Institut für Parallele und Verteilte Systeme

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit Nr. 56

Hierarchische Transformationen und der Q-Zyklus

Steffen Hirschmann

Studiengang:	Informatik
Prüfer/in:	Jun.-Prof. Dr. Dirk Pflüger
Betreuer/in:	Dr. Stefan Zimmer
Beginn am:	6. Mai 2013
Beendet am:	5. November 2013
CR-Nummer:	G.1.8

Kurzfassung

In dieser Arbeit wird ein Mehrgitterverfahren für elliptische partielle Differentialgleichungen zweiter Ordnung basierend auf dem Q-Zyklus vorgestellt, das auf dünnen Gittern und deren zugrunde liegenden hierarchischen Basen arbeitet. Hierzu werden effiziente Vergröbern- und Verfeinern-Funktionen vorgestellt, die es erlauben, dünne Gitter anisotropisch zu vergröbern bzw. verfeinern. Diese Funktionen verwenden die dimensionsweise ANOVA-Zerlegung einer Dünngitterfunktion, die ebenfalls vorgestellt wird. An einem Beispielproblem werden Konvergenzraten dieses Mehrgitterverfahrens gezeigt.

Inhaltsverzeichnis

1 Einleitung

Bei der Diskretisierung elliptischer partieller Differentialgleichungen mittels Finite-Elemente-Methode — siehe z.B. Hackbusch [Hac96] — entstehen lineare Gleichungssysteme, die man hocheffizient mit sog. Mehrgitterverfahren lösen kann, siehe Griebel [Gri90].

Die Verwendung von sog. dünnen Gittern, siehe Bungartz und Griebel [BG04], die anhand eines Tensorproduktansatzes aus einer eindimensionalen hierarchischen Basis aufgebaut werden, bringt den Vorteil, dass diese deutlich weniger Freiheitsgrade als volle Gitter besitzen ($\mathcal{O}(N(\log N)^{d-1})$ im Gegensatz zu $\mathcal{O}(N^d)$) bei nur logarithmisch größerem Fehler. Dafür sind dünne Gitter vor allem für Probleme geeignet, die mehr als zwei Dimensionen besitzen.

Das Finite-Elemente-Verfahren mit hierarchischen Ansatzfunktionen ist nicht trivial, da das entstehende lineare Gleichungssystem nicht dünn besetzt ist. Um dieses trotzdem schnell zu lösen, kann z.B. das Verfahren der konjugierten Gradienten (CG) verwendet werden. Dieses benötigt nur die Matrixmultiplikation mit der Steifigkeitsmatrix, welche schnell mittels des sog. Up-Down-Schemas implementiert werden kann.

Wir werden den Ansatz des Up-Down-Schemas verfolgen und es auf mehrere Dimensionen verallgemeinern. Zusätzlich werden wir Vergröbern- und Verfeinern-Funktionen vorstellen, die es erlauben, hierarchische Überschüsse während des Durchlaufs zu ändern und trotzdem alle Residuen konsistent zu halten. Dies wird es erlauben, Mehrgitterzyklen auf hierarchischen Basen durchzuführen. Dies ist insbesondere möglich, da die Vergröbern- und Verfeinern-Funktionen in einzelne Richtungen anisotropisch arbeiten können, wie es bei Peherstorfer und Bungartz als „semi-coarsening“ vorgestellt wurde [PB12, Peh10]. Allerdings benutzten diese keine dünnen Gitter, sodass sie hochdimensionale Probleme aufgrund des „Fluchs der Dimensionen“ nicht berechnen konnten.

Hierzu wird die sog. ANOVA-Zerlegung nach Efron und Stein [ES81] zur dimensionsweisen Zerlegung einer Dünngitterfunktion („dimensionsweise ANOVA-Zerlegung“) nach Griebel [Gri06] verwendet, um das Residuum während des Mehrgitterzyklus effizient zu errechnen. Frühere Ansätze, Mehrgitterverfahren auf hierarchischen Basen durchzuführen, z.B. bei Pflaum [Pfl98] oder Bungartz und Peherstorfer [PB12], berücksichtigen nicht alle ANOVA-Komponenten und können damit entweder nur gewisse Gleichungen lösen oder approximieren die Lösung nur.

Gliederung

Zuerst werden in Kapitel 2 die Grundlagen dieser Arbeit beschrieben. Zum besseren Verständnis des mehrdimensionalen Falls wird zuerst der eindimensionale in Kapitel 3 und 4 eingeführt.

Anschließend folgt die Verallgemeinerung der Finiten-Elemente-Methode in Kapitel 5 und der hierarchischen Transformationen in Kapitel 6. Danach wird die dimensionsweise ANOVA-Zerlegung in Kapitel 7 vorgestellt, mit der eine effiziente Auswertung der Gleichungen möglich wird. Das entstehende mehrdimensionale Up-Down-Schema wird dann in Kapitel 8 vorgestellt und in Kapitel 9 werden weiterführende Erläuterungen gegeben.

In Kapitel 10 folgen Implementierungshinweise und numerische Ergebnisse des Verfahrens. Schließlich wird in Kapitel 11 ein Resümee gezogen.

2 Grundlagen

In diesem Kapitel sollen die nötigen Grundlagen für diese Arbeit präsentiert werden. Dies sind dünne Gitter und Mehrgitterverfahren.

2.1 Dünne Gitter

Um effizient mit hochdimensionalen Problemen umgehen zu können, verwenden wir dünne Gitter. Da hier nur das nötigste vorgestellt wird, sei der Leser für eine ausführliche Beschreibung an [BG04] verwiesen.

Als Grundlage dienen stückweise lineare, eindimensionale Hütchenfunktionen, die als Basisfunktionen benutzt werden:

$$\phi_{l,i}(x) = 1 - \left|2^l x - i\right| \tag{2.1}$$

Hierbei ist l das sog. Level und i der Index der Basisfunktion.

Mittels eines Tensorproduktansatzes werden höherdimensionale Basisfunktionen konstruiert:

$$\phi_{\underline{l},\underline{i}}(x) = \prod_{j=1}^{d} \phi_{l_j,i_j} \tag{2.2}$$

Level $\underline{l}$ und Index $\underline{l}$ sind hierbei Multiindizes. Bildlich dargestellt ist dies in Abbildung 2.1a, in der man anhand eines zweidimensionalen Beispielplots die eindimensionalen Basisfunktionen in x- und y-Richtung erkennt.

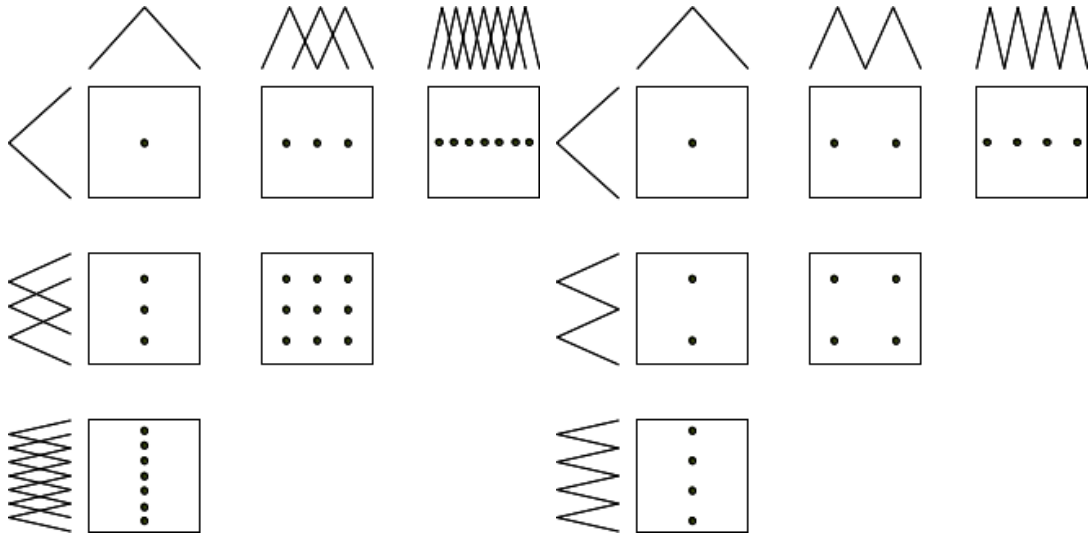
Die sog. Knotenbasis $\mathbf{B}_{\underline{l}}$ eines Levels $\underline{l}$ spannt den Raum $V_{\underline{l}} = \text{span}\{\mathbf{B}_{\underline{l}}\}$ der stückweisen d-linearen Funktionen auf $]0,1[$ auf:

$$\mathbf{B}_{\underline{l}} = \{\phi_{\underline{l},\underline{i}} : i \in I_{\underline{l}}\}$$

Wobei die Indexmenge $I_{\underline{l}}$ eines Levels $\underline{l}$ (also die Menge aller Indizes dieses Levels) wie folgt definiert ist:

$$I_k = \{1, \ldots, 2^k - 1\}$$

$$I_{\underline{k}} = \{(i_1, \ldots, i_n) : i_1 \in I_{k_1} \wedge \cdots \wedge i_n \in I_{k_n}\} = I_{k_1} \times I_{k_2} \times \cdots \times I_{k_n}$$



(a) Hierarchische Unterräume **(b)** Hierarchische Inkremente

Abbildung 2.1: Zweidimensionale Unterraumschemata der Tiefe 3. Links bestehend aus $V_{(1,1)}$, $V_{(1,2)}$, $V_{(1,3)}$, $V_{(2,1)}$, $V_{(3,1)}$, $V_{(2,2)}$. Rechts aus den hierarchischen Inkrementen $W_{(1,1)}$, $W_{(1,2)}$, $W_{(1,3)}$, $W_{(2,1)}$, $W_{(3,1)}$, $W_{(2,2)}$. Eindimensionale Basisfunktionen in x- und y-Richtung jeweils in Zeile bzw. Spalte gezeichnet. Man erkennt, dass die Basisfunktionen der einzelnen $W_{\underline{l}}$ keinen gemeinsamen Träger haben. Basisfunktionen in den Punkten ergeben sich als Produkt der beiden eindimensionalen Basisfunktionen.

Abbildung 2.1a zeigt eben diese Knotenbasis auf verschiedenen Leveln. Die sog. hierarchischen Inkremente $W_{\underline{l}} = \text{span}\{\hat{\mathbf{B}}_{\underline{l}}\}$ (dargestellt in Abbildung 2.1b) werden aufgespannt durch:

$$\hat{\mathbf{B}}_{\underline{l}} = \{\phi_{\underline{l},\underline{i}} : i \in \hat{I}_{\underline{l}}\}$$

Mit:

$$\hat{I}_{\underline{k}} = \{i \in I_{\underline{k}} : \forall j \in \{1, \ldots, d\} : i_j \text{ ungerade}\}$$

Nach [BG04] gilt:

$$V_{\underline{l}} = \bigoplus_{\underline{k} \leq \underline{l}} W_{\underline{k}} \tag{2.3}$$

Wobei das Prädikat $\underline{k} \leq \underline{l}$ wie folgt definiert ist:

$$\underline{l} \geq \underline{k} \Leftrightarrow \forall i \in \{1, \ldots, n\} : l_i \geq k_i$$

Dies definiert die sog. hierarchische Basis des Raumes $V_{\underline{l}}$:

$$\bigcup_{\underline{k} \leq \underline{l}} \hat{\mathbf{B}}_{\underline{k}} = \{\phi_{\underline{k},\underline{i}} : \underline{k} \leq \underline{l} \wedge \underline{i} \in \hat{I}_{\underline{k}}\}$$

Die Koeffizienten in einer Linearkombination bezüglich der hierarchischen Basis nennt man hierarchische Überschüsse.

Wir definieren die sog. Levelmenge eines dünnen Gitters anhand einer maximalen Tiefe $t_{\max}$ nach [BG04]:

$$\mathcal{L} = \{\underline{l} \in \mathbb{N}^d : |\underline{l}|_1 \leq t_{\max} + d - 1\} \tag{2.4}$$

Wobei die 1-Norm eines Levels gegeben ist durch die Summe der Komponenten: $|\underline{l}|_1 = \sum_{i=1}^{d} l_i$.

Mit der Levelmenge lässt sich der eigentlich Raum des dünnen Gitters definieren:

$$V_{t_{\max}}^{(1)} = \bigoplus_{\underline{l} \in \mathcal{L}} W_{\underline{l}}$$

2.1.1 Inhomogene Randwerte

Will man Randwerte ebenfalls interpolieren, so reicht die Basis $\mathbf{B}_{\underline{l}}$ offensichtlich nicht aus, da Funktionen aus ihrem Spann nur Träger auf $]0,1[^d$ haben.

Dies kann relativ einfach gelöst werden, indem man Funktionen hinzunimmt, die die Randwerte interpolieren. Hierzu wird die Indexmenge I_l erweitert: $I_l \cup \{0, 2^d\}$. Außerdem werden Gitter mit ausschließlich Randwerten hinzugenommen. Dies wird erreicht, indem in Gleichung (2.4) als für $\mathcal{L}$ zugrundeliegende Menge $\mathbb{N}_0^d$ genommen wird. Dieses Vorgehen erzeugt auf jedem Gitter $\mathcal{O}(\left|I_{\underline{l}}\right|_1)$ Randwerte, was für manche Anwendungszwecke suboptimal ist. Unter anderem kann man andere Ansatzfunktionen wählen, um das Problem in den Griff zu bekommen. Siehe z.B. [Pfl10, S. 12ff].

In dieser Arbeit wird jedoch davon ausgegangen, dass Randwerte auf jedem Gitter in der eben vorgestellten Form interpoliert werden.

2.2 Mehrgitterverfahren

Zum Lösen von Gleichungssystemen, wie sie bei der Diskretisierung elliptischer partieller Differentialgleichungen entstehen, können Mehrgitterverfahren verwendet werden. Diese sind eine der schnellsten Typen von Lösern für solche linearen Gleichungssysteme. Sie lösen die LGS unabhängig von der Gittergröße bzw. Maschenweite und konvergieren damit in $\mathcal{O}(1)$, sprich: mit konstanter Anzahl an Iterationen.

Die Idee dahinter ist, dass man auf einem Gitter hochfrequente Fehler hinreichend glätten kann und niederfrequente auf gröberen Gittern wiederum glätten kann. Anschließend wird die Feingitterlösung durch die Lösung auf dem groben Gitter korrigiert. Für eine generelle Einführung und weiterführende Erläuterungen siehe [Hac91].

In dieser Arbeit wird ein Mehrgitterverfahren basierend auf dem Q-Zyklus unter Verwendung der hierarchischen Basis vorgestellt.

2.2.1 Q-Zyklus

Der Q-Zyklus ist ein Mehrgitterzyklus für Dünne Gitter bestehend aus V-Zyklen in jede Dimension; siehe [PB12]. Der hier vorgestellte Q-Zyklus ist eine Verallgemeinerung der in [Pfl98] vorgestellten zweidimensionalen Variante. Der Q-Zyklus ist in Algorithmus 2.3 definiert. Die zugrundeliegenden V-Zyklen (Algorithmus 2.2) arbeiten stets nur in eine Richtung. Eine Hilfsfunktion Apply-Codim1-Entity (Algorithmus 2.1) wird eingeführt, um die später vorgestellten Algorithmen Vergröbern und Verfeinern nicht rekursiv in mehreren Dimensionen definieren zu müssen.

Algorithmus 2.1 `Apply-Codim1-Entity(Dimension` $\tilde{d}$`, Operation op)`

1: **for all** $\underline{l} = (1, \ldots, 1, k_{\tilde{d}}, \ldots, k_d)$ **to** $(|I_{k_1}|, \ldots, \left|I_{k_{\tilde{d}-1}}\right|, k_{\tilde{d}}, \ldots, k_d)$ **do**
2: op($\tilde{d}$, $\underline{l}$)
3: **end for**

Hierbei ist `op` ein Transferoperator zwischen zwei Gittern. Dieser berechnet Komponenten der Gitter $\underline{l}$ und $\underline{l} + e_{\hat{d}}$ (welches das in Richtung $\hat{d}$ feinere Gitter ist) anhand der des jeweils anderen. Diese Komponenten sind Größen $r^I_{\underline{k}}$, die später in Kapitel 7 eingeführt werden. Die beiden Operatoren, die wir benutzen werden sind Vergröbern und Verfeinern.

Das Level $(|I_{k_1}|, \ldots, \left|I_{k_{\tilde{d}-1}}\right|, k_{\tilde{d}}, \ldots, k_d)$ ist hierbei definiert als das größte Level, das $k_{\tilde{d}}$ bis k_d als letzte Einträge besitzt:

$$l = (|I_{k_1}|, \ldots, \left|I_{k_{\tilde{d}-1}}\right|, k_{\tilde{d}}, \ldots, k_d) \iff l \in \mathcal{L} \wedge l_{\tilde{d}} = k_{\tilde{d}} \wedge \cdots \wedge l_d = k_d \wedge |l|_1 \text{ maximal} \quad (2.5)$$

Algorithmus 2.2 `V-Zyklus(Dimension` $\tilde{d}$`)`

1: **for all** $\underline{k} = (1, \ldots, 1, 1, k_{\tilde{d}+1}, \ldots, k_d)$ **to** $(1 \ldots, 1, \left|I_{k_{\tilde{d}}}\right| - 1, k_{\tilde{d}+1}, \ldots, k_d)$ **do**
2: Q-Zyklus($\tilde{d} - 1$)
3: Apply-Codim1-Entity($\tilde{d}$, nd-Verfeinern)
4: **end for**
5: Q-Zyklus($\tilde{d} - 1$)
6: **for all** $\underline{k} = (1, \ldots, 1, \left|I_{k_{\tilde{d}}}\right|, k_{\tilde{d}+1}, \ldots, k_d)$ **downto** $(1, \ldots, 1, 2, k_{\tilde{d}+1}, \ldots, k_d)$ **do**
7: Apply-Codim1-Entity($\tilde{d}$, nd-Vergröbern)
8: Q-Zyklus($\tilde{d} - 1$)
9: **end for**

Hierbei ist das Level $(1, \ldots, 1, \left|I_{k_{\tilde{d}}}\right|, k_{\tilde{d}+1}, \ldots, k_d)$ analog zu Gleichung (2.5) definiert, außer, dass die ersten $\tilde{d} - 1$ Einträge 1 sind.

Algorithmus 2.3 `Q-Zyklus(Dimension` $\tilde{d}$ `=` d`)`

1: **if** $\tilde{d} = 0$ **then**
2: Relaxiere Level $\underline{k}$
3: **else**
4: V-Zyklus($\tilde{d}$)
5: **end if**

Initial gilt $\underline{k} = (1, \ldots, 1)$. Als Relaxationsmethode wähle man einen hinreichend guten Glätter, wie z.B. Jacobi- oder Gauss-Seidel-Iteration(en), siehe [Hac91]. Abbildung 2.2 zeigt eine Hälfte der Besuchsreihenfolge der Gitter des Q-Zyklus. Danach werden die Gitter nochmals im umgekehrter Reihenfolge besucht.

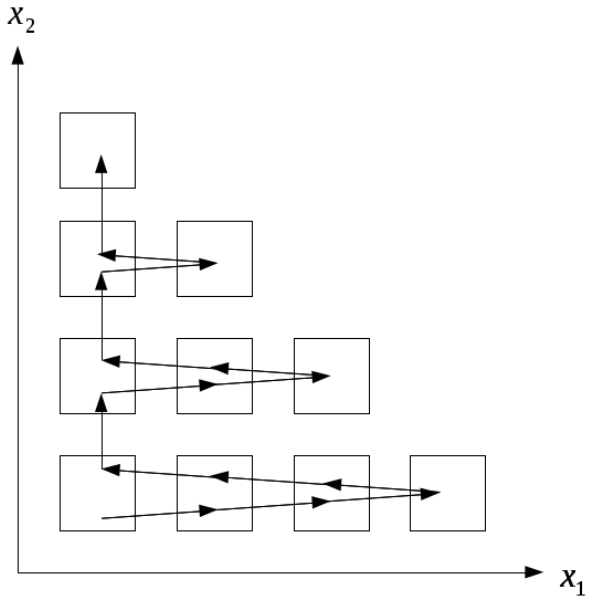


Abbildung 2.2: Besuchsreihenfolge des Q-Zyklus bis zum in x_2-Richtung feinsten Gitter. Anschließend besucht der Q-Zyklus nochmals alle Level in genau der umgekehrten Reihenfolge.

3 Eindimensionales Finite-Elemente-Verfahren

Als ersten wollen wir den eindimensionalen Fall betrachten. In diesem Kapitel wird in Kürze das benötigte Vorwissen für den eindimensionalen Fall des Finite-Elemente-Verfahrens vorgestellt. Als zweites soll dann betrachtet werden, wie man einzelne Unterräume relaxiert.

Wir wählen einen Ritz-Galerkin-Ansatz mit der Knotenbasis $\mathbf{B}_l$ als Test- und Ansatzraum. Folglich gilt es, Gleichung (3.1) zu lösen, vgl. [Hac96, S. 115ff]:

$$\forall \phi_{l,j} \in \mathbf{B}_l : a(u, \phi_{l,j}) = b(\phi_{l,j}) \tag{3.1}$$

Hierbei ist $a(\cdot, \cdot)$ eine Bilinearform; ist also mit Hinsicht auf beide Argumente eine Linearform. $b(\cdot)$ ist die rechte Seite. Diese wird für jede Basisfunktion im Testraum ausgewertet werden und stellt damit keine Herausforderung dar. Das eigentliche Problem ist — wie wir im Nachfolgenden sehen werden — die Auswertung der Bilinearform. Damit setzen wir der Einfachheit halber: $b(\phi_{l,i}) = 0$ für alle l und i.

Die Bilinearform ist abhängig von der Variationsformulierung der elliptischen partiellen Differentialgleichung. Wir werden das L^2-Skalarprodukt benötigen:

$$a(u, v) = \int_{\mathcal{R}} u \cdot v \, \mathrm{d}x \tag{3.2}$$

Sowie für die zweite Ableitung zusätzlich:

$$a(u, v) = \int_{\mathcal{R}} \nabla u \cdot \nabla v \, \mathrm{d}x \tag{3.3}$$

Mit der Darstellung von u im Ansatzraum als Linearkombination $u = \sum_{i \in I_l} u_i \phi_{l,i}$ ergibt sich folgendes lineares Gleichungssystem:

$$\forall \phi_{l,j} \in \mathbf{B}_l : \sum_{i \in I_l} a(\phi_{l,i}, \phi_{l,j}) u_i = b(\phi_{l,j}) = 0$$

Hierbei bezeichnet man die entstehende Matrix $A = \left(a(\phi_{l,i}, \phi_{l,j})\right)_{i,j}$ als „Steifigkeitsmatrix“. Diese wollen wir im Folgenden genauer untersuchen.

3.1 Eindimensionaler Operator A

Sind $v = \sum_{i \in I_l} v_i \phi_{l,i}$ und $w = \sum_{i \in I_l} w_i \phi_{l,i}$ beliebige Linearkombinationen, so gilt für eine Bilinearform:

$$a(v,w) = a\left(\sum_{i \in I_l} v_i \phi_{l,i}\,, \sum_{j \in I_l} w_j \phi_{l,j}\right) = \sum_{i \in I_l} v_i \sum_{j \in I_l} w_j a(\phi_{l,i}, \phi_{l,j}) = \sum_{i \in I_l} v_i \sum_{j \in I_l} w_j A_{ij} = v^T A w$$

Hier bezeichnen wir mit v sowohl die Funktion, als auch den Vektor der Koeffizienten $v = (v_i)_{i \in I_l}$, um keine zusätzliche Notation einführen zu müssen.

Um den Operator A explizit aufzustellen, muss man folglich die Bilinearformen berechnen. Die Basisfunktion $\phi_{l,i}$ hat den Träger $]\frac{i-1}{2^l}, \frac{i+1}{2^l}[$. Folglich gilt: $\forall j \geq 2 : a(\phi_{l,i}, \phi_{l,i\pm j}) = 0$ und die Matrix ist dünn besetzt — sie ist tridiagonal. Die Diagonale und die beiden Nebendiagonalen müssen berechnet werden. Dies soll exemplarisch für das L^2-Skalarprodukt geschehen. Zuerst die Elemente der Hauptdiagonalen:

$$a(\phi_{l,i}, \phi_{l,i}) = \int_{\frac{i-1}{2^l}}^{\frac{i+1}{2^l}} \left(1 - \left|2^l x - i\right|\right)^2 \mathrm{d}x = \frac{2}{3} 2^{-l}$$

Nun die Elemente der Nebendiagonalen:

$$a(\phi_{l,i}, \phi_{l,i+1}) = \int_{\frac{i}{2^l}}^{\frac{i+1}{2^l}} \left(1 - \left|2^l x - i\right|\right) \left(1 - \left|2^l x - (i+1)\right|\right) \mathrm{d}x = \frac{1}{6} 2^{-l}$$

Somit ergibt sich für das L^2-Skalarprodukt aus Gleichung (3.2) der wohlbekannte Operatorstern:

$$\frac{h}{6} [1\ 4\ 1]$$

Für Gleichung (3.3) ergibt sich der Operatorstern:

$$\frac{1}{h} [-1\ 2\ -1]$$

3.2 Hierarchische Basis

Da das Ziel ein Mehrgitterzyklus auf dünnen Gittern ist, müssen wir im eindimensionalen die hierarchische Basis betrachten, das heißt insbesondere alle Level $l \in \mathcal{L}$, die nicht in der Knotenbasis von Level l enthalten sind. Zusätzlich müssen für ein Mehrgitterverfahren einzelne Level relaxiert werden können und nicht alle gemeinsam.

Stellen wir u also nicht mehr in einer Knotenbasis, sondern vielmehr in der hierarchischen Basis dar, so erhalten wir:

$$u = \sum_{l \in \mathcal{L}} \sum_{i \in \hat{I}_l} u_{l,i} \phi_{l,i}$$

Hierbei sind die $u_{l,i}$ die vorher genannten hierarchischen Überschüsse.

Wir teilen also den Raum $V^{(1)}_{t_{\max}}$ bezüglich des zu relaxierenden Levels l auf in V_l — also den Spann der Knotenbasis des Levels l — und $\hat{V}_l$, sodass gilt: $V^{(1)}_{t_{\max}} = V_l \bigoplus \hat{V}_l$.

Dies ergibt eine natürliche Aufteilung der Funktionen $u \in V^{(1)}_{t_{\max}}$ nach [Pfl98, S. 149]:

$$u \in V^{(1)}_{t_{\max}} : u = \tilde{u} + \hat{u} \text{ , sodass } \tilde{u} \in V_l \wedge \hat{u} \in \hat{V}_l \tag{3.4}$$

Wobei für $\hat{u}$ gilt:

$$\hat{u} = \sum_{k>l} \sum_{i \in \hat{I}_k} u_{k,i} \phi_{k,i} \tag{3.5}$$

3.2.1 Entstehendes LGS

Es soll nun u bezüglich Level l relaxiert werden. Das bedeutet, dass als Testraum nun die Knotenbasis $\mathbf{B}_l$ des Levels l genommen wird. Das führt auf folgendes Gleichungssystem:

$$\forall \phi_{l,i} \in \mathbf{B}_l : a(u, \phi_{l,i}) = 0 \tag{3.6}$$

Mit der Zerlegung für u aus Gleichung (3.4) in der Bilinearform: $a(u, \phi_{l,i}) = a(\tilde{u} + \hat{u}, \phi_{l,i}) = a(\tilde{u}, \phi_{l,i}) + a(\hat{u}, \phi_{l,i})$ ergibt sich das LGS:

$$\forall \phi_{l,i} \in \mathbf{B}_l : a(\tilde{u}, \phi_{l,i}) = -a(\hat{u}, \phi_{l,i}) \tag{3.7}$$

Für die Relaxation von Level l gilt es also, die rechte Seite $a(\hat{u}, \phi_{l,i})$ zu berechnen. Diese hängt von allen Level $k > l$ ab. Mit Gleichung (3.5) ergibt sich:

$$\forall \phi_{l,i} \in \mathbf{B}_l : a(\tilde{u}, \phi_{l,i}) = -\sum_{k>l} \sum_{j \in \hat{I}_k} u_{k,j} a(\phi_{k,j}, \phi_{l,i}) \tag{3.8}$$

Das bedeutet, die rechte Seite des LGS zu Level l ändert sich mit jeder Änderung der Funktion u auf einem Level k mit $k > l$. Ein triviales Auswertungsschema müsste die rechte Seite folglich jedes mal neu berechnen, wenn die Funktion auf einem höheren Level geändert wurde. Dies wäre ein zu hoher rechnerischer Aufwand, weshalb im nächsten Kapitel ein effizientes Auswertungssystem vorgestellt werden soll.

4 Eindimensionales Up-Down-Schema

Dieses Kapitel soll das eindimensionale Up-Down-Schema erklären als ersten Schritt zum mehrdimensionalen Up-Down-Schema in Kapitel 8. Dazu werden erst die benötigten Operatoren erklärt und anschließend das essentielle Konzept der hierarchischen Transformationen (Hierarchisierung und Dehierarchisierung) nach [Gri90]. Danach wird das eindimensionale Up-Down-Schema als schnelle Auswertungsalternative der rechten Seite des LGS (3.8) hergeleitet und vorgestellt.

4.1 Operatoren

Seien ein Grobgitter (Level l) und ein feineres Gitter (Level $l+1$) gegeben. Betrachtet man eine Basisfunktion der Knotenbasis des Grobgitters ϕ_l, so lässt sich diese überschussfrei in der Knotenbasis des Feingitters darstellen. Dies wird durch ein dünnes Gitter bzw. die hierarchische Basis gewährleistet. Genauer gesagt, existiert eine Linearkombination der Grobgitterbasisfunktion auf dem feinen Gitter:

$$\phi_l = \sum_i \alpha_i \phi_{l+1,i}$$

Diese Linearkombination ist durch die verwendete Hütchenbasis vorgegeben und nach Definition der Basisfunktionen in Gleichung (2.1):

$$\phi_{l,i} = \frac{1}{2}\phi_{l+1,2i-1} + \phi_{l+1,2i} + \frac{1}{2}\phi_{l+1,2i+1} \tag{4.1}$$

Dieser Zusammenhang wird in Abbildung 4.1 dargestellt.

Anhand dieser Linearkombination definieren wir einen Restriktionsoperator $R : V_F \to V_G$ und einen Prolongationsoperator $P : V_G \to V_F$. Durch die überschussfreie Darstellung einer Grobgitterfunktion auf dem feinen Gitter gilt die Forderung:

$$RP = \mathbb{I} \tag{4.2}$$

Wir wählen R und P nach Zusammensetzung einer Grobgitterbasisfunktion aus Feingitterbasisfunktionen aus Gleichung (4.1):

$$R_{i,j} = \begin{cases} 1, & \text{falls } i = 2j \\ 0, & \text{sonst} \end{cases}$$

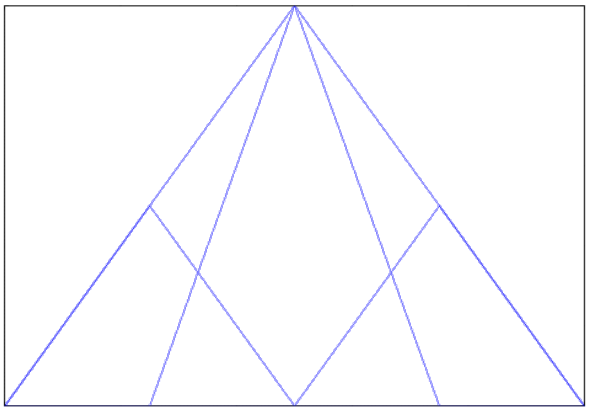


Abbildung 4.1: Zusammensetzung der Grobgitterbasisfunktion aus Feingitterbasisfunktionen.

$$P_{i,j} = \begin{cases} 1, & \text{falls } j = 2i \\ \frac{1}{2}, & \text{falls } j = 2i-1 \vee j = 2i+1 \\ 0, & \text{sonst} \end{cases}$$

Es lässt sich leicht nachprüfen, dass diese Operatoren R und P die Forderung (4.2) erfüllen. Diese Wahl ist nicht verpflichtend. Jede Kombination aus R und P, die Forderung (4.2) erfüllt, ist zulässig. Warum wir R und P so gewählt haben, wird in Kapitel 4.3 deutlich.

Beispielhaft sollen hier der Operator $R_{3\to 2}$ für die Restriktion von Level 3 auf Level 2 und $P_{2\to 3}$ für die Prolongation von Level 2 auf Level 3 vorgestellt werden (die Level werden jeweils ohne Randwerte angenommen):

$$R_{3\to 2} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \qquad P_{2\to 3} = \begin{pmatrix} \frac{1}{2} & 0 & 0 \\ 1 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 \\ 0 & 1 & 0 \\ 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 1 \\ 0 & 0 & \frac{1}{2} \end{pmatrix}$$

4.2 Hierarchische Transformationen

Betrachten wir wiederum eine Feingitterfunktion $u_F \in V_F$ in der Knotenbasis; diese kann dargestellt werden per Grobgitterfunktion $u_G \in V_F \setminus W_F$ (diese ist auf dem Grobgitter V_G darstellbar) und hierarchischen Überschüssen auf dem Feingitter $v_F \in W_F$. Die hierarchischen

Überschüsse sind der Anteil der Funktion u_F, die nicht in u_G enthalten ist und sind somit nicht auf dem Grobgitter darstellbar:

$$u_F = u_G + v_F, \quad Rv_F = 0 \tag{4.3}$$

Die Gitterfunktion u_F kann hierarchisiert werden, indem die Grobgitteranteile angezogen werden. Die Hierarchisierung ist nach [Gri90, S. 17] wie folgt definiert:

$$u_F \mapsto u_F - PRu_F = u_G + v_F - \underbrace{PR}_{=\mathbb{I}} u_G - P \underbrace{Rv_F}_{=0} = v_F$$

Diese Abbildung erfüllt die Forderung (4.3), da:

$$R(u_F - PRu_F) = Ru_F - \underbrace{RP}_{=\mathbb{I}} Ru_F = Ru_F - Ru_F = 0$$

Außerdem darf eine Grobgitterfunktion $u_G = Ru_F$ keine Überschüsse auf dem feinen Gitter haben:

$$u_F = \underbrace{PRu_F}_{u_G} + \underbrace{(u_F - PRu_F)}_{v_F}$$

Hierzu wurde Forderung (4.2) aus vorherigem Kapitel benötigt.

Schließlich kann eine hierarchisierte Darstellung von u_F mittels u_G und v_F einfach wieder in eine nicht hierarchisierte Darstellung überführt werden. Die Operation, die dies bewerkstelligt heißt Dehierarchisierung und wird wie folgt durchgeführt:

$$u_F = u_G + v_F = \underbrace{PRu_F}_{u_G} + \underbrace{(u_F - PRu_F)}_{v_F}$$

4.3 Bilinearformauswertung

Betrachten wir nochmals die Definition der Bilinearform $a(\cdot, \cdot)$ aus Gleichung (3.2):

$$a(u, w) = u^T A w$$

Diese gilt, wenn u, w und A vom selben Level sind. Mit der Wahl der Restriktion und Prolongation anhand der Zusammensetzung der Basisfunktionen, können die Matrizen eines gröberen Levels anhand der des feineren berechnet werden:

$$u, w \in V_G : a(u, w) = (Pu)\, A_F P w = u^T \underbrace{P^T A_F P}_{A_G} w$$

Betrachtet man eine Grobgitterfunktion und eine Feingitterfunktion in der Bilinearform, so erhält man mit Zerlegung der Feingitterfunktion in Grobgitteranteil und hierarchischen Überschüssen:

$$\begin{aligned} a(w_G, u_F) &= a(w_G, u_G) + a(w_G, v_F) \\ &= w_G^T A_G u_G + (P w_G)^T A_F v_F \\ &= w_G^T A_G u_G + w_G^T P^T A_F v_F \\ &= w_G^T (A_G u_G + P^T A_F v_F) \end{aligned} \tag{4.4}$$

Offensichtlich kann man ein auf dem feineren Gitter berechnetes Residuum $A_F u_F$ auf dem groben Gitter weiter verwenden! Wir definieren $r_G = P^T A_F v_F$. Vom höheren Level aus kommend, berechnet man also r_G mithilfe des Residuums $A_F v_F$. Diese Idee funktioniert auch über mehrere Level hinweg, indem Gleichung (4.4) in sich selbst eingesetzt wird. Sei $k > l$:

$$\begin{aligned} a(w_l, u_k) &\overset{\text{Gl. (4.4)}}{=} \underbrace{a(w_l, u_{k-1})}_{\text{weiter zerlegen}} + a(w_l, v_k) \\ &\overset{\text{Gl. (4.4)}}{=} a(w_l, u_l) + \sum_{i=l+1}^{k} a(w_l, v_i) \\ &= w_l (A_l u_l + \sum_{i=l+1}^{k} \left(P^T\right)^{i-l} A_i v_i \\ &= w_l (A_l u_l + P^T (A_{l+1} u_{l+1} + P^T (\cdots + \underbrace{P^T (A_k v_k)}_{r_{k-1}} \ldots))) \\ &\qquad \underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}_{r_{l+1}} \\ &\qquad \underbrace{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad}_{r_l} \end{aligned} \tag{4.5}$$

Die letzte Zeile dieser Gleichung ist eine Auswertung der Bilinearform analog zum Horner-Schema für Polynome mit Zwischenergebnissen r_i.

Alleine von feinen zu groben Leveln zu laufen, reicht jedoch nicht, da die hierarchische Basis verwendet wird. Das bedeutet, grobe Gitter haben einen Einfluss auf die Funktionswerte der feinen Gitter. Damit die Funktion u folglich verändert werden kann, müssen Grobgitteränderungen zuerst auf feinere Gitter gerechnet werden.

4.4 Algorithmen

Das eindimensionale Up-Down-Schema entsteht durch direkte Implementierung der im vorherigen Kapitel hergeleiteten Gleichung (4.5). Funktionswerte werden in der hierarchischen Basis gespeichert. Das Schema ist zweigeteilt: Zuerst kommt der Aufstieg, indem die Funktionswerte dehierarchisiert werden. Anschließend folgt der Abstieg, währenddessen die Residuen berechnet werden und die Funktionswerte wieder dehierarchisiert werden. Die

Funktion für den Aufstieg heißt *Verfeinern* und ist in Algorithmus 4.1 dargestellt; die für den Abstieg *Vergröbern*, siehe Algorithmus 4.2.

Algorithmus 4.1 `1d-Verfeinern(Level k)`

$u_{k+1} \leftarrow Pu_k + v_{k+1}$ // Dehierarchisierung

Algorithmus 4.2 `1d-Vergröbern(Level k)`

$u_k \leftarrow Ru_{k+1}$
$v_{k+1} \leftarrow u_{k+1} - Pu_k$ // Hierarchisierung
$r_k \leftarrow P^T(A_{k+1}v_{k+1} + r_{k+1})$ // Residuum nach Gleichung (4.4)

Man setze hierbei $r_{l_{\max}} = 0$

Das Schema wird häufig verwendet, um eine Matrixmultiplikation mit einer kompletten Dünngitterfunktion schnell zu berechnen; in unserem Fall eine Multiplikation mit der Steifigkeitsmatrix. Hierfür führe man Algorithmus 4.3 aus und berechne anschließend das Residuum auf jedem Level, indem man auf r_k das Residuum $A_k u_k$ von Level k addiert. Vorausgesetzt wird eine korrekte Initialisierung aller hierarchischen Überschüsse.

Algorithmus 4.3 `1d-Up-Down-Schema`

Voraussetzung: Initialisierung
Ergebnis: r_k für $k \in \mathcal{L}$
for $k = 1, \ldots, l_{\max} - 1$ **do**
    Verfeinern(k)
end for
for $k = l_{\max} - 1, \ldots, 1$ **do**
    Vergröbern(k)
end for

Jedoch kann man mit diesem Schema auch levelweise relaxieren. Dieser Ansatz ist eine eindimensionale Version der „Hierarchischen-Transformations-Mehrgitter-Methode“ nach [Gri90]. Der eindimensionale Mehrgitteransatz wird in Algorithmus 4.4 gezeigt. Er ist dem normalen Up-Down-Schema sehr ähnlich mit der Ausnahme, dass wir an dem Ergebnis der kompletten Matrixmultiplikation nur bedingt interessiert sind und eben levelweise relaxieren. Man verwende für die Funktion Relaxieren ein Iterationsverfahren, wie in Kapitel 2 beim Q-Zyklus angesprochen.

Die Laufzeiten der Algorithmen Verfeinern und Vergröbern werden nach Vorstellung der multidimensionalen Äquivalente in Kapitel 8 diskutiert. Diese gelten selbstverständlich auch für den eindimensionalen Fall.

Algorithmus 4.4 `1d-Mehrgitter-Up-Down-Schema`

Voraussetzung: Initialisierung
  for $k = 1, \ldots, l_{\max} - 2$ **do**
    Relaxieren(k)
    Verfeinern(k)
  end for
  Relaxieren($l_{\max} - 1$)
  for $k = l_{\max} - 1, \ldots, 2$ **do**
    Vergröbern(k)
    Relaxieren(k)
  end for

5 Mehrdimensionales Finite-Elemente-Verfahren

Da nun das eindimensionale Konzept bekannt ist, folgt die Verallgemeinerung — das mehrdimensionale Up-Down-Schema. Das eindimensionale Konzept kann hier teilweise übertragen werden.

In diesem Kapitel wird der Grundstein für das mehrdimensionale Up-Down-Schema gelegt, indem die zu Grunde liegenden Gleichungen vorgestellt sowie deren Probleme aufgezeigt werden.

Als Basis für $[0,1]^n$ wird der Tensorproduktansatz basierend auf der Hütchenbasis aus Gleichung 2.2 nach [BG04] gewählt. Im Mehrdimensionalen entstehen neue Verhältnisse der Level zueinander. Für diese führen wir das Prädikat $\succ$ ein, wobei $\underline{l} \succ \underline{k}$ die Negation von $\underline{l} \leq \underline{k}$ ist:

$$\underline{l} \geq \underline{k} \Leftrightarrow \forall i \in \{1, \ldots, n\} : l_i \geq k_i$$
$$\underline{l} \succ \underline{k} \Leftrightarrow \exists i \in \{1, \ldots, n\} : l_i > k_i$$

Das zu lösende LGS (3.6) überträgt sich aus dem eindimensionalen, jedoch sind Test- und Ansatzraum nun die Tensorproduktknotenbasis des jeweiligen Levels:

$$\forall \phi_{\underline{l},\underline{i}} \in \mathbf{B}_{\underline{l}} : a(u, \phi_{\underline{l},\underline{i}}) = 0 \tag{5.1}$$

Die Zerlegung aus Gleichung (3.4): $u = \tilde{u} + \hat{u}$ bezüglich eines Levels $\underline{l}$ gilt weiterhin. Im Mehrdimensionalen bestehen die einzelnen Komponenten jedoch aus deutlich mehr Summanden:

$$\tilde{u} = \sum_{\underline{k} \leq \underline{l}} \sum_{\underline{j} \in \hat{I}_{\underline{k}}} u_{\underline{l},\underline{j}} \phi_{\underline{k},\underline{j}} \in V_{\underline{l}} \quad \hat{u} = \sum_{\underline{k} \succ \underline{l}} \sum_{\underline{j} \in \hat{I}_{\underline{k}}} u_{\underline{l},\underline{j}} \phi_{\underline{k},\underline{j}} \in \hat{V}_{\underline{l}}$$

Abbildung 5.1 zeigt anhand eines zweidimensionalen Beispiels, welche Bereiche des Unterraumschemas in welche der beiden Funktionen fallen.

Man beachte, dass die Darstellung von $\tilde{u}$ nach Gleichung (2.3) immer noch in der Knotenbasis $\mathbf{B}_{\underline{l}}$ erfolgen kann. Das entstehende LGS ist:

$$\forall \phi_{\underline{l},\underline{i}} \in \mathbf{B}_{\underline{l}} : a(\tilde{u}, \phi_{\underline{l},\underline{i}}) = -\sum_{\underline{k} \succ \underline{l}} \sum_{\underline{j} \in I_{\underline{k}}} u_{\underline{l},\underline{j}} a(\phi_{\underline{k},\underline{j}}, \phi_{\underline{l},\underline{i}}) \tag{5.2}$$

Um für die rechte Seite dieses LGS ein effizientes Auswertungsschema angeben zu können, benötigen wir — analog zum eindimensionalen Fall — zuerst grundlegende Operatoren wie

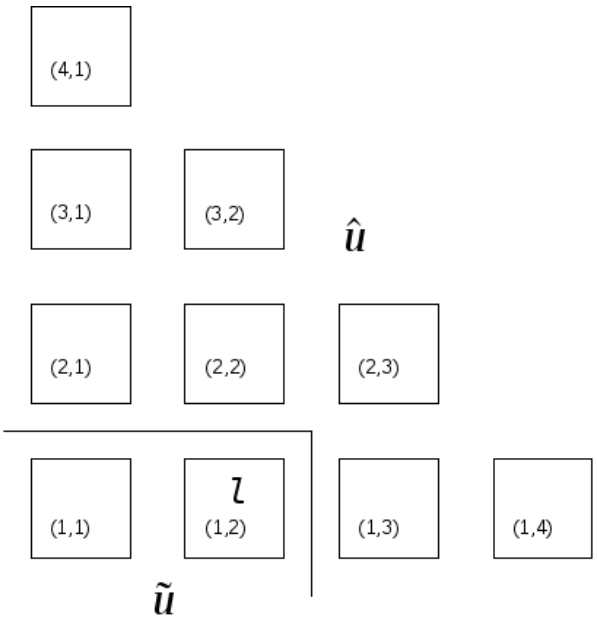


Abbildung 5.1: Zerlegung des Unterraumschemas in V_l und $\hat{V}_l$ anhand von Level $(1,2)$. Es gilt: $\tilde{u} \in V_{\underline{l}}$ und $\hat{u} \in \hat{V}_{\underline{l}}$. $\tilde{u}$ enthält Beiträge der Level $(1,1)$, $(1,2)$ und $\hat{u}$ alle restlichen.

multidimensionale Restriktion und Prolongation, sowie hierarchische Transformationen im Mehrdimensionalen.

6 Mehrdimensionale Hierarchische Transformationen

Zuerst sollen die eindimensionalen Operationen aus Kapitel 4.1 für den mehrdimensionalen Fall erweitert werden. Anschließend wird das Konzept der hierarchischen Transformationen in einer Richtung nach [PB12] ins Mehrdimensionale übertragen.

Analog zu den eindimensionalen Operatoren R, P, P^T und A definieren wir multidimensionale Äquivalente $R^{(i)}$, $P^{(i)}$, ${P^T}^{(i)}$ $A^{(i)}$. Diese arbeiten anisotropisch, also nur in Richtung i. In [PB12] wurde dieses Konzept als „semi-coarsenenig“ vorgestellt. Bildlich dargestellt, arbeiten diese multidimensionalen Operatoren zeilenweise auf Zeilen in Dimension i und wenden auf diesen den jeweiligen eindimensionalen Operator an. Zusätzlich können sie einen Index tragen, um das eindimensionalen Level in Wirkungsrichtung zu spezifizieren: $R^{(i)}_{k+1,k}$ restringiert in Dimension i das eindimensionale Level $k+1$ auf Level k. Eine formale Definition sieht wie folgt aus:

$$R^{(i)}_{k_i,k_i-1} v = \left(R \begin{pmatrix} v_{j_1,\dots,j_{i-1},0,j_{i+1},\dots,j_d} \\ \vdots \\ v_{j_1,\dots,j_{i-1},|I_{k_i}|-1,j_{i+1},\dots,j_d} \end{pmatrix} \right)_{j_1,\dots,j_{i-1},j_{i+1},\dots,j_d \in I_{k_1} \times \cdots \times I_{k_{i-1}} \times I_{k_{i+1}} \times \cdots \times I_{k_d}} \tag{6.1}$$

Im Falle eines eindimensionalen Raums gilt nach Definition offensichtlich $R^{(1)}_{k,k-1} = R_{k \to k-1}$, wobei letzteres der altbekannte eindimensionale Restriktionsoperator für Level k ist.

$P^{(i)}_{k_i,k_i+1}$, ${P^T_{k_i+1,k_i}}^{(i)}$, $A^{(i)}_k$ werden analog zu Gleichung (6.1) definiert, wenden jedoch die eindimensionalen Operatoren P bzw. P^T bzw. A an. Außerdem gilt:

$$\left(P^{(i)}_{k_i,k_i+1}\right)^T = {P^T_{k_i+1,k_i}}^{(i)}$$

Beispielhaft wird die multidimensionale Prolongation eines Feldes berechnet:

$$P^{(2)} \begin{pmatrix} a & b & c \\ d & e & f \\ f & h & i \end{pmatrix} = \begin{pmatrix} \frac{1}{2}a & a & \frac{1}{2}(a+b) & b & \frac{1}{2}(b+c) & c & \frac{1}{2}c \\ \frac{1}{2}d & d & \frac{1}{2}(d+e) & e & \frac{1}{2}(e+f) & f & \frac{1}{2}f \\ \frac{1}{2}g & g & \frac{1}{2}(g+h) & h & \frac{1}{2}(h+i) & i & \frac{1}{2}i \end{pmatrix}$$

Der Einfachheit definieren wir n-fache Operatoren. Hier beispielhaft anhand der Restriktion. N-fache Prolongation $P^{(i)}_{k,l}$, sowie transponierte Prolongation $P^{T\,(i)}_{l,k}$ sind analog definiert.:

$$R^{(i)}_{l,k} = R^{(i)}_{l,l-1} R^{(i)}_{l-1,l-2} \cdots R^{(i)}_{k+1,k} = \prod_{j=l}^{k+1} R^{(i)}_{j,j-1}$$

Man beachte, es übertragen sich anhand der Definition in Gleichung (6.1) die Eigenschaften der eindimensionalen Operatoren:

$$R^{(i)}_{k_i+1,k_i} P^{(i)}_{k_i,k_i+1} = \mathbb{I}$$

Wir wollen nun folgenden Satz zeigen:

Satz 1. *Beliebige multidimensionale Operatoren in unterschiedlicher Richtung sind vertauschbar.*

Beweis. Man betrachte o.B.d.A. zwei Dimensionen. In höherdimensionalen Räumen betrachte man einfach alle zweidimensionalen Ebenen, in der die beiden Operatoren zusammen wirken, separat.

Gegeben sei das zweidimensionale Feld $v = (v_{k,l})_{k=0,\ldots,N;l=0,\ldots,M}$, sowie die Operatoren $X^{(1)}$ und $Y^{(2)}$. Es gilt:

$$X^{(1)}Y^{(2)}v = X^{(1)} \left(\sum_{i=1}^{M} y_{k,i} v_{i,l} \right)_{k,l} = \left(\sum_{j=1}^{N} x_{l,j} \sum_{i=1}^{M} y_{k,i} v_{i,j} \right)_{k,l}$$
$$= \left(\sum_{i=1}^{M} y_{k,i} \sum_{j=1}^{N} x_{l,j} v_{i,j} \right)_{k,l} = Y^{(2)} \left(\sum_{j=1}^{N} x_{l,j} v_{k,j} \right)_{k,l} = Y^{(2)} X^{(1)} v$$

□

Anmerkung: Für beliebige Operatoren in *derselben* Richtung gilt dies selbstverständlich nicht.

Die hierarchischen Transformationen Hierarchisierung und Dehierarchisierung werden wir — wie im eindimensionalen Fall ebenfalls — als Grundlage für eine effiziente Auswertung der Gleichungen benötigen, weshalb sie hier definiert werden. Analog zu den multidimensionalen Operatoren Restriktion und Prolongation arbeiten auch diese in einer Richtung und sind wie die eindimensionalen Äquivalente in Kapitel 4.2 definiert.

Sei $u_{\underline{k}}$ Gitterfunktion zur Knotenbasis des Levels $\underline{k}$. Eine Hierarchisierung nach [Gri90, PB12] in Richtung i kann wiederum erreicht werden durch die Subtraktion der Grobgitteranteile ebendiese Richtung:

$$u_{\underline{k}} \mapsto u_{\underline{k}} - P^{(i)}_{k_i-1,k_i} R^{(i)}_{k_i,k_i-1} u_{\underline{k}}$$

Genau diese Abzüge müssen für eine Dehierarchisierung wieder hinzuaddiert werden:

$$u_{\underline{k}} = P^{(i)}_{k_i-1,k_i} R^{(i)}_{k_i,k_i-1} u_{\underline{k}} + \left(u_{\underline{k}} - P^{(i)}_{k_i-1,k_i} R^{(i)}_{k_i,k_i-1} u_{\underline{k}} \right)$$

Nun haben wir die grundlegenden Bausteine beisammen, um eine effizient berechenbare Zerlegung von u kennen zu lernen.

7 Dimensionsweise ANOVA-Zerlegung

In diesem Kapital soll die dimensionsweise ANOVA-Zerlegung einer Funktion nach [Gri06] vorgestellt werden. Diese baut auf der ANOVA-Zerlegung nach [ES81] auf.

Betrachten wir zuerst nochmals die eindimensionale Zerlegung aus Gleichung (3.4). Führen wir für die Zerlegung eine andere Notation ein, nämlich eine, die anzeigt, in welche Dimension die in der Funktion enthaltenen Level größer sind als das Referenzlevel l, so lautet die Zerlegung:

$$u_l = u_l^{\emptyset} + u_l^{\{1\}}$$

Hierbei steht die Indexmenge I in u_l^I für die Dimensionen, in denen ein Level k größer sein muss als l, um als Beitrag in u_l^I einzufließen. Somit sind $u_l^{\emptyset}$ die Beiträge aller Level, die in keiner Richtung größer sind als l; $u_l^{\{1\}}$ sind Beiträge von Gittern, die in Dimension 1 (der einzig vorhandenen Dimension) größer sind als l.

Dieses Idee wollen wir ins Mehrdimensionale überführen. Hierzu müssen wir jede Menge I berücksichtigen, in der ein Level im Mehrdimensionalen größer sein kann als ein anderes. Das bedeutet: Alle $I \subseteq 2^d$, wobei 2^d die Potenzmenge der Menge aller Dimensionen ist: $2^d = \{J : J \subseteq \{1, \ldots, d\}\}$. Danach ergibt sich die mehrdimensionale Zerlegung zu:

$$u_{\underline{l}} = \sum_{I \subseteq 2^d} u_{\underline{l}}^I \tag{7.1}$$

Diese Zerlegung nennen wir „dimensionsweise ANOVA-Zerlegung“ [Gri06]. Abbildung 7.1 zeigt diese Zerlegung anhand eines zweidimensionalen Beispiels. Für die einzelnen Komponenten $u_{\underline{l}}^I$ gilt nach obiger Beschreibung:

$$u_{\underline{l}}^I = \sum_{\substack{\underline{k} \in \mathcal{L} \\ k_i > l_i \text{ für } i \in I \\ k_i \leq l_i \text{ sonst}}} v_{\underline{k}} = \sum_{\underline{k} \succ_I \underline{l}} v_{\underline{k}} \tag{7.2}$$

Wobei wir das Prädikat $\succ_I$ zur Vereinfachung des Summationsindexes nutzen. $\underline{k} \succ_I \underline{l}$ bedeutet: $\underline{k}$ ist in Dimensionen $i \in I$ echt größer als $\underline{l}$ und in allen anderen Dimensionen nicht größer. $v_{\underline{k}}$ ist die Funktion der Überschüsse: $v_{\underline{k}} = \sum_{i \in \hat{I}_{\underline{k}}} v_{\underline{k},i} \phi_{\underline{k},i}$ mit $(v_{\underline{k},i})_{\underline{i} \in \hat{I}_{\underline{k}}}$ den eigentlichen Überschüssen.

Gleichung (7.1) beschreibt die Zerlegung, die wir für eine effiziente Evaluation des LGS (5.1) benutzen werden. Wollen wir das LGS lösen, müssen wir die Bilinearformen (besonders eben die rechte Seite) effizient auswerten können. Wie dies geschieht, wird im Folgenden erklärt.

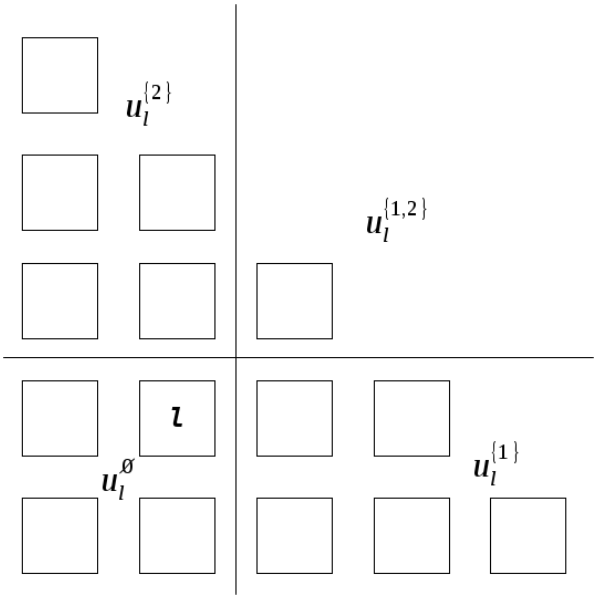


Abbildung 7.1: ANOVA-Zerlegung einer Funktion u bezüglich des Levels l: $u = u_{\underline{l}}^{\emptyset} + u_{\underline{l}}^{\{1\}} + u_{\underline{l}}^{\{2\}} + u_{\underline{l}}^{\{1,2\}}$. Alle vier ANOVA-Komponenten sind eingezeichnet. Das Kreuz verdeutlicht die Trennung zwischen den einzelnen Komponenten.

Auswertung der Bilinearform

Mithilfe der ANOVA-Zerlegung der Funktion u können wir das LGS aus Gleichung (5.1) nun wie folgt auswerten:

$$\forall \phi_{\underline{l},\underline{i}} \in \mathbf{B}_{\underline{l}} : \sum_{I \subseteq 2^d} a(u_{\underline{l}}^I, \phi_{\underline{l},\underline{i}}) = 0 \tag{7.3}$$

Betrachten wir den allgemeinen Fall: w sei eine Funktion zur Knotenbasis von Level $\underline{k}$ und wir wollen die Bilinearform $a(w, u_{\underline{k}}^I)$ auswerten. $u_{\underline{k}}^I$ sei eine ANOVA-Komponente zu Level $\underline{k}$, definiert nach Gleichung (7.2); enthält also die Überschüsse aller Level $\underline{l} \succ_I \underline{k}$. Zur Auswertung benutzen wir selbstverständlich die Definition selbst:

$$a(w, u_{\underline{k}}^I) = \sum_{\underline{l} \succ_I \underline{k}} a(w, v_{\underline{l}}) \tag{7.4}$$

Folglich bleiben Bilinearformen $a(w, v_{\underline{l}})$ auszuwerten. Wir werten diese anhand ihrer Koeffizientenvektoren $(w_{\underline{i}})_{\underline{i} \in I_{\underline{k}}}$ und $(v_{\underline{l},\underline{i}})_{\underline{i} \in I_{\underline{k}}}$ — wie oben bereits erwähnt bezeichnen wir diese der Einfachheit halber auch mit w und $v_{\underline{l}}$ — bezüglich der Knotenbasis ihres Levels aus. Da wir Funktionswerte niemals restringieren dürfen, ohne die Interpolationseigenschaften bezüglich des aktuellen Gitters zu verlieren, jedoch gefahrlos prolongieren können, müssen wir sie sozusagen auf dem Level des „Kleinsten Gemeinsamen Nenners“ auswerten. Dieses ist $\underline{m} = (\max(l_1, k_1), \ldots, \max(l_d, k_d))$. So stellen wir sicher, dass wir w und $v_{\underline{l}}$ stets nur

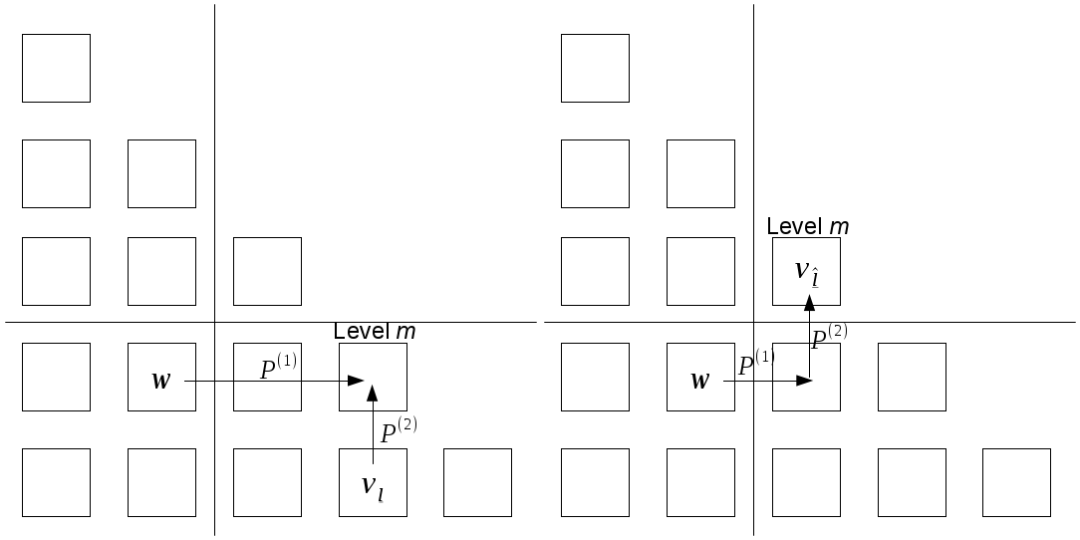


Abbildung 7.2: Auswertung der Bilinearformen $a(w, v_{\underline{l}})$ und $a(w, v_{\underline{\hat{l}}})$ schematisch dargestellt. Ausgewertet wird auf Level $\underline{m}$. Hierzu müssen im linken Beispiel w und $v_{\underline{l}}$ jeweils in die angegebenen Richtungen prolongiert werden. Im rechten Beispiel fallen $\underline{m}$ und $\underline{\hat{l}}$ zusammen, sodass nur w prolongiert werden muss.

prolongieren, jedoch in keine Richtung restringieren müssen. Level $\underline{m}$ lässt sich genauer bestimmen:

$$m_i = \begin{cases} l_i & \text{, falls } i \in I \\ k_i & \text{, sonst} \end{cases} \tag{7.5}$$

Ein zweidimensionales Beispiel wird in Abbildung 7.2 gezeigt. Man beachte, dass gelten kann $\underline{m} = \underline{l}$.

Um auf das Auswertungslevel $\underline{m}$ zu kommen, müssen wir w in Richtungen $i \in I$ prolongieren und $v_{\underline{l}}$ in Richtungen $i \notin I$. Damit ergibt sich für Gleichung (7.4):

$$\begin{aligned} a(w, u^I_{\underline{k}}) &= w^T \sum_{\underline{l} \succ_1 \underline{k}} \left(\prod_{i \in I} P^{T\ (i)}_{l_i,k_i} \right) \left(\prod_{i \in 2^d} A^{(i)}_{m_i} \right) \left(\prod_{i \notin I} P^{(i)}_{l_i,k_i} \right) v_{\underline{l}} \\ &\overset{\text{Gl. (7.5)}}{=} w^T \sum_{\underline{l} \succ_1 \underline{k}} \left(\prod_{i \in I} P^{T\ (i)}_{l_i,k_i} \right) \left(\prod_{i \in I} A^{(i)}_{l_i} \right) \left(\prod_{i \notin I} A^{(i)}_{k_i} \right) \left(\prod_{i \notin I} P^{(i)}_{l_i,k_i} \right) v_{\underline{l}} \\ &\overset{\text{Satz 1}}{=} w^T \left(\prod_{i \notin I} A^{(i)}_{k_i} \right) \underbrace{\sum_{\underline{l} \succ_1 \underline{k}} \left(\prod_{i \in I} P^{T\ (i)}_{l_i,k_i} A^{(i)}_{l_i} \right) \left(\prod_{i \notin I} P^{(i)}_{l_i,k_i} \right) v_{\underline{l}}}_{r^I_{\underline{k}}} \\ &= w^T \left(\prod_{i \notin I} A^{(i)}_{k_i} \right) r^I_{\underline{k}} \end{aligned} \tag{7.6}$$

Hier fassen wir jeweils die Summe zu neuen Größen $r^I_{\underline{k}}$ zusammen:

$$r^I_{\underline{k}} = \sum_{\underline{l} \succ_I \underline{k}} \left(\prod_{i \in I} {P^T_{l_i,k_i}}^{(i)} A^{(i)}_{l_i} \right) \left(\prod_{i \notin I} P^{(i)}_{l_i,k_i} \right) v_{\underline{l}} \tag{7.7}$$

Diese Größen können wir — analog zum Eindimensionalen — während des Durchlaufs mit Relaxation auf Unterräumen konsistent halten, denn es gilt folgender Zusammenhang:

$$r^I_{\underline{k}} = {P^T_{k_i,k_i+1}}^{(i)} \left(A^{(i)}_{k_i+1} r^{I \setminus \{i\}}_{\underline{k}+l_i} + r^I_{\underline{k}+l_i} \right) \tag{7.8}$$

Wobei Level $\underline{k} + l_i$ das in Dimension i eins feinere Level zu $\underline{k}$ ist. Diese Formel ist analog zum Eindimensionalen, außer, dass wir diese Größen in alle Dimensionen konsistent halten müssen. Ebenfalls wie im Eindimensionalen gilt offensichtlich: $r^{\emptyset}_{\underline{k}} = u_{\underline{k}}$.

Durch die mehrdimensionalen Operatoren, die nur in je ein Richtungen wirken, haben die $r^I_{\underline{k}}$ eine interessante Eigenschaft. Sie sind ihrer Definition nach in Richtungen $i \in I$ Residuen; in Richtungen $i \notin I$ jedoch noch Funktionswerte. Diese Eigenschaft machen wir uns zunutze, indem wir sie in den Richtungen, in denen sie Funktionswerte sind, sprich: $i \notin I$, hierarchisiert speichern — wie die Überschüsse $v_{\underline{l}}$ selbst. Das ist insofern nützlich, als damit Beiträge von Leveln, die kleiner als $\underline{k}$ sind, beim Aufstieg durch eine Dehierarchisierung in den $r^I_{\underline{k}}$ berücksichtigt werden — ebenfalls analog zum eindimensionalen Fall; hier wurden die Funktionswerte hierarchisiert gespeichert, um genau dieses Szenario auch abzudecken.

Da wir nun die Zerlegung kennen, wollen wir im nächsten Kapitel das eigentliche Auswertungsschema kennen lernen.

8 Mehrdimensionales Up-Down-Schema

In diesem Kapitel wird das mehrdimensionale Up-Down-Schema vorgestellt. Dazu wird zuerst auf die benötigten Größen eingegangen und dann die Algorithmen zum effizienten Traversieren des Unterraumschemas vorgestellt.

8.1 Unterraumschema

Wie in Kapitel 7 bereits erwähnt, benötigen wir die Größen $r_{\underline{k}}^{I}$ für einen effizienten Durchlauf durch das Unterraumschema. Von diesen gibt es auf jedem Level $\underline{k}$ offensichtlich 2^d Stück. Alle Komponenten des Unterraumschemas sind damit:

$$\left[r_{\underline{k}}^{I}\right]_{\underline{k}\in\mathcal{L},I\in 2^d}$$

Die $r_{\underline{k}}^{I}$ sind ihrerseits multidimensionale Koeffizientenvektoren — wie $u_{\underline{k}}$ — zum Gitter von Level $\underline{k}$; sprich: so viele Elemente wie Basisfunktionen in der Knotenbasis von Level $\underline{k}$.

8.2 Vorgehensweise

Das Konzept zum Lösen eines Randwertproblems ist wie folgt:

1. Anlegen des Schemas mit Initialisierung auf Null:

$$\forall I \subseteq 2^d, \underline{k} \in \mathcal{L} : r_{\underline{k}}^{I} = (0)_{\underline{i}\in\mathcal{I}_{\underline{k}}}$$

2. Q-Zyklus-Durchlauf zum Setzen der Initialwerte. Dies sind in unserem Fall nur Randwerte (es würde also reichen, über alle i, die Randindizes sind, zu iterieren). Die Randwerte für Level $\underline{l}$ werden gesetzt, wenn der Q-Zyklus Level $\underline{l}$ relaxiert.

$$\forall \underline{l} \in \mathcal{L}, \underline{i} \in I_{\underline{l}} : \left(r_{\underline{l}}^{\emptyset}\right)_{\underline{i}} = g(x_{\underline{l},\underline{i}})$$

3. Relaxierende Q-Zyklus-Durchläufe als Mehrgitterverfahren zum eigentlichen Lösen des Randwertproblems.
4. Fertig. (Evtl. volles Gitter interpolieren o.ä.)

8.3 Algorithmen

Die Algorithmen zum Durchlauf durch das Schema heißen wie im Eindimensionalen: *Verfeinern* und *Vergröbern*. Zum Verfeinern in Richtung i dehierarchisieren wir diejenigen $r^I_{\underline{k}}$, für die gilt: $i \notin I$; denn diese Richtungen i sind die $r^I_{\underline{k}}$ Funktionswerte. Die Funktion zum n-dimensionalen Vergröbern wird in Algorithmus 8.1 dargestellt.

Algorithmus 8.1 `nd-Verfeinern(Dimension i, Level k)`

for all $I \in 2^d : i \notin I$ **do**
  $r^I_{\underline{k}+l_i} \leftarrow r^I_{\underline{k}+l_i} + P^{(i)}_{k_i,k_i+1} r^I_{\underline{k}}$ // Dehierarchisierung
end for

Analog zum eindimensionalen Fall, ist der Arbeitsaufwand beim Vergröbern etwas höher. Neben der obligatorischen Hierarchisierung von Größen, die in Vergröberungsrichtung i Funktionswerte darstellen, werden hier in einer Richtung aus Funktionswerten Residuen errechnet. Dies geschieht nach Gleichung (7.8). Die Funktion zum n-dimensionalen Vergröbern wird in Algorithmus 8.2 dargestellt.

Algorithmus 8.2 `nd-Vergröbern(Dimension i, Level k)`

for all $I \in 2^d : i \notin I$ **do**
  $r^I_{\underline{k}} \leftarrow R^{(i)}_{k_i+1,k_i} r^I_{\underline{k}+l_i}$
  $r^I_{\underline{k}+l_i} \leftarrow r^I_{\underline{k}+l_i} - P^{(i)}_{k_i,k_{i+1}} r^I_{\underline{k}}$ // Hierarchisierung
end for
for all $I \in 2^d : i \in I$ **do**
  $r^I_{\underline{k}} \leftarrow {P^T_{k_i,k_i+1}}^{(i)} \left(A^{(i)}_{k_i+1} r^{I\setminus\{i\}}_{\underline{k}+l_i} + r^I_{\underline{k}+l_i} \right)$
end for

Als Durchlaufschema nehmen wir hier den Q-Zyklus. Dieser ist in Algorithmus 2.3 darstellt. Für ein Mehrgitterzyklus müssen wir einzelne Level relaxieren. Hierfür benötigen wir auf gegebenem Level das Residuum. Dieses berechnen wir nach Gleichung (7.6). Die Berechnung ist nochmals in Algorithmus 8.3 dargestellt.

Algorithmus 8.3 `nd-Residuum(Level k)`

$$r_{\underline{k}} = \sum_{I \in 2^d} \left(\prod_{i \notin I} A^{(i)}_{k_i} \right) r^I_{\underline{k}}$$

8.4 Laufzeit

Die nd-Vergröbern-Funktion benötigt offensichtlich mehr Operationen als das nd-Verfeinern. Betrachten wir also nur erstere. Sei $N_{\underline{k}}$ die Anzahl der Freiheitsgrade der Knotenbasis auf Level $\underline{k}$. Die Anwendung eines mehrdimensionalen Operators kostet höchstens $3N_{\underline{k}}$ Operationen, wenn die Matrizen — wie in unserem Fall — höchstens tridiagonal sind (R, P und P^T besitzen echt weniger Einträge). Die Addition zweier $r_{\underline{k}}^I$ benötigt trivialerweise $N_{\underline{k}}$ Operationen. In beiden Schleifen werden zwei Operatoren angewendet und eine Addition. Zusammen sind dies höchstens $7N_{\underline{k}}$ Operationen. Beide Schleifen iterieren zusammen genau über alle $I \subseteq 2^d$, sodass sich eine Gesamtlaufzeit ergibt von höchstens $2^d 7N_{\underline{k}}$ Operationen, also $\mathcal{O}(2^d N_{\underline{k}})$, was linear in $N_{\underline{k}}$ ist.

Die Berechnung des Residuums aus den ANOVA-Komponenten ist etwas komplizierter: Evaluiert man die Formel, indem auf jedes $r_{\underline{l}}^I$ zuerst alle Operatoren angewendet werden und danach alles addiert wird, ergibt sich folgende Laufzeit: Es gibt 2^d Summationen. Die Anzahl der Operatoranwendungen für $|I| = i$ ist $d - i$. Da es $\binom{d}{i}$ Teilmengen von $\{1, \ldots, d\}$ mit i Elementen gibt, ergibt sich folgende Formel für die Kosten aller Operatoranwendungen:

$$\sum_{i=0}^{d} \binom{d}{i} (d-i) 3N_{\underline{k}} = 3N_{\underline{k}} \sum_{i=0}^{d} \binom{d}{i} i < 3N_{\underline{k}} 2^d d$$

Der Gesamtaufwand für die Berechnung des Residuums läge somit in $\mathcal{O}(d2^d N_{\underline{k}})$, also linear in $N_{\underline{k}}$.

Das Residuum kann aber schneller berechnet werden: Indem man sukzessive $A^{(i)}$ ausklammert, können Operatoranwendungen gespart werden. Jeder Operator muss nur noch 2^{i-1} mal angewendet werden. Die Kosten für die Operatoranwendungen sind also:

$$\sum_{i=1}^{d} 2^{i-1} 3N_{\underline{k}} = 3N_{\underline{k}}(2^d - 1) \in \mathcal{O}(2^d N_{\underline{k}})$$

Zusammen mit den $2^d N_{\underline{k}}$ Operationen für die Summationen, ergibt sich ein Gesamtaufwand für die Berechnung des Residuums von $\mathcal{O}(2^d N_{\underline{k}})$; wiederum linear in $N_{\underline{k}}$, jedoch deutlich schneller als das direkte Auswertungsschema.

9 Erläuterungen

In diesem Kapitel werden Erläuterungen zum vorgestellten mehrdimensionalen Up-Down-Schema gegeben.

9.1 Korrektheit

Um zu zeigen, dass das Schema auf Level $\underline{k}$ das korrekte Residuum zur levelweisen Relaxation berechnet, muss die Gültigkeit von Gleichung (7.7), also die Definition einer Komponente $r_{\underline{k}}^I$, wenn der Q-Zyklus das Level $\underline{k}$ besucht, bewiesen werden. Mit gültiger Gleichung (7.7) und der Definition der dimensionsweisen ANOVA-Zerlegung aus Gleichung (7.1) würde folgen:

$$a(w,u) \overset{\text{Gl. (7.1)}}{=} \sum_{I \in 2^d} a(w, u_{\underline{k}}^I) \overset{\text{Gl. (7.6), (7.7)}}{=} w^T \sum_{I \in 2^d} \left(\prod_{i \notin I} A_{k_i}^{(i)} \right) r_{\underline{k}}^I \overset{\text{Alg. 8.3}}{=} w^T r_{\underline{k}}$$

Der Beweis von Gleichung (7.7) steht aus. Betrachtet man jedoch die in Kapitel 8.2 vorgestellte Vorgehensweise zur Berechnung eines Randwertproblems, so reicht es, zu zeigen, dass ein Q-Zyklus-Durchlauf mit levelweiser Änderung der Überschüsse nichts an der Gültigkeit der Gleichung (7.7) ändert, da sie nach dem ersten Schritt (Initialisierung aller $r_{\underline{l}}^I$ auf Null) trivialerweise gültig ist. Die danach folgende Initialisierung sowie Relaxationen stellen eine levelweise Änderung der Überschüsse dar.

Im Folgenden werden Erläuterungen zum Algorithmus gegeben, die nahe legen, dass diese Gleichung gilt, jedoch keinen formalen Beweis darstellen.

9.2 Q-Zyklus-Graph

Wir betrachten den Q-Zyklus-Durchlauf als Graphen. Die hierarchischen Unterräume sind die Knoten. Diesen ist also ein Level zugeordnet. Die Kanten sind gegeben durch die Beziehungen zwischen den Level, die sich durch Vergröberungen und Verfeinerungen ergeben. Formal also folgenden Graphen G:

$$G = (V, E) \text{ mit: } V = \mathcal{L} \text{ und } E = E_G \cup E_F$$

Wobei:

$$E_F = \{(i,j) \in V \times V : \text{ Verfeinern } i \to j \text{ wird aufgerufen}\}$$
$$E_G = \{(i,j) \in V \times V : \text{ Vergröbern } i \to j \text{ wird aufgerufen}\}$$

Beobachtung 1. *In diesem Graph besitzt jeder Knoten Verfeinern- und Vergröbern-Kanten mit all seinen 2^d Nachbarn (Randknoten je nachdem weniger).*

Begründung. Die Dimensionsrekursion des Q-Zyklus zusammen mit der Schleife von Apply-Codim1-Entity, die einen kompletten Unterraum der Kodimension 1 verfeinert bzw. vergröbert, und der des V-Zyklus, die in einer Dimension vergröbert bzw. verfeinert, ergibt direkt, dass alle Level $\underline{k}$ in jede Dimension, in die ein Nachfolger existiert verfeinert und in jede Dimension, in der ein Vorgänger existiert, vergröbert werden. □

Aus dieser Beobachtung und der Definition der Levelmenge $\mathcal{L}$ folgt direkt, dass der Graph G stark zusammenhängend ist.

9.3 Pfade im Graphen

Wir betrachten die Pfade in G, auf denen die Überschüsse $v_{\underline{l}}$ eines Levels $\underline{l}$ nach $\underline{k}$ propagiert werden. Sei Level $\underline{l} = (l_1, \ldots, l_n)$ bereits besucht und der Q-Zyklus befindet sich aktuell bei Level $\underline{k} = (k_1, \ldots, k_n)$. Wir definieren die Mengen G und K, wobei G die Menge der Richtungen ist, in der $\underline{k}$ größer als $\underline{l}$ und K die Menge der Richtungen, in der $\underline{k}$ kleiner als $\underline{l}$ ist:

$$i \in G \Leftrightarrow l_i < k_i$$

$$i \in K \Leftrightarrow l_i > k_i$$

Man beachte, dass diese Mengen per Definition disjunkt sind ($G \cap K = \emptyset$) und je eine Teilmenge der Menge aller Richtungen sind: $G \subseteq \{1, \ldots, n\}$, $K \subseteq \{1, \ldots, n\}$.

Durch die Definition der Levelmenge $\mathcal{L}$ und die Existenz aller Kanten zwischen benachbarten Knoten sollte klar sein, dass es Pfade von Level $\underline{l}$ nach $\underline{k}$ gibt, die nach Manhattan-Metrik optimal sind.

Beobachtung 2. *Die Überschüsse $v_{\underline{l}}$ von Level $\underline{l}$ werden auf einem nach Manhattan-Metrik optimalen Pfad transportiert*

Begründung. Annahme, die Überschüsse würden nicht auf einem solchen Pfad propagiert werden. Dann gibt es eine Richtung i, in die Vergröbert und Verfeinert wird. Es werden nun beide Fälle betrachtet:

Verfeinern(i) nach Vergröbern(i):
Nach dem Vergröbern in Richtung i sind die Überschüsse von Level $\underline{l}$ Summand in allen r^I mit $i \in I$. Dieses i wird in keinem Vergröbern- oder Verfeinern-Aufruf, der nun folgt, mehr aus den Indexmengen I genommen. Das dann folgende Verfeinern(i) übernimmt nur $r^{I'}$ mit $i \notin I'$ und damit nicht den Überschuss $v_{\underline{l}}$.

Vergröbern(i) nach Verfeinern(i):
Nach der Verfeinerung in Richtung i ist nochmals eine Vergröberung in Richtung i notwendig, d.h. der Apply-Codim1-Entity-Aufruf von Zeile 3 im V-Zyklus kann nicht nach Level $\underline{k}$ führen.

Damit wird — bevor eine eventuell nötige Operation in eine andere Richtung durchgeführt wird — die komplette Kodimension-1-Entität wieder vergröbert und damit auch Level $\underline{l} + e_i$ nach Level $\underline{l}$. Dabei werden die Überschüsse $v_{\underline{l}}$ wieder von $r^{\emptyset}_{\underline{l}+e_i}$ abgezogen. Somit können diese von Level $\underline{l} + e_i$ nicht weiter propagiert werden.

□

Ein Pfad π, auf dem Überschüsse propagiert werden, hat jedoch eine zusätzliche Eigenschaften: Er beginnt mit allen nötigen Vergröberungen und danach erst wird verfeinert. Intern haben die Vergröberungen und Verfeinerungen auch eine Reihenfolge. Diese ist durch den Q-Zyklus vorgegeben. Im Falle von Algorithmus 2.3 werden die Dimensionen während des Vergröberns aufsteigend und während des Verfeinerns absteigend behandelt.

Dieser Pfad π, der nach Manhattan-Metrik optimal ist und erst nach Dimension aufsteigend vergröbert und danach nach Dimension absteigend verfeinert, ist für jedes Levelpaar $\underline{l}$, $\underline{k}$ einzigartig.

Für Überschüsse, die auf genau diesem Pfad π transportiert werden, können wir folgendes beobachten:

Beobachtung 3. *Wurden die Überschüsse $v_{\underline{l}}$ von Level $\underline{l}$ nach $\underline{k}$ auf genanntem Pfad π propagiert, so enthält $r^K_{\underline{k}}$ den Summanden*

$$\left(\prod_{i\in I} {P^T_{k_i,l_i}}^{(i)} A^{(i)}_{l_i}\right)\left(\prod_{i\notin I} P^{(i)}_{l_i,k_i}\right) v_{\underline{l}} \tag{9.1}$$

Begründnug. O.B.d.A. sei $K = \{i_1, \ldots, i_n\}$ und die Dimensionen werden in der Reihenfolge $i_1, \ldots, i_n$ vergröbert. Das Level $\underline{m}$ ergibt sich aus den nötigen Vergröberungen in Richtungen $i \in K$ von Level $\underline{l}$ aus, also:

$$m_i = \begin{cases} k_i & \text{, falls } i \in K \\ l_i & \text{, sonst} \end{cases}$$

Sei $I \subseteq K$ die Menge der Indizes, nach der bereits vergröbert wurde. Für $I = \emptyset$, also auf Level $\underline{l}$ selbst, gilt offensichtlich: $r^{\emptyset}_{\underline{l}}$ enthält $v_{\underline{l}}$ als Summand. Per Induktion über I mithilfe von Gleichung (7.8) (also der Vergröberungsvorschrift) folgt dann, dass $r^K_{\underline{m}}$ folgenden Summanden enthält:

$$\left(\prod_{i\in K} {P^T_{m_i,l_i}}^{(i)} A^{(i)}_{l_i}\right) v_{\underline{l}} \tag{9.2}$$

Betrachten wir nun die Verfeinerungen. Sei $G = \{j_1, \cdots, j_p\}$. Da gilt: $\forall j \in G : j \notin K$, wird der Summand (9.2) nach Verfeinerungsvorschrift in alle Richtungen $j \in G$ prolongiert.

Damit enthält $r^K_{\underline{k}}$ mithilfe von Satz 1 folgenden Summanden:

$$\left(\prod_{i\in G} P^{(i)}_{l_i,k_i}\right)\left(\prod_{i\in K} P^{T}_{k_i,l_i}{}^{(i)} A^{(i)}_{l_i}\right) v_{\underline{l}} = \left(\prod_{i\in K} P^{T}_{k_i,l_i}{}^{(i)} A^{(i)}_{l_i}\right)\left(\prod_{i\notin I} P^{(i)}_{l_i,k_i}\right) v_{\underline{l}}$$

Die Gleichheit kommt daher, dass in Richtungen $i \notin K \cup G$ gilt: $k_i = l_i$ und damit für die Prolongation gilt: $P^{(i)}_{l_i,k_i} = \mathbb{I}$.

□

Fordern wir an dieser Stelle, dass Gleichung (7.7) beim letzten Q-Zyklus-Durchlauf galt, so müssen die Beiträge von Leveln $\underline{l}$, die in diesem Durchlauf noch nicht besucht wurden, noch korrekt in $r^K_{\underline{k}}$ vorhanden sein. Eventuell wurden sie jedoch bei einer beim letzten Q-Zyklus-Durchlauf erfolgten Vergröberung subtrahiert (durch eine Hierarchisierung). Da spätestens Level $\underline{m}$ mit $m_i = \min(l_i, k_i)$ diese Beiträge jedoch speichert und dieses Level vor $\underline{k}$ besucht wird, sollten die Beiträge von $\underline{l}$ wieder korrekt dehierarchisiert vorliegen.

Deshalb sollte nach dieser Betrachtung ein Q-Zyklus-Durchlauf mit levelweiser Änderung der Überschüsse nichts an der Gültigkeit der Gleichung (7.7) ändern. Wie oben bereits erwähnt, ist dies jedoch noch formal zu beweisen.

10 Numerische Ergebnisse

In diesem Kapitel werden zuerst Hinweise für eine Implementierung gegeben. Anschließend wird ein Beispielproblem vorgestellt sowie die Lösung und Konvergenzraten, die sich mit dem vorgestellten Verfahren ergeben.

10.1 Implementierungshinweise

Hier sollen einige Hinweise zu einer effizienten Implementierung der vorgestellten Algorithmen gegeben werden. Eine der wichtigsten Designfragen ist, ob man die eindimensionalen Operatoren R, P, P^T, A (als Grundlage für die benötigten mehrdimensionalen Operationen) als Matrizen oder als nativen Code implementiert. Ersteres erlaubt eine einfache Adaption auf neue Basen, letzteres sorgt erstens für eine deutlich schnellere Auswertung ($\mathcal{O}(N)$ im Gegensatz zu $\mathcal{O}(N^2)$ bei den in Kapitel 4 definierten Operatoren — solange man sie als volle Matrizen speichert) und keinen zusätzlichen Speicherverbrauch zur Laufzeit. Angenommen, die maximale Tiefe im Unterraumschema ist $t_{\max}$, so benötigen wir eine Matrix A, die genau auf diesem eindimensionalen Level arbeitet. Somit wäre der Speicheraufwand für die Matrix A $\mathcal{O}(2^{t_{\max}^2})$. Das größte Gitter hat einen Speicherbedarf von $\mathcal{O}\left(2^{\left(\frac{t_{\max}}{d}\right)^d}\right)$. Damit folgt: Solange man in zwei und weniger Dimensionen arbeitet, ist die Optimierung des Speicherbedarfs der Matrizen sehr viel wichtiger als die der Gitter. Arbeitet man in mindestens drei Dimensionen, so ist der Speichherbedarf für die eindimensionalen Matrizen *asymptotisch* vernachlässigbar.

Um beide Varianten zu ermöglichen, möge man ein gemeinsames Interface benutzen. In Sprachen, die Operatorüberladung ermöglichen, ist dies naheliegend durch die Multiplikation-Operation zu realisieren. So können Klassen, die dieses Interface implementieren sehr einfach mit eventuell gespeicherten Matrizen multiplizieren oder eine in Code implementierte Routine aufrufen.

Zur Einordnung des zeitlichen Vorteils bei der Nutzung von nativem Code wurde eine Zeitmessung für beide Verfahren durchgeführt. Die Matrizen wurden je als `numpy.Matrix` gespeichert und zur Laufzeit erstellt. Außerdem wurde die Matrix-Vektor-Multiplikation in C-Code implementiert, der mittels Foreign-Function-Call-Bibliothek `ctypes` aus dem Python-Code (der das mehrdimensionale Up-Down-Schema implementiert) heraus aufgerufen. Die Zeiten für das Matrix-Vektor-Produkt in beiden Varianten und die Erstellung der Matrizen befinden sich geplottet für verschiedene Gittergrößen in Abbildung 10.1. Wie man erkennen kann, ist das Matrix-Vektor-Produkt der numpy-Bibliothek für kleine Gittergrößen schneller

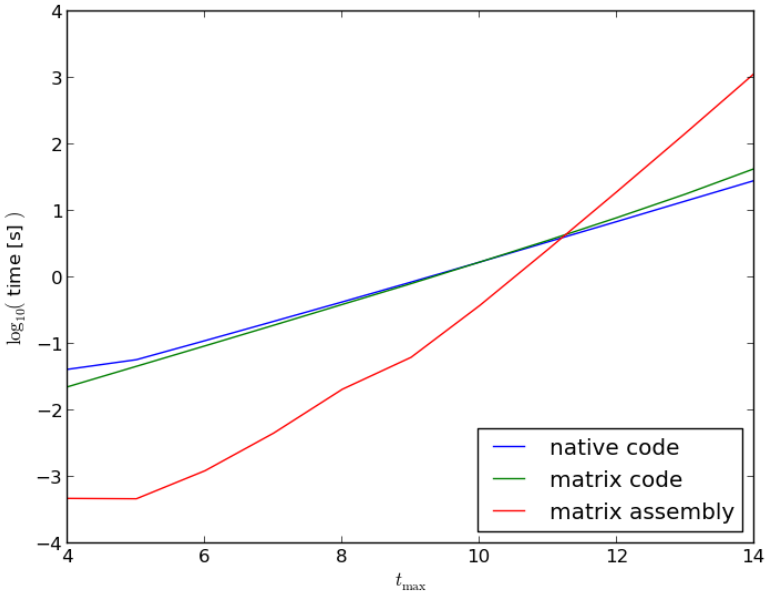


Abbildung 10.1: Vergleich der Zeiten für einen Q-Zyklus-Durchlauf ohne Glättung auf einem dünnen Gitter. Für Gittergrößen ab Tiefe 11 liegt der zeitliche Gewinn der Implementierung der Operationen als Code vor allem im Wegfall der Matrixaufstellung. Die verwendete Matrix-Implementierung `numpy.Matrix` ist performanzmäßig sehr gut, kann es für größere Gittergrößen jedoch nicht mit einer direkten Implementierung als Code aufnehmen. Zeitskala logarithmisch!

als die native Implementierung, was vermutlich auf den Foreign-Function-Call-Overhead zurückzuführen ist. Für große Gittergrößen kann man erkennen, dass der zeitliche Vorteil eher bei der nativen Implementierung liegt; insbesonderen, wenn man die benötigte Zeit für die Erstellung der Matrizen beachtet.

Eine Parallelisierung der eindimensionalen Operationen lohnt im Allgemeinen nicht, da diese meistens auf zu kleinen Gittern arbeiten. Dies wurde auch experimentell mittels einer Shared-Memory-Parallelisierung (unter Verwendung von OpenMP) im C-Code überprüft. Eine adaptive Variante, die nur auf sehr feinen Gitter parallelisiert wäre denkbar und könnte eventuell einen Performanzgewinn bringen. Ferner könnte überprüft werden, ob sich die Parallelisierung der mehrdimensionalen Operationen (sprich: mehrere eindimensionale Operationen zur selben Zeit auf dem selben Gitter durchführen) oder der nd-Vergröbern- bzw. nd-Verfeinern-Algorithmen (Parallelisierung über die Indexmengen, die zu bearbeiten sind).

10.2 Beispielproblem

Als Beispielproblem wird die folgende Laplacegleichung (10.1) genommen. Die Randwerte sind gegeben durch Gleichung (10.2).

$$\begin{aligned} \Delta u &= 0, \quad && \text{in } \Omega \\ u &= g, && \text{auf } \delta\Omega \end{aligned} \tag{10.1}$$

Wobei:

$$g(\boldsymbol{x}) = \mathrm{e}^{-\sqrt{d-1}\pi x_d} \prod_{j=1}^{d-1} \sin(\pi x_j) \tag{10.2}$$

Um eine mehrdimensionale Lösung zu errechnen, müssen wir den Laplace-Operator in Tensorproduktoperatoren zerlegen:

$$\Delta = \sum_{d=1}^{n} \partial_d^2 \otimes \bigotimes_{j \neq d} \mathbb{I}^{(j)} = \sum_{d=1}^{n} A_{\partial^2} \otimes \bigotimes_{j \neq d} A_{L^2}$$

Wobei die eindimensionalen Operatoren A_{∂^2} und A_{L^2} durch die beiden in Kapitel 3.1 angegebenen Operatorsterne $\frac{h}{6}\,[1\ 4\ 1]$ und $\frac{1}{h}\,[-1\ 2\ -1]$ gegeben sind.

Die zweidimensionale Lösung dieses Randwertproblems auf $\Omega = [0,1]^2$ ist in Abbildung 10.2 dargestellt. Sie wurde mit einem zweidimensionalen dünnen Gitter der Tiefe 4 mit Freiheitsgraden auf dem Rand berechnet.

Konvergenzergebnisse für das Mehrgitterverfahren in Abhängigkeit von der Anzahl der Q-Zyklus-Durchläufe sind in Abbildung 10.3 dargestellt. Wie man erkennen kann, zeigt der Graph die typische Eigenschaft eines Mehrgitterverfahrens: Die Konvergenzrate ist unabhängig von der maximalen Tiefe, sprich der Gittergröße des größten Gitters. Sie liegt beim zweidimensionalen Problem in etwa bei einer Größenordnung pro Q-Zyklus und im dreidimensionalen Problem bei einer Größenordnung pro zwei Q-Zyklen. Bei Peherstorfer und Bungartz [PB12] zeigen sich etwas bessere Konvergenzraten für ein zweidimensionales Problem. Sie verwenden jedoch volle Gitter mit Gauss-Seidel-Iterationen. Vergleichbar sind die erlangten Ergebnisse auch zu [Pfl98, Peh10].

10.3 Einordnung

Der verfügbare Speicher limitiert selbstverständlich die maximale Tiefe des dünnen Gitters. Mit einer direkten Implementierung des hier vorgestellten Schemas ohne zusätzlichen Aufwand, Speicher bei den $r_{\underline{l}}^{I}$ zu sparen, können bereits praxistaugliche Problemgrößen berechnet werden. Aufgeführt sind diese maximalen Tiefen in Tabelle 10.1.

Ein Problem dieses Schemas ist, dass es nicht trivial ist, nach Aktualisierung eines Freiheitsgrades das Residuum effizient zu aktualisieren. Bei einfacher Implementierung der

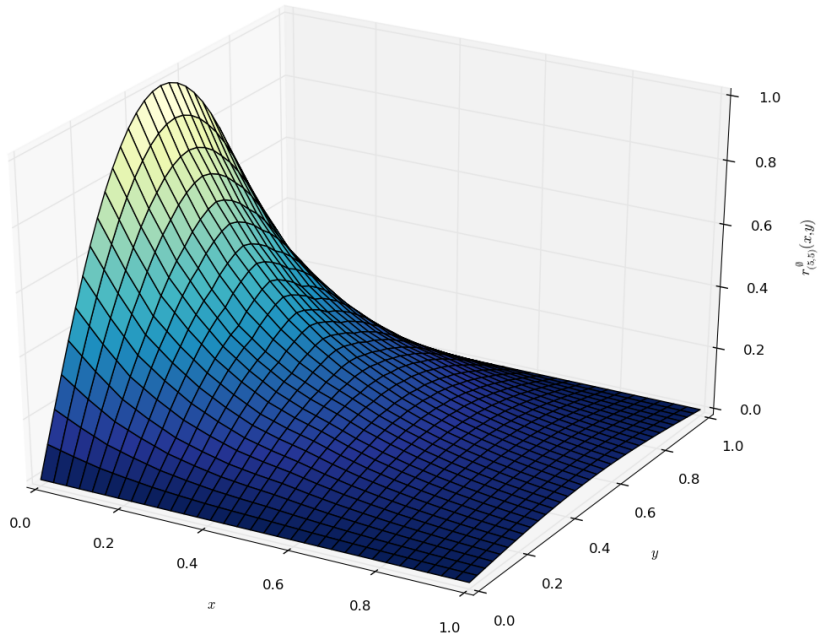
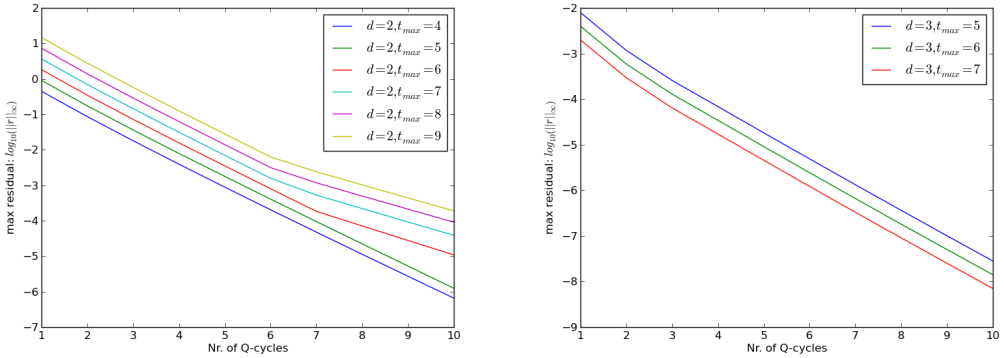


Abbildung 10.2: Lösung der Laplacegleichung auf $\Omega = [0,1]^2$ auf einem dünnen Gitter der Tiefe 4 mit Freiheitsgraden auf dem Rand berechnet. Zu Darstellungszwecken wurde das volle Gitter mit 33×33 Punkten interpoliert.

Dimensionen	maximale Tiefe	tatsächlicher Speicherbedarf
2	22	9 GB
3	17	8 GB
4	14	15 GB
5	10	6 GB
6	8	9 GB

Tabelle 10.1: Maximale berechenbare Tiefe eines dünnen Gitters mit 16GB RAM. Der tatsächliche Speicherbedarf ist gerundet.



(a) Zweidimensional **(b)** Dreidimensional

Abbildung 10.3: Residuum zur Maximumsnorm auf die Anzahl Q-Zyklen abgetragen. Links auf einem zweidimensionalen Gebiet; rechts dreidimensional. Wie deutlich zu erkennen ist, ist die Konvergenzrate (Steigung) unabhängig von der Gittergröße. Als Glätter wurde je eine Jacobi-Iteration benutzt. Die beiden abweichenden Konvergenzraten im zweidimensionalen Fall sind die kleinsten Gitter. Damit kann davon ausgegangen werden, dass diese nicht repräsentativ sind, da sie zu wenig Gitterpunkte haben.

Art	Zeit ($t_{\max} = 10$)	Zeit ($t_{\max} = 11$)	Zeit ($t_{\max} = 12$)
ohne Glätter	6.2s	11.9	23.6
zehn Jacobi-Iterationen	66.2s	136.0s	280.9s
zehn Jacobi-Iteration (C)	11.5s	23.1	47.0s
zehn Gauss-Seidel-Iteration	509.1s	2260.1	9603.2s

Tabelle 10.2: Zeit für Q-Zyklus-Durchläufe mit und ohne Glättung. Gemessen wurde auf einem Intel®Xeon®E7540 mit 2.0GHz. Das dünne Gitter hat zwei Dimensionen die maximale Tiefe ist in der obersten Zeile angegeben. Die dritte Zeit ist gemessen mit in C implementierten Jacobi-Iterationen (per `ctypes` aufgerufen). Alle Zeiten auf eine Nachkommastelle gerundet. Die Gauss-Seidel-Implementierung berechnet das Residuum nach jeder Aktualisierung eines Freiheitsgrades neu.

Residuumsberechnung, muss es jedes mal neu berechnet werden. Diese Neuberechnung liegt zwar nur in $\mathcal{O}(N)$; müsste jedoch nach jedem der N Freiheitsgrade neu durchgeführt werden. Dies macht es nicht effizient möglich, Gauss-Seidel-Iterationen als Glätter bei einer einfachen Implementierung einzusetzen. Jacobi-Iterationen hingegen benötigen während der Berechnung das neue Residuum nicht. Deshalb sind diese auf dem vorgestellten Schema einfach durchzuführen. Zur Einordnung der Kosten von Iterationsverfahren , sind in Tabelle 10.2 die Zeiten für Q-Zyklus-Durchläufe ohne Glättung, mit je zehn Jacobi-Iterationen und je zehn Gauss-Seidel-Iteration dargestellt.

Wie man erkennen kann, ist das Zeitaufwändigste die Relaxation der Gitter. Aus Tabelle 10.2 kann man entnehmen, dass die Zeit für eine Jacobi-Iteration pro Level in etwa so teuer ist, wie ein Q-Zyklus-Durchlauf selbst. Mit der C-Implementierung kann dies um einen deutlichen Faktor verringert werden. Eine Jacobi-Iteration ist offensichtlich linear in $2^{t_{\max}}$. Deutlich erkennbar ist auch, dass Gauss-Seidel-Operationen quadratisch in $2^{t_{\max}}$ sind. Dies ist jedoch — wie oben bereits erwähnt — nicht inhärent im Schema verankert, sondern viel eher ein Problem der Implementierung.

11 Zusammenfassung und Ausblick

In dieser Arbeit wurde zuerst eine auf eindimensionale Probleme begrenzte Einführung in das Thema zum besseren Problemverständnis gegeben.

Anschließend wurden Verfeinern- und Vergröbern-Algorithmen vorgestellt, mit denen es möglich ist, hierarchische Basen mit dem Q-Zyklus effizient zu traversieren und dabei während des Durchlaufs die Überschüsse zu modifizieren. Die Algorithmen sind hierbei auf die Lösung von elliptischen partiellen Differentialgleichungen mittels des vorgestellten Mehrgitterschemas beschränkt. Hierzu wurde die dimensionsweise ANOVA-Zerlegung vorgestellt und gezeigt, dass die Algorithmen alle ANOVA-Komponenten zum Berechnen des Residuums benutzen. Des weiteren sind die Operatoren in den vorgestellten Funktionen frei wählbar, solange sie die genannten Forderungen erfüllen, was eine einfache Adaption auf neue Basen und Bilinearformen ermöglicht.

Der Ablauf des Q-Zyklus mit den vorgestellten Funktionen wurde genauer betrachtet. Hierbei wurde insbesondere auf die Bedingung für die Korrektheit des vorgestellten mehrdimensionalen Up-Down-Schema eingegangen.

Die Kosten für das Berechnen des Residuums und das Vergröbern bzw. Verfeinern wurden abgeschätzt. Diese sind linear in der Anzahl der Gitterpunkte, was optimal ist. Damit ist eine effiziente Traversierung des Schemas mithilfe des Q-Zyklus garantiert.

Als letzter Punkt wurden die Konvergenzraten des entstandenen Verfahrens experimentell mittels der Laplace-Gleichung bestimmt. Diese haben sich als unabhängig von der Gittergröße herausgestellt, was typisch für Mehrgitterverfahren ist.

Ausblick

Das vorgestellte mehrdimensionale Up-Down-Schema ist beschränkt auf den Q-Zyklus als Traversierungsschema, da die mehrdimensionalen Vergröbern- und Verfeinern-Algorithmen auf eben diesen ausgelegt sind. Deshalb könnte untersucht werden, ob ein multidimensionales Up-Down-Schema entwickelt werden kann, dass — wie dieses — alle ANOVA-Komponenten verwendet, jedoch für allgemeine Traversierungsschemata gültig ist. Wenngleich das wohl am weitesten verbreitete Mehrgitterschema für hierarchische Basen der Q-Zyklus sein dürfte.

Eine wichtige Erweiterung des vorgestellten mehrdimensionalen Up-Down-Schemas wäre die Verwendung von adaptiven Dünnen Gittern. Da diese erlauben, die Anzahl an Freiheitsgraden in gewissen Gebieten weiter zu reduzieren, in denen die Funktion sehr glatt ist und sich wenig ändert.

Des weiteren könnte an dem vorgestellten Schema untersucht werden, wo und in welcher Form sich Parallelisierung lohnt. Diese ist notwendig, will große Probleme auf aktuellen Rechnern effizient lösen. Wobei nicht nur Shared-Memory-Parallelisierung eine Rolle spielen sollte, sondern auch eine Untersuchung, inwieweit sich das Verfahren für eine Distributed-Memory-Parallelisierung eignet.

Dünne Gitter besitzen zwar signifikant weniger Freiheitsgrade als volle Gitter, trotzdem ist der Speicherbedarf des hier vorgestellten Unterraumschemas recht hoch. Deshalb sollte untersucht werden, ob der Speicherbedarf verringert werden kann. Hier könnten Schrumpfung von Koeffizienten anhand von Schwellwerten überprüft werden (dieses Verfahren ist wohlbekannt in anderen Basen, insbesondere z.B. der ebenfalls hierarchischen Haar Wavelets). Auch verlustfreie Verfahren wie Huffman- oder Lauflängenkodierung wären denkbar.

Schließlich steht der eigentliche Korrektheitsbeweis des Schemas aus. In Kapitel 9 wurden hierfür bereits einige Erläuterungen und Ansätze gegeben.

Literaturverzeichnis

[BG04] H.-J. Bungartz, M. Griebel. Sparse grids. *Acta Numerica*, 13:1–123, 2004. (Zitiert auf den Seiten 7, 9, 10, 11 und 25)

[ES81] B. Efron, C. Stein. The Jackknife Estimate of Variance. *Annals of Statistics*, 9:586–596, 1981. (Zitiert auf den Seiten 7 und 31)

[Gri90] M. Griebel. *Zur Lösung von Finite-Differenzen- und Finite-Element-Gleichungen mittels der Hierarchischen-Transformations-Mehrgitter-Methode*. Dissertation, Institut für Informatik, Technische Universität München, 1990. (Zitiert auf den Seiten 7, 19, 21, 23 und 28)

[Gri06] M. Griebel. Sparse Grids and Related Approximation Schemes for Higher Dimensional Problems. In *Foundations of Computational Mathematics*. Cambridge University Press, 2006. (Zitiert auf den Seiten 7 und 31)

[Hac91] W. Hackbusch. *Iterative Lösung grosser schwachbesetzter Gleichungssysteme*. Teubner Verlag, 1991. (Zitiert auf den Seiten 11 und 13)

[Hac96] W. Hackbusch. *Theorie und Numerik elliptischer Differentialgleichungen*. Teubner Verlag, 1996. (Zitiert auf den Seiten 7 und 15)

[PB12] B. Peherstorfer, H.-J. Bungartz. Semi-Coarsening in Space and Time for the Hierarchical Transformation Multigrid Method. In *Proceedings of the International Conference on Computational Science*, Band 9 von *Procedia Computer Science*. Omaha, Nebraska, USA, 2012. (Zitiert auf den Seiten 7, 12, 27, 28 und 45)

[Peh10] B. Peherstorfer. *HT-Multigrid-compatible Time/Space Discretizations of the Stokes Equations in a Pressure Gradient Formulation*. Master's Thesis, Technische Universität München, 2010. (Zitiert auf den Seiten 7 und 45)

[Pfl98] C. Pflaum. A multilevel algorithm for the solution of second order elliptic Differential Equations on sparse grids. *Numerische Mathematik*, 79:141–155, 1998. (Zitiert auf den Seiten 7, 12, 17 und 45)

[Pfl10] D. Pflüger. *Spatially Adaptive Sparse Grids for High-Dimensional Problems*. Dissertation, Technische Universität München, 2010. (Zitiert auf Seite 11)

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift