} \leq C |||w||| \cdot \left(\sum_{e \in \mathcal{E}^1_{leaf}} h_e^{-1} \|[v]\|^2_{0,e} \right)^{\frac{1}{2}}.$$

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:

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

Damit folgt die Beschränktheit von 2).

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

zu 4)

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

mit $C =$ max. Anzahl von Intersections für eine codim 0 Entität.

zu 5)

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

zu 6) analog zu 5) unter Verwendung von (2.9)

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

Details findet man in [Arnold, Brezzi, Cockburn, Marini, 2002]

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 sch 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)

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

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

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

Damit folgt mit der Youngschen Ungleichung

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

Also folgt, dass alle Verfahren stabil sind, falls β_{min} groß genug ist.

Bemerkung 2.22

1) Scharfe Abschätzungen für β_{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 Ω, 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. Seien $(u_h, \sigma_h) \in V_h^p \times \Sigma_h^p$ 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 $h := \max_{e \in \mathcal{E}_{leaf}^0} h_e, h_e := diam(e)$.

Beweis:

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

(ii)

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

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

wenn man eine 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 $u \in H^{p+1}(\Omega)$:

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

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

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

Dann folgt aus der adjungierten Konsistenz von $B_{(\hat{u},\hat{\sigma})}$ nach Lemma 2.15:

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

Sei v_I eine stückweise lineare Interpolierende von v und setze $\varphi_h = u - u_h$ in (2.11), so folgt:

$$\begin{aligned}
\|u - u_h\|_{0,\Omega}^2 &= B_{(\hat{u},\hat{\sigma})}(u - u_h, v) \\
&\overset{Konsistenz}{=} B_{(\hat{u},\hat{\sigma})}(u - u_h, v - v_I) \\
&\overset{Stablitt}{\leq} C_b|||u - u_h|||||v - v_I||| \\
&\leq Ch|v|_{2,\Omega}|||u - u_h|||nonumber \qquad (2.12) \\
&\overset{Theorem2.23}{\leq} Ch|v|_{2,\Omega}h^p|u|_{p+1,\Omega} \\
&\overset{ellipt.Regularitt}{\leq} C\|u - u_h\|_{0,\Omega}h^{p+1}|u|_{p+1,\Omega}.
\end{aligned}$$

Korollar 2.25 (Stabilitäts- und Approximationseigenschaften der LDG-Verfahren aus Bsp. 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.
†: 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:

$$\|u - u_h\|_{0,\Omega} =< f, v - v_h >_\Omega - B_{(\hat{u},\hat{\sigma})}(u_h, v - v_h)$$

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:

$$B_{(\hat{u},\hat{\sigma})}(\varphi_h, v) =< u - u_h, \varphi_h > \ \forall \varphi \in V().$$

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

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

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

$$\|u -_h \|_{0,\Omega} \leq C \cdot \eta,$$

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

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

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

Beweis: Nach Theorem 2.26 gilt:

$$
\begin{aligned}
\|u-u_h\|_{0,\Omega}^2 =& < f, v-v_h > - B_{(\hat{u},\hat{\sigma})}(u_h, v-v_h) \\
=& < f, v-v_h >_\Omega - < \nabla_h u_h, \nabla_h (v-v_h) >_\Omega \\
& - \left(\int_{\Gamma\setminus\partial\Omega} [\hat{u}-u_h]\{\nabla_h(v-v_h)\cdot n\} - \{\hat{\sigma}\cdot n\}\cdot[v-v_h] \right) \\
& - \int_{\Gamma\setminus\partial\Omega} (\{\hat{u}-u_h\}[\nabla_h(v-v_h)\cdot n] - [\hat{\sigma}\cdot n]\{v-v_h\}) \\
& - \int_{\partial\Omega} ((\hat{u}-u_h)\nabla_h(v-v_h)\cdot n - \hat{\sigma}\cdot n(v-v_h)) \\
\overset{part.Integration}{=}& < f+\Delta_h u_h, v-v_h >_\Omega \\
& - \int_{\Gamma\setminus\partial\Omega} ([\hat{u}-u_h]\{\nabla_h\cdot(v-v_h)\cdot n\} - \{(\hat{\sigma}-\nabla_h u_h)\cdot n\}\cdot[v-v_h]) \\
& - \int_{\Gamma\setminus\partial\Omega} (\{\hat{u}-u_h\}[\nabla_h(v-v_h)\cdot n] - [(\hat{\sigma}-\nabla_h u_h)\cdot n]\{v-v_h\}) \\
& - \int_{\partial\Omega} ((\hat{u}-u_h)\nabla_h(v-v_h)\cdot n - (\hat{\sigma}-\nabla_h u_h)\cdot n(v-v_h)) \\
\leq & \sum_{e\in\mathcal{E}^0_{leaf}} \|f+\Delta u_h\|_{0,e}\cdot\|v-v_h\|_{0,e} \\
& + \sum_{\substack{e^1\in\mathcal{E}^1_{leaf},\\ e^1\not\subset\partial\Omega}} \left(\|[\hat{u}-u_h]\|\cdot\|\{(\nabla v_h - v_h)\cdot n)\}\|_{0,e^1} + \|\{(\hat{\sigma}-\nabla_h u_h)\cdot n\}\|_{0,e^1}\|[v-v_h]\|_{0,e^1} \right) \\
& + \sum_{\substack{e^1\in\mathcal{E}^1_{leaf},\\ e^1\not\subset\partial\Omega}} \left(\|\{\hat{u}-u_h\}\|\cdot\|[(\nabla v_h - v_h)\cdot n)]\|_{0,e^1} + \|[(\hat{\sigma}-\nabla_h u_h)\cdot n]\|_{0,e^1}\|\{v-v_h\}\|_{0,e^1} \right) \\
& + \sum_{\substack{e^1\in\mathcal{E}^1_{leaf},\\ e^1\subset\partial\Omega}} \left(\|\hat{u}-u_h\|_{0,e^1}\cdot\|(\nabla v_h - v_h)\cdot n)\|_{0,e^1} + \|(\hat{\sigma}-\nabla_h u_h)\cdot n\|_{0,e^1}\|v-v_h\|_{0,e^1} \right).
\end{aligned}
$$

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

$$
\begin{aligned}
&\|v-I_h v\|_{0,e} \leq C h_e^s \cdot |v|_{s,e},\ \forall e\in\mathcal{E}^0_{leaf}, \\
&\|v-I_h\|_{0,e^1} \leq C h_e^{s-\frac{1}{2}} |v|_{0,e},\ \forall e^1\in\mathcal{E}^1_{leaf}, e_1\subset\partial e.
\end{aligned}
$$

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

$$
\begin{aligned}
\|u-u_h\|_{0,\Omega}^2 \leq C \sum_{e\in\mathcal{E}^0_{leaf}} & \Bigg(h_e^2\|f+\Delta u\|_{0,e}|v|_{2,e} \\
& + \frac{1}{2} \sum_{\substack{e^1\in\mathcal{E}^1_{leaf},\\ e^1\subset\partial e\setminus\partial\Omega}} \left(h_e^{\frac{1}{2}}\|[\hat{u}-u_h]\|_{0,e^1} + h_e\frac{3}{2}\|\{(\hat{\sigma}-\nabla_h u_h)\cdot n\}\|_{0,e^1} \right)\cdot|v|_2,e \\
& + \frac{1}{2} \sum_{\substack{e^1\in\mathcal{E}^1_{leaf},\\ e^1\subset\partial e\setminus\partial\Omega}} \left(h_e^{\frac{1}{2}}\|\{\hat{u}-u_h\}\|_{0,e^1} + h_e\frac{3}{2}\|[(\hat{\sigma}-\nabla_h u_h)\cdot n]\|_{0,e^1} \right)\cdot|v|_2,e \\
& + \frac{1}{2} \sum_{\substack{e^1\in\mathcal{E}^1_{leaf},\\ e^1\subset\partial e\cap\partial\Omega}} \left(h_e^{\frac{1}{2}}\|\hat{u}-u_h\|_{0,e^1} + h_e\frac{3}{2}\|(\hat{\sigma}-\nabla_h u_h)\cdot n\|_{0,e^1} \right)\cdot|v|_2,e \Bigg)
\end{aligned}
$$

Mit Cauchy-Schwarz gilt damit:

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

Kürzen liefert die Behauptung.

Beispiel 2.28 (a-posteriori Fehlerabschätzung für das IP-Verfahren (LDG_4))
Für das LDG_4-Verfahren aus Bsp. 2.17 gilt:

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

mit

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

Dabei hängt die Konstante C vom Polynomgrad p ab.

Beweis: Für LDG_4 ist

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

Also folgt
(i)

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

(ii)

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

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 [11] 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 [2] und [6]*)

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

Definition 2.30 (adaptives LDG-Verfahren)
***Ziel**: Startend mit einem "groben" Gitter $\mathcal{T}_h^0$ sol lokal verfeinertes Gitter $\mathcal{T}_h^j$ konstruiert werden, so dass für eine vorgegebene Toleranz TOL gilt:*
Ist (u_h^j, σ_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 existieren a-posteriori Fehlerabschätzungen der Form

$$\|u - u_h\|_{\#} \leq C_{\#}\eta, \qquad \eta^2 = \sum_{e \in \mathcal{E}_{leaf}^0} \eta_2^2. \tag{2.13}$$

***Notation**:*

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

Mit (2.13) und der Definition von TOL^j folgt: Ist

$$\left(\eta_e^j\right) \leq TOL^j \; \forall e \in \left(\mathcal{E}_{leaf}^0\right)^j \Longrightarrow C_{\#}\eta^j \leq TOL.$$

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

Gegeben: $\mathcal{T}_h^0, \theta, TOL, LDG(\hat{u}, \hat{\sigma})$
Berechne: (u_h^0, σ_h^0) und η^0, η_e^0 auf $\mathcal{T}_h^0$ und setze $j = 0$.
Solange $(C_{\#}\eta^j > TOL)\{$

1) $\forall e \in \left(\mathcal{E}_{leaf}^0\right)^j$: falls

$$\left(\eta_e^j\right)^j > TOL^j,$$

dann markiere e zur Verfeinerung, falls $(\eta_e^j)^j < \theta TOL^j$, dann 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, wo 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 $j = j + 1$.

}

2.4 LDG-Verfahren für nichtlineare Probleme

Definition 2.31 (allgemeines nichtlineares Problem in Divergenzform)
Sei $\Omega \subset \mathbb{R}^n$ 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 $a : \Omega \times \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^n 4$ und $f : \Omega \times \mathbb{R} \to \mathbb{R}$ gegeben. Wir suchen eine Lösung $u \in H^1(\Omega)$ mit

$$\begin{aligned} -\nabla \cdot a(\cdot, u(\cdot), \nabla u(\cdot)) &= f(\cdot, u(\cdot)) \text{ in } \Omega \\ u &= g_D \text{ auf } \Gamma_D \\ a(\cdot, u(\cdot), \nabla u(\cdot)) \cdot u &= g_N \text{ auf } \Gamma_N. \end{aligned} \tag{2.14}$$

Beispiel 2.32
A) Lineare Advektions-Diffusionsgleichung:

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

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

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:

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

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

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

$$\theta = \nabla u, \qquad \sigma = a(\cdot, u, \theta).$$

Dann ist (u, θ, σ) eine Lösung von Problem 2.31, falls gilt:

$$\left.\begin{aligned} \sigma &= a(\cdot, u, \theta) \\ \theta &= \nabla u \\ -\nabla\sigma &= F(\cdot, u) \end{aligned}\right\} \text{ auf } \Omega, \tag{2.15}$$

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

Definition 2.34 (schwache Formulierung von (2.17))
Sei $\mathcal{T}_h$ ein hierarchisches Gitter auf Ω. Dann heißt $(u, \theta, \sigma) \in H^1(\Omega) \times [L^2(\Omega)]^n \times [H^1(\Omega)]^n$ schwache Lösung von (2.17), falls $\forall\, e \in \mathcal{E}^0_{leaf}$ gilt:

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

und

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

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 $(u_h, \theta_h, \sigma_h) V_h^p \times \Sigma_h^q \times \Sigma_h^r$ heißt LDG-Verfahren der Ordnung (p, q, r) zu den Flüssen $\hat{u}, \hat{\sigma}$, falls $\forall\, e \in \mathcal{E}^0_{leaf}$ gilt:

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

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:

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

Kapitel 3

Interface Programming in C++

Inhalt

Schritte des Programmentwurfs

1) "Proof of concecpt": Bsp. Template Klassen für einen festen Datentyp implementieren und testen. Erstmal einfaches Verfahren in 1D testen. Dabei immer auf fertige Bibliotheken zurückgreifen (etwa STL).
2) "Profiling": Aufgspüren von Engpässen im Code. Nicht den Teil optimieren, der nur 1% der Laufzeit ausmacht (gprof,...)
3) Optimierungsoptionen des Compilers studieren. Bsp. loopunrolling, Return-value optimization, inline-Grenzen,...

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.*

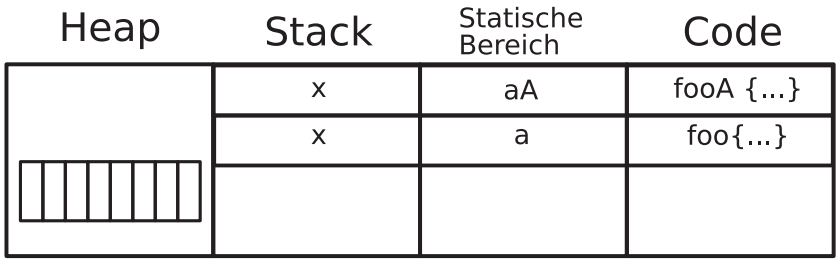
3.1 Funktionsweise des Compilers

3.1.1 Speicheraufbau

A: Aus Programmierersicht

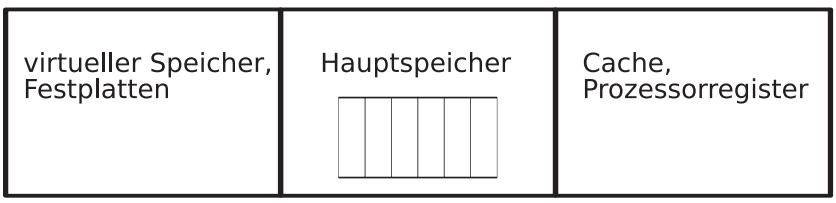
```
class A {
```

```
  static double a;
  double b;
  void foo();
};
void foo() {
  double * heap = new double[100];
  double x,y;
  A a;
}
```



Definition (Stack, Heap)
*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 einmal pro Templateparameter erzeugt, d.h. die Kombination A<int> ist ein völlig anderer Typ als A<double>.*

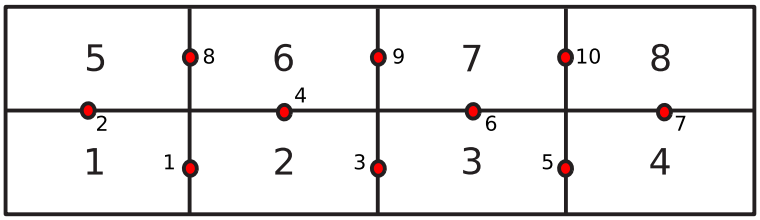
B: Aus Hardwaresicht



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.

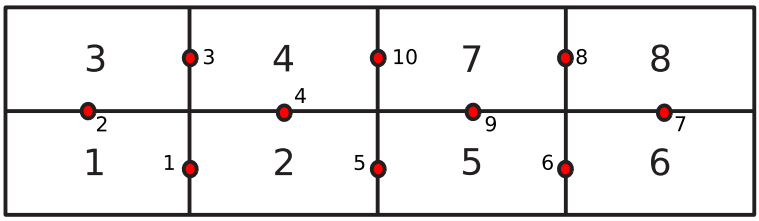
Definition (Cacheeffizeinz)
Eine cache effiziente Implementierung minimiert die Anzahl der cache misses.

Beispiel(Berechnung von Flüssen über Elementkanten):



Annahme: cache hat zwei Seiten und zwei Zeilen pro Seite. Wichtig: Transparenz für Benutzer ⟶ Interface

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

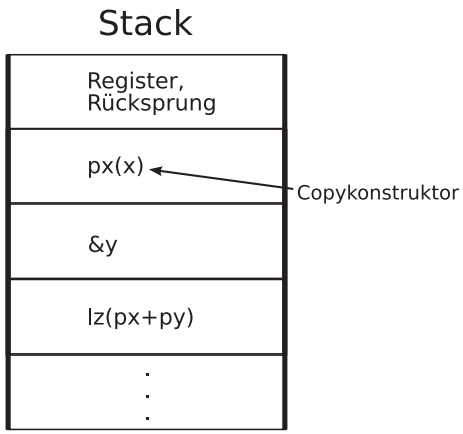


Fluss	Cache		Miss
1,2,3,4	1 2	3 4	2
5	1 2	5 6	1
6,7,8,9	7 8	5 6	1
10	7 8	5 6	1
			5

Bemerkung: Bei unstrukturierten Gittern wird Cacheeffizienz ignoriert. Gruppe *AMR* (adaptive mesh refinement, Blockstrukturiert) versuchen, Cacheeffizienz umzusetzen.

3.1.2 Methodenaufrufe

```
double foo(double px, double &py) {
  double lz = px =py;
}

int main () {
  double x,y,z;
  z = foo(x,y);
}
```



Aufräumen des Stacks, "="-operator auf z aufrufen (bei der Anweisung `double z = foo(x,y);` wird der copy Konstuktor aufgerufen). Bei Klassenmethoden wird aus `A::foo(x,y); fooA(A* , double x, double y);` Return value optimization: Compiler wandelt `double foo(double, & double)` um in `void foo(double, &double, * double)`. Dadurch wird das Anlegen einer lokalen Instanz (hier lz) und der Copykonstruktor 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)

$\longrightarrow$ Inlining
Der Compiler kann (muss aber nicht) den Code von foo an die Stelle des Aufrufs einfügen. Die Benuztzung von inline kann (muss aber nicht) zu schnelleren/größeren Programmen führen.
Bsp: Warum langsamer? Durch große Methoden und große Hacks kann es zu mehr Cache misses kommen.
Warum kleiner? Keinen Code zur Zustandssicherung auf dem Stack.
Was der Compiler inlinet, kann durch Compilerflags beeinflusst werden.
Virtuelle Funktionen: Für jede Klasse mit virtuellen Methoden wird eine **vTable** im statischen Speicherbereicht angelegt und pro Instanz ein Pointer auf die vTables des dynamischen Typs. Die vTable besteht aus Funktionspointern.

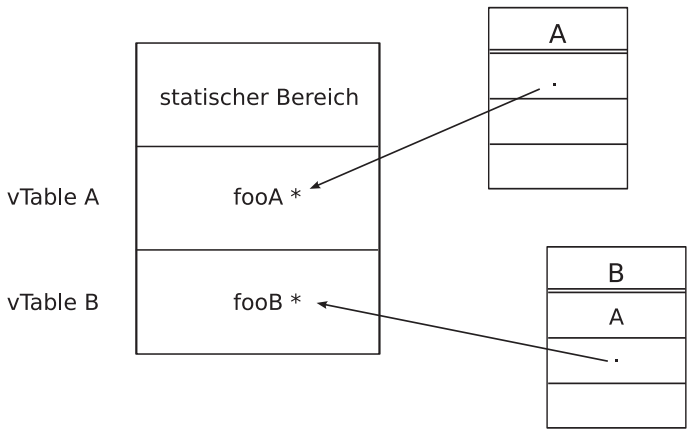
```
class A {
  virtual void foo();
};

class B: public A {
  virtual void foo();
};
```

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

Bemerkung: Virtuelle Funktionen erzeugen kaum zusatzlichen Speicheraufwand.
Methodenaufruf:

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

Effizienzverlust besonders bei kleinen Funktionen liegt an fehlender Möglichkeit zu inlinen.



3.1.3 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/inline Funktionen müssen im Header stehen.)

Desweiteren ernthalten Objectfiles die unaufgelösten Synmbole (Bsp. extern,...).
Linken: verbinden unaufgelöste Symbole mit den exportierten Symbolen. Bibliotheken sind einfach eine Ansammlung von Objectfiles.

3.2 Standardbibliotheken

Im Standard aufgenommen wurde 1997 die STL (Standard Template Library) definiert:

- Container
- Iteratoren
- Algorithmen

1) **Container**

Speicherstrukturen für gleichartige Objekte (werden über Template Argumente festgelegt).

Bei der STL werden keine Datenstrukturen vorgeschrieben, sondern Komplexitäten von elementaren Operatationen und Arten des Zugriffs auf die Elemente.

Beispiel:

`vector<T>`

vorgeschrieben: konsekutiv im Speicher, mit `&(v[0])` = Adresse des ersten Elements, aber `&(v[0])` $\neq$ `&(v.begin())`.

Sequenzen	`vector`	`dequeue`	`list` (einfachverkettet)
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 ("Schnelle Suche, kein wahlfreier Zugriff"):

`set`,	`multiset`,	`map`,	`multimap`	
↑	↑	↑	↑	Elemente eines Containers gehören zu Klassen
key	key	(key, value)	(key, value)	
(eindeutig)		(eindeutig)		

mit

- Default Konstruktor,
- Copy Konstruktor,
- =operator.

Bei assoziativen Containern muss für den Schlüssel <-operator vorhanden sein.
Kein Container mit Referenzen möglich, etwa:`vector<T &>` nicht möglich, `vector<T *>` benutzen.
Container stellen Iteratoren zur Verfügung: `begin()`, `end()`.
Iteratoren in der STL sind verallgemeinerte C-Pointer, d.h. enthalten keine eigenen Methoden, sondern nur Operatoren `++, *, ->, ==,...`

Alternatives Design Pattern:

```
template<Container> Iterator {
  void first();
  void next();
  bool done();
  T& item();
};
```

Spezialisierung von Template Argumenten:

```
template<>
class Iterator<vector> {
...
};
```

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 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 $\mathcal{E}_{leaf}^c$ 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 $\mathcal{E}_l^{codim}$,

   b) `LeafIterator<codim>`: durchläuft die Menge $\mathcal{E}_{leaf}^{codim}$,

   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 Entiät $e \in \mathcal{E}_l^0$,

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

5) Index- und Id-Mengen

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

   b) `LeafIndexSet`: repräsentiert die Indexmenge I_{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 $\mathcal{E}_{level}^{codim}$,

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

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

Index- und IdSets

`levelIdexSet()`
`leafIdexSet()`
`globalIdSet()`
`localIdSet()`
} liefern Objekte der entsprechenden IndexSets bzw. IdSet Klassen.

Adaptivität

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

`mark(refCount,e`: markiert Entität e zur Verfeinerung/Vergröberung. Dabei gilt:

$$refCount = \begin{cases} 1 & \rightarrow \text{ verfeinern} \\ -1 & \rightarrow \text{ vergröbern} \end{cases}$$

`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 collectionCommunication.

2) Verwendung der Iteratoren

Bsp.: Durchlauf der Blattentitäten der Kodimension 0

```
template<GridImp> {
void traverseElements(GridImp & grid)
{
  typedef typename GridImp::template Codim<0> LeafIterator ElementLeafIteratorType;
  ElementLeafIteratorType it = grid.template leafbegin<0>();
  for(it; it!= grid.template leafend<0>(); ++it)
  { ... }
};
```

Eine Anleitung zur Verwendung der Gitterschnittstelle von DUNE ist [4].

3) Kopplung von Daten an Gitterentitäten Die Index- und IdSets liefern bijektive Abbildungen von Entitätenmengen in geordnete Indexmengen, z.B.

$$\begin{aligned} i_l^c : \quad & \mathcal{E}_l^c \longrightarrow I_l^c \qquad \text{"in LevelIndexet"}, \\ i_{leaf}{}^c : & \mathcal{E}_l^c \longrightarrow I_{leaf}^c \qquad \text{"in LeafIndexet"}. \end{aligned}$$

Ist

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

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

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

falls $d : I_{\mathcal{E}'} \rightarrow D(\mathcal{E}')$ 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,

`ibegin()`, `iend()` } 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.

3.3.5 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)`: entspicht F^{-1},

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

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

`volume()`: liefert Volumen der Entität,

`jacobianInverseTransposed(x_local)`: liefert $\nabla F(x\_local)^T$.

3.3.6 Die Iteratorenklassen

1) Methoden von LevelIterator undLeafIterator

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

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

`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

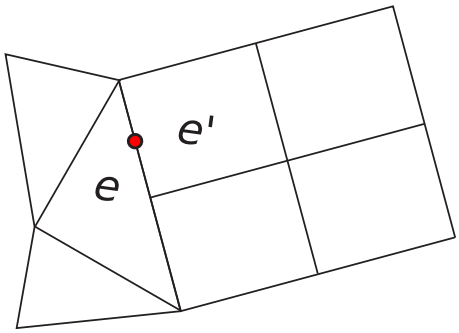


Abbildung 3.1: "inside" Entity e und "outside" Entity e'

`level()`: liefert Level der "inside" Entity,

`operator++()`: Preincrement operator,

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

`leafNeighbor()`: "true", falls Blattnachbar existiert,

`levelNeighbor()`: "true", falls Levelnachbar 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,

`numerInSelf()`: liefert lokale Nummer der codim 1 Subentität von "inside", die die Intersection enthält,

`numerInNeighbor()`: 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.

3.3.7 Die Referenzelement Klassen

1) Methoden von ReferenceElement

`size(codim)`: liefert die Anzahl der Subentiäten der Kodimension "codim" (entspricht k_{codim} in Def. 2.3),

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

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

`position(i,c)`: liefert die Position/Koordinate der Entität $\hat{e}_i^c$,

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

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

`volume()`: liefert Volumen des Referenzelements.

2) Methode der Klasse ReferenceElementContainer

`operator()(geometryType)`: liefert das Referenzelement vom Type "geometryType".

3.3.8 Diskrete Funtionen in DUNE

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 $e \in \mathcal{E}_{leaf}^0$ des Gitters definiert sind.

Ist $\hat{e}$ das Referezelement von $e \in \mathcal{E}^0_{leaf}$ 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 $V_e := span\left\{\varphi \circ F_e^{-1} \mid \varphi \in V_{\hat{e}}\right\}$.
V_h ist dann gegeben durch:

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

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,DomainDim,RangeDim>`: repräsentiert den Funktionenraum V mit $\mathbb{K}_D$ = DomainField, $\mathbb{K}_R$ = RangeField, d = DomainDim, n = RangeDim.
2) `Function<FunctionSpace>`: repräsentiert eine Funktion $u \in V$. Wichtigste Methode ist:
`evaluate(x, ret)`: $ret = u(x)$, für $u \in V$.
3) `DiscreteFunctionSpace<FunctionSpace,Grid,BaseFunktionSet>`: 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 Null Entitäten des Gitters ($\mathcal{E}^0_{leaf}$), sowie folgende Methoden:
   a) `mapToGlobal(e,nLocal)`: entspricht der Abbildung g_e und bildet die lokale DOF Nummer "nLocal" auf die zugehörige globale DOF Nummer ab,
   b) `getBaseFunctionSet(e)`: gibt $V_{\hat{e}}$ zu e zurück.
4) `DiscreteFunction<DiscreteFunctionSpace,LocalFunction>`: 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 $lf = u_e$ zurück.
5) `LocalFunction<DiscreteFunctionSpace>`: repräsentiert die lokale Funktion u_e. Die wichtigsten Methoden sind:
   a) `numberOfDofs()`: gibt $dim(V_e)$ zurück,
   b) `operator[](i)`: gibt den i-ten Freiheitsgrad zurück (entspricht u_{e,φ_i}),
   c) `evaluateLocal(e,x,ret)`: berechnet $ret = u_e(x)$,
   d) `jacobianLocal(e,x,ret)`: berechnet $ret = \nabla u_e(x)$,
   e) `evaluate(e,quad,quadPoint,ret)`: wertet u_e im Quadraturpunkt quad.point(quadPoint, ret) aus,
   f) `jacobian(e,quad,quadPoint,ret)`: wertet ∇u_e in quad.point(quadPoint) aus.

Die Dokumentation zu den in DUNE enthaltenen Klassen findet sich auf der Website des Projekts [1].

3.4 Weiterführende Techniken

3.4.1 Template Meta Programming

Idee: Compiler soll Code generieren. Bei C Präpozessormakros (Nachteil: Keine Typsicherheit, schlecht wartbar).
In C++: Template Meta Programming. 1994 entdeckt: C++ Compiler ist Turing vollständig,
1) kann Berechnungen durchführen,

2) Codegenerierung

Beobachtung in der Numerik ist, dass Vektoren in zwei sehr unterschiedlichen Größen vorkommen:

→ Koordinatenvektoren ($N = 2, 3$),

→ DOF Vektoren ($N = 10^6$).

Bsp.: `Mult<3,Vec>:apply(a,b)`
$\overset{generriert}{\longrightarrow}$ $a[2] * b[2] + a[1] * b[1] + a[0] * b[0]$.

Bsp.: `for(int i= `α`; i<`β`; ++i` ret += a[i]*b[i];

3) Traits: Umwandlung von Typen. Traits sind Klassen, die Typinformationen zur Verfügung stellen.

```
template<Pair>
void foo(Pair & p1, Pair & p2) {
  Mult<Pair::Type1>(p1.first, p2.first);
}
```

3.4.2 Expression Templates

Beispiel:
Berechne

$$x = 0.5 * z + (y * z)y - z', \; x, y, z, y' \in \mathbb{R}^{10^6}.$$

Version 1 :

$$\left.\begin{array}{rcl} t1 & = & 0.5 * z; \\ t_2 & = & (y * z); \\ t_3 & = & t_2 y; \\ t_4 & = & t_3 - z'; \\ x & = & t_4 + t_1; \end{array}\right\} \text{4 temporäre Variablen, 5 Schleifen}$$

Version 2:

```
for(int i=0; i<size; ++i) {
x[i]=0.5*y[i] ... ;
}
```

$\Longrightarrow$ 1 Schleife, keine temporären Variablen.
Nachteil: Lesbarkeit und Verständlichkeit nimmt ab.

Expression Templates erlauben (1) zu schreiben und (2) zu generieren.
→ "Boost Library"

Kapitel 4

Evolutionsgleichungen

Inhalt

Problem A: Löse das Anfangswertproblem

$$\partial_t u(t,x) + L[u(t,\cdot)](x) = 0 \tag{4.1}$$

mit $(t,x) \in (0,T) \times \Omega$, $\Omega \subset \mathbb{R}^n$ offen, beschränkt. L ist ein Differentialoperator (im Ort), etwa

$$L[v](x) = -\nabla \cdot (a(x, v(x), \nabla v(x))) - f(x, v(x)))$$

(vgl. 2.23)
Damit (4.1) wohlgestellt ist, benötigen wir geeignete Randbedingungen. (vlg. 2.23) auf $(0,T) \times \partial\Omega$ und Anfangsbedingungen

$$u(0,x) = u_0(x) \text{ für } x \in \Omega \tag{4.2}$$

Bsp.:

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

Annahme: Wir gehen im weiteren davon aus, dass das Problem A wohlgestellt ist.

4.1 Method of lines

Idee: Bezüglich einer Koordinate (hier die Zeit) haben wir es mit einem gewöhnlich Anfangwertproblem zu tun, für welche sehr gute numerische Methoden und Softwarepakete existieren.
Die Diskretisierung erfolgt daher in zwei Schritten (s. Abb. 4.1)
Für das semidirekte Problem (d.h. diskret im Ort, kontinuierlich in der Zeit) nehmen wir an, dass wir zu L eine Approximation L_h gegeben ist mit $L_h : V_h \to L$
Problem B:

$$\partial_t \overline{u}_h + L_h[\overline{u}_h(t,\cdot)](x) = 0 \tag{4.3}$$

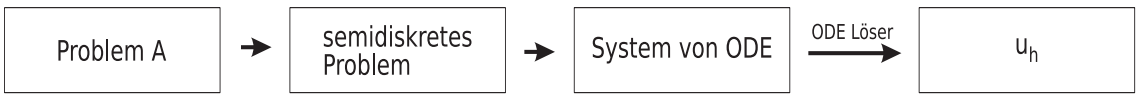


Abbildung 4.1: Schritte der Diskretisierung

Sei $V_h := span\{\varphi_1, \ldots, \varphi_r\}$, $\varphi_i : \Omega \to \mathbb{R}$, dann sei $\overline{u}_h(t,\cdot) \in V_h$, d.h. $\overline{u}_h = \sum_{j=1}^r \alpha_j(t)\varphi_j(x), \alpha_j : [0,T) \to \mathbb{R}$. Die Variationsformulierung von (4.3) ist dann gegeben durch:

$$(\partial_t \overline{u}_h, \varphi_i)_\Omega + (L_h[\overline{u}_h(t,\cdot)](\cdot), \varphi_i)_\Omega = 0 \tag{4.4}$$

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

$$\sum_{j=1}^r \alpha_j'(t) \int_\Omega \varphi_j(x)\varphi_i(x) = -\int_\Omega L_h[\overline{u}_h(t,\cdot)](x)\varphi_i(x),\ i = 1, \ldots, r.$$

Sei $L_h[\overline{u}_h(t,\cdot)](x) = \sum_{j=1}^r w_j(t)\varphi_j(x)$, dann

$$\sum_{j=1}^r \alpha_j'(t) \int_\Omega \varphi_j(x)\varphi_i(x) = \sum_{j=1}^r w_j(t) \int_\Omega \varphi_j(x)\varphi_i(x)$$

Setze

$$\begin{aligned} M &:= \left(\int_\Omega \varphi_j\varphi_i\right)_{1\le i,j\le r} \qquad (Massenmatrix), \\ \overline{U}(t) &:= (\alpha_j(t)), \\ a(t,\alpha_1(t),\ldots,\alpha_r(t)) &= a(t,(\overline{U}(t)) = (w_1(t),\ldots,w_r(t)). \end{aligned}$$

Dann ist (4.4) äquivalent zu

$$\begin{aligned} &MU'(t) = Ma(t,U(t)), \\ &\text{bzw. } U'(t) = a(t,U(t)). \end{aligned} \tag{4.5}$$

Wir erhalten also ein System von gewöhnlichen Differentialgleichungen für $U = (\alpha_1, \ldots, \alpha_r)$.
Zur Definition der Anfangsbedingungen sei π_h ein Projektionsoperator mit Bild in V_h. Sei $\overline{u}_h(0,x) = \pi[u_0](x)$ für $x \in \Omega$ bzw. $(\overline{u}_h(0,\cdot),\varphi_i) = (\pi_h[u_0](\cdot),\varphi_i), 1 \le i \le r$. Dies ist äquivalent zu

$$\sum_{jk=1}^r \alpha_j(-) \int_\Omega \varphi_j\varphi_i = \int_\Omega \pi_h[u_h]\varphi_i$$

und daher den Anfangsbedingungen

$$\begin{aligned} u(0) &= M^{-1}u_0 \text{ mit} \\ u_0 &= (\int_\Omega \pi_h[u_h]\varphi_i)_{1\le i\le r}. \end{aligned}$$

π_h kann Lagrangeinterpolation, L^2-Projektion, ..., sein.

Bemerkung 4.1
In Kapitel 2 haben wir L^{-1} approximiert (also etwa den inversen Laplaceoperator). Hier benötigen eine Approximation von L. In Kapitel 2 haben wir allerdings immer aufbauend auf einer Approximation von L durch Iteration eine Approximation von L^{-1} konstruiert. Daher wissen wir, wie L_h definiert werden kann. ODE können wir auch lösen.

Beispiel 4.2 (Finite Diskretisierung in 1D)

Sei $\Omega = [a,b]$, $L[v] = -\Delta v$ und

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

mit $v_h(x) = \sum_{j=1}^{r} v_j\varphi_j(x)$, φ_j Hütchenfunktion zu einer Zerlegung Δ_h von $[a,b]$.

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

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

Damit erhalten wir das System von ODE's

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

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

Beispiel 4.3 (Finite Elemente Diskretisierung für $\Omega \subset \mathbb{R}^n$)
$L = -\Delta u$. Für gegebenes v_h 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.
$(w_h,\varphi_i) = MW$, d.h. $W = M^{-1}SV$, W Koeffizientenvektor von w_h.

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

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

$$\begin{aligned} U^{n+1} = U^n + \Delta t \cdot M^{-1}SU^{n+1} = (I + \Delta t M^{-1}S)U^n, \\ \Rightarrow (I - \Delta t M^{-1}S)U^{n+1} = U^n, \text{ bzw. } U^{n+1} = (I - \Delta t M^{-1}S)^{-1}U^n. \end{aligned}$$

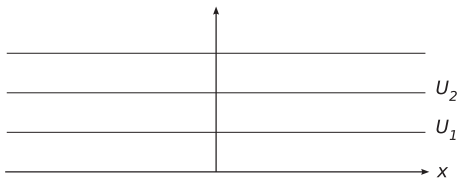


Abbildung 4.2: Diskretisierung nur in der Zeit, kontinuierlich für jeden Zeitpunkt

Beispiel 4.4 (Erhaltungsgleichungen in 1D)
Sei $L[v] = \partial_x f(v)$, $L_h[v_h] = \frac{1}{h}(g(v_{i+1}, v_i) - g(v_i, v_{i-1}))$,
(Bezeichnungen wie in 4.3), mit $g(\cdot,\cdot)$ numerische Flussfunktion.

$$\Rightarrow U'(t) = -\frac{1}{h}(g(\alpha_{j+1}(t) - \alpha_j(t)) - g(\alpha_j(t), \alpha_{j-1}(t))), 1 \leq j \leq r.$$

Mit explizitem Eulerverfahren ($\alpha_j^n = U_j^n$) ergibt sich dann

$$U_j^{n+1} = U_j^n - \frac{\Delta t}{h}(g(U_{j+1}^n, U_j^n) - g(U_j^n, U_{j-1}^n)),$$

(1. Ordnung Finite Volumen Verfahren)

4.2 Semidiskrete LDG-Verfahren

Wir betrachten eine Advektions-Diffusionsgleichung

$$\partial_t u + \nabla \cdot (f(u) - d(u)\nabla\sigma(u)) - S(u) = 0.$$

Es sind mehrere LDG-Ansätze möglich.

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

$$u_k = S_k(u_{k-1}, \ldots, u_0) - \nabla \cdot f(u_{k-1}, \ldots, u_0) - \sum_{l=0}^{k-1} a_{kl}(u_{k-1}, \ldots, u_0)\nabla u_l \qquad \text{(H)}$$

für $k = 1, \ldots K, u_0 \equiv u$. Dazu kommt eine Evolutionsgleichung

$$\partial_t u = F(u_K, \ldots, u_0).$$

Ansatz 1: $(k = 2)$

$$\begin{aligned} f_1(u_0) &= \sigma(u_0), \qquad S_1 = a_0 = 0 \\ f_2(u_1, u_0) &= f(u_0) + d(u_0)u_1, \qquad S_2(u_1, u_0) = S(u_0), a_{20} = a_{21} = 0 \\ F(u_2, u_1, u_0) &= u_2 \end{aligned}$$

Bemerkung 4.5
Es ist noch eine andere Zerlegung sinnvoll,

$$\begin{aligned} v_0 &= u_0, \qquad v_1 = -\nabla \cdot f(u_0) \\ w_0 &= v_0, \qquad w_1 = \nabla\sigma(w_0), w_2 - \nabla(d(w_0)w_1) \\ \partial_t u &= S(u) + v_1 + w_2 \end{aligned}$$

In diesem Fall kann man v_1 in der Zeit explizit diskretisieren, w_2 implizit (führt zu besseren Stabilitätseigenschaften).

Definition 4.6 (LDG Formulierung)
Wir betrachten

$$\partial_t u + L[u] = 0$$

und nehman an, dass L zerlegt werden aknn in der Form:

$$\begin{aligned} L[u_0] &= -F(L_k[L_{k-1}[\ldots L_1[u_0]\ldots]), \\ L_k &: (u_{k-1}, \ldots, u_0) \longrightarrow (u_k, \ldots, u_0), \end{aligned}$$

u ist somit Lösung von

$$\partial_t u = F(u_k, \ldots, u_1, u_0).$$

Die Operatoren L_k seien von einem "elementaren" Typ, etwa vom Typ (H).

Beispiel:
$\partial_t u + \partial_x f(u) = \varepsilon\partial_x^2 u + \eta\partial_x^3 u + \ldots$

$$\begin{aligned} u_0 &= u, \qquad u_1 = \partial_x u_0 \\ u_2 &= \partial_x u_1, \qquad u_3 = \partial_x(f(u_0) + \varepsilon u_1 - \eta u_2) \\ \partial_t u &= u_3 \end{aligned}$$

Die Gleichungen sind jetzt in der Form von Def. 4.6 und alle L_k sind vom Typ (H).

Beispiel:

$$\begin{aligned} \partial_t u + \nabla \cdot f(u) &= \nabla p, \\ \Delta p &= S(u). \end{aligned}$$

$$\begin{aligned} \partial_t u &= L_2[L_1[u]], \\ L_1[u] &= \Delta^{-1} S(u), \\ L_2[u_1, u] &= -\nabla \cdot (f(u) + u_1 I. \end{aligned}$$

Elementaroperatoren vom Type (H) und inverser Laplaceoperator.

Lemma 4.7 (Variationsformulierung für (H))
Sei $\mathcal{T}_h$ ein Gitter auf Ω, φ eine Testfunktion, dann

$$\begin{aligned}\sum T \in \mathcal{T}_h \int_T u_k \varphi &= \sum_T \int_T \left(S_k - \nabla \cdot f_k - \sum_l a_{kl} \nabla u_l \right) \varphi \\ &= \sum_T \left\{ \int_T \left(S_k - \sum_l a_{kl} \nabla u_l \right) \varphi + f_k \nabla \varphi - \int_{\partial T} \left(\tilde{f}_k \cdot n\varphi + \sum_l \widetilde{a_{kl}\varphi} \cdot n[u_l] \right) \right\},\end{aligned}$$

mit $\tilde{f}_k$: numerische Flussfunktion,
$\widetilde{a_{kl}\varphi}$: Mittel,
$[u_l]$: Sprung.

Bemerkung: Der Term $a_{kl}\nabla u_l$ ist nicht in Divergenzform und ist für stückweise stetige Funktionen wohldefiniert, wenn er linear ist unter dem Gradienten.
Die Diskretisierung beinhaltet den Elementanteil und einen "Maßanteil" auf den Elementrändern.
Zur Diskretisierung wählen wir

$$u_{k,h} \in V_{k,h}, \qquad \text{mit } u_{k,h}|_T \in V_{k,h}(T),$$

mit $V_{k,h}(T) = span\left\{ \varphi^T_{k,1}, \ldots, \varphi^T_{k,r^T_k} \right\}$.
Sei $U^T_k = (U^T_{k,j})^{r^T_k}_{j=1}$ der Koeffizientenvektor. Dann betrachten wir die diskrete Variationsformulierung

$$\begin{aligned}\sum_{j=1}^{r^T_k} U^T_{k,j} \int_T \varphi^T_{k,j} \varphi^T_{k,i} = \int_T &\left(S_k - \sum_l a_{k,l} \nabla u_l \right) \varphi^T_{k,i} \\ &+ f_k \cdot \nabla \varphi^T_{k,i} - \int_{\partial T} \tilde{f}_k \cdot n \varphi^T_{k,i} + \sum_l \widetilde{a_{k,l}\varphi^R_{k,i}} n[u_j].\end{aligned}$$

Mit üblicher Vektornotation

$$U^T_k = (U^T)^{-1} b^T_k.$$

Im Gegensatz zum elliptischen Fall braucht man hier keine Matrizen aufzustellen, sondern kann den Koeffizientenvektor direkt bestimmen.
Einschub:

$$\begin{aligned}\partial_t u + \partial_x (\frac{1}{2} u^2) + 0, \\ \partial_t u + u \partial_x u = 0, \\ a(u) \partial_x u\end{aligned}$$

Mit $\varphi = 1, U|_T = u_T$

$$\begin{aligned}\int_T \partial_t u + \int_T u \partial_x u + \_ \partial T \tilde{u} n[u] = 0, \\ (\partial_x u = 0) \int_t \partial_t u + \int_{\partial T} \tilde{u} n[u] = 0, \\ bzw. \int_T \partial_t u + \int_{\partial T} (\frac{1}{2} u^2) \cdot n = 0.\end{aligned}$$

Wählt man $\tilde{u}$ ungünstig, erhält man zentrale Differenzen Diskretisierung (instabil). D.h. in diesem Fall muss $\tilde{a}$ (evtl. auch $[u]$) geschickt gewält werden, um upwinding zu garantieren. Ist $a\nabla u$ im Vergleich zu $\nabla \cdot f(u)$ weniger dominatn, dann reicht häufig $\tilde{a} = \frac{1}{2}(a_l + a_r)$.
In Ansatz 2 tritt dieser Term zur Berechnung der Diffusion auf, s.d. zentrale Diskretisierung kein Problem ist.
Stabilisierung: Wie beim höherer Ordnung Finite Volumen Verfahren brauchen wir beim DG-Verfahren höherer Ordnung eine Limitierung der Gradienten (minmod, ...). Allerdings keinen Reduktionsschritt.

Definition 4.8 (Semi-diskrete LDG-Formulierung mit Stabilisierung)
Sei dazu

$$t^0 = 0 < t^1 < \ldots < t^N = T,$$

eine Zerlegung von $[0,T]$. *Auf* (t^n, t^{n+1}) *seien FE-Räume*

$$V_{h,n} = \{v_h \in L^\infty(\Omega) \mid v_h|_T \in P_p(T)\ \forall T \in \mathcal{T}_h\},$$

wobei $\mathcal{T}_h$ *ein Gitter auf* Ω *ist.*
Seien

$$\Lambda_h^{n,t} : V_{h,n} \longrightarrow V_{h,n},$$

Projektionsoperatoren für $t \in (t^n, t^{n+1})$ *und* $\Lambda n, t^n : V_{h,n-1} \to V_{h,n}$.
Dann definieren wir eine Approximation u_h^n *von* u *auf* $[t^n, t^{n+1})$ *durch*

$$u_h^n(t^n) = \Lambda_h^{n,t^n}[u_h^{n-1}(t^n)],$$
$$\frac{d}{dt}(u_h^n, \varphi) = -\left(L[\Lambda_h^{n,t}(u_h^n(t))], \varphi_h\right),\ \forall \varphi \in V_{h,n}.$$

Bemerkung: Λ_h^{n,t^n} beschreibt die Prolongation 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, etwa in 1D

$$\Lambda_h^{n,t}(u_h(t))|_T = minmod(\nabla u_h, \frac{1}{h}(\overline{u}_h|_{T'} - \overline{u}_h|_T) \cdot n),\ T' \in N(T),$$

für DG Diskretisierung mit $p = 1$.

Bemerkung: Vergleich DG vs. FV-Verfahren für

$$\partial_t u + \nabla \cdot f(u) = 0.$$

1) Leicht formal höhere Ordnung zu definieren.

2) Besonders bei linearen Gleichungen DG wenig numerische Diffusion und ohne Limiter stabil.

3) Zur Stabilisierung müssen die Integrale mit ausreichender Ordnung berechnet werden (Formel von Cockburn-Shu).
Etwa für $p = 1$: int_T mit Quadratur, welche exakt ist für Polynome 1. Grades, $\int_{\partial T}$ muss mit 2. Ordnung approximiert werden (z.B. in 2d durch 2 Punkt Gauss -Formel). D.h. pro Kante zwei numerische Flussauswertungen, bei Finiten Volumen mit Rekonstruktion reicht eine.
$\Rightarrow$ doppelter Aufwand.

4) Auf unstrukturierten Gittern sind noch keine wirklich guten Limiter für DG auf dem Markt (bei nicht glatten, nicht linearen Problemen).

4.3 Zeitdiskretisierung

Die "method of lines" führt auf ein System von gewöhnlichen Differentialgleichungen der Form

Definition 4.9 (System von gewöhnlichen DGLn)
Sei

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

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

$$\frac{d}{dt}U(t) = L(U(t), t) \text{ für } t \in (t^n, t^{n+1}), \tag{4.6}$$

mit der Anfangsbedingung

$$U(t^n) = U^n. \tag{4.7}$$

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

Bemerkung 4.10
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.

4.3.1 Explizite SSP-Runge-Kutta Verfahren

Lit.: C.-W. Shu, *A survey of strong stability preserving high order time discretisations*, [12].

Definition 4.11 (Explizites m-Schritt Runge-Kutta Verfahren)
Ein allgemeines m-Schritt Runge-Kutta Verfahren zur approximativen Lösung von (4.6), (4.7) ist gegeben durch

$$\tilde{U}_l := U_n + \Delta t \sum_{k=1}^{m} a_{lk} L_k,\ l = 1, \dots m,$$
$$L^l := L(\tilde{U}_l, t^n + c_l \Delta t),$$
$$U^{n+1} := U^n + \Delta t \sum_{k=1}^{m} b_k L^k.$$

Das Verfahren heißt konvergent, 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.

c_1	0			0
c_2	a_{21}	$\ddots$		
$\vdots$	$\vdots$	$\ddots$	$\ddots$	
c_m	a_{2m}	$\dots$	$a_{m,m-1}$	0
	b_1	$\dots$	b_{m-1}	b_m

Definition 4.12 (Explizites Eulerverfahren)
Das Euler-Verfahren, definiert durch

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

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

Definition 4.13 (SSP-Verfahren)
Sei $\|\cdot\|$ eine (semi-)Norm auf $\mathbb{R}^N$. Ein numerisches Verfahren zur Approximation von (4.6), (4.7) heißt stabil bzgl. $\|\cdot\|$, falls gilt

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

Ein numerisches Verfahren heißt SSP-Verfahren, falls gilt:

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

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

Definition 4.14 (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:

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

Definition 4.14 (Fortsetzung)

i) 3 Schritt Verfahren der Ordnung 3:

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

ii) 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 4.15: 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

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

für ein $\tilde{p} \leq m$?

Satz 4.16 (Natural Continuous Extension)
Jedes m-Schritt Runge-Kutta Verfahren der Ordnung $\tilde{m}$ besitzt eine "natürlicht stetige Erweiterung" u_h vom Polynomgrad $\tilde{p}$ mit

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

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

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

gibt, so dass gilt

$$U_h(t^n) = U^n,$$
$$U_h(t^{n+1}) = U^{n+1},$$
$$U_h(t^n + s\Delta t) := U^n + \Delta t \sum_{k=1}^{n} b_k(s) c^k,\ 0 \leq s \leq 1. \tag{4.8}$$

Falls die exakte Lösung U von (4.6), (4.7) glatt genug ist, gilt für U_h

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

Beweis: siehe [14].

Definition 4.17 (NCE-SSP-RK Verfahren mit bis 4 Schritten)

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

$$\begin{array}{c|c} 0 & \\ \hline & b_1(s) = s \end{array}$$

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

$$\begin{array}{c|cc} 0 & & \\ 1 & 1 & \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array} \qquad \begin{aligned} b_1(s) &= (b_1 - 1)s^2 + s, \\ b_2(s) &= b_2 s^2. \end{aligned}$$

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

$$\begin{array}{c|ccc} 0 & & & \\ 1 & 1 & & \\ \frac{1}{2} & \frac{1}{4} & \frac{1}{4} & \\ \hline & \frac{1}{6} & \frac{1}{6} & \frac{2}{3} \end{array} \qquad \begin{aligned} b_i(s) &= 3(2c_i - 1)b_i s^2 + 2(2 - 3c_i)b_i s, \\ &\textit{für } i = 1, 2, 3. \end{aligned}$$

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

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

$$\begin{array}{c|cccc} 0 & & & & \\ 1 & 1 & & & \\ \frac{1}{2} & \frac{1}{4} & \frac{1}{4} & & \\ 1 & \frac{1}{6} & \frac{1}{6} & \frac{2}{3} & \\ \hline & \frac{1}{6} & \frac{1}{6} & \frac{2}{3} & 0 \end{array} \qquad \begin{aligned} b_1(s) &= \tfrac{2}{3}s^3 - \tfrac{2}{3}s^2 + s, \\ b_2(s) &= -\tfrac{1}{3}s^3 + \tfrac{1}{2}s^2, \\ b_3(s) &= -\tfrac{4}{3}s^3 + 2s^2, \\ b_4(s) &= s^3 - s^2. \end{aligned}$$

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

$$\begin{array}{c|cccc} 0 & & & & \\ \frac{1}{2} & \frac{1}{2} & & & \\ 1 & \frac{1}{2} & \frac{1}{2} & & \\ \frac{1}{2} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \\ \hline & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{2} \end{array} \qquad \begin{aligned} b_1(s) &= 2(1 - 4b_1)s^3 + 3(b_1 - 1_s^2 + s, \\ b_i(s) &= 4(3c_i - 2)b_i s^3 + 3(3 - 4c_i)b_i s^2, \\ &\textit{für } i = 2, 3, 4. \end{aligned}$$

Bemerkung 4.18 (ODE für NCE-RK Verfahren)

1) NCE-Verfahren 4.5. Ordnung findet man in [Zennaro "86, Owren-Zennaro "95]

2) Sei U_h gegeben als Lösung eines m-Schritt NCE-RK Verfahrens von (4.6), (4.7). 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 (4.8)

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

4.3.2 Implizite Runge-Kutta Verfahren und Kollokationsverfahren

Lit.: Strehnel/Weiner, *Numerik gewöhnlicher DGL*, [13].

Definition 4.19 (Implizites m-Schritt Runge-Kutta Verfahren)
Ein implizites Runge-Kutta Vefahren ist definiert duch ein Buchtertableau

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

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 4.20 (Kollokationsverfahren)
Eine Approximation $U_h \in [\mathbb{P}^m(t^n, t^{n+1})]^N$ heißt Lösung des Kollokationsverfahrens mit den Kollokationspunkten $t^n + c_i\Delta t, i = 1, \ldots, m$ für das Problem (4.6), (4.7), 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 4.21 (Kollokationsverfahren sind stetige RK Verfahren (SIRK))
Ein Kollokationsverfahren mit m Stufen und Kollokationspunkten $c_i, i = 1, \ldots, m$, ist äquivalent zu einem RK-Verfahren definiert durch

$$\begin{aligned} c_i, i = 1, \ldots, m, \\ a_{ij} &= \int)0^{c_i} l_j(\theta)d\theta, \\ b_j &= \int_0^1 l_j(\theta)d\theta, \end{aligned}$$

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

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

Beweis: s. Deuflhard/Bornemann, *Scientific computing with ordinary differential equations*, [7].

Theorem 4.22 (Ordnung von Kollokations RK-Verfahren)
Ein implizites Runge-Kutta Verfahren, das surch ein Kollikationsverfahren gegeben ist, hat bei genügend glattem L die Konsistenzordnung p, gdw. die Quadraturformel mit Punkten $c_i, i = 1, \ldots, m$, und Gewichten $b_i, i = 1, \ldots, m$, die Ordnung p hat.
Beweis: s. [7]

Beispiel 4.23 Gauss Kollokationsverfahren

i) $m = 1, p = 2$

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

ii) $m = 2, p = 4$

$$\begin{array}{c|cc} \frac{1}{2} - \frac{1}{\sqrt{12}} & \frac{1}{4} & \frac{1}{4} - \frac{1}{\sqrt{12}} \\ \frac{1}{2} + \frac{1}{\sqrt{12}} & \frac{1}{4} + \frac{1}{\sqrt{12}} & \frac{1}{4} \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array}$$

4.3.3 IMEX-RK Verfahren

Lit.: Pareschi/Russo, *Implicit-explicit Runge-Kutta schemes and applications to hyperbolic systems with relaxation*, [10].

Definition 4.24 (IMEX-RK Verfahren)
Sei $L = L_e + L_i$, wobei $L : \mathbb{R}^N \times \mathbb{R}^+ \longrightarrow \mathbb{R}^N$ die rechte Seite von (4.6), (4.7) ist (d.h. $\frac{d}{dt}U = L_e(U,t) + L_i(U,t)$). Ein implizites-explizites Runge-Kutta Verfahren (IMEX) ist von der Form

$$\begin{aligned} \tilde{U}^l &:= U^n + \Delta t \left(\sum_{k=1}^{l-1} a^e_{lk} L^k_e + \sum_{k=1}^{m} a^i_{lk} L^k_i \right), \\ L^l_{e/i} &:= L_{e/i}(\tilde{U}^l, t^n + c^{e/i}_l \Delta t), \\ U^{n+1} &:= U^n + \Delta t \sum_{k=1}^{m} \left(b^e_k L^k_e + b^i_k L^k_i \right). \end{aligned}$$

Das Verfahren ist charakterisiert duch die Angabe von 2 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: Eine notwendige Bedingung um zu garantieren, dass L_e nur explizit ausgewertet wird ist

1) $a^e_{lk} = 0, k \geq l$,

2) $a^i_{lk} = 0, k > l$.

Ein Runge-Kutta Verfahren mit der Eingenschaft 1) ist ein explizites Runge-Kutta Verfahren. Ein Runge-Kutta Verfahren mit der Eigenschaft 2) heißt Diagonalimplizites Verfahren (DIRK-Verfahren).
Glit für ein DIRK Verfahren

$$a_{ii} = a_{jj}, \qquad \forall i \neq j,$$

so heißt das Verfahren SDIRK Verfahren (S : simple).

Algorithmus 4.25 (IMEX-DIRK Verfahren)

Für $l = 1, \ldots, m$ {
A) Berechne: 1) falls (l-1 $\geq$ 1):
$L_e(\tilde{U}^{l-1}, t^n + c^e_{l-1}\Delta t)$
2) $\hat{U}^l = U^n + \Delta t \sum_{k=1}^{l-1} a^e_{lk} L^k_e$

B) Berechne: 1) falls (l-1 $\geq$ 1):

$$L_i^{l-1} = L(\tilde{U}^{l-1}, t^n + c_{l-1}^i \Delta t)$$

2) Berechne $\tilde{U}^l$ als Lösung von

(4.9) $$\tilde{U}^l = \hat{U} + \Delta t \sum_{k=1}^{l-1} a_{lk}^i L_i^k + \Delta t a_{ll}^i L_i(\tilde{U}^l, t^n + c_l^i \Delta t).$$

}

Setze $U^{n+1} = U^n + \Delta t \sum_{k=1}^m (b_e^k L_e^k + b_i^k L_i^k)$.

Beispiel 4.26 (SSP-DIRK Verfahren)
Wir geben IMEX Verfahren an, die sich aus SSP und DIRK Verfahren zusammensetzen.
Notation:

SSP-DIRK (m_1, m_2, p):
- m_1 : Stufen von SSP-RK,
- m_2: Stufen von DIRK,
- 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
	$\frac{1}{2}$	$\frac{1}{2}$

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

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
	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{3}$

$\frac{1}{4}$	$\frac{1}{4}$		
$\frac{1}{4}$	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	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{2}{3}$

α	α			
0	$-\alpha$	α		
1	0	$1-\alpha$	α	
$\frac{1}{2}$	β	η	$\frac{1}{2}-\alpha-\beta-\eta$	α
	0	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{2}{3}$

mit $\alpha := 0.24169426078821$,
$\beta := 0.06042356519705$,
$\eta := 0.12915286960590$.

Algorithmus 4.27 Lösung von IMEX Verfahren mit dem Newton Algorithmus
Schreiben wir (4.9) in der Form

$$F((\tilde{U})^l) = 0,$$

so kann das (nichtlineare) Gleichungssystem durch das Newton Verfahren gelöst werden

$$\tilde{U}^{l,0} := U^n.$$

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, das z.B. mit einem iterativen Löser gelöst werden kann.

Problem: Das Verfahren benötigt in jeder Iteration die Jacobimatrix $DF(\tilde{U}^{l,i})$.

Mögliche Vereinfachungen sind

1) Ersetze überall $DF(\tilde{U}^{l,i})$ durch $DF(U^n)$.

2) Approximiere $DF(U^n)$ durch Differenzenquotienten

$$\begin{aligned} DF(U^n)_{ij} &= \frac{\partial F_i(U^n)}{\partial U_i} \\ &\approx \frac{F_i(U^n + \delta e_j) - F_i(U^n)}{\delta}. \end{aligned}$$

3) In iterativen Verfahren für 1) benötigt man nur das Produkt $DF(U^n) \cdot V$. Dies kann approximiert werden durch

$$\frac{F_i(U^n + \delta V) - F_i(U^n)}{\delta}.$$

4.4 Implementierung in DUNE

Ziel: Implementieren von Operator $L_h : V_h \longrightarrow V_h$ mit

$$L_h = L_h^N[L_h^{N-1}[\ldots L_h^1[\ldots]\ldots].$$

L_h^i sind vorgegebene "einfache" Operatoren.

Klassen in Dune zur Beschreibung von Operatoren

- Mapping: $L : V \longrightarrow W$ (Abbildung von VR in VR)

  ```
  typedef DomainType,
          DomainFieldType,
          RangeType,
          RangeFieldType,
  ```

  mit V `DomainType`, W `RangeFieldType`.

- Operationen:

  ```
  operator () (const DomainType & org, RangeType dest);
  ```

  Desweiteren: `operator +, +=, * (RangeFieldType)` . Abgeleitet sind

  - `Function` (mit DomainType ist (DomainFieldType)^n, RangeType ist (RangeFieldType)^m),
  - `operator` (mit DomainType, RangeType sind DiscreteFunctions).

  Ein spezieller Operator ist `DiscreteOperator`, beschreibt Operatoren der Form

  $$L_h = L_{post}\,(L_e|e \in \mathcal{T}_h) \cdot L_{pre},$$

  $(L_e|e \in \mathcal{T}_h)$: Auswirkung des Operators auf den Eintitäten des Gitters (LocalOperator).

  Beispiel (FV Verfahren):

L_e: Berechne den Updatevektor upd,
L_{post}: $Dest - arg + \Delta t * upd$
(Operation auf DOF-Vektor),
L_{pre}: $upd = 0$.

Vorteil: Verknüpfung wie + können auf den lokalen Operatoren durchgeführt werden, falls

$$L^2_{pre} \circ L^1_{post} = Id,$$
$$L^2_h + L^1_h = L^2_{post}(L^1_e + L^2_e) \circ L^1_{pre}.$$

Dadurch werden Gitterdurchläufe reduziert.

Bemerkung:
$L_h = L^2_h + L^1_h$ ist eine Mapping.
$L_h += L^3_h$ operiert auf globaler Ebene und nicht lokal. Anders: $L_h = L^3_h + L^2_h + L^1_h$.

Das Pass Konzept

Realisierung von Verknüpfungen der Form $L^N \circ \dots L^1$.

Ansatz 1:

$$L^{i+1}(U^i, U^0) \longrightarrow (U^{i+1}, U^0),$$

aber dann wäre L^{i+1} kein DUNE-Operator.

Ansatz 2:

$$L^{i+1} : U^0 \longrightarrow U^{i+1},$$

und $(U^i, \dots, U^1)$ sind Parameter,

$$U^{i+1} = L^{i+1}(U^0; U^i, \dots, U^1).$$

Damit ist L^{i+1} ein DUNE-Operator.

Implementiert ist Ansaztz 2 in der Klasse Pass.
Verwaltung wie verkettete Liste, die zur Compilezeit aufgebaut wird. Template Argumente sind

`ProblemType`: Interface zur Problembeschreibung, etwa Quellterm, Diffusionsmatrix, numerischer Fluss,

`PreviousPass`: Typ von L_i, ist von Pass abgeleitete Klasse oder Start Pass. Im Konstruktor übergibt man eine Referenz auf eine Instanz von PreviousPass und von ProblemType.

```
L^{i+1}: operator() (arg, dest) {
           previousPass(arg, dest);
           compute(dest, previousPass.parameters());
         }
```

`parameters()` (= Tupel mit $(U^i, \dots, U^1, U^0)$) gibt `Pair<`U^{i+1}`, previousPass.parameters()>` zurück. StartPass hat leere `compute()` Methode und `parameters()` gibt `Pair<Arg, Nil>` zurück.

`compute` ist eine virtuelle Methode etwa zur Durchführung von

(DG Pass) $V \approx S(U) - \nabla \cdot f(u)$, oder
(Laplace Pass) $V \approx \Delta^{-1} f(U)$.

Parallelisierung

1) *Vektorrechner* (SIMD = single inter multiple data), sehr effizient für Finite Differenzen auf strukturierten Gittern. Ungeeignet für unstrukturierte Gitter.

2) *Shared memory Rechner* (MIMD), mehrere Prozessoren greifen auf gemeinsamen Speicher zu. Parallelisierung von Code ist einfach, etwa statt `for(i = 0; i < N; ++i)` schreibe `for(i = q; i < (q+1)(N/p); ++i)` auf Prozessor q (bei p Prozessoren). **Probleme**:

   a) Gemeinsamer schreibenden Zugriff auf gleiche Speicherstelle.
   Bsp.: Fluss zwischen T_i, T_j, T_i gehört zu Prozessor p_1, T_h zu Prozessor p_2.

   $$(*) \quad \left.\begin{array}{lll} upd_i & += & g_{ij} \\ upd_j & -= & g_{ij} \end{array}\right\} \text{ wird durchgeführt von } p_1.$$

   Parallel berechnet p_2 den Fluss zwischen T_j, T_k aus unf führt

   $$upd_j += g_{jk}$$

   aus.
   Lösung: Entweder berechne g_{ij} sowohl auf p_1 und p_2, dh. führe $(*)$ auf p_1 nicht durch oder arbeite mit *Locks*.

p_1	p_2
hole Lock für upd_j	hole Lock für upd_j
$upd_j += g_{ij}$	warten, bis Lock wieder weg
schreibe upd_j	
Freigabe von upd_j	
	hole Lock
	$upd + j += g_{jk}$
	schreibe upd_j.

   Beide Versionen führen zu parallelem Overhead.

   b) Reentrent, etwa statische Variablen.
   Bsp.: findToken, findNextToken "END"

`char *`			E	N	D				E	N	D		

   Position des zuletzt gefundenen Tokens wird in statischer Variable gespeichert. Verwenden zwei Prozessoren die Methode gleichzeitig, verwenden sie denselben Speicher → nicht *Thread safe*.
   Bsp.: `adapt();`
   muss `adapt` parallelisieren, wird nur von einem Thread aufgerufen. Nachteil: Serieller Programmanteil, Reduktion der Effizienz.

Allgemein:

Sei $T_{ser}(N)$ Laufzeit des seriellen Codes. Es gelte

$$T_{ser}(N) = \underbrace{T^1_{ser}(N)}_{\text{perfekt parallelisieren}} + \underbrace{T^2_{ser}(N).}_{\text{seriell ausgeführt}}$$

Dann gilt

$$T_{par}(N,p) = \frac{T^1_{ser}(N)}{p} + T^2_{ser}(N) + T_{overhead}(N,p).$$

$$\text{Speedup: } S_p(N) = \frac{T_{ser}(N)}{T_{par}(N,p)}, \text{ optimal: } p.$$

$$\text{Effizienz: } E_p(N) = \frac{S_p(N)}{p}, \text{ optimal: } 1.$$

Nachteile *shared memory Rechner*: mit vielen Prozessoren sehr teuer und Bandbreite zum Speicher wir bei großen Rechnungen zum Problem.

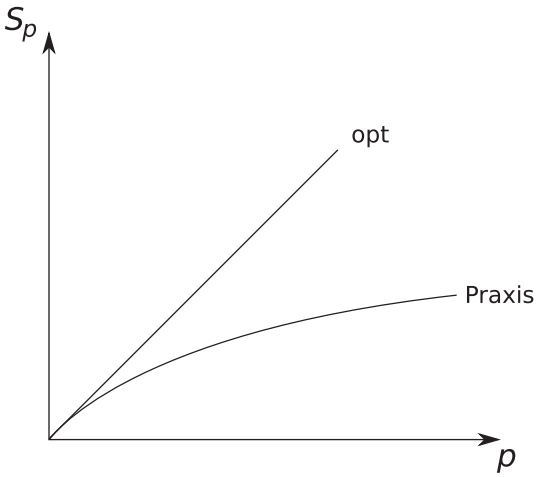


Abbildung 4.3: Speedup bei festem N

3) *Distributed memory*: Keine Probleme mit gemeinsamem Speicherzugriff, *Thread safe*.

Allerdings: werden Daten von anderen Prozessoren benötigt (etwa Nachbarschaften), so müssen diese ausgetauscht werden
$\longrightarrow$ Synchronisation, d.h. warten auf den Langsamsten. Aufwand muss gut verteilt werden, s.d. jeder Prozessor gleich lange braucht, "Lastverteilung".

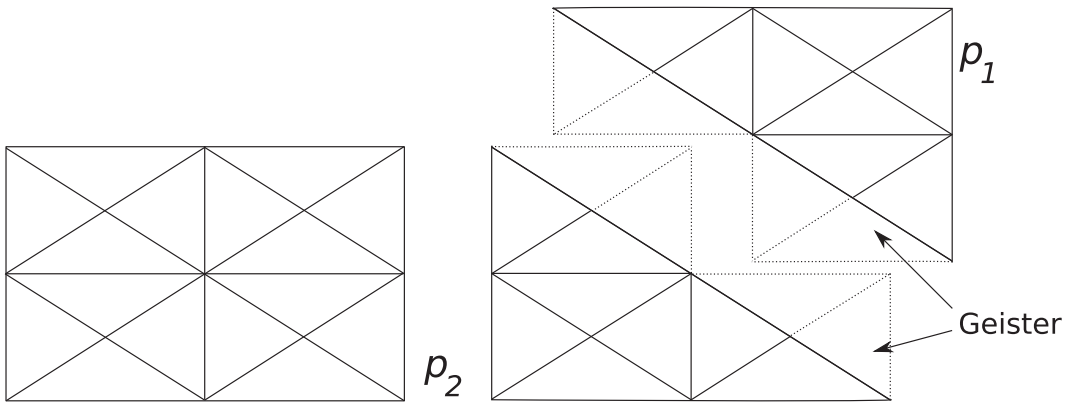


Abbildung 4.4: aufgeteiltes Gebiet mit Geisterzellen

1. Schritt: Datenaustausch *interior* $\rightarrow$ *ghost*,

2. Schritt: Numerik auf interior,

3. Schritt: Synchronisation mit globalen Größen (etwa Δt).

zu 1) Bei Adaption wird Lastgleichgewicht verletzt $\longrightarrow$ Repartitionierung.

Literaturverzeichnis

[1] DUNE – Distributed and Unified Numerics Environment. `http://dune-project.org/`.

[2] Mark Ainsworth. Robust a posteriori error estimation for nonconforming finite element approximation. *SIAM J. Numer. Anal.*, 42(6):2320–2341, 2005.

[3] Douglas N. Arnold, Franco Brezzi, Bernardo Cockburn, and L.Donatella Marini. Unified analysis of discontinuous Galerkin methods for elliptic problems. *SIAM J. Numer. Anal.*, 39(5):1749–1779, 2002.

[4] Peter Bastian, Markus Blatt, Andreas Dednder, Christian Engwer, Robert Klöfkorn, Mario Ohlberger, and Oliver Sander. DUNE grid howto. `http://www.dune-project.org/doc/grid-howto/grid-howto.html`.

[5] Peter Bastian, Mark Droske, Christian Engwer, Robert Klöfkorn, Thimo Neubauer, Mario Ohlberger, and Martin Rumpf. Towards a unified framework for scientific computing. In *Kornhuber, Ralf (ed.) et al., Domain decomposition methods in science and engineering. Selected papers of the 15th international conference on domain decomposition, Berlin, Germany, July 21-25, 2003. Berlin: Springer. Lecture Notes in Computational Science and Engineering 40, 167-174* . 2005.

[6] Rommel Bustinza, Gabriel N. Gatica, and Bernardo Cockburn. An a posteriori error estimate for the local discontinuous Galerkin method applied to linear and nonlinear diffusion problems. 2005.

[7] Peter Deuflhard and Folkmar 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.

[8] Willy Dörfler. Orthogonale Fehler-Methoden. `http://www.mathematik.uni-freiburg.de/IAM/homepages/willy/paper01.html`.

[9] Wolfgang Hackbusch. *Iterative Lösung großer schwachbesetzter Gleichungssysteme. (Iterative solution of large sparse systems of equations).2., überarb. u. erw. Aufl.* Stuttgart: Teubner. 404 S. , 1993.

[10] Lorenzo Pareschi and Giovanni Russo. Implicit-explicit runge-kutta schemes and applications to hyperbolic systems with relaxation. *Journal of Scientific computing*, 25(1):129–155, 2005.

[11] B. Rivière 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.

[12] Chi-Wang Shu. A survey of strong stability preserving high order time discretizations. `citeseer.ist.psu.edu/shu01survey.html`.

[13] Karl Strehmel and Rüdiger Weiner. *Numerik gewöhnlicher Differentialgleichungen. (Numerical methods for ordinary differential equations).* Teubner Studienbücher: Mathematik. Stuttgart: Teubner. 462 p. , 1995.

[14] M. Zennaro. Natural continuous extensions of Runge-Kutta methods. *Math. Comput.*, 46:119–133, 1986.

Index