Diplomarbeit

Dünngitterverfahren für hochdimensionale elliptische partielle Differentialgleichungen

Angefertigt am Institut für Numerische Simulation

Vorgelegt der

Mathematisch-Naturwissenschaftlichen Fakultät der

Rheinischen Friedrich-Wilhelms-Universität Bonn

April 2005

Von

Christian Feuersänger

aus Düsseldorf

Inhaltsverzeichnis

Kapitel 1

Einführung

Die numerische Simulation von physikalischen Prozessen ist wichtiger Bestandteil moderner Forschung und Entwicklung. Ingenieursanwendungen wie die Simulation von Strömungen oder Elastizität werden durch partielle Differentialgleichungen modelliert, welche wiederum rechnergestützt gelöst werden. Da diese im allgemeinen nicht analytisch lösbar sind, diskretisiert man klassischerweise die kontinuierlichen Funktionen auf strukturierten Gittern. Neben physikalischen Anwendungen werden zunehmend auch Problemstellungen aus dem Finanzbereich, der stochastischen Modellierung, der Simulation von Warteschlangen oder in der Quantenmechanik mithilfe von partiellen Differentialgleichungen modelliert. Diese hängen von deutlich mehr als $d = 3$ Variablen ab, die Problemstellungen sind hochdimensional.

Jedoch ist die Diskretisierung von partiellen Differentialgleichungen mit konventionellen Methoden, bedingt durch Speicheranforderungen und Berechnungskomplexität, auf drei oder vier Dimensionen beschränkt. Der Grund hierfür ist der sogenannte „Fluch der Dimensionen“: die Kosten zur Lösung eines Problems bis zu einer vorgegebenen Genauigkeit hängen exponentiell von der Dimension ab. Bei Verwendung klassischer Finite Elemente oder Finite Differenzen Verfahren mit stückweise d-polynomiellen Funktionen über einem beschränkten Gebiet auf uniformen Gittern benötigt man $N = O(N_1^d)$ Unbekannte, wobei N_1 die Zahl der Punkte in eine Koordinatenrichtung bezeichnet. Die damit erreichbare Genauigkeit hängt von der Maschenweite $O(N_1^{-1})$ ab und beträgt $O(N_1^{-\gamma})$, wobei die Ordnung γ von der Glattheit der betrachteten Funktion, dem Polynomgrad der Ansatzfunktionen und der verwendeten Norm bestimmt wird. Im Verhältnis zur Zahl der Unbekannten entspricht dies einer Genauigkeit von $O(N^{-\frac{\gamma}{d}})$. Selbst, wenn die Rechnerkapazitäten die exponentiell steigenden Kosten bewältigen könnten – das Verhältnis zur erzielbaren Genauigkeit degeneriert mit der Dimension, was die klassischen Methoden untauglich für hochdimensionale Anwendungen macht.

Hier stellen dünne Gitter eine Abhilfe dar. Dazu verwendet man eine eindimensionale Multilevelbasis, bezüglich der Funktionen mit einer Veränderlichen dargestellt werden können. Für glatte Funktionen fallen die Basiskoeffizienten mit zunehmendem Level ab. Durch die Bildung einer Tensorproduktbasis läßt sich dies auf beliebige Dimensionen übertragen. Dabei entstehen anisotrope Basisfunktionen, die in jede Richtung unterschiedliche Levels l_m aufweisen. Es kann gezeigt werden, daß die Basiskoeffizienten einer

Funktion mit beschränkten gemischten zweiten Ableitungen wie $2^{-\sum_{m=1}^{d} l_m}$ abfallen. Läßt man systematisch diejenigen Basisfunktionen weg, die in jeder Richtung ein hohes Level haben und daher kaum Beitrag leisten, so erhält man dünne Gitter. Diese haben lediglich $N = O(N_1 \log^{d-1} N_1)$ viele Punkte, erreichen damit jedoch bis auf einen Logarithmus dieselbe Approximationsgüte wie volle Gitter. Da auf diese Weise der „Fluch der Dimensionen“ zumindest teilweise gebrochen wird, stellen dünne Gitter geeignete Kandidaten zur Lösung hochdimensionaler Problemstellungen dar. Während Dünngitterverfahren beispielsweise bei der hochdimensionalen Quadratur mit Erfolg eingesetzt werden (vgl. [13, 18, 19]), sind Verfahren für hochdimensionale Differentialgleichungen nur für Spezialfälle bekannt, so etwa in [31] und [32]. In dieser Arbeit befassen wir uns mit der Entwicklung einer effizienten hochdimensionalen Galerkindiskretisierung für elliptische Dirichletrandwertprobleme zweiter Ordnung.

Seitdem dünne Gitter 1991 erstmals in [20] und [41] zur Lösung von partiellen Differentialgleichungen eingesetzt wurden, ist viel Anstrengung in die Entwicklung geeigneter Verfahren investiert worden. Bei Verwendung von Finiten Elementen mit Dünngitteransatzfunktionen liegt eine Hauptschwierigkeit in der schnellen Lösung des auftretenden linearen Gleichungssystems. Die hohe Zahl von Unbekannten setzt eine Zeitkomplexität von $O(N)$ voraus, obwohl bereits die Zahl der nicht-verschwindenden Matrixeinträge größer als $O(N)$ ist. Dies führt zur Verwendung von iterativen Gleichungssystemlösern, die lediglich die Multiplikation der Matrix mit einem Vektor benötigen. Für die dreidimensionale Poissongleichung wird erstmals in [7, 8] ein Verfahren vorgestellt, das diese Multiplikation ohne Aufstellung der Matrix durchführt und so jede Iteration in der Zeit $O(N)$ durchführen kann. Das Verfahren wird in [4] auf beliebige Dimensionen und Operatoren vom Helmholtztyp erweitert. Die Anzahl der Iterationen zum Lösen des Gleichungssystems wird in [23] durch die Verwendung eines algorithmisch einfachen Vorkonditionierers unabhängig von der Maschenweite – und damit unabhängig von N – beschränkt, was auch die Lösung des Systems in Zeit $O(N)$ möglich macht. Eine Verallgemeinerung auf Ansatzfunktionen höheren Grades wird in [9] vorgestellt. Für allgemeine elliptische partielle Differentialgleichungen wird in [14] eine Diskretisierung mithilfe stückweise linearer Ansatzfunktionen entwickelt und in [1] auf Polynome höheren Grades fortgeführt.

Ein entscheidender Nachteil jedoch macht das Verfahren für die Anwendung auf hochdimensionale Problemstellungen ungeeignet: die Koeffizienten in der $O(\cdot)$-Notation der Zeitkomplexität hängen exponentiell von der Dimension ab. Für beliebige Dimension d sind $O(2^{d+1}N)$ Operationen zur Multiplikation mit der Matrix notwendig. Ziel dieser Arbeit ist daher die Weiterentwicklung des bestehenden Verfahrens, sodaß bei gleichem Allgemeinheitsgrad derselbe Nutzen mit deutlich günstigeren Zeitkosten $O(\mathrm{Poly}(d)N)$ mit einem akzeptablem $\mathrm{Poly}(d)$ erzielt werden kann. Ähnlich wie das in [31] für den speziellen Fall der Helmholtzgleichung entwickelte Verfahren nutzen wir dazu gezielt strukturelle Eigenschaften der Matrixstruktur, wobei wir jedoch die Allgemeinheit des grundlegenden Verfahrens bewahren. Zum effizienten Lösen hochdimensionaler partieller Differentialgleichungen ist auch der sparsame Umgang mit Speicher notwendig, weshalb wir flexible, schnelle und vor allem speichersparende Datenstrukturen zur Verwaltung hochdimensionaler dünner Gitter entwickeln.

Dazu führen wir dünne Gitter in Kapitel 2 durch systematische Auswahl aus Vollgit-

terpunkten ein und assoziieren damit die stückweise lineare hierarchische Hutbasis. Neben den hierarchischen Transformationen fassen wir die Interpolationsfehlerabschätzungen aus der Literatur zusammen. Vorbereitend für die Diskretisierung in Kapitel 3 definieren wir die hierarchische Prewaveletbasis, die in dieser Arbeit eine wesentliche Rolle bei der Reduktion der Laufzeitkomplexität spielt.

In Kapitel 3 führen wir die Galerkindiskretisierung elliptischer partieller Differentialgleichungen durch, indem wir das in [4] eingeführte grundlegende Verfahren anwenden. Dieses setzt die Multiplikation der Steifigkeitsmatrix mit einem Vektor rekursiv aus eindimensionalen Teilalgorithmen zusammen. Neben der detaillierten Herleitung der Rekursion untersuchen wir Zeit- und Speicherkosten und leiten daraus unter Einbeziehung weiteren Wissens über die Struktur der Differentialgleichungen Erweiterungen ab, die die Laufzeit signifikant reduzieren. Dazu benötigen wir Ansatzfunktionen mit abgeschwächten Orthogonalitätseigenschaften bezüglich des L_2-Produktes, wofür wir hier die hierarchische Prewaveletbasis verwenden.

Kapitel 4 widmet sich einer Analyse der bekannten Datenstrukturen zur Verwaltung dünner Gitter und führt eine neue Datenstruktur ein, die schnellen Zugriff mit geringstmöglichem Speicheraufwand kombiniert.

Das Verfahren wird in Kapitel 5 vervollständigt, indem wir anhand der Konditionierung der Steifigkeitsmatrix die Anzahl der Iterationen eines iterativen Lösers von linearen Gleichungssystemen untersuchen. Hier spielt der aus [23] für die Prewaveletbasis bekannte Vorkonditionierer eine entscheidende Rolle. Des weiteren untersuchen wir die Approximationsgüte der diskreten Lösung anhand einiger Modellprobleme. Der Kernpunkt ist dabei die Abschätzung der Verfahrensordnung. Mithilfe eines im Rahmen dieser Arbeit entwickelten C++-Verfahrens untersuchen wir, inwieweit die Ordnung des Verfahrens durch den Interpolationsfehler abschätzbar ist. In dieser Arbeit betrachten wir dabei erstmalig auch den Einfluss der Dimension auf die Abschätzung.

Kapitel 6 fasst die gesammelten Erkenntnisse in einem Fazit zusammen und zeigt einen Ausblick auf weitere Forschungen.

An dieser Stelle möchte ich meinen Dank aussprechen. Mein Dank gilt Gott für mein Leben, die Befähigung zu kreativer Entwicklung und die vielen Ideen. Besonders bedanke ich mich bei Prof. Dr. Michael Griebel für die Überlassung des Themas, die fruchtbringenden, richtungsweisenden Diskussionen und die hervorragenden Arbeitsbedingungen am Institut. Ich bedanke mich bei Prof. Dr. Michael Clausen für die Übernahme des Zweitgutachtens. Weiterhin danke ich Daniel Oeltz für die gewissenhafte und vorbildliche Betreuung, die bereichernden Diskussionen sowie die vielen konstruktiven Anregungen bei der Durchsicht des Manuskripts. Zudem gilt mein Dank Thomas Mertens für den wertvollen Austausch von Ideen während der Arbeit an unseren Diplomen. Mein Dank gebührt auch den Institutsangehörigen im allgemeinen und meinen Bürokollegen im besonderen für die erfrischend fröhliche und angenehme Arbeitsatmosphäre, die Hilfsbereitschaft und das gute Klima. Schliesslich bedanke ich mich bei meiner Frau Kerstin für die Ermutigung bei Rückschlägen und das Mitfreuen bei Erfolgen und damit für die moralische Unterstützung.

Kapitel 2

Grundlagen

In den folgenden Abschnitten führen wir dünne Gitter sowie die Interpolation mit hierarchischen Basen auf dünnen Gittern ein. Der Schwerpunkt liegt dabei auf der Darstellung dünner Gitter als strukturierte Punktmengen mit hierarchischen Bezeichnungsweisen und -Relationen für Punkte, was Inhalt von Abschnitt 2.1 ist. In Abschnitt 2.2 assoziieren wir stückweise lineare Basisfunktionen mit den Gitterpunkten. Die hierarchische Gitterstruktur führt uns damit zu hierarchischen Basen, mit denen die Approximation von Funktionen möglich ist.

2.1 Dünne Gitter

Zur Einführung dünner Gitter beginnen wir mit dem eindimensionalen Fall und führen eine hierarchische Bezeichnungsweise für Gitterpunkte aus dem Innern des Einheitsintervalls $(0, 1)$ ein. Diese hierarchische Sichtweise erweitern wir auf den d-dimensionalen Einheitswürfel $(0, 1)^d$ und definieren dünne Gitter als Menge aller Punkte bis zu einem speziell ausgewählten hierarchischen Level. Schliesslich erweitern wir dünne Gitter auf den Rand des Einheitswürfels rekursiv, indem wir auf einer m-dimensionalen Untermannigfaltigkeit des Randes (beispielsweise Flächen, Kanten, etc.) die Punkte eines m-dimensionalen dünnen Gitters einfügen.

2.1.1 Innere Punkte

Wir gehen aus von einem vollen Gitter in Dimension $d = 1$ mit Machenweite $h_l := \frac{1}{2^l}$, $l \in \mathbb{N}$ und betrachten nur innere Punkte. Die Punkte des Gitters bezeichnen wir mit

$$x_{l,i} := i \cdot h_l,$$

mit $i = 1, \ldots, 2^l - 1$. Wir bezeichnen l als „Level“ des Gitters und i als „Ort“ von $x_{l,i}$. Wir bezeichnen ferner die Menge der zugehörigen Indizes mit

$$V_l := \{(l, i) | i = 1, \ldots, 2^l - 1\}$$

und schreiben $x \bar{\in} V_l$, falls es in V_l Indizes (l, i) gibt mit $x_{l,i} = x$.

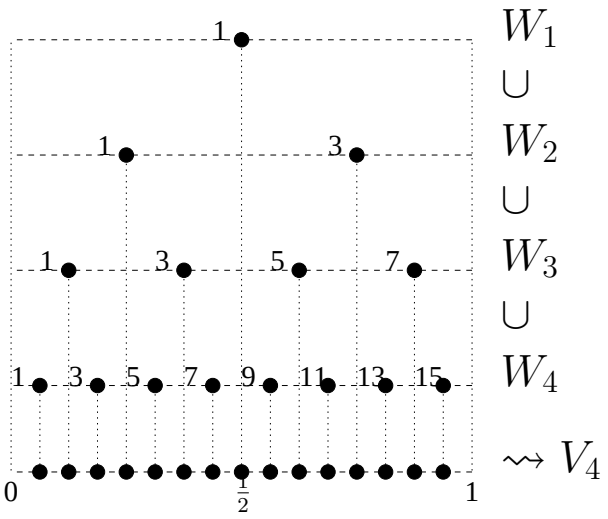

Abbildung 2.1: Darstellung eines eindimensionalen vollen Gitters durch eine hierarchische Sichtweise

Nun lassen sich die zu V_l gehörenden Punkte auch alternativ durch eine hierarchische Sichtweise beschreiben, wenn wir die Unterschiede von einem Level l auf das nächste durch Überschüsse beschreiben, d. h. wir interessieren uns für die Punkte $x_{l,i} \in V_l$, die noch nicht durch V_{l-1} beschrieben wurden. Wir schreiben

$$W_l := \{(l,i) \in V_l | x_{l,i} \bar{\notin} V_{l-1}\}.$$

Definieren wir noch $W_1 := V_1$, so können wir alle Punkte $x \bar{\in} V_l$ auch mittels der disjunkten Zerlegung

$$G_l^1 := \bigcup_{k=1}^{l} W_k$$

beschreiben. Dies veranschaulichen wir anhand von Abbildung 2.1. Die Überschüsse W_l sind die ungeraden Ortsindizes von V_l. Durch die Vereinigung der dargestellten Überschussmengen erhalten wir die Multiindizes

$$\{(1,1),(2,1),(2,3),(3,1),(3,3),\ldots,(4,15)\},$$

deren zugehörige Punktkoordinaten $x_{l,i}$ auch durch V_4 beschrieben wurden, d. h. $x_{l,i} \bar{\in} V_4$. Im Eindimensionalen ist dies nichts anderes als ein Binärbaum mit Wurzel $(1,1)$ sowie dem linken Sohn $(l+1, 2i-1)$ und dem rechten Sohn $(l+1, 2i+1)$ auf den entsprechenden Levels. Um die hierarchischen Relationen zwischen Punkten $(l,i),(k,j) \in G_n^1$ eines dünnen Gitters zu charakterisieren, führen wir an dieser Stelle einige Bezeichnungen ein. Von besonderem Interesse für die noch folgende Algorithmik ist, ob ein Punkt ein „Nachfahre“ oder ein „Vorfahre“ eines anderen ist. Ein Vorfahre von (l,i) ist dabei ein Punkt, der auf dem eindeutigen Pfad zur Wurzel des Baumes liegt. Entsprechend ist ein Nachfahre eines Punktes (l,i) dadurch definiert, daß er auf einem Pfad von (l,i) zu einem Blatt

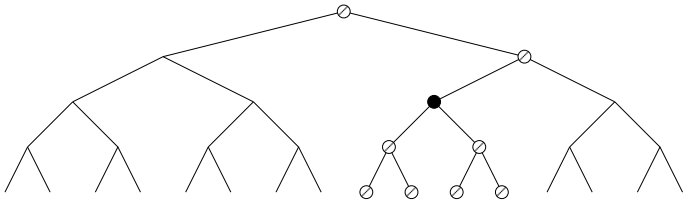

Abbildung 2.2: Hierarchische Relationen im Eindimensionalen. Markiert sind alle Vor- und Nachfahren des schwarz markierten Punktes.

des Baumes liegt. Wir veranschaulichen die Definition anhand von Abbildung 2.2. Zu dem eingezeichneten Punkt (l, i) sind oberhalb alle Vorfahren markiert und unterhalb alle Nachfahren.

Die so gewonnene Darstellung lässt sich auf mehrdimensionale volle Gitter erweitern, wobei wir nun verschiedene Levels l_m pro Richtung $m \in \{1, \ldots, d\}$ erlauben. Die Verallgemeinerung des Begriffs des „Levels“ für die d-dimensionale Konstruktion verschieben wir dabei zunächst auf später. Zu einem d-dimensionalen Levelindex $\mathbf{l} = (l_1, \ldots, l_d)$ definieren wir in Analogie zum eindimensionalen Fall

$$V_{\mathbf{l}} := \{(\mathbf{l}, \mathbf{i}) | \mathbf{i} := (i_1, \ldots, i_d), i_m = 1, \ldots, 2^{l_m} - 1 \text{ für alle } m = 1, \ldots, d\}.$$

Um die so beschriebenen Punkte $x_{\mathbf{l},\mathbf{i}} := (x_{l_1,i_1}, \ldots, x_{l_d,i_d})^T$ wieder hierarchisch zu betrachten gehen wir nun koordinatenweise vor. Wir streben dabei an, daß alle Punkte auf einem beliebigen Strahl parallel zu einer der Achsen durch ein volles Gitter (d. h. auf einer Linie des Gitters) die eindimensionale hierarchische Struktur widerspiegeln. In *jeder* Richtung m soll dazu ein mehrdimensionaler Überschussraum mit Level $\mathbf{l}$ ausschliesslich die Punkte enthalten, die in keinem der $V_{\mathbf{k}}$ mit $k_m < l_m$ vorkommen. Dazu definieren wir

$$W_{\mathbf{l}} := \{(\mathbf{l}, \mathbf{i}) \in V_{\mathbf{l}} | \text{ für alle } m = 1, \ldots, d \text{ mit } l_m > 1 : x_{\mathbf{l},\mathbf{i}} \bar{\notin} V_{\mathbf{l}-\mathbf{e}_m}\}.$$

Dabei sei $\mathbf{e}_m$ der m.te Einheitsvektor in d Dimensionen. Wiederum enthält $W_{\mathbf{l}}$ ausschliesslich die ungeraden Indizes aus $V_{\mathbf{l}}$,

$$W_{\mathbf{l}} = \{(\mathbf{l}, \mathbf{i}) \in V_{\mathbf{l}} | i_m \text{ ungerade für alle } m = 1, \ldots, d\}.$$

Der Einführung im Eindimensionalen folgend können wir so wieder alle Punkte eines vollen Gitters $V_{\mathbf{l}}$ eindeutig beschreiben mittels einem Level $\mathbf{l}$ und einem Ort $\mathbf{i}$. Mit der Definition $W_{\mathbf{1}} := V_{\mathbf{1}}$ beschreibt die – wiederum disjunkte – Zerlegung

$$G_{\mathbf{l}}^d := \bigcup_{k_1=1}^{l_1} \cdots \bigcup_{k_d=1}^{l_d} W_{k_1,\ldots,k_d}$$

genau dieselben Punkte wie $V_{\mathbf{l}}$. Dies veranschaulichen wir mittels Abbildung 2.3. Der Vollgitterraum $V_{3,2}$ ist eindeutig zerlegbar in die markierten Überschussräume $W_{\mathbf{k}}$. Betrachten wir in diesem Beispiel alle Punkte $x_{\mathbf{l},\mathbf{i}}$ mit $l_2 = 2$ und $i_2 = 1$ (d. h. $x_2 = \frac{1}{4}$), so liegen diese auf einer Geraden parallel zur x_1-Achse. Sie bilden eine Gitterlinie, für die

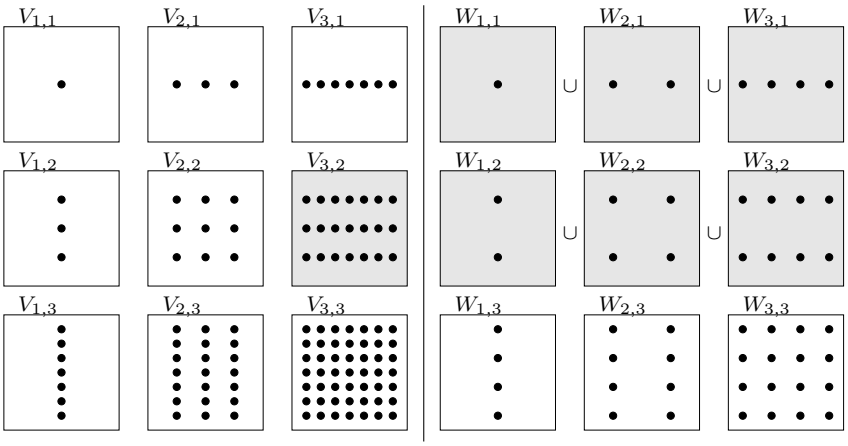

Abbildung 2.3: Darstellung eines zweidimensionalen vollen Gitters mit unterschiedlichen Maschenweiten in jede Richtung durch die hierarchische Sichtweise. Linkerhand sehen wir die verschiedenen Vollgitterräume $V_{\mathbf{l}}$ und rechterhand die Überschussräume $W_{\mathbf{l}}$. Die rechts markierten Räume beschreiben dieselben Punkte wie der links markierte $V_{3,2}$.

per Konstruktion die eindimensionale Sichtweise gilt und formen daher einen Binärbaum. Die Betrachtung einzelner solcher Linien durch das Gitter spielt bei vielen Algorithmen – ob auf dünnen Gittern oder auf vollen – eine wichtige Rolle und wird uns in Kapitel 3 eingehend beschäftigen.

Wir kommen nun auf die Definition des „Levels“ eines Gitters zurück. Im eindimensionalen Fall hatten wir das Level des Gitters als den höchsten vorkommenden Levelindex l definiert. Im Mehrdimensionalen gibt es nun d Richtungen mit jeweils unterschiedlichen Maschenweiten und Levels. Wir definieren nun das Level des *vollen* Gitters $G^d_{n,\dots,n}$ als n. Abkürzend schreiben wir dafür

$$\bar{G}^d_n := G^d_{n,\dots,n} = \bigcup_{1 \le |\mathbf{l}|_\infty \le n} W_{\mathbf{l}}.$$

Verwenden wir zur Auswahl der Überschussräume die $|\cdot|_1$-Norm für die Levelindizes $\mathbf{l}$, so erhalten wir das reguläre *dünne* Gitter in Dimension d mit Level n. Damit auch im Mehrdimensionalen der Punkt in $W_{\mathbf{1}}$ auf Level 1 bleibt, definieren wir

$$G^d_n := \bigcup_{1 \le |\mathbf{l}-\mathbf{1}|_1 + 1 \le n} W_{\mathbf{l}}. \tag{2.1}$$

Dies entspricht denjenigen Levelindizes mit $|\mathbf{l}|_1 - d + 1 \le n$. Entsprechend definieren wir

$$n(\mathbf{l}) := |\mathbf{l}|_1 - d + 1$$

als das Level von $\mathbf{l}$ in einem dünnen Gitter. In Abbildung 2.3 finden sich die entsprechenden $W_{\mathbf{l}}$ unterhalb der Diagonalen. Abbildung 2.4 zeigt das Gitter G^2_3.

Wir definieren nun entsprechend den auf Seite 6 im Eindimensionalen eingeführten hierarchischen Relationen auch für Punkte eines mehrdimensionalen Gitters hierarchische

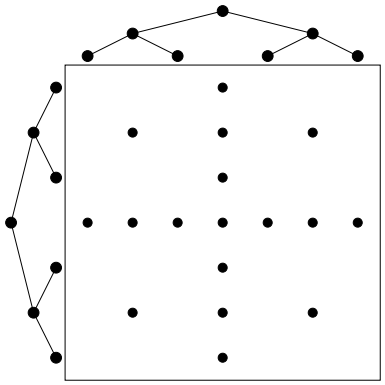

Abbildung 2.4: Das reguläre dünne Gitter in Dimension 2 mit Level $n = 3$.

Relationen. Ein Punkt $(\mathbf{l}, \mathbf{i})$ ist dabei Vorfahr (Nachfahre) von $(\mathbf{k}, \mathbf{j})$, wenn diese Relation für *alle* Richtungen $m \in \{1, \ldots, d\}$ gilt. Formal führen wir dazu die hierarchischen Relationen

$$
\begin{aligned}
(\mathbf{l}, \mathbf{i}) \underset{H}{>} (\mathbf{k}, \mathbf{j}) &:\Leftrightarrow \forall m \text{ ist } (l_m, i_m) \text{ Vorfahre von } (k_m, j_m), \\
(\mathbf{l}, \mathbf{i}) \underset{H}{<} (\mathbf{k}, \mathbf{j}) &:\Leftrightarrow \forall m \text{ gilt:} \\
&\qquad (l_m, i_m) = (k_m, j_m) \text{ oder } (l_m, i_m) \text{ ist Nachfahre von } (k_m, j_m), \\
(\mathbf{l}, \mathbf{i}) \underset{H}{\leq} (\mathbf{k}, \mathbf{j}) &:\Leftrightarrow (\mathbf{l}, \mathbf{i}) \underset{H}{>} (\mathbf{k}, \mathbf{j}) \text{ oder } (\mathbf{l}, \mathbf{i}) \underset{H}{<} (\mathbf{k}, \mathbf{j})
\end{aligned}
$$

ein. Man beachte, daß damit keine vollständige Ordnung definiert ist, sondern lediglich hierarchische Relationen angegeben werden.

2.1.2 Randpunkte

Um die bislang lediglich für innere Punkte definierten vollen Gitter $\bar{G}_n^d$ und dünnen Gitter G_n^d auch auf Randpunkte zu verallgemeinern, bedienen wir uns einer rekursiven Definition. Sei $\tilde{G}_n^d \in \{G_n^d, \bar{G}_n^d\}$. An dieser Stelle definieren wir dann $\underline{\tilde{G}}_n^d$ als das um Randpunkte erweiterte Gitter mithilfe der folgenden Rekursion. Dazu bezeichnen wir zu gegebenem Multiindex $(\mathbf{l}, \mathbf{i})$ mit $(\bar{\mathbf{l}}^m, \bar{\mathbf{i}}^m)$ denjenigen Multiindex, der aus Streichung der m-ten Komponente von $(\mathbf{l}, \mathbf{i})$ hervorgeht. Wir definieren also die Erweiterung von $\tilde{G}_n^d$ um Randpunkte als

$$
\begin{aligned}
\underline{\tilde{G}}_n^d := \tilde{G}_n^d \cup \bigcup_{m=1}^{d} &\{(\mathbf{l}, \mathbf{i}) | l_m = 0, i_m = 0, (\bar{\mathbf{l}}^m, \bar{\mathbf{i}}^m) \in \underline{\tilde{G}}_n^{d-1}\} \\
&\cup \{(\mathbf{l}, \mathbf{i}) | l_m = 0, i_m = 1, (\bar{\mathbf{l}}^m, \bar{\mathbf{i}}^m) \in \underline{\tilde{G}}_n^{d-1}\}
\end{aligned}
$$

mit dem Abschluß der Rekursion in Dimension $d = 1$,

$$
\underline{\tilde{G}}_n^1 := \tilde{G}_n^1 \cup \{(0, 0), (0, 1)\}.
$$

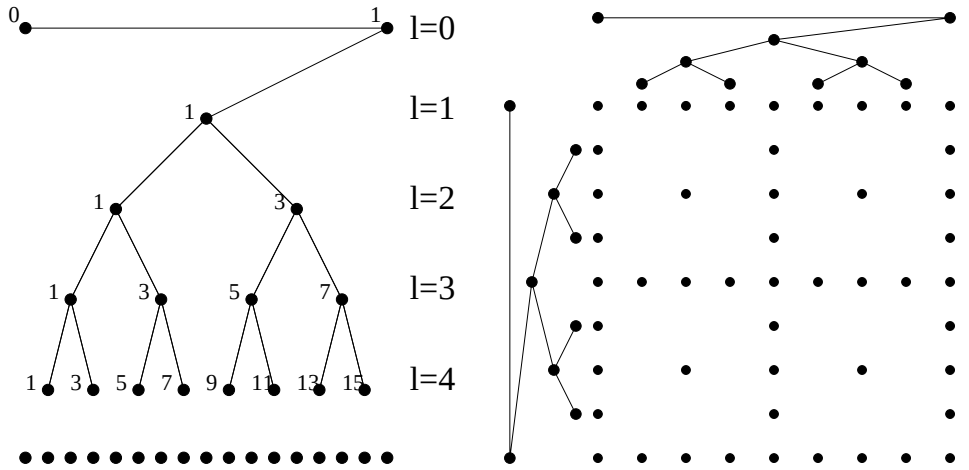

Abbildung 2.5: Dünne Gitter mit Randpunkten in Dimensionen $d = 1$ und $d = 2$.

Damit erhalten wir im Eindimensionalen lediglich die zwei Randpunkte des Intervalls $[0, 1]$ zusätzlich. In Dimension $d = 2$ erhalten wir sowohl für volle als auch für dünne Gitter zusätzlich die vier Eckpunkte von $[0, 1]^2$ sowie auf den Gebietsrändern $x_1 \in \{0, 1\}$ und $x_2 \in \{0, 1\}$ eindimensionale volle Gitter. In Dimension $d = 3$ unterscheiden sich dünne und volle Gitter auch auf dem Rand, denn nun ist jede der sechs Seitenflächen wiederum eine zweidimensionale dünne beziehungsweise volle Gitterstruktur.

Dies entspricht für volle Gitter nichts anderem als der Hinzunahme aller Überschussräume, für die $l_m = 0$ erlaubt ist – ansonsten ändert sich hier nichts. Für dünne Gitter ist ein Schema wie in Abbildung 2.3 zwar auch unter Einbeziehung der Randpunkte möglich, erlaubt jedoch aufgrund der Rekursion nicht mehr die Angabe einer einfachen Auswahlnorm wie in (2.1). Abbildung 2.5 illustriert die Hinzunahme der Randpunkte für $d = 1$ und $d = 2$. Die bereits erwähnte Binärbaumstruktur ergänzen wir dazu um zwei Randknoten, die wir über der eigentlichen Wurzel eintragen.

Für Randpunkte ergänzen wir die Definition des Levels eines Punktes mit Levelindex $\mathbf{l}$. Ist $\mathbf{l} = \mathbf{0}$, so ist das Level ebenfalls 0, ansonsten verwenden wir das Level $n(\tilde{l})$, wobei $\tilde{l}$ lediglich die nicht-verschwindenden Komponenten enthält. Formal schreiben wir

$$K(\mathbf{l}) := |\{m | l_m = 0\}|, \tag{2.2}$$

$$n(\mathbf{l}) := \sum_{\substack{m=1,\ldots,d \\ l_m \neq 0}} (l_m - 1) + 1 = |\mathbf{l}|_1 - \big(d - K(\mathbf{l})\big) + 1, \tag{2.3}$$

sowie der Vollständigkeit halber

$$n(\mathbf{0}) := 0. \tag{2.4}$$

2.1.3 Komplexität

Es stellt sich die Frage, wieviele Punkte die Gitter $\bar{G}_n^d$ beziehungsweise G_n^d enthalten. Im Eindimensionalen sind die beiden Mengen gleich mächtig und enthalten $O(N_1)$ mit

$N_1 := 2^n$ viele Elemente. Für volle Gitter multipliziert sich dies auf und wir erhalten

$$|\bar{G}_n^d| = O(N_1^d) = O(2^{n \cdot d})$$

Elemente in d Dimensionen. Das in der Dimension exponentiell steigende Wachstum verhindert allein durch den enormen Speicherverbrauch die Verwaltung entsprechender Gitter bereits ab $d = 4$.

Dünne Gitter benötigen substanziell weniger Punkte als volle Gitter. In Dimension $d > 1$ erhält man die Zahl der Punkte durch die Rekursionsvorschrift

$$|G_n^d| = \sum_{k=0}^{n-1} 2^k |G_{n-i}^{d-1}| \tag{2.5}$$

ohne Randpunkte, sowie mit Randpunkten

$$|G_n^d| = 3 \cdot |G_n^{d-1}| + \sum_{k=1}^{n-1} 2^k |G_{n-i}^{d-1}|. \tag{2.6}$$

Für detaillierte Untersuchungen und Beweise zu der Komplexität dünner Gitter sei auf [8, 9] verwiesen. Asymptotisch entspricht dies (bei konstanter Dimension)

$$|G_n^d| = O(N_1 \cdot \log^{d-1} N_1) = O(2^n n^{d-1}).$$

Die exponentielle Abhängigkeit von der Dimension verschwindet somit in einem Logarithmus. Dieses asymptotische Verhalten der Zahl benötigter Punkte macht dünne Gitter zu Kandidaten für Löser von hochdimensionalen Problemen. Wie wir in Abschnitt 2.2 untersuchen werden, entspricht die Interpolationsgenauigkeit mit hierarchischen Basen trotz der drastisch reduzierten Zahl der Punkte bis auf einen Logarithmus der Interpolationsgenauigkeit auf vollen Gittern. Aus diesem Grund werden wir zur Lösung von hochdimensionalen partiellen Differentialgleichungen dünngitterbasierte Verfahren entwickeln.

An dieser Stelle sei erwähnt, daß die Zahl der Punkte in einem dünnen Gitter durch eine auf der Energienorm basierten Auswahl von Überschussräumen $W_\mathbf{l}$ auf $O(N_1)$ reduziert werden kann. Da diese Form jedoch lediglich in der Energienorm gute Interpolationseigenschaften aufweist, nutzen wir hier ausschliesslich das eingeführte Gitter G_n^d. Für energienorm-basierte Gitter sei auf [4, 8, 27, 28] u. a. verwiesen.

2.2 Hierarchische Basen auf dünnen Gittern

Ziel dieses Abschnittes ist es, Funktionen $\mathbf{u}\colon [0,1]^d \to \mathbb{R}$ geeignet durch einen Interpolanten auf dem dünnen Gitter G_n^d zu approximieren. Dabei betrachten wir zunächst die hierarchische Hutbasis, die eng verwandt ist mit der klassischen stückweise linearen Finite-Element-Basis, insbesondere bei Multilevelmethoden, siehe [21] und die darin enthaltenen Zitate. Der von der hierarchischen Hutbasis auf dünnen Gittern aufgespannte Raum wird u. a. in [4, 8, 14] als Ansatz- und Testraum für ein Galerkinverfahren verwendet und in Folgearbeiten auf Polynome höheren Grades verallgemeinert, siehe beispielsweise [1, 9]. Wie wir

in Kapitel 3 sehen werden, eignen diese sich nicht für Löser hochdimensionaler Problemstellungen, weswegen wir hier die aus Hutfunktionen zusammengesetzte Prewaveletbasis untersuchen. Diese wird bereits in [31] zur Lösung der Helmholtzgleichung herangezogen und hat durch Semiorthogonalität und die Verfügbarkeit von darauf basierenden Vorkonditionierern (siehe dazu [23]) algorithmische Vorteile.

Zu Beginn definieren wir noch einige grundlegende Begriffe. Wir betrachten im Folgenden Funktionen mit d Veränderlichen, $\mathbf{u}\colon [0,1]^d \to \mathbb{R}$, mit existierenden gemischten Ableitungen

$$D^\alpha \mathbf{u} := \frac{\partial^{|\alpha|_1}\mathbf{u}}{\partial^{\alpha_1}\dots\partial^{\alpha_d}}$$

bis zu einem gewissen Grad $r > 0$. Dabei ist $\alpha = (\alpha_1, \dots, \alpha_d) \in \mathbb{N}_0^d$ ein d-dimensionaler Index. Mit

$$|\alpha|_1 = \sum_{m=1}^{d} \alpha_m$$

bezeichnen wir wie zuvor die Summe der Einträge und mit

$$|\alpha|_\infty := \max_{m \le d} \alpha_m$$

den größten Eintrag. Uns interessieren insbesondere Funktionen, deren gemischte Ableitungen bis zur zweiten Ordnung existieren und stetig sind. Wir definieren daher den Raum

$$X(\Omega) := \{\mathbf{u}\colon \Omega \to \mathbb{R} | D^\alpha \mathbf{u} \in C^0(\Omega), |\alpha|_\infty \le 2\}$$

sowie den Teilraum von Funktionen, die auf dem Rand $\Gamma := \partial\Omega$ verschwinden,

$$X_0(\Omega) := \{\mathbf{u} \in X(\Omega) | \mathbf{u}|_\Gamma = 0\}.$$

Für $\mathbf{u} \in X_0(\Omega)$ und α mit $|\alpha|_\infty \le 2$ führen wir zudem die Seminormen

$$|\mathbf{u}|_{\alpha,\infty} := \|D^\alpha \mathbf{u}\|_\infty,$$
$$|\mathbf{u}|_{\alpha,2} := \|D^\alpha \mathbf{u}\|_2 = \sqrt{\int_\Omega |D^\alpha \mathbf{u}|^2 dx}$$

ein.

2.2.1 Die hierarchische Hutbasis

Die hierarchische Hutbasis entsteht aus der Translation und Stauchung der eindimensionalen Funktion $\max\{0, 1 - |x|\}$. Wir assoziieren mit jedem (l,i) die stückweise lineare Funktion $\phi_{l,i}$, welche an $x_{l,i}$ den Wert 1 annimmt und linear zu den Nachbarn $(l, i \pm 1)$ auf 0 abfällt. Außerhalb dieser Intervalle ist $\phi_{l,i}$ konstant 0. Formal definieren wir

$$\phi_{l,i}(x) := \begin{cases} \frac{1}{h_l} \cdot x + 1 - \frac{x_{l,i}}{h_l} & \text{für } x \in [x_{l,i} - h_l, x_{l,i}) \\ -\frac{1}{h_l} \cdot x + 1 + \frac{x_{l,i}}{h_l} & \text{für } x \in [x_{l,i} + h_l, x_{l,i}] \\ 0 & \text{sonst.} \end{cases}$$

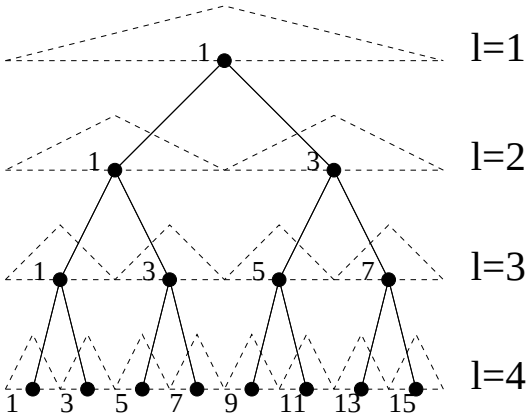

Abbildung 2.6: Die eindimensionale hierarchische Hutbasis Φ_4, dargestellt zusammen mit den Gitterpunkten aus G_4^1.

Die Funktionen $\{\phi_{l,i}|(l,i) \in V_n\}$ bilden die bekannte eindimensionale, stückweise lineare *nodale* Basis mit Maschenweite h_n. Die grundlegende Idee bei der Einführung von hierarchischen Basen ist, anstelle der nodalen Basis Funktionen mit unterschiedlich breiten Trägern zur Darstellung heranzuziehen. Mithilfe der im vorigen Abschnitt eingeführten Indexmengen W_l lassen sich diese handhaben. Wir definieren die hierarchische Hutbasis auf dem dünnen Gitter G_n^1 mittels

$$\Phi_n^1 := \{\phi_{l,i}|(l,i) \in G_n^1\}.$$

Die entsprechenden Basisfunktionen sind für $n = 4$ in Abbildung 2.6 dargestellt. Ebenso wie mit der nodalen Basis ist es so möglich, eine Funktion $\mathbf{u}$ durch ihren Dünngitterinterpolanten

$$u(x) = I_{G_n^1}\mathbf{u}(x) = \sum_{(l,i)\in G_n^1} u_{l,i}\phi_{l,i}(x) \tag{2.7}$$

zu approximieren. Die Koeffizienten $u_{l,i}$ auf einem Level $n(l) = r \leq n$ beschreiben dabei genau die Differenz der kontinuierlichen Funktion $\mathbf{u}$ zu dem Interpolanten bis einschließlich Level $r-1$ an der Stelle $x_{l,i}$,

$$u_{l,i} = \mathbf{u}(x_{l,i}) - I_{G_{r-1}^1}\mathbf{u}(x_{l,i}) \tag{2.8}$$

mit Knotenwerten auf dem gröbsten Level,

$$u_{1,1} = \mathbf{u}(x_{1,1})$$

(beziehungsweise für Level $l = 0$ falls $\mathbf{u}_\Gamma \neq 0$). Wir bezeichnen die Koeffizienten $u_{\mathbf{l},\mathbf{i}}$ als „hierarchische Koeffizienten“ oder auch „hierarchische Überschüsse“.

Für die Dimension d verallgemeinern wir nun die eindimensionalen Funktionen $\phi_{l,i}$ mithilfe von Tensorprodukten, die die Einträge aus Multiindizes $(\mathbf{l},\mathbf{i})$ nutzen. Wir schreiben

$$\phi_{\mathbf{l},\mathbf{i}}(x) := \prod_{m=1}^{d} \phi_{l_m,i_m}(x_m).$$

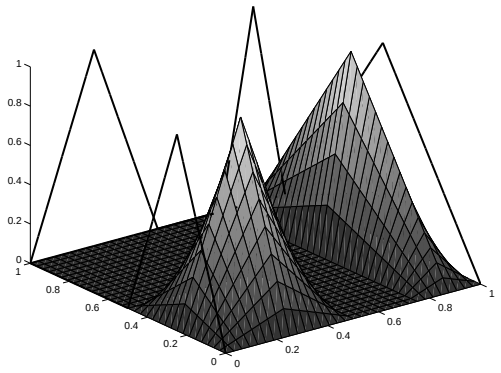

Abbildung 2.7: Die Basisfunktionen der hierarchischen Hutbasis $\phi_{(3,1|7,1)} = \phi_{3,7} \cdot \phi_{1,1}$ und $\phi_{(2,2|1,1)} = \phi_{2,1} \cdot \phi_{2,1}$.

Abbildung 2.7 zeigt als Beispiel die zu den Punkten $(\mathbf{l}, \mathbf{i}) = (2, 2|1, 1)$ und $(\mathbf{l}, \mathbf{i}) = (3, 1|7, 1)$ gehörenden Basisfunktionen. Analog zum eindimensionalen Fall definieren wir die hierarchische Hutbasis auf dem dünnen Gitter G_n^d mittels

$$\Phi_n^d := \{\phi_{\mathbf{l},\mathbf{i}} | (\mathbf{l}, \mathbf{i}) \in G_n^d\}.$$

Damit lassen sich Funktionen von d Veränderlichen durch ihren Dünngitterinterpolanten

$$u(x) = I_{G_n^d} \mathbf{u}(x) = \sum_{(\mathbf{l},\mathbf{i}) \in G_n^d} u_{\mathbf{l},\mathbf{i}} \phi_{\mathbf{l},\mathbf{i}}(x)$$

approximieren, wobei die hierarchischen Überschüsse $u_{\mathbf{l},\mathbf{i}}$ aus der Verallgemeinerung der Differenz (2.8) entstehen.

Interpolationsfehler

Hierarchische Basen auf dünnen Gittern (nicht nur die hierarchische Hutbasis) haben eine entscheidende Interpolationseigenschaft: Für Funktionen $\mathbf{u} \in X(\Omega)$ mit beschränkten gemischten Ableitungen bis zur zweiten Ordnung fallen die hierarchischen Koeffizienten mit zunehmendem Level sehr stark gegen 0 hin ab, vgl. [9, 11]. Daher ist es möglich, glatte Funktionen mit der vorteilhaft geringen Zahl der Gitterpunkte gut zu interpolieren.

Dazu geben wir die Resultate aus [9] wieder. Für Dimension d und Level n definieren wir die Hilfsgröße

$$A(d, n) := \sum_{m=0}^{d-1} \binom{n+d-1}{k}.$$

Für $\mathbf{u} \in X_0(\Omega)$ gilt für den Dünngitterinterpolationsfehler

$$\|\mathbf{u} - I_{G_n^d}\mathbf{u}\|_{L_\infty} \leq \frac{2 \cdot |\mathbf{u}|_{\mathbf{2},\infty}}{8^d} \cdot 2^{-2n} \cdot A(d,n), \tag{2.9}$$

$$\|\mathbf{u} - I_{G_n^d}\mathbf{u}\|_{L_2} \leq \frac{2 \cdot |\mathbf{u}|_{\mathbf{2},2}}{12^d} \cdot 2^{-2n} \cdot A(d,n), \tag{2.10}$$

$$\|\mathbf{u} - I_{G_n^d}\mathbf{u}\|_a \leq \frac{d \cdot |\mathbf{u}|_{\mathbf{2},\infty}}{2 \cdot 3^{(d-1)/2} \cdot 4^{d-1}} \cdot 2^{-n}, \tag{2.11}$$

wobei $\mathbf{2} = (2, \ldots, 2) \in \mathbb{N}^d$ ein Index für die in jede Richtung gemischten zweiten Ableitungen ist. Dabei ist

$$\|\mathbf{u}\|_a := \sqrt{\sum_{m=1}^{d} \int_\Omega (\frac{\partial \mathbf{u}}{\partial x_m})^2 dx}$$

die Energienorm.

Oftmals betrachtet man lediglich die Asymptotik $n \to \infty$ (d. h. $h \to 0$) und behandelt die Dimension als Konstante. In diesem Fall erhält man mit

$$A(d,n) = \frac{n^{d-1}}{(d-1)!} + O(n^{d-2})$$

die Abschätzungen

$$\|\mathbf{u} - I_{G_n^d}\mathbf{u}\|_{L_2} = O(2^{-2n} n^{d-1}), \tag{2.12}$$

$$\|\mathbf{u} - I_{G_n^d}\mathbf{u}\|_{L_\infty} = O(2^{-2n} n^{d-1}) \tag{2.13}$$

$$\tag{2.14}$$

sowie

$$\|\mathbf{u} - I_{G_n^d}\mathbf{u}\|_a = O(2^{-n}).$$

Äquivalent erhalten wir mit $h := 2^{-n} = N_1^{-1}$ für feste Dimension

$$\|\mathbf{u} - I_{G_n^d}\mathbf{u}\|_{L_2} = O(h^2 \log^{d-1} \frac{1}{h}),$$
$$\|\mathbf{u} - I_{G_n^d}\mathbf{u}\|_{L_\infty} = O(h^2 \log^{d-1} \frac{1}{h}),$$

bzw.

$$\|\mathbf{u} - I_{G_n^d}\mathbf{u}\|_a = O(h).$$

Damit haben dünne Gitter eine hohe Interpolationsgüte, denn sie weisen (bis auf einen Logarithmus) dieselbe Fehlerordnung wie volle Gitter auf, haben jedoch substantiell weniger Punkte. Somit erfahren wir ein vorteilhaftes Kosten/Nutzen-Verhältnis: bei Verwendung von lediglich $O(2^n n^{d-1})$ Punkten erhalten wir einen Interpolationsfehler von nahezu der gleichen Ordnung wie bei $O(2^{n \cdot d})$ vielen Punkten. Das Verhältnis der zu erlangenden Kosten und den dazu notwendigen Kosten lässt sich mithilfe der ϵ-Komplexität beschreiben,

siehe dazu [11] und die darin enthaltenen Zitate. Für die Interpolation auf vollen Gittern bedeutet dies, dass mit N Gitterpunkten eine Genauigkeit von $\epsilon_M(N)$ erreicht werden kann, wobei wir für die Normen $M \in \{L_2, L_\infty, a\}$ die Beziehungen

$$\begin{aligned} \epsilon_{L_2}(N), \epsilon_{L_\infty}(N) &= O(N^{-\frac{d}{2}}), \\ \epsilon_a(N) &= O(N^{-d}) \end{aligned}$$

aus [9, 11] entnehmen. Für dünne Gitter wird die exponentielle Abhängigkeit von der Dimension bedeutend reduziert: Mit $N = |G_n^d|$ Gitterpunkten lassen sich die Interpolationsfehlerabschätzungen

$$\begin{aligned} \epsilon_{L_2}(N), \epsilon_{L_\infty}(N) &= O(N^{-2} \cdot |\log_2 N|^{3\cdot(d-1)}), \\ \epsilon_a(N) &= O(N^{-1} |\log_2 N|^{d-1}), \end{aligned}$$

angeben, siehe [9, 11] für nähere Erläuterungen. Da nun die exponentielle Dimensionsabhängigkeit nur noch im Logarithmus auftaucht, eignen sich dünne Gitter auch zur Darstellung hochdimensionaler Funktionen. Die Diskussion der dimensionsabhängigen Koeffizienten im Verhältnis zur Seminorm $|\mathbf{u}|_{\mathbf{2},M}$ für $M \in \{\infty, 2\}$ spielt jedoch eine wichtige Rolle bei hochdimensionalen Problemen. Mit entsprechenden Untersuchungen werden wir uns in Kapitel 5 beschäftigen.

Adaptivität

Will man Funktionen geringerer Glattheit als $X_0(\Omega)$ durch ihren Dünngitterinterpolanten approximieren, so erhält man ein deutlich schlechteres Kosten/Nutzen-Verhältnis. Zur genauen Erfassung von Singularitäten benötigt man sehr feine Gitter, also ein hohes Level n. Gleichzeitig ist die für glatte Funktionen angegebene Interpolationsordnung nicht erreichbar. Um dennoch Funktionen geringerer Glattheit effizient darstellen zu können, bedient man sich der Beobachtung, daß singuläre Phänomene in der Regel lokal beschränkt sind, während derartige Funktionen im Rest des Gebiets den von uns gewünschten Glattheitsannahmen genügen. Die Genauigkeit wird dort erhöht (d. h. Gitterpunkte mit Basisfunktionen werden eingefügt), wo die Funktion singulär ist, während die glatten Bestandteile durch Gitter niedrigen Levels gut approximiert werden. Dieses Vorgehen ist als h-Adaptivität bekannt, da lokal die Maschenweite angepasst wird (siehe [11, 25]). Durch adaptives Vorgehen ist es möglich, das Kosten/Nutzen Verhältnis wie bei der Interpolation glatter Funktionen mit regulären dünnen Gittern wiederherzustellen.

Im Fall einer hierarchischen Basis Φ können wir schlichtweg durch Hinzunahme weiterer Basisfunktionen auf feineren Leveln die Genauigkeit erhöhen. Notwendig sind alle Punkte, deren Überschuss $u_{\mathbf{l},\mathbf{i}}$ einen gewissen Schwellwert $\epsilon > 0$ übersteigt. Der Interpolant enthält dann nicht mehr die Punkte der regulären Indexmenge G_n^d, sondern diejenigen einer „aktiven“ Indexmenge

$$G_\epsilon^d := \{(\mathbf{l}, \mathbf{i}) \in G_\infty^d \,|\, \|u_{\mathbf{l},\mathbf{i}} \cdot \phi_{\mathbf{l},\mathbf{i}}\| \geq \epsilon\}.$$

Die Wahl der Norm $\|\cdot\|$ erlaubt die Generierung von adaptiven Gittern, die gute Approximationseigenschaften bezüglich dieser Norm aufweisen.

Sind zusätzlich zu den Knoten der aktiven Indexmenge G_ϵ^d auch alle hierarchischen Väter im Gitter enthalten, so können sämtliche Algorithmen auch adaptiv ausgeführt werden – die Traversierungsmethoden besuchen grundsätzlich nur Söhne beziehungsweise Väter (dies werden wir in Kapitel 3 eingehend untersuchen).

Nun ist die Erstellung der Menge aktiver Indizes zu einer gegebenen Funktion $\mathbf{u}$ schwierig, zumal in den folgenden Kapiteln $\mathbf{u}$ nur näherungsweise während des Lösungsprozesses einer Differentialgleichung verfügbar ist. Ausgehend von einem regulären dünnen Gitter muss entschieden werden, an welchen Orten im Gitter weitere Punkte aus G_ϵ^d eingefügt werden müssen. Für jeden Index $(\mathbf{l}, \mathbf{i}) \in G_\epsilon^d$ stehen also lediglich Informationen an Punkten geringeren Levels zur Entscheidung zur Verfügung. Dazu ist ein Fehlerschätzer notwendig, der zu jedem bereits eingefügten Punkt $(\mathbf{k}, \mathbf{j})$ erkennt, ob auf höheren Levels ein Nachfahre $(\mathbf{l}, \mathbf{i}) \in G_\epsilon^d$ von $(\mathbf{k}, \mathbf{j})$ mitsamt allen auf den Levels $(n(\mathbf{k}) + 1)$ bis $(n(\mathbf{l}) - 1)$ liegenden hierarchischen Vorfahren eingefügt werden muss. Der Fehlerschätzer muß zuverlässig sein (d. h. der Interpolationsfehler muß gegen 0 konvergieren) und gleichzeitig effizient, also nicht unnötig viele Punkte in das erstellte Gitter einfügen.

In [22, 36] wird ein einfacher Fehlerindikator basierend auf der hierarchischen Hutbasis vorgestellt. Dazu nutzt man zur Entscheidung, ob $(\mathbf{l}, \mathbf{i})$ verfeinert werden soll, lediglich den gewichteten hierarchischen Koeffizienten $u_{\mathbf{l},\mathbf{i}}$ in der Bewertungsnorm $\|\cdot\|$, d. h. $\|u_{\mathbf{l},\mathbf{i}} \cdot \phi_{\mathbf{l},\mathbf{i}}\| > \epsilon$. Diese Vorgehensweise beruht auf der Beobachtung, daß die hierarchischen Überschüsse diskrete zweite Ableitungen repräsentieren und somit ein Maß für die Glattheit von $\mathbf{u}$ an $x_{\mathbf{l},\mathbf{i}}$ darstellen, vgl. [8, 9, 11, 36]. Diese einfache Heuristik erlaubt das sukzessive Einfügen neuer Punkte anhand der bereits in die aktive Indexmenge eingefügten Punkte. Die Effizienz dieses Schätzers ist theoretisch nicht klar, jedoch beobachtet man in der Praxis gute Ergebnisse. Hat man eine Funktion mit einer sehr feinen Spitze (d. h. eine Spitze, für die hierarchische Überschüsse von sehr feinem Level in G_ϵ^d notwendig sind), so wird diese Heuristik jedoch fehlschlagen. Unter Umständen beeinflusst eine solche Spitze die hierarchischen Koeffizienten auf groben Levels nicht, sodaß der Fehlerschätzer zu früh terminiert. Dennoch beobachtet man insgesamt gute Resultate in der Praxis bei Verwendung dieses Fehlerindikators. Abbildung 2.8 zeigt als Beispiel die Funktion

$$\mathbf{u}(x_1, x_2) = 5 \cdot e^{-10x_1 - 10x_2},$$

dargestellt mit einem adaptiven dünnen Gitter mit 3774 Punkten. Zugrunde liegt der L_∞-Fehlerschätzer mit $\epsilon = 10^{-4}$. Da wir in dieser Arbeit primär die Entwicklung von Algorithmik zu gegebenem (adaptiven) Gitter im Blick haben, sei für detailliertere Ausführungen zur Gittergenerierung und Fehlerschätzung auf [11] verwiesen.

Hierarchische Transformationen

Hat man ein adaptives oder reguläres dünnes Gitter G^d, so stellt sich die Frage, wie man zu gegebener Funktion $\mathbf{u}\colon [0,1]^d \to \mathbb{R}$ den Dünngitterinterpolanten $u(x) = \sum_{(\mathbf{l},\mathbf{i}) \in G} u_{\mathbf{l},\mathbf{i}} \phi_{\mathbf{l},\mathbf{i}}(x)$ ermittelt. Im Falle der nodalen Basis ist diese Aufgabe trivial, denn an jedem Gitterpunkt übernimmt man lediglich die Knotenwerte von $\mathbf{u}$. Dies verallgemeinern wir auf hierarchische Basen und lernen in diesem Abschnitt schnelle Algorithmen kennen, die in d Durchläufen durch G^d alle hierarchischen Koeffizienten $u_{\mathbf{l},\mathbf{i}}$ aus den Knotenwerten

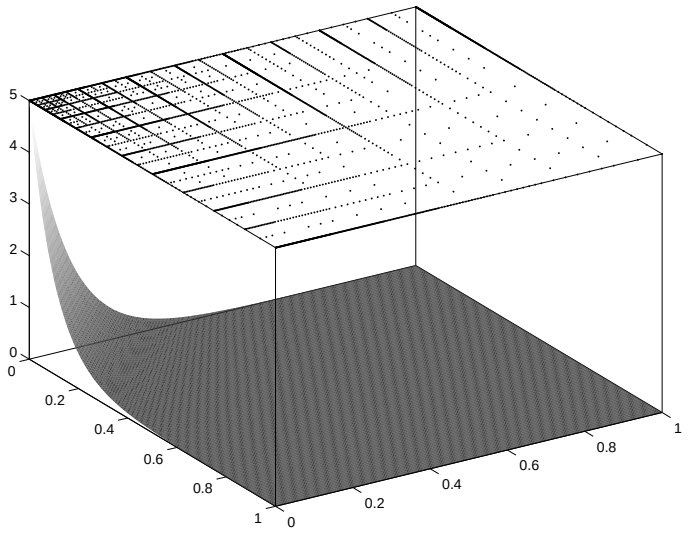

Abbildung 2.8: Die Funktion $\mathbf{u}(x_1, x_2) = 5 \cdot e^{-10x_1-10x_2}$ und das mithilfe des L_∞-Fehlerschätzers generierte Gitter mit $\epsilon = 10^{-4}$ (3774 Punkte)

$\mathbf{u}(x_{\mathbf{l},\mathbf{i}}) = u(x_{\mathbf{l},\mathbf{i}})$ errechnen. Diese Transformationen dienen ebenso dazu, zu einer beliebigen Dünngitterfunktion effizient alle Funktionswerte an den Knoten zu bestimmt. Dazu bedienen wir uns der bereits betrachteten Eigenschaft

$$u_{\mathbf{l},\mathbf{i}} = \mathbf{u}(x_{\mathbf{l},\mathbf{i}}) - I_{(G^d_{r-1} \cap G^d)}\mathbf{u}(x_{\mathbf{l},\mathbf{i}})$$

für einen Gitterpunkt $(\mathbf{l}, \mathbf{i})$ auf Level $n(\mathbf{l}) = r$.

Weiterhin nutzen wir aus, daß man aufgrund des Tensorproduktansatzes

$$\phi_{\mathbf{l},\mathbf{i}}(x) = \prod_{m=1}^{d} \phi_{l_m,i_m}(x_m)$$

die einzelnen Richtungen $m = 1, \ldots, d$ *nacheinander* bearbeiten kann. Dabei werden Zwischenergebnisse lediglich weiter transportiert. Die Tensorproduktstruktur erlaubt diese Art von Kommutativität, denn die verschiedenen Koordinatenrichtungen stehen in einem kartesischen Koordinatensystem orthogonal aufeinander und beeinflussen einander daher nicht, vgl. [22]. In Kapitel 3 werden wir dieses Schema eingehend in Bezug auf die Lösung von Differentialgleichungen untersuchen. Die hier entwickelten Algorithmen sind letztlich einfach Spezialfälle der in Kapitel 3 diskutierten Algorithmik.

Mithilfe dieser Beobachtung ziehen wir uns zurück auf den eindimensionalen Fall und wenden den daraus resultierenden Algorithmus lediglich nacheinander in jede Richtung auf alle Linien des Gitters G^d an. Im Eindimensionalen können wir aus gegebenen hierarchischen Werten des Interpolanten auf Level $(r-1)$,

$$u_{r-1} := I_{(G^d_{r-1} \cap G^d)}\mathbf{u},$$

alle hierarchischen Werte auf Level r ermitteln, denn der Interpolant ist bis Level $(r-1)$ stückweise linear. Betrachten wir den Punkt (r,i), so stellen wir fest, daß u_{r-1} gerade auf dem Träger von $\phi_{r,i}$ stückweise linear ist. Der Träger von $\phi_{r,i}$ ist aber $[x_{r,i}-h_r, x_{r,i}+h_r]$. Damit gilt gemäß (2.8)

$$u_{r,i} = \mathbf{u}(x_{r,i}) - \frac{1}{2}\big(u_{r-1}(x_{r,i}-h_r) + u_{r-1}(x_{r,i}+h_r)\big). \tag{2.15}$$

Allerdings existieren per Konstruktion Punkte an den Koordinaten $(x_{r,i}\pm h_r)$ in $G^d_{r-1}\cap G^d$. Damit läßt sich ein Verfahren angeben, das auch bei der angesprochenen Erweiterung auf d Dimensionen funktioniert. Zunächst werten wir die Funktion $\mathbf{u}$ an jedem Gitterpunkt aus und bezeichnen den Wert mit $\tilde{u}_{\mathbf{l},\mathbf{i}} := \mathbf{u}(x_{\mathbf{l},\mathbf{i}})$. Anschliessend traversieren wir das eindimensionale Gitter (d. h. einen Binärbaum) von „oben“ $\equiv r=1$ nach „unten“ $\equiv r=n$ und ermitteln jeweils den Koeffizienten $u_{\mathbf{l},\mathbf{i}}$. Die Knotenwerte $u_{r-1}(x_{r,i}\pm h_r)$ sind nichts anderes als bestimmte $\tilde{u}_{\mathbf{l},\mathbf{i}}$ und können bei der Baumtraversierung als temporäre Hilfswerte von Vätern an die Söhne einfach weitergereicht werden. Im Eindimensionalen entspricht dieser Algorithmus gerade der Basistransformation einer Funktion von nodaler in hierarchische Basisdarstellung. Die andere Richtung ist ebenfalls möglich. Dabei verwendet man lediglich Addition anstelle von Subtraktion,

$$u_{r,i} = \mathbf{u}(x_{r,i}) + \frac{1}{2}\big(u_{r-1}(x_{r,i}-h_r) + u_{r-1}(x_{r,i}+h_r)\big). \tag{2.16}$$

Die Werte $u_{r-1}(x_{r,i}\pm h_r)$ müssen nun erst ermittelt werden, bevor sie an die Söhne weitergereicht werden können.

Diese Vorgehensweise, die von oben nach unten die Binärbaumstruktur traversiert und dabei aus akkumulierbaren/verfügbaren Werten an den Vätern die Werte für die Söhne ermittelt, spielt eine wichtige Rolle bei allen Algorithmen auf dünnen Gittern und kann hier exemplarisch für die noch folgenden Algorithmen gesehen werden. Dies ist wesentlicher Teil der Betrachtungen in Kapitel 3.

Das Erzeugendensystem

Oftmals ist nicht nur die Basisdarstellung einer Dünngitterfunktion u interessant, sondern die Verwendung eines Erzeugendensystems, welches auf jedem Level auch diejenigen Hutfunktionen zur Darstellung hinzunimmt, die bei der Einführung der hierarchischen Indexmengen in Abschnitt 2.1 weggelassen wurden. Dies spielt insbesondere bei Mehrgitterverfahren eine entscheidende Rolle, siehe dazu [21]. Wir benötigen diese Knoten lediglich aus algorithmischen Gründen zum Ablegen von temporären Zwischenergebnissen.

Im eindimensionalen Fall führten wir die hierarchische Sichtweise ein, indem wir alle durch V_n beschriebenen Punkte eindeutig darstellten durch $\bigcup_{r=1}^{n} W_r$. Der Raum V_n führt uns zur nodalen Hutbasis, das Aufteilen in Überschussräume W_r zur hierarchischen Hutbasis. Nehmen wir jedoch die redundante Darstellung

$$E_n^1 := \bigcup_{r=1}^{n} V_r,$$

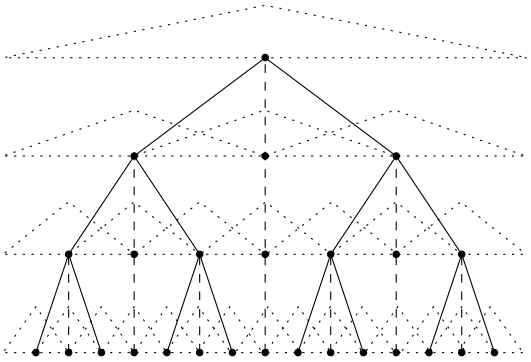

Abbildung 2.9: Das Erzeugendensystem im Eindimensionalen für $n = 4$.

so beschreiben wir nach wie vor dieselben Punkte wie V_n. Die Darstellung lässt sich nun auch mit dem Erzeugendensystem $\bar{\Phi}_n^1 := \{\phi_{l,i} | (l,i) \in E_n^1\}$ durchführen. Abbildung 2.9 zeigt die Elemente von $\bar{\Phi}_n^1$ am Beispiel $n = 4$.

Oftmals ist es notwendig, Zwischenergebnisse auf den zum Erzeugendensystem gehörenden Knoten abzuspeichern, um effektiv die Algorithmik durchzuführen. Wie sich herausstellen wird, ist dies nur für eindimensionale Strukturen notwendig, da die gesamte Algorithmik auf die Hintereinanderausführung von eindimensionalen Algorithmen ausgelegt ist und somit nur Linien in einer d-dimensionalen Struktur bearbeitet werden müssen. Dadurch ist es nicht notwendig, dünne Gitter in d Dimensionen um Erzeugendensystempunkte zu erweitern: es ist ausreichend, ggf. eindimensionale Baumstrukturen bereitzustellen. Im Gegensatz zu einem Binärbaum benötigen diese lediglich einen Faktor 2 an zusätzlichen Knoten für das Erzeugendensystem.

Hat man eine Darstellung von u bezüglich des Erzeugendensystems,

$$u = \sum_{(l,i) \in E_n^1} \tilde{u}_{l,i} \phi_{l,i},$$

und möchte die Redundanz eliminieren zu $u = \sum_{(l,i) \in G_l^1} u_{l,i} \phi_{l,i}$, so entspricht dies der Anwendung eines Sterns $\left[-\frac{1}{2} \quad 1 \quad -\frac{1}{2}\right]$ auf jeden Punkt der Basis. Algorithmisch lässt sich dies durch die Einführung von temporären Hilfsgrößen $t_{l,i}$ an jedem Knoten (l,i) aus dem Erzeugendensystem durchführen. Diese betragen

$$t_{l,i} := \tilde{u}_{l,i} + t_{l+1,2 \cdot i},$$

und damit

$$u_{l,i} = t_{l,i} - \frac{1}{2} t_{l,i+1} - \frac{1}{2} t_{l,i-1}.$$

Dabei sei $t_{l,i} := 0$ falls $(l,i) \neq E_n^1$.

2.2.2 Die hierarchische Prewaveletbasis

Die hierarchische Prewaveletbasis ist eng verwandt mit der hierarchischen Hutbasis. Sie entsteht als Lösung des Problems „finde eine eindimensionale hierarchische Basis Ψ_n^1, die

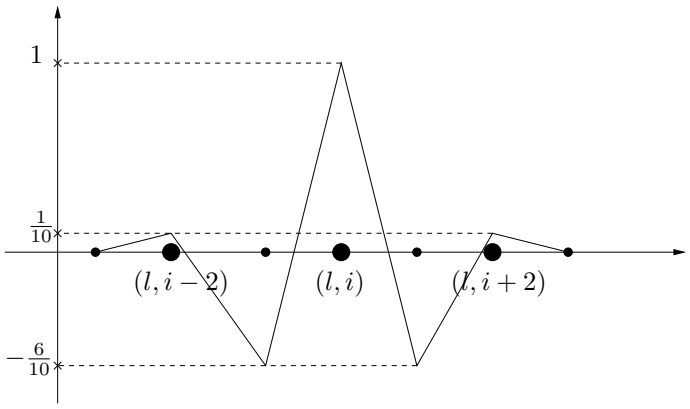

Abbildung 2.10: Randfernes Prewavelet im Eindimensionalen.

Linearkombination nodaler Basisfunktionen ist und L_2-orthogonal zwischen zwei Levels $l \neq k$ ist“ (siehe dazu [23, 31]). Dabei postuliert man, daß die Basisfunktion $\psi_{l,i}$ eine Linearkombination von benachbarten Funktionen $\phi_{l,i\pm j}$ aus dem Erzeugendensystem auf Level l ist. Als Resultat dieses Vorgehens erhält man

$$\psi_{l,i} := \frac{1}{10}\phi_{l,i-2} - \frac{6}{10}\phi_{l,i-1} + 1 \cdot \phi_{l,i} - \frac{6}{10}\phi_{l,i+1} + \frac{1}{10}\phi_{l,i+2},$$

siehe Abbildung 2.10. Für randnahe Knoten errechnet man

$$\psi_{l,1} := \frac{9}{10}\phi_{l,1} - \frac{6}{10}\phi_{l,2} + \frac{1}{10}\phi_{l,3}$$

sowie der symmetrischen Modifikation für $\psi_{l,2^l-1}$. Die Funktionen auf den Levels $l \leq 1$ sind identisch mit den entsprechenden der hierarchischen Hutbasis. Die Verallgemeinerung auf höhere Dimensionen erfolgt völlig analog wie für die Hutbasis über den Tensorproduktansatz

$$\psi_{\mathbf{l},\mathbf{i}}(x) := \prod_{m=1}^{d} \psi_{l_m,i_m}(x_m).$$

Abbildung 2.11 zeigt das Prewavelet $\psi_{3,4|5,7}$.

Bedingt dadurch, daß die Prewaveletbasis eine Linearkombination von Hutfunktionen ist, wird die Interpolationsgenauigkeit gegenüber der hierarchischen Hutbasis nicht erhöht. Auch die Prewaveletbasis bleibt ein stückweise linearer Interpolant und im Falle regulärer dünner Gitter spannen beide den gleichen Raum auf; im adaptiven Fall gibt es aufgrund der größeren Trägerbreite der Prewavelets kleine Unterschiede.

Hierarchische Transformationen

Um eine Funktion $\mathbf{u}\colon [0,1]^d \to \mathbb{R}$ durch die hierarchische Prewaveletbasis zu interpolieren, benötigen wir auch hier eine schnelle Transformation von den Knotenwerten $\mathbf{u}(x_{\mathbf{l},\mathbf{i}})$ zu hierarchischen Koeffizienten $u_{\mathbf{l},\mathbf{i}}$ des Interpolanten

$$u = \sum_{(\mathbf{l},\mathbf{i})\in G^d} u_{\mathbf{l},\mathbf{i}}\psi_{\mathbf{l},\mathbf{i}}$$

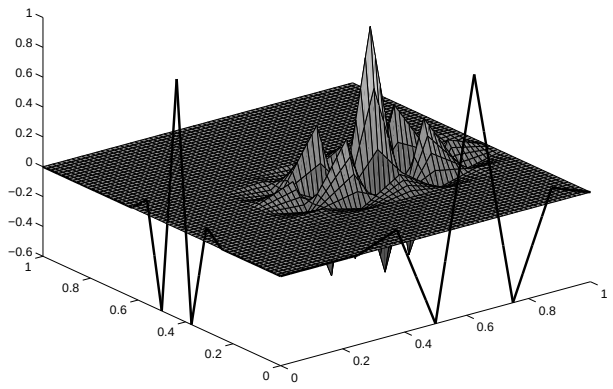

Abbildung 2.11: Das Prewavelet $\psi_{3,4|5,7} = \psi_{3,5} \cdot \psi_{4,7}$.

sowie umgekehrt von hierarchischer Darstellung in Knotenwerte an den Gitterpunkten. In diesem Abschnitt entwickeln wir entsprechende Algorithmen. Da die Prewaveletbasis aus Linearkombination mehrerer Hutfunktionen besteht, ist es notwendig, zunächst Algorithmen zum Basiswechsel von der Prewaveletbasis in die hierarchische Hutbasis und umgekehrt zu entwickeln. Für die Transformation von Knotenwerten auf hierarchische Prewaveletdarstellung transformieren wir dann zunächst in die hierarchische Hutbasis mit den bereits kennengelernten Algorithmen und anschliessend in die Prewaveletbasis. Wir bezeichnen in diesem Abschnitt die hierarchischen Prewaveletkoeffizienten mit $u_{\mathbf{l},\mathbf{i}}$ und die hierarchischen Hutbasiskoeffizienten mit $h_{\mathbf{l},\mathbf{i}}$. Wie auch bei den Transformationen für die hierarchische Hutbasis ist es ausreichend, die eindimensionale Transformation zu formulieren und diese lediglich auf jede Linie des mehrdimensionalen Gitters in jede Richtung hintereinander anzuwenden.

Wir betrachten zunächst die Transformation von der Prewaveletdarstellung in die hierarchische Hutbasis,

$$\sum_{(l,i)\in G_n^d} u_{l,i}\psi_{l,i} \mapsto \sum_{(l,i)\in G_n^d} h_{l,i}\phi_{l,i}.$$

Sind $\bar{u}$ beziehungsweise $\bar{h}$ die zugehörigen Koeffizientenvektoren, so bezeichnen wir mit P die Transformationsmatrix, für die gilt $\bar{h} = P \cdot \bar{u}$. Aufgrund der Linearkombination von Hutfunktionen mittels des Sterns

$$\begin{bmatrix} \frac{1}{10} & -\frac{6}{10} & 1 & -\frac{6}{10} & \frac{1}{10} \end{bmatrix}$$

können wir die Koeffizienten algorithmisch auf die Knoten des Erzeugendensystems verteilen. Dies erlaubt die Darstellung $P = S \cdot Q$, wobei $Q \cdot \bar{u}$ schlichtweg jeden $u_{\mathbf{l},\mathbf{i}}$ gewichtet mit den Koeffizienten $\frac{1}{10}, -\frac{6}{10}, 1$ usw. auf die Knoten des Erzeugendensystems addiert und S die aus $Q \cdot u$ resultierende Redundanz im Erzeugendensystem wieder auflöst, siehe oben.

Fasst man die Anwendung von $S \cdot Q \cdot \bar{u}$ zusammen, so erhält man

$$h_{l,i} = u_{l,i} + \frac{1}{10}u_{l,i\pm2} + t_{l+1,2i} - \frac{1}{2}t_{l,i\pm1} \qquad \text{falls } (l,i) \text{ randfern}, \tag{2.17}$$

$$h_{l,i} = \frac{9}{10}u_{l,i} + \frac{1}{10}u_{l,i\pm2} + t_{l+1,2i} - \frac{1}{2}t_{l,i\pm1} \qquad \text{falls } (l,i) \text{ randnah} \tag{2.18}$$

mit temporären Werten für $(l,i) \neq G_n^1$,

$$t_{l,i} := -\frac{6}{10}u_{l,i\pm1} + t_{l+1,2i}.$$

Für die Knoten auf Level $l \leq 1$ sind die Hutbasis- und Prewaveletkoeffizienten identisch.

Zur Entwicklung eines Verfahrens für die inverse Transformation, d. h. von hierarchischer Hutbasis auf Prewaveletdarstellung, ist mehr Arbeit zu leisten. Formal entspricht dies der Invertierung der Matrix P. Zur algorithmischen Umsetzung lösen wir das Gleichungssystem $P \cdot \bar{u} = \bar{h}$ zu gegebenem $\bar{h}$. Dies entspricht der Auflösung von (2.17) und (2.18) nach $u_{l,i}$. Wir setzen die Definition von $t_{l,i}$ einmal ein, um in der Gleichung für Level l nur temporäre Werte von Level $(l+1)$ oder höher berücksichtigen zu müssen, und erhalten so durch Umsortierung das System für Level l

$$\begin{aligned}
\frac{16}{10}u_{l,i} + \frac{4}{10}u_{l,i\pm2} &= h_{l,i} - t_{l+1,2i} + \frac{1}{2}t_{l+1,2(i\pm1)},\\
\frac{12}{10}u_{l,1} + \frac{4}{10}u_{l,3} &= h_{l,1} - t_{l+1,2} + \frac{1}{2}t_{l+1,4},\\
\frac{12}{10}u_{l,2^l-1} + \frac{4}{10}u_{l,2^l-3} &= h_{l,2^l-1} - t_{l+1,2(2^l-1)} - \frac{1}{2}t_{l+1,2(2^l-2)}
\end{aligned}$$

Die Idee ist nun, bei dem höchsten Level zu beginnen und Level für Level dieses tridiagonale System zu lösen. Hat man die Lösung für ein festes Level gefunden, so ist damit auch die rechte Seite des Systems für das nächste Level aufstellbar. Auf Level 1 schließlich ist nichts mehr zu tun, da die Koeffizienten hier ohnehin identisch sind. Schnelle Löser für tridiagonale Systeme sind bekannt (siehe [34]) und benötigen zwei Durchläufe durch das Gleichungssystem – in unserem Falle also zwei Durchläufe durch jedes Level.

Mit den hiermit abgeschlossenen Transformationen ist nun – für die hierarchische Hutbasis wie für die Prewaveletbasis – die Ermittlung eines Interpolanten einfach möglich. Man wertet die darzustellende Funktion an den (Dünn-)Gitterpunkten aus und führt die Transformationen durch. Für Funktionen mit genügender Glattheit konvergiert der Interpolant gegen die wahre Funktion, wie wir auf Seite 2.2.1 aus der Literatur zusammengefasst haben. Andere Funktionen erfordern die Generierung eines adaptiven Gitters, was in der Formulierung der Algorithmik nur geringe Änderungen erfordert. Dies führt uns im nächsten Schritt zur Darstellung der diskreten Lösung von partiellen Differentialgleichungen mithilfe von Dünngitterfunktionen.

Kapitel 3

Diskretisierung

In diesem Kapitel setzen wir dünne Gitter ein, um numerisch Lösungen für hochdimensionale elliptische partielle Differentialgleichungen der Form

$$L\mathbf{u}(x) = -\sum_{m,r=1}^{d} c_{m,r}\partial^{m,r}\mathbf{u}(x) + \sum_{m=1}^{d} b_m\partial^m\mathbf{u}(x) + \lambda\mathbf{u}(x) = f(x) \tag{3.1}$$

auf einem rechteckigen Gebiet

$$\Omega := \bigotimes_{m=1}^{d} [a_m, b_m]$$

mit Dirichletrandwerten $g(x), x \in \partial\Omega$ zu gewinnen. Wir bezeichnen dabei mit $\partial^m\mathbf{u}$ die Ableitung von $\mathbf{u}$ nach x_m, $\frac{\partial}{\partial x_m}\mathbf{u}$, sowie mit $\partial^{m,r}\mathbf{u}$ die zweite Ableitung von $\mathbf{u}$ nach m und r, $\frac{\partial^2}{\partial x_m \partial x_r}\mathbf{u}$. Die Koeffizienten λ, b_m sowie $c_{m,r}$ seien dabei reelle Konstanten.

Numerische Verfahren zur Lösung derartiger Differentialgleichungen sind bekannt für niedrigdimensionale Anwendungen, d. h. für $d = 2, 3$, siehe [12]. In der Regel verwendet man (adaptive) volle Gitter in Finite Differenzen- oder Finite Element Ansätzen zur Approximation der Lösung, was jedoch durch die hohe Zahl von Punkten zu einem in der Dimension exponentiellen Speicher- und Zeitverbrauch führt. In dieser Arbeit verwenden wir gezielt dünne Gitter, um durch Ausnutzung der vorteilhaft geringen Zahl der Punkte Löser für hohe Dimensionen zu konstruieren. Wir verwenden hier ein Galerkinverfahren mit hierarchischen Dünngitterbasen als Ansatz- und Testfunktionen. Der Schwerpunkt liegt dabei auf der Entwicklung von schnellen Algorithmen und effizienten Datenstrukturen, die die entstehenden diskreten Gleichungssysteme lösen können, denn die komplexe Struktur dünner Gitter erschwert die Umsetzung der Algorithmik insbesondere in hohen Dimensionen. Dies führt zu einer exponentiellen Abhängigkeit der Laufzeit von der Dimension, siehe [1, 4, 9, 33]. Unser Ziel ist es, diese durch neue Algorithmen zu reduzieren.

Dazu fassen wir zunächst die Grundlagen des Galerkinverfahrens zusammen und entwickeln in darauf folgenden Abschnitten algorithmische Umsetzungen, die insbesondere die auftretende exponentielle Abhängigkeit der Laufzeit von der Dimension bei Dünngitterverfahren brechen.

Wir betrachten die zu (3.1) gehörende schwache Form

$$a(\mathbf{u}, v) = \int_\Omega f(x)v(x)dx, \tag{3.2}$$

die wir durch das Testen gegen die Funktionen $v\colon \Omega \to \mathbb{R}$ aus einem Testraum V mit $v|_\Gamma = 0$ mit anschließender partieller Integration erhalten. Die zugehörige Bilinearform $a(\cdot,\cdot)$ ergibt sich damit zu

$$\begin{aligned} a(\mathbf{u}, v) = \sum_{m,k=1}^{d} c_{mk} \int_\Omega \partial^k \mathbf{u}(x)\partial^m v(x)dx + \sum_{m=1}^{d} b_m \int_\Omega \partial^m \mathbf{u}(x)v(x)dx + \\ \lambda \int_\Omega \mathbf{u}(x)v(x)dx. \end{aligned} \tag{3.3}$$

Nach dem Ansatz von Galerkin suchen wir nun eine Funktion $\mathbf{u} \in V$, sodaß für alle $v \in V$ die Gleichung (3.2) und die Randbedingung $\mathbf{u}|_\Gamma = g$ mit $\Gamma := \partial\Omega$ erfüllt ist. Im Gegensatz zur starken Form ist dabei eine Reduktion der Glattheitsanforderungen möglich: an die Stelle von starken Ableitungen treten schwache Ableitungen, die wir wiederum mit $\partial^k \mathbf{u}(x)$ bezeichnen. Nach dem Lemma von Lax-Milgram existiert eine eindeutige Lösung, sofern die Bilinearform beschränkt und V-elliptisch ist, sowie die rechte Seite ebenfalls beschränkt ist, vergleiche [12].

Um unsere hierarchische Basen aus dem letzten Kapitel nun verwenden zu können, benötigen wir als Gebiet das Einheitsquadrat $[0,1]^d$. Bei einem Gebiet $\Omega := \bigotimes_{m=1}^{d}[a_m, b_m]$ lässt sich dies jedoch leicht durch eine d-lineare Koordinatentransformation $\kappa\colon [0,1]^d \to \Omega$,

$$\kappa_m(x_m) := q_m x_m + p_m$$

mit $q_m := b_m - a_m$ und $p_m := a_m$ bewerkstelligen. Wir erhalten damit die Äquivalenz der Problemformulierungen

$$\begin{aligned} \forall v \in V : a(\mathbf{u}, v) &= \int_\Omega f(x)v(x)dx \\ \Leftrightarrow \forall \tilde{v} \in \tilde{V} : \tilde{a}(\mathbf{u} \circ \kappa, \tilde{v}) &= \int_{[0,1]^d} (f \circ \kappa)(x)\tilde{v}(x)dx \end{aligned}$$

wobei $\tilde{V}$ den entsprechenden Testraum mit Funktionen $\tilde{v}\colon [0,1]^d \to \mathbb{R}$ bezeichne. Dabei enthält $\tilde{a}$ modifizierte konstante Koeffizienten, die durch Anwendung der Kettenregel auf $u \circ \kappa$ entstehen. Die bei der Transformation auftauchende Determinante der Jakobimatrix von κ lässt sich wegkürzen (für Details der Transformation sei an dieser Stelle auf [14] verwiesen). So können wir im Folgenden ohne Einschränkung der Allgemeinheit annehmen $\Omega = [0,1]^d$.

Als Ansatz- und Testraum verwenden wir nun den von der hierarchischen Hutbasis[1] Φ_n^d aufgespannten, endlichdimensionalen Teilraum $V_n := \operatorname{span}\left\{\Phi_n^d\right\}$ des Sobolevraums erster

[1] Das Verfahren läßt sich völlig analog mit einer anderen hierarchischen Basis durchführen.

Ordnung mit homogenen Randbedingungen, $V := H_0^1$ (siehe [2] für die Definition von Sobolevräumen). Inhomogene Dirichletrandbedingungen werden wir gesondert diskretisieren. Wir suchen eine Approximation $u = \sum_{(\mathbf{l},\mathbf{i})\in G_n^d} u_{\mathbf{l},\mathbf{i}}\phi_{\mathbf{l},\mathbf{i}} \in V_h$ an $\mathbf{u}$, die die diskrete Problemformulierung

$$a(u, \phi_{\mathbf{k},\mathbf{j}}) = \sum_{(\mathbf{l},\mathbf{i})\in G_n^d} u_{\mathbf{l},\mathbf{i}} a(\phi_{\mathbf{l},\mathbf{i}}, \phi_{\mathbf{k},\mathbf{j}}) = \int_{[0,1]^d} f(x)\phi_{\mathbf{k},\mathbf{j}}(x)dx$$

für alle $\phi_{\mathbf{k},\mathbf{j}} \in \Phi_n^d$ erfüllt. Definieren wir nun die Matrix $A = (a_{(\mathbf{l},\mathbf{i}),(\mathbf{k},\mathbf{j})})$ mit

$$a_{(\mathbf{l},\mathbf{i}),(\mathbf{k},\mathbf{j})} := a(\phi_{\mathbf{l},\mathbf{i}}, \phi_{\mathbf{k},\mathbf{j}}), \tag{3.4}$$

so entspricht dies der Lösung des linearen Gleichungssystems

$$A^T \bar{u} = \bar{f},$$

wobei $\bar{u}$ die Koeffizienten von u enthalte und

$$\bar{f}_{\mathbf{k},\mathbf{j}} := \int_{[0,1]^d} f(x)\phi_{\mathbf{k},\mathbf{j}}(x)dx.$$

Inhomogene Dirichletranddaten

Wollen wir das Verfahren für inhomogene Dirichletranddaten durchführen, so nutzen wir eine Standardtechnik, indem wir $\mathbf{u}$ in zwei ausreichend glatte Bestandteile $\mathbf{u}^I$ und $\mathbf{u}^R$ mit $\mathbf{u}^I|_\Gamma = 0$ aufteilen. Dann nämlich gilt bezüglich der starken Form

$$L\mathbf{u}(x) = f(x) \Leftrightarrow L\mathbf{u}^I(x) = f(x) - L\mathbf{u}^R(x),$$

wobei $L\mathbf{u}^R(x)$ durch die Dirichletdaten fixiert ist und das restliche Problem homogene Dirichletranddaten hat.

Wir können diese Technik diskret anwenden, da für $\mathbf{u}|_\Gamma \neq 0$ der Interpolant u auch Randpunkte benötigt und schreiben

$$u = \sum_{(\mathbf{l},\mathbf{i})\in G_{n,I}^d} u_{\mathbf{l},\mathbf{i}}\phi_{\mathbf{l},\mathbf{i}} + \sum_{(\mathbf{l},\mathbf{i})\in G_{n,R}^d} u_{\mathbf{l},\mathbf{i}}\phi_{\mathbf{l},\mathbf{i}} \tag{3.5}$$

mit einer Aufteilung in innere Punkte, $G_{n,I}^d$, und Randpunkte, $G_{n,R}^d$. Für $(\mathbf{l},\mathbf{i}) \in G_{n,R}^d$ lassen sich die hierarchischen Koeffizienten leicht durch eine Restriktion der hierarchischen Transformation auf den Rand ermitteln, wenn man nur die Knotenwerte $\mathbf{u}(x_{\mathbf{l},\mathbf{i}})$ von $\mathbf{u}$ (d. h. die Dirichletdaten) kennt. Damit gilt

$$a(u, \phi_{\mathbf{k},\mathbf{j}}) = \bar{f}_{\mathbf{k},\mathbf{j}} \Leftrightarrow a(u_I, \phi_{\mathbf{k},\mathbf{j}}) = \bar{f}_{\mathbf{k},\mathbf{j}} - a(u_R, \phi_{\mathbf{k},\mathbf{j}})$$

mit einer (3.5) entsprechenden Aufteilung $u = u_I + u_R$. Im Folgenden betrachten wir daher nur homogene Randbedingungen und wenden uns erst in Abschnitt 3.1.4 wieder der algorithmischen Umsetzung dieser Modifikation der rechten Seite zu.

3.1 Das Unidirektionale Prinzip

Wir wenden uns nun den Algorithmen zur Lösung des linearen Gleichungssystems

$$A^T \bar{u} = \bar{f} \tag{3.6}$$

mit der Steifigkeitsmatrix

$$A = (a_{(\mathbf{l},\mathbf{i}),(\mathbf{k},\mathbf{j})})_{(\mathbf{l},\mathbf{i}),(\mathbf{k},\mathbf{j})\in G^d},$$
$$a_{(\mathbf{l},\mathbf{i}),(\mathbf{k},\mathbf{j})} = a(\phi_{\mathbf{l},\mathbf{i}}, \phi_{\mathbf{k},\mathbf{j}})$$

und der rechten Seite $\bar{f}$ zu. Dabei ist $u \in V_n = \text{span}\left\{\Phi_n^d\right\}$, $\bar{u}$ der Vektor der Koeffizienten und $a(u,v)$ ist die durch die schwache Form gegebene Bilinearform der Gleichung, siehe oben.

Bei Verwendung von klassischen Finite-Element-Basen hat die Steifigkeitsmatrix üblicherweise $O(N), N = |G^d|$ nicht-verschwindende Einträge. Daher ist es in den entsprechenden Finite Element Paketen üblich, die Matrix in einem Assemblierungsschritt aufzustellen und anschliessend das System (3.6) effizient zu lösen. Mit Hilfe eines iterativen Lösers und einem levelunabhängigen Vorkonditionierer (beispielsweise durch Multileveltechniken oder Wavelets, siehe [21, 23]) erreicht man so eine Gesamtlaufzeit von $O(N)$ für das Lösen des Gleichungssystems.

Im Fall von hierarchischen Ansatzfunktionen hat A jedoch deutlich mehr nicht-verschwindende Einträge als $O(N)$ – es entsteht ein Besetzungsmuster mit einer sogenannten Fingerstruktur, siehe Abbildung 3.1 (vgl. auch [4]). Bei Verwendung der üblichen iterativen Gleichungssystemlöser wie CG o.ä. ist jedoch lediglich die Multiplikation von A^T mit einem Vektor vonnöten. Inhalt dieses Kapitels ist es daher, effiziente Algorithmen zur Errechnung der Multiplikation $A^T \cdot \bar{u}$, $\bar{u} \in \mathbb{R}^N$ zu entwickeln, die trotz der hohen Zahl von Matrixeinträgen lediglich die Zeit $O(c \cdot N)$ benötigen. Mit einem levelunabhängigen Vorkonditionierer (wie oben angedeuted) erhalten wir so auch für das gesamte Verfahren eine Laufzeit von $O(c \cdot N)$. Die Algorithmen hierfür wurden bereits u. a. in [1, 4, 9] entwickelt. Da wir uns in dieser Arbeit gezielt mit hochdimensionalen Problemen beschäftigen, ist die Abhängigkeit der Konstanten c von der Dimension wesentlich. Die bisher entwickelten Algorithmen benötigen eine Laufzeit von $O(d^2 2^d N)$ – zu langsam für einen effizienten hochdimensionalen Löser (vgl. [1]). Wesentlicher Teil dieser Arbeit ist daher die Entwicklung neuer Verfahren, deren Laufzeit wiederum linear in der Zahl der Unbekannten aber nur polynomiell in der Dimension skaliert. Zunächst führen wir jedoch die bekannten Algorithmen ein.

Wie in (3.3) zu sehen, besteht die zu ermittelnde Bilinearform aus Summanden der Gestalt

$$\langle u, \phi_{\mathbf{k},\mathbf{j}} \rangle := \int_{[0,1]^d} D^\alpha u(x) D^\beta \phi_{\mathbf{k},\mathbf{j}}(x) dx \tag{3.7}$$

mit schwachen Ableitungsoperatoren D^α und D^β.

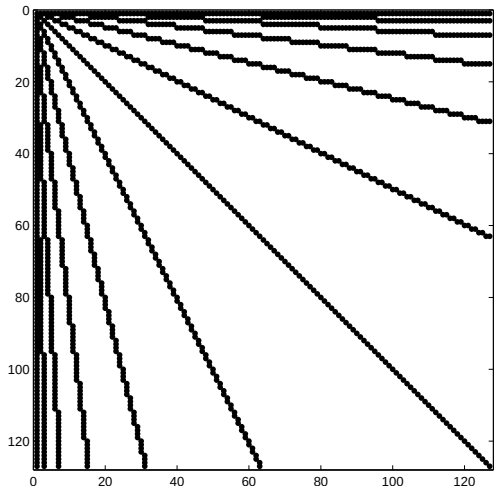

Abbildung 3.1: Besetzungsmuster der Massenmatrix $\int_{[0,1]^d} \phi_{\mathbf{l},\mathbf{i}}(x)\phi_{\mathbf{k},\mathbf{j}}(x)dx$ für $d = 1, n = 7$

Setzt man die Basisdarstellung $u = \sum_{(\mathbf{l},\mathbf{i})\in G^d} u_{\mathbf{l},\mathbf{i}}\phi_{\mathbf{l},\mathbf{i}}$ sowie die multiplikative Struktur der Basisfunktionen ein, so erhält man

$$\langle u, \phi_{\mathbf{k},\mathbf{j}}\rangle = \int_{[0,1]^d} \cdots \int \sum_{(\mathbf{l},\mathbf{i})\in G^d} u_{\mathbf{l},\mathbf{i}} \cdot \phi^{(\alpha_1)}_{l_1,i_1}(x_1) \cdots \phi^{(\alpha_d)}_{l_d,i_d}(x_d) \cdot \phi^{(\beta_1)}_{k_1,j_1}(x_1) \cdots \phi^{(\beta_d)}_{k_d,j_d}(x_d) dx_1 \cdots dx_d. \quad (3.8)$$

Teilt man nun die Summe über $(\mathbf{l},\mathbf{i})$ in die Einzelkomponenten $(l_m, i_m), m = 1, \ldots, d$ auf und klammert diese aus, so erhält man

$$\langle u, \phi_{\mathbf{k},\mathbf{j}}\rangle = \int_0^1 \sum_{\substack{(l_1,i_1)\\ \exists(\mathbf{l}',\mathbf{i}'):\\ (l_1,\mathbf{l}'|i_1,\mathbf{i}')\in G^d}} \left(\int_0^1 \sum_{\substack{(l_2,i_2)\\ \exists(\mathbf{l}',\mathbf{i}'):\\ (l_1,l_2,\mathbf{l}'|i_1,i_2,\mathbf{i}')\in G^d}} \left(\cdots \int_0^1 \sum_{\substack{(l_d,i_d):\\ (l_1,\ldots,l_d|i_1,\ldots,i_d)\in G^d}} u_{(l_1,\ldots,l_d|i_1,\ldots,i_d)} \cdot \phi^{(\alpha_d)}_{l_d,i_d}(x_d) \cdot \phi^{(\beta_d)}_{k_d,j_d}(x_d) dx_d \right) \cdots \right) \cdot \phi^{(\alpha_1)}_{l_1,i_1}(x_1) \cdot \phi^{(\beta_1)}_{k_1,j_1}(x_1) dx_1. \quad (3.9)$$

Hält man $(d-1)$ Komponenten eines Multiindex $(\mathbf{l},\mathbf{i})$ fest und betrachtet alle Gitterpunkte im Gitter G^d, die in diesen Komponenten mit $(\mathbf{l},\mathbf{i})$ übereinstimmen, so liegen diese auf einer Gitterlinie. Dies legt die Vermutung nahe, alle Werte $\langle u, \cdot\rangle$ auf einmal durch Anwendung von eindimensionalen Algorithmen von innen nach außen zu errechnen, siehe Schema 3.1.1.

Jedoch ist ein solches Schema noch nicht ausreichend für dünne Gitter, obwohl die zugrunde liegende Idee der Schlüssel zu effizienten Algorithmen ist. Wir veranschaulichen das Schema zunächst für volle Gitter in Dimension $d = 2$. Für jede x_1-Koordinate, zu der es Gitterpunkte gibt, durchläuft man in x_2-Richtung die Struktur. Dabei errechnet man

Algorithmus 3.1.1 Schema zur Berechnung aller Werte $\langle u, \cdot \rangle$ aus Gleichung (3.9)

for $m := d$; $m \geq 1$; $m := m - 1$ **do**
  for all Gitterlinien in Richtung m **do**
    wende einen eindimensionalen Algorithmus auf die Linie an;
    überschreibe dabei die alten Werte auf der Linie mit den (in diesem Kontext eindimensionalen) Werten $\langle u, \cdot \rangle$
  end for
end for

für jeden Punkt $(\mathbf{k}, \mathbf{j})$ die innersten Integrale in (3.9) und speichert das Ergebnis in $u_{\mathbf{k},\mathbf{j}}$ ab. Anschliessend durchläuft man in x_1-Richtung die Struktur, und errechnet die äußeren Integrale. Für Dimensionen $d > 2$ wird jede Richtung entsprechend einmal durchlaufen. Wir werden Algorithmen kennenlernen, die während eines Durchlaufes in eine spezielle Richtung alle benötigten Integrale in Zeit $O(N)$ errechnen können. Bei d Richtungen hat solch ein Schema für volle Gitter Laufzeit $O(d \cdot N)$.

Durch solch ein Vorgehen vermeidet man teure und komplizierte Operationen, die auf Nachbarpunkte in jeder Richtung zugreifen müssen – insbesondere bei anisotropen Gittern. Dieses Vorgehen, einen mehrdimensionalen Algorithmus auf die Hintereinanderausführung zahlreicher eindimensionaler Algorithmen zurückzuführen, wird als „Unidirektionales Prinzip“ bezeichnet. Die Voraussetzung für die Anwendung dieses Schemas ist, daß die sich akkumulierenden Zwischenergebnisse auch auf geeigneten Knoten abgelegt werden können und beim Durchlauf durch die nachfolgenden Richtungen richtig weitertransportiert werden. Nun sind aber dünne Gitter gerade so konstruiert, daß alle Linien in einer festen Richtung m unterschiedlich viele Punkte haben; die für den Transport notwendigen Punkte fehlen. Das Schema ist dann zwar technisch auf die vorhandenen Punkte anwendbar, liefert jedoch falsche Resultate (vgl. [4]). Um dies zu skizzieren, betrachten wir ein Beispiel in Abbildung 3.2. Abstrakt gehen wir davon aus, daß Daten von dem eingezeichneten Punkt $(\mathbf{l}, \mathbf{i})$ zu dem Punkt $(\mathbf{k}, \mathbf{j})$ des dünnen Gitters transportiert werden müssen – ebenso andersherum. Bei einem Vorgehen wie in Schema 3.1.1 wird erst in Richtung x_2 und anschliessend in Richtung x_1 traversiert, wobei ein Datentransport in irgendeiner Art geschehen kann. Gemäß der eingezeichneten Pfeile kann einer der beiden Transporte stattfinden, während der andere aufgrund der Dünngitterstruktur nicht möglich ist: der zum Transport notwendige Zwischenknoten fehlt.

Ein Algorithmus, der nicht einem ähnlichen Schema folgt, wird Probleme mit der exponentiell in der Dimension steigenden Zahl der Nachbarpunkte bekommen. Aus diesem Grunde wurden Verallgemeinerungen von Schema 3.1.1 für dünne Gitter entwickelt. Dazu verwendet man eindimensionale Algorithmen, die die Operation in zwei Teile zerlegen. Diese wendet man so an, daß Zwischenergebnisse immer nur über Dünngitterknoten transportiert werden. Die Ergebnisse kombiniert man in richtiger Art und Weise und erhält so das d-dimensionale Resultat.

Ein solches Schema wird in [5] vorgestellt und ist unter dem Name „Überschussschleuder“ bekannt. Jedoch benötigt es zur Errechnung von $\langle u, \phi_{\mathbf{k},\mathbf{j}} \rangle$ für alle Gitterpunkte $(\mathbf{k}, \mathbf{j}) \in G^d$ einen Speicher (und damit auch Laufzeit) von $O(2^d N)$ und ist daher für die Lösung

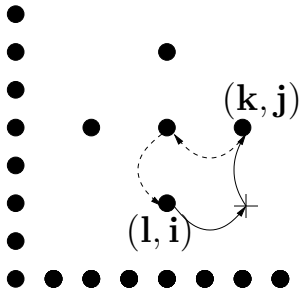

Abbildung 3.2: Fehlende Knoten bei dünnen Gittern, um einen Datentransport gemäß Schema 3.1.1 durchzuführen.

hochdimensionaler Problemstellungen ungeeignet.

Eine sparsamere, rekursive Variante wurde in [4] vorgestellt. Diese benötigt lediglich einen Speicher von $O(d \cdot N)$ und spielt eine wichtige Rolle in Galerkinverfahren für dünne Gitter. Es findet Anwendung in [1, 4, 9] (sowie weiteren Arbeiten) und ist für dünne Gitter unter dem Namen „Unidirektionales Prinzip“ bekannt. Wir werden das Verfahren im folgenden Abschnitt 3.1.1 diskutieren. Die Kombination der erwähnten beiden eindimensionalen Algorithmen im Laufe der Rekursion benötigt jedoch, wie wir noch sehen werden, $O(2^d)$ Durchläufe durch die Datenstruktur. In dieser Arbeit setzen wir hier an und entwickeln das Verfahren weiter: in den Abschnitten 3.1.2 und 3.1.3 werden wir durch konsequente Ausnutzung weiterer Struktur die exponentielle Laufzeitabhängigkeit $O(2^d N)$ auf $O(dN)$ reduzieren. Damit verbessern wir auch den Speicherbedarf von $O(dN)$ auf $O(N)$. Doch zunächst diskutieren wir das Unidirektionale Prinzip aus [4].

3.1.1 Algorithmen für die hierarchische Hutbasis auf dünnen Gittern

Ein für dünne Gitter passendes algorithmisches Prinzip lässt sich angeben, wenn man in der Algorithmik zusätzlich zur Aufteilung in Koordinaten die hierarchischen Relationen berücksichtigt. Dies werden wir im Folgenden ausführen. Unser Ziel ist es, für alle $(\mathbf{k}, \mathbf{j}) \in G^d$ den Bestandteil

$$\langle u, \phi_{\mathbf{k},\mathbf{j}} \rangle = \int_{[0,1]^d} D^\alpha u(x) D^\beta \phi_{\mathbf{k},\mathbf{j}}(x) dx \tag{3.10}$$

eines Matrix-Vektor-Produktes zu berechnen. Die Laufzeit des Verfahrens soll dabei linear in der Zahl der Unbekannten sein, mit einer (dimensionsabhängigen) Konstanten, also $O(T(d) \cdot N)$, $N = |G^d|$. Den Einfluss der Dimension $T(d)$ lassen wir dabei zunächst unberücksichtigt. Die grundsätzliche Idee entspricht dabei dem oben angedachten Schema: für jede Linie in einer festen Richtung $m \in \{1, \ldots, d\}$ wenden wir einen eindimensionalen Algorithmen an. Ein Durchlauf durch die gesamte Datenstruktur kostet $O(N)$ Operationen. Es gilt nun, eindimensionale Algorithmen zu finden, die zu dem mehrdimensionalen Algorithmus zur Bestimmung aller Werte in (3.10) kombiniert werden können. Die Anzahl der notwendigen Ausführungen wird dann der Koeffizient $T(d)$. In der Herleitung folgen wir weitgehend [1].

Der eindimensionale Fall

Sei eine eindimensionale Funktion $u \in V_h = \operatorname{span}\left\{\phi_{l,i} | (l,i) \in G^d\right\}$ mit einem adaptiven oder regulären dünnen Gitter G^d gegeben mittels

$$u(x) = \sum_{(l,i)\in G^d} u_{l,i}\phi_{l,i}(x).$$

Für festes $(k,j) \in G^d$ können wir aufgrund der Trägerbreite folgern, daß zur Auswertung von u an einer beliebigen Stelle $x \in \operatorname{supp}\phi_{k,j}$ lediglich Anteile der hierarchischen Vor- und Nachfahren (siehe Seite 6) von (k,j) benötigt werden. Wir schreiben für $x \in \operatorname{supp}\phi_{k,j}$

$$u(x) = \overline{u}_{k,j}(x) + \underline{u}_{k,j}(x) \tag{3.11}$$

mit

$$\overline{u}_{k,j} := \sum_{\substack{(l,i)\in G^d \\ (l,i)\underset{H}{>}(k,j)}} u_{l,i}\phi_{l,i}(x), \quad \underline{u}_{k,j} := \sum_{\substack{(l,i)\in G^d \\ (l,i)\underset{H}{\leq}(k,j)}} u_{l,i}\phi_{l,i}(x).$$

Damit erhalten wir

$$\begin{aligned}\langle u, \phi_{k,j}\rangle &= \int_0^1 u^{(\alpha)}(x)\phi_{k,j}^{(\beta)}(x)dx \\ &= \langle \overline{u}_{k,j}, \phi_{k,j}\rangle + \langle \underline{u}_{k,j}, \phi_{k,j}\rangle.\end{aligned}$$

Nutzen wir zudem konsequent aus, daß zu festem (k,j) die Träger aller hierarchischen Nachfahren von (k,j) vollständig in $\operatorname{supp}\phi_{k,j}$ enthalten sind, so erhalten wir zwei hocheffiziente Algorithmen zur Bestimmung *aller* Bestandteile $\langle \overline{u}_{k,j}, \phi_{k,j}\rangle$ sowie $\langle \underline{u}_{k,j}, \phi_{k,j}\rangle$. In nur einer Prä-Order Traversierung durch den Baum lassen sich alle $\langle \overline{u}_{k,j}, \phi_{k,j}\rangle$ auf einmal errechnen – in nur $O(N)$ Schritten. Durch die Reihenfolge von oben nach unten wird dieser Algorithmus üblicherweise „TopDown“ genannt, vgl. [1, 9, 11]. Entsprechend lassen sich die Anteile der Nachfahren durch nur eine Post-Order Traversierung alle auf einmal ermitteln, was man entsprechend „BottomUp“ nennt. Tatsächlich besteht die Kunst der Algorithmik darin, zu gegebener Bilinearform $a(\cdot,\cdot)$ diese eindimensionalen Algorithmen zu entwickeln. Für mehrdimensionale Anwendungen setzt man diese lediglich in die noch folgende Kombinationsvorschrift ein.

Es würde an dieser Stelle zu weit führen, die entsprechenden Algorithmen für die Hutbasis im einzelnen vorzustellen. Für das Verständnis der weiteren Abschnitte reicht es völlig aus, sich die beiden Algorithmen TopDown und BottomUp als Black-Box vorzustellen: zu gegebenem Gitter G^d, einer Form $\langle\cdot,\cdot\rangle = \langle\cdot,\cdot\rangle_{\alpha,\beta}$ und $u = \sum_{(l,i)\in G^d} u_{l,i}\phi_{l,i}$ errechnen diese $v = \sum_{(k,j)\in G^d} \tilde{v}_{k,j}\phi_{k,j}$. Dabei ist $\tilde{v}_{k,j} = \langle \overline{u}_{k,j}, \phi_{k,j}\rangle$ beziehungsweise $\tilde{v}_{k,j} = \langle \underline{u}_{k,j}, \phi_{k,j}\rangle$. Algorithmisch überschreiben wir dabei die Speicher der Koeffizienten $u_{l,i}$ mit $\tilde{v}_{l,i}$. Das Schema für TopDown ist dargestellt in Abbildung 3.3. Konkrete Beschreibungen von Algorithmen für die Hutbasis finden sich beispielsweise für $\langle u, v\rangle = \int u(x)v(x)dx$ und $\langle u, v\rangle = \int u'(x)v'(x)dx$ in [4, Kapitel 3.1]. Am Ende dieses Kapitels lernen wir weitere derartige eindimensionale Algorithmen kennen.

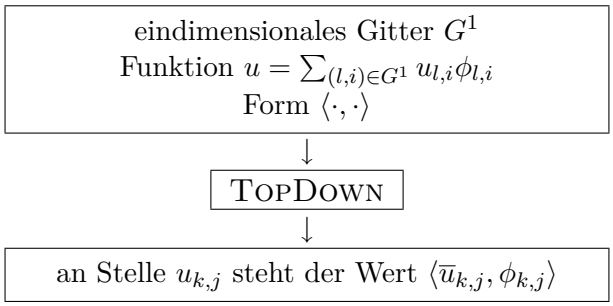

Abbildung 3.3: Ein- und Ausgabeschema des eindimensionalen Algorithmus Topdown.

Wir betrachten nun die Kombination von eindimensionalen TopDown- und BottomUp-Durchläufen in verschiedenen Richtungen, um dieselbe Algorithmik auch für höhere Dimensionen bereitzustellen.

Der mehrdimensionale Fall

Zur Entwicklung eines Algorithmus zur Berechnung von Gleichung (3.10) besteht bei dünnen Gittern die Schwierigkeit, Zwischenergebnisse so auf den vorhandenen Gitterpunkten abzulegen, daß sie nach Abschluss einer Sequenz von *eindimensionalen* Algorithmen schließlich die gewünschten Werte enthalten. Das direkte koordinatenweise Vorgehen führt wie eingangs beschrieben nicht zum Ziel, daher müssen auch im Mehrdimensionalen hierarchische Relationen verwendet werden. Um die gesamte Algorithmik nur auf die Kombination eindimensionaler Bestandteile (TopDown und BottomUp) zurückzuführen, betrachten wir eine Darstellung der Funktionswerte $u(x)$, welche nur entlang *einer* Richtung in hierarchische Vor- und Nachfahren trennt. In Analogie zu der Vorgehensweise im eindimensionalen Fall erhalten wir so für festes $(\mathbf{k},\mathbf{j}) \in G^d$ und $x \in \operatorname{supp}\phi_{\mathbf{k},\mathbf{j}}$ die Aufteilung

$$\begin{aligned} u(x) = &\sum_{\substack{(l_1,i_1)>_H(k_1,j_1):\\ \exists(\mathbf{l}',\mathbf{i}')\lessgtr(\mathbf{k}',\mathbf{j}'),\\ (l_1,i_1)\times(\mathbf{l}',\mathbf{i}')\in G^d}} \sum_{\substack{(\mathbf{l}',\mathbf{i}'):\\ (\mathbf{l}',\mathbf{i}')\lessgtr(\mathbf{k}',\mathbf{j}'),\\ (l_1,i_1)\times(\mathbf{l}',\mathbf{i}')\in G^d}} u_{(l_1,i_1)\times(\mathbf{l}',\mathbf{i}')}\cdot\phi_{(l_1,i_1)\times(\mathbf{l}',\mathbf{i}')}(x) \\ &+ \sum_{\substack{(l_1,i_1)\le_H(k_1,j_1):\\ \exists(\mathbf{l}',\mathbf{i}')\lessgtr(\mathbf{k}',\mathbf{j}'),\\ (l_1,i_1)\times(\mathbf{l}',\mathbf{i}')\in G^d}} \sum_{\substack{(\mathbf{l}',\mathbf{i}'),\\ (\mathbf{l}',\mathbf{i}')\lessgtr(\mathbf{k}',\mathbf{j}'):\\ (l_1,i_1)\times(\mathbf{l}',\mathbf{i}')\in G^d}} u_{(l_1,i_1)\times(\mathbf{l}',\mathbf{i}')}\cdot\phi_{(l_1,i_1)\times(\mathbf{l}',\mathbf{i}')}(x). \end{aligned} \tag{3.12}$$

Dabei bezeichne

$$(l_1,i_1)\times(\mathbf{l}',\mathbf{i}') := (\mathbf{l},\mathbf{i})$$

mit $\mathbf{l} = (l_1,\mathbf{l}')$, $\mathbf{i} = (i_1,\mathbf{i}')$ die Konkatenation zweier Multiindizes. Man überzeugt sich leicht davon, das auf diese Weise alle Basisfunktionen $\phi_{\mathbf{l},\mathbf{i}}$ mit $\operatorname{supp}\phi_{\mathbf{k},\mathbf{j}} \cap \operatorname{supp}\phi_{\mathbf{l},\mathbf{i}} \neq \emptyset$ berücksichtigt werden, denn auch im Mehrdimensionalen kommen zur Darstellung von $u(x)$ nur diejenigen Bestandteile $u_{\mathbf{l},\mathbf{i}}\phi_{\mathbf{l},\mathbf{i}}$ in Frage, welche von hierarchischen Vor- oder Nachfahren stammen. Der linke Teil von Abbildung 3.4 zeigt die zu berücksichtigenden Punkte für festes $(\mathbf{k},\mathbf{j})$ anhand eines zweidimensionalen Beispiels.

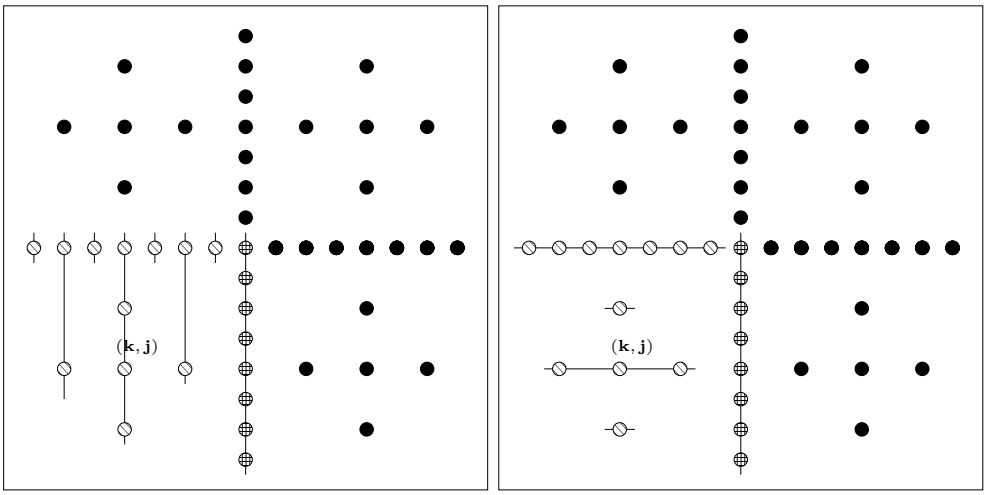

Abbildung 3.4: Darstellung der in (3.12) (links) und (3.13) (rechts) auftauchenden Indizes für festes $(\mathbf{k},\mathbf{j}) \in G^d$ im zweidimensionalen Fall. Die mit vertikal und horizontal gestrichelten Punkte gehören jeweils zur ersten Doppelsumme und die diagonal gestrichelten zur zweiten; die Linien zeigen die Reihenfolge der Summation auf.

Jedoch führt diese Aufteilung nicht zu einer Anwendbarkeit der eindimensionalen Algorithmik. Diese war so gebaut, daß nach Ende eines Algorithmus (beispielsweise TopDown) der Bestandteil $\langle \overline{u}_{\mathbf{k},\mathbf{j}}, \phi_{\mathbf{k},\mathbf{j}} \rangle$ an Knoten $(\mathbf{k},\mathbf{j})$ abgelegt war. Das soeben eingeführte Splitting enthält keinerlei Struktur, die zum Weitertransport derartiger eindimensionaler Ergebnisse genutzt werden kann – und so führt es letztlich auf dieselben Schwierigkeiten wie das bereits eingangs verworfene Schema (3.1.1).

Abhilfe schafft eine kleine Modifikation, die die Ausnutzung der Dünngitterstruktur zum Datentransport erlaubt. Dazu betrachten wir alle Knoten, deren letzte $(d-1)$ Komponenten mit denen von $(\mathbf{k},\mathbf{j})$ übereinstimmen, sowie diejenigen Knoten, deren erste Komponente mit der von $(\mathbf{k},\mathbf{j})$ übereinstimmt. Wir schreiben für $x \in \operatorname{supp} \phi_{\mathbf{k},\mathbf{j}}$

$$
\begin{aligned}
u(x) = & \sum_{\substack{(l_1,i_1) \underset{H}{>} (k_1,j_1): \\ (l_1,i_1)\times(\mathbf{k}',\mathbf{j}') \in G^d}} \; \sum_{\substack{(\mathbf{l}',\mathbf{i}'): \\ (\mathbf{l}',\mathbf{i}') \lessgtr (\mathbf{k}',\mathbf{j}'), \\ (l_1,i_1)\times(\mathbf{l}',\mathbf{i}') \in G^d}} u_{(l_1,i_1)\times(\mathbf{l}',\mathbf{i}')} \cdot \phi_{(l_1,i_1)\times(\mathbf{l}',\mathbf{i}')}(x) \\
& + \sum_{\substack{(\mathbf{l}',\mathbf{i}'): \\ (\mathbf{l}',\mathbf{i}') \lessgtr (\mathbf{k}',\mathbf{j}'), \\ (k_1,j_1)\times(\mathbf{l}',\mathbf{i}') \in G^d}} \; \sum_{\substack{(l_1,i_1) \underset{H}{<} (k_1,j_1): \\ (l_1,i_1)\times(\mathbf{l}',\mathbf{i}') \in G^d}} u_{(l_1,i_1)\times(\mathbf{l}',\mathbf{i}')} \cdot \phi_{(l_1,i_1)\times(\mathbf{l}',\mathbf{i}')}(x),
\end{aligned}
\tag{3.13}
$$

vergleiche [1]. Diese Aufteilung berücksichtigt wiederum alle Bestandteile, wie man sich anhand des rechten Teils von Abbildung 3.4 veranschaulicht. Setzen wir (3.13) in unsere

Zielform $\langle \cdot, \cdot \rangle$ ein, so erhalten wir für die erste Doppelsumme

$$\int_0^1 \sum_{\substack{(l_1,i_1) \underset{H}{>} (k_1,j_1): \\ (l_1,i_1)\times(\mathbf{k}',\mathbf{j}')\in G^d}} \left\{ \int_{[0,1]^{d-1}} \sum_{\substack{(\mathbf{l}',\mathbf{i}'): \\ (\mathbf{l}',\mathbf{i}') \underset{\geq}{\leq} (\mathbf{k}',\mathbf{j}'), \\ (l_1,i_1)\times(\mathbf{l}',\mathbf{i}')\in G^d}} u_{(l_1,i_1)\times(\mathbf{l}',\mathbf{i}')} \cdot \phi^{(\alpha')}_{(\mathbf{l}',\mathbf{i}')}(x') \cdot \right.$$

$$\left. \cdot \phi^{(\beta')}_{(\mathbf{k}',\mathbf{j}')}(x')dx' \right\} \cdot \phi^{(\alpha_1)}_{(l_1,i_1)}(x_1) \cdot \phi^{(\beta_1)}_{(k_1,j_1)}(x_1)dx_1 \quad (3.14)$$

oder abkürzend

$$= \int_0^1 \sum_{\substack{(l_1,i_1) \underset{H}{>} (k_1,j_1): \\ (l_1,i_1)\times(\mathbf{k}',\mathbf{j}')\in G^d}} \tilde{u}_{(l_1,i_1)\times(\mathbf{k}',\mathbf{j}')} \cdot \phi^{(\alpha_1)}_{(l_1,i_1)}(x_1) \cdot \phi^{(\beta_1)}_{(k_1,j_1)}(x_1)dx_1$$

mit den Werten in den geschweiften Klammern

$$\tilde{u}_{(l_1,i_1)\times(\mathbf{k}',\mathbf{j}')} := \int_{[0,1]^{d-1}} \sum_{\substack{(\mathbf{l}',\mathbf{i}'), \\ (\mathbf{l}',\mathbf{i}') \underset{\geq}{\leq} (\mathbf{k}',\mathbf{j}'), \\ (l_1,i_1)\times(\mathbf{l}',\mathbf{i}')\in G^d}} u_{(l_1,i_1)\times(\mathbf{l}',\mathbf{i}')} \cdot \phi^{(\alpha')}_{(\mathbf{l}',\mathbf{i}')}(x') \cdot \phi^{(\beta')}_{(\mathbf{k}',\mathbf{j}')}(x')dx'.$$

Entsprechendes gilt für die zweite Doppelsumme. Wir erhalten

$$\int_{[0,1]^{d-1}} \sum_{\substack{(\mathbf{l}',\mathbf{i}'): \\ (\mathbf{l}',\mathbf{i}') \underset{\geq}{\leq} (\mathbf{k}',\mathbf{j}'), \\ (k_1,j_1)\times(\mathbf{l}',\mathbf{i}')\in G^d}} \left\{ \int_0^1 \sum_{\substack{(l_1,i_1) \underset{H}{\leq} (k_1,j_1): \\ (l_1,i_1)\times(\mathbf{l}',\mathbf{i}')\in G^d}} u_{(l_1,i_1)\times(\mathbf{l}',\mathbf{i}')} \cdot \phi^{(\alpha_1)}_{(l_1,i_1)}(x_1) \cdot \right.$$

$$\left. \cdot \phi^{(\beta_1)}_{(k_1,j_1)}(x_1)dx_1 \right\} \cdot \phi^{(\alpha')}_{(\mathbf{l}',\mathbf{i}')}(x') \cdot \phi^{(\beta')}_{(\mathbf{k}',\mathbf{j}')}(x')dx' \quad (3.15)$$

oder wie oben abkürzend

$$\int_{[0,1]^{d-1}} \sum_{\substack{(\mathbf{l}',\mathbf{i}'): \\ (\mathbf{l}',\mathbf{i}') \underset{\geq}{\leq} (\mathbf{k}',\mathbf{j}'), \\ (k_1,j_1)\times(\mathbf{l}',\mathbf{i}')\in G^d}} \hat{u}_{(k_1,j_1)\times(\mathbf{l}',\mathbf{i}')} \cdot \phi^{(\alpha')}_{(\mathbf{l}',\mathbf{i}')}(x') \cdot \phi^{(\beta')}_{(\mathbf{k}',\mathbf{j}')}(x')dx'$$

mit den entsprechenden Werten der geschweiften Klammern,

$$\hat{u}_{(k_1,j_1)\times(\mathbf{l}',\mathbf{i}')} := \int_0^1 \sum_{\substack{(l_1,i_1) \underset{H}{\leq} (k_1,j_1), \\ (l_1,i_1)\times(\mathbf{l}',\mathbf{i}')\in G^d}} u_{(l_1,i_1)\times(\mathbf{l}',\mathbf{i}')} \cdot \phi^{(\alpha_1)}_{(l_1,i_1)}(x_1) \cdot \phi^{(\beta_1)}_{(k_1,j_1)}(x_1)dx_1.$$

Die Werte für $\tilde{u}$ werden wir durch eine Rekursion über d berechnen. Hat man diese, so reicht eine Anwendung von TopDown durch diejenige Gitterlinie in Richtung 1, welche als letzte $(d-1)$ Koordinaten gerade $(\mathbf{k}', \mathbf{j}')$ hat, aus, um die erste Doppelsumme für unser $(\mathbf{k}, \mathbf{j})$ zu ermitteln. Analog lassen sich die $\hat{u}$ durch Anwendung von BottomUp auf *alle* Linien entlang Richtung 1 durch das Gitter errechnen. Eine Rekursion behandelt dann die verbleibenden dimensionsabhängigen Integrale der zweiten Doppelsumme. Eine Rekursion des Verfahrens ist dadurch möglich, daß die $(d-1)$ dimensionalen Bestandteile exakt die gleiche Form haben wie das ursprüngliche Problem – es sind im Falle der zweiten Doppelsumme lediglich viele solcher Probleme zu lösen. Auf diese Weise lässt sich $\langle u, \phi_{\mathbf{k},\mathbf{j}} \rangle$ *für alle* Gitterpunkte $(\mathbf{k}, \mathbf{j})$ errechnen, wobei lediglich die eindimensionalen Bestandteile TopDown und BottomUp auf jeweils alle Gitterlinien angewandt werden müssen. Die Richtung der Gitterlinien wird durch die Rekursion gesteuert. Die Rekursion endet, wenn die zu behandelnden Probleme nur noch eindimensional sind. Der Datentransport findet aufgrund dieses Splittings nur über Dünngitterknoten statt. Stellt man bei der adaptiven Verfeinerung sicher, daß in alle d Richtungen die hierarchischen Vorfahren existieren, so funktioniert dieses Prinzip auch adaptiv. Da die TopDown- und BottomUp-Durchläufe durch alle Linien lediglich eine Zeit linear in der Zahl der Knoten benötigen, ist die Gesamtalgorithmik also auch linear in der Zahl der Knoten möglich – die Zahl der Durchläufe ist nur noch von der Dimension des Problems abhängig.

Wir fassen die Algorithmik zusammen. Notwendig ist die Eingabe der eindimensionalen Formen für TopDown und BottomUp in jede Richtung, das Gitter G^d sowie die Funktion u. Das Splitting in die Doppelsummen erfordert das Nutzen eines temporären Vektors der Länge $|\bar{u}| = N$. Schliesslich wendet man die Rekursion für die erste Doppelsumme gefolgt von einem TopDown an, sowie einen BottomUp für die zweite Doppelsumme mit anschliessender Rekursion. Der Ablauf im Detail ist in Schema 3.1.2 beschrieben.

Kostendiskussion

Bei fester Dimension d ist damit eine Matrix-Vektor-Multiplikation mit einer Galerkinmatrix $A_{(\mathbf{l},\mathbf{i}),(\mathbf{k},\mathbf{j})} = a(\phi_{\mathbf{l},\mathbf{i}}, \phi_{\mathbf{k},\mathbf{j}})$ zusammensetzbar aus Summanden der Form (3.10) und effizient in linearer Zeit $O(N)$, $N = |G^d|$, errechenbar mittels Algorithmus 3.1.2. Der Speicherverbrauch ist bedingt durch die d Kopien beschränkt durch $O(dN)$. Da in dieser Arbeit insbesondere die Dimension für die Laufzeit wichtig ist, wollen wir den Vorfaktor $T(d)$ in der Gesamtlaufzeit $O(T(d) \cdot N)$ des Verfahrens untersuchen. Entscheidend ist dabei die Anzahl der Baumdurchläufe. Wir definieren

$$T(d) := \text{Baumtraversierungen bei verbleibender Rekursionstiefe } d$$

und erhalten damit wegen je einem TopDown- und BottomUp-Durchlauf und zwei rekursiven Aufrufen

$$T(d) = 2 + 2 \cdot T(d-1)$$

und $T(1) = 2$. Induktiv zeigt man leicht, daß $T(d) = 2^{d+1} - 2$. Die Gesamtlaufzeit beträgt daher $O(2^d \cdot N)$. Diese enorme Zahl von Baumdurchläufen pro Matrix-Vektor-Multiplikation verhindert den Zugang zu hohen Dimensionen, obwohl die Zahl der Punkte dies

Algorithmus 3.1.2 UniDir für dünne Gitter. Als Eingabe wird erwartet: das Gitter G^d, α und β sowie die Funktion $u = \sum_{(\mathbf{l},\mathbf{i})\in G^d} u_{\mathbf{l},\mathbf{i}}\phi_{\mathbf{l},\mathbf{i}}$. Nach Ende des Algorithmus ist u überschrieben worden, es gilt dann $u_{\mathbf{l},\mathbf{i}} = \langle u, \phi_{\mathbf{l},\mathbf{i}}\rangle$ für alle $(\mathbf{l},\mathbf{i}) \in G^d$. Die Integrationsrichtung ist vom Hauptprogramm aus auf $m = 1$ zu setzen; sie ist lediglich ein interner Rekursionsparameter.

```
c := copy(u)
for all Gitterlinien G_m ⊂ G^d in Richtung m do
   führe BottomUp auf G_m mit den entsprechenden Koeffizienten von u sowie der Form
   ⟨·,·⟩ = ⟨·,·⟩_{α_m,β_m} durch
end for
if m < d then
   rufe UniDir mit G^d, u, den Operatoren sowie Richtung m + 1 auf
   rufe UniDir mit G^d, c, den Operatoren sowie Richtung m + 1 auf
end if
for all Gitterlinien G_m ⊂ G^d in Richtung m do
   führe TopDown auf G_m mit den entsprechenden Koeffizienten von c sowie der Form
   ⟨·,·⟩ = ⟨·,·⟩_{α_m,β_m} durch
end for
u := u + c
```

durchaus erlaubt. Die Notwendigkeit von $O(2^d)$ vielen Baumdurchläufen ergibt sich ausschliesslich aus dem eindimensionalen hierarchischen Väter/ Söhne Splitting (3.13). Dies führt zu den TopDown und BottomUp Durchläufen, die in der oben beschriebenen Weise rekursiv kombiniert werden müssen. Will man die bezüglich der Zahl der Unbekannten lineare Laufzeit beibehalten und dennoch einen besseren Vorfaktor bezüglich der Dimension erreichen, so muß man die Rekursion in Algorithmus 3.1.2 brechen. In dieser Arbeit verfolgen wir diese Richtung gezielt, indem wir zusätzliches Wissen über die Struktur der betrachteten Differentialgleichungen in die Algorithmik einbeziehen.

3.1.2 Einbeziehung von Struktur der Differentialgleichung

Zur Entwicklung einer verbesserten Algorithmik betrachten wir erneut den zugrundeliegenden Differentialoperator zweiter Ordnung

$$L\mathbf{u}(x) = -\sum_{m,r=1}^{d} c_{m,r}\partial^{m,r}\mathbf{u}(x) + \sum_{m=1}^{d} b_m\partial^m\mathbf{u}(x) + \lambda\mathbf{u}(x).$$

Die im vorigen Unterabschnitt entwickelte Algorithmik basiert auf der Anwendung eindimensionaler Teilalgorithmen, die in geeigneter Weise kombiniert werden. Wir sahen bereits, daß die eindimensionale Aufteilung in zwei Teilalgorithmen die teure Rekursion verursacht. Wir wenden uns daher gezielt dem Einfluss eindimensionaler Bestandteile der Bilinearform zu. Wir betrachten zunächst den Diffusionsteil

$$S_{m,r}(u, \phi_{\mathbf{k},\mathbf{j}}) := \int_{\Omega} \partial^m u(x)\partial^r \phi_{\mathbf{k},\mathbf{j}}(x)dx.$$

Wir setzen dazu die Basisdarstellung $u(x) = \sum_{(\mathbf{l},\mathbf{i})\in G^d} u_{\mathbf{l},\mathbf{i}}\phi_{\mathbf{l},\mathbf{i}}(x)$ sowie die Tensorproduktstruktur der $\phi_{\mathbf{l},\mathbf{i}}$ ein und erhalten für $m \neq r$

$$S_{m,r}(u,\phi_{\mathbf{k},\mathbf{j}}) = \sum_{(\mathbf{l},\mathbf{i})\in G^d} u_{\mathbf{l},\mathbf{i}} \int_0^1 \phi'_{l_m,i_m}(x_m)\phi_{k_m,j_m}(x_m)dx_m \cdot \int_0^1 \phi_{l_r,i_r}(x_r)\phi'_{k_r,j_r}(x_r)dx_r \cdot \prod_{\substack{a=1,\ldots,d\\ a\neq m,r}} \int_0^1 \phi_{l_a,i_a}(x_a)\phi_{k_a,j_a}(x_a)dx_a$$

sowie für $m = r$

$$S_{m,m}(u,\phi_{\mathbf{k},\mathbf{j}}) = \sum_{(\mathbf{l},\mathbf{i})\in G^d} u_{\mathbf{l},\mathbf{i}} \int_0^1 \phi'_{l_m,i_m}(x_m)\phi'_{k_m,j_m}(x_m)dx_m \cdot \prod_{\substack{a=1,\ldots,d\\ a\neq m}} \int_0^1 \phi_{l_a,i_a}(x_a)\phi_{k_a,j_a}(x_a)dx_a.$$

Die gesamte Abhängigkeit von der Dimension wird ausgedrückt durch eindimensionale Massenanteile beziehungsweise Identitäten in der starken Form $L\mathbf{u}(x)$. Dieselbe Struktur ergibt sich für den Konvektionsteil und erst recht für den Reaktionsteil.

Um die starke Dimensionsabhängigkeit des Unidirektionalen Prinzips zu brechen, verfolgen wir in dieser Arbeit das Ziel, die Multiplikation mit eindimensionalen Massenanteile ohne Aufteilung in zwei Teilalgorithmen zu realisieren und dabei weiterhin mit $O(N)$ Operationen für einen Durchlauf auszukommen. Nun benötigt aber die bislang verwendete hierarchische Hutbasis zwei Durchläufe für die Multiplikation mit der eindimensionalen Massenmatrix, damit beim Übergang zum Mehrdimensionalen der Datentransport vorgenommen werden kann (siehe [4]). Dies gilt analog auch für Polynombasen höheren Grades, die aus Verallgemeinerung der hierarchischen Hutbasis entstehen, siehe [1, 9].

Deshalb verfolgen wir in dieser Arbeit den Ansatz mit der hierarchischen Hutbasis nicht weiter, sondern wenden uns (eindimensionalen) Ansatzfunktionen zu, die bezüglich des inneren Produktes $\langle u, v\rangle_{L_2} = \int_0^1 u(x)v(x)dx$ zumindest nahezu orthogonal sind. Dazu ist es ausreichend, daß das L_2-Produkt zweier Ansatzfunktionen auf unterschiedlichen Levels verschwindet: in diesem Fall sind keine Interaktionen zwischen den Levels zu betrachten und man braucht lediglich einen Durchlauf, um die Multiplikation mit der Massenmatrix effizient zu errechnen. Diese Eigenschaft bezeichnet man als Semiorthogonalität.

Dabei stoßen wir direkt an die Grenzen des „Unidirektionalen Prinzips“, denn Orthogonalität derartiger stückweise definierter Polynome ist nur mit breiteren Trägern möglich. Grundlage des Unidirektionalen Prinzips ist eine Aufteilung der eindimensionalen Operation in hierarchische Väter und Söhne, was im Mehrdimensionalen schließlich den korrekten Datentransport gewährleistet. Nehmen wir nun aber Ansatzfunktionen, die semiorthogonal bezüglich des L_2-Produktes sind und deshalb breitere Träger als die Hutbasis haben, so benötigen wir für den Diffusions- und Konvektionsteil ebenfalls TopDown- und BottomUp-

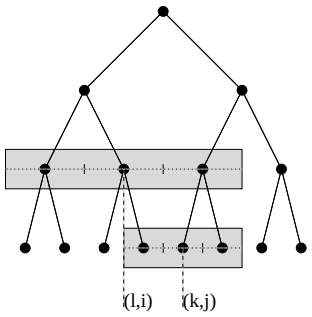

Abbildung 3.5: Bei breiten Ansatzfunktionen überlappen die Träger, obwohl keine hierarchische Vater/Sohn Relation gilt.

Algorithmen, die nun nicht mehr auf diesem notwendigen Splitting beruhen können. Dazu betrachten wir ein Beispiel. Wir gehen von einer Mutterfunktion $\tilde{\phi}$ aus, die auf Level $l > 1$ eine Trägerbreite von $4 \cdot h$ hat. Dann schneiden sich die Träger einer Funktion auf Level l und einer auf Level $l + 1$, obwohl keine direkte hierarchische Relation gilt – vergleiche Abbildung 3.5. Im folgenden Unterabschnitt entwickeln wir daher eine Erweiterung des Unidirektionalen Prinzips für Ansatzfunktionen mit breiten Trägern. Schliesslich verwenden wir die hierarchische Prewaveletbasis aus Kapitel 2, die semiorthogonal bezüglich des L_2-Produktes ist und so die Rekursion vermeidbar macht und gleichzeitig nur Randmodifikationen mit geringem Aufwand erfordert. Die Prewaveletbasis wird bereits in [31] zur Lösung von

$$\begin{aligned} -\Delta u + \lambda u &= f \text{ in } [0,1]^d, \\ u &= 0 \text{ auf } \Gamma = \partial[0,1]^d, \end{aligned}$$

verwendet. Dabei wird die Semiorthogonalität der Prewavelets zur Behandlung der eindimensionalen Massenanteile ausgenutzt und die Orthogonalität der hierarchischen Hutbasis bezüglich der Form

$$(\mathbf{u}, \mathbf{v}) = \int_0^1 \mathbf{u}'(x)\mathbf{v}'(x)dx$$

zur Behandlung der Steifigkeitsmatrix. In dieser Arbeit gehen wir hier weiter und erweitern das allgemeine „Unidirektionale Prinzip“ auf Ansatzfunktionen mit breiteren Trägern.

3.1.3 Erweiterung für Ansatzfunktionen mit breiten Trägern

Wie wir bereits sahen, ist die Durchführung des Unidirektionalen Prinzips für Ansatzfunktionen mit breiten Trägern nicht direkt durchführbar, da die grundlegende Voraussetzung eines Väter/Söhne Splittings nicht gegeben ist. Dies hat zunächst Auswirkungen auf die Auswertung *eindimensionaler* Bilinearformen $\langle \cdot, \cdot \rangle$ mittels TopDown und BottomUp, denn für diese müssen weit mehr überlappende Träger als bei dem bekannten Vater/Sohn Splitting berücksichtigt werden. Konnte man vorher TopDown- und BottomUp-Algorithmen

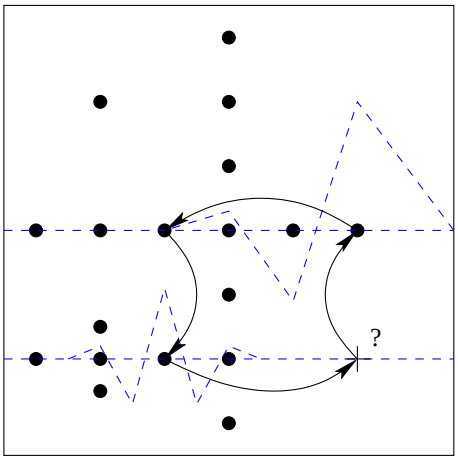

Abbildung 3.6: Für Zwischenergebnisse notwendige Knoten bei Verwendung der Prewaveletbasis.

entwickeln, die für eine feste, gegebene Form $\langle \cdot, \cdot \rangle$ innerhalb einer Preorder - (beziehungsweise Postorder -) Traversierung durch den eindimensionalen Binärbaum errechnet werden konnten, so ist nun bestenfalls eine levelweise Traversierung zu erwarten. Am Ende dieses Abschnitts werden wir Beispiele für die hierarchische Prewaveletbasis sehen, bei der durch levelweise Baumtraversierung die eindimensionalen Matrix-Vektor-Multiplikationen durchgeführt werden können.

Doch die entscheidende Auswirkung der breiteren Träger macht sich beim Datentransport im Mehrdimensionalen bemerkbar. Bei der hierarchischen Basis (oder vergleichbar breiten Trägern) wirken adaptive Gitterstruktur und Trägerbreite derart zusammen, daß durch die aufwändige Rekursion alle Bestandteile transportiert werden können. Die adaptive Gitterstruktur stellt sicher, daß zu jedem Knoten sein Vater (in jede Richtung) im Gitter existiert und die Algorithmik nutzt dies aus.

Nun sind im Eindimensionalen i. a. auch Interaktionen von (l, i) mit hierarchischen Vor- und Nachfahren (k, j) der *benachbarten* Punkte $(l, i \pm 2)$ zu berücksichtigen. Die Auswirkungen auf den mehrdimensionalen Datentransport erläutern wir anhand von Abbildung 3.6. Dargestellt ist ein adaptives dünnes Gitter sowie zwei ausgezeichnete Punkte, deren Interaktion betrachtet wird. Als Basis wird die hierarchische Prewaveletbasis verwendet (siehe Kapitel 2). In Richtung der x_1-Achse überlappen die Träger der an den Punkten fixierten Basisfunktionen und es muss ein Datentransport stattfinden, was in unserem Unidirektionalen Prinzip nur entlang von achsenparallelen Gitterlinien möglich ist. Da das adaptive dünne Gitter jedoch den dafür entscheidenden Punkt nicht enthält, kann eine der beiden Interaktionen nicht berücksichtigt werden. Man beachte, daß in dem betrachteten Beispiel ein reguläres dünnes Gitter den fehlenden Punkt enthalten würde. Diese Beobachtung gilt allgemein, denn die Punkte $(l, i \pm 2 \cdot r), r \in \mathbb{N}$, existieren im Fall regulärer dünner Gitter (wobei in Randnähe entsprechende Modifikationen vorgenommen werden müssen). Ebenso existieren deren Vorfahren. Somit lässt sich das Unidirektionale Prinzip für den regulären Fall ausweiten auf ein Splitting, welches analog zu (3.13) auch

die ferneren Nachbarn berücksichtigt. Es verbleibt, die Algorithmik für den adaptiven Fall zu erweitern.

Dazu passen wir in dieser Arbeit die adaptive Gitterstruktur an. Zusätzlich zu den hierarchischen Vätern fügen wir für jeden neu eingefügten Punkt alle benötigten Nachbarn sowie deren hierarchische Väter in die Datenstruktur ein[2]. Diese zusätzlichen Knoten sind jedoch zu unterscheiden von der Basis für den Funktionenraum, sie haben keinen Anteil an der Darstellung der Funktionen. Statt dessen fungieren sie lediglich als Transporthilfe für die adaptive Algorithmik. Entsprechend sind die dazu gehörenden Koeffizienten zu Anfang und Ende der Matrix-Vektor-Multiplikationen auf Null zu setzen. Die auf Gitterlinien operierenden eindimensionalen Algorithmen arbeiten dabei so, als ob die Zwischenknoten wie alle anderen zum adaptiven Gitter gehören würden. Der weitere Datentransport in die anderen Richtungen wird dabei genau wie oben durch das Unidirektionale Prinzip vorgenommen.

Um die Zahl der zusätzlich notwendigen Punkte im Gitter abzuschätzen, nehmen wir an, daß zu jedem Punkt des Gitters die maximale Zahl von Nachbarn eingefügt werden muss. Da dies in jeder Richtung zu geschehen hat, steigt damit die Zahl der zu verwaltenden Gitterpunkte um einen Faktor von maximal $\text{const} \cdot d$ an. Zusätzlich hat man zu jedem neu eingefügten Knoten alle hierarchischen Vorfahren einzufügen, um den Knoten für die Algorithmik auffindbar zu machen. Diese jedoch sind bereits eingefügt worden. Um dies zu verifizieren, reicht die Untersuchung des eindimensionalen Falls, da neue Knoten ohnehin nur entlang der Koordinatenachsen, aber nicht in die Diagonalen, eingefügt werden. Hat man zu (l, i) beispielsweise den Nachbarn $(l, i+2)$ eingefügt, so ist dessen Vater $(l-1, \frac{i+2}{2})$. Dies ist aber entweder der Vater von (l, i) oder aber einer dessen Nachbarn und wurde entsprechend schon eingefügt. Analoges gilt für fernere Nachbarn. Da im Falle $d > 1$ Nachbarn in jede Richtung eingefügt werden, ist dieses Argument verallgemeinerbar. Somit ist ein adaptives Gitter zu Ansatzfunktionen mit breiten Trägern um maximal einen Faktor $\text{const} \cdot d$ größer als ein adaptives Gitter zur Hutbasis. Die Konstante hängt dabei von der konkreten Trägerbreite ab: bei der Prewaveletbasis überlappen die Träger bis zum zweiten Nachbarn in jede Richtung, es sind also in unserem Fall bis zu vier zusätzliche Punkte pro Richtung und Punkt einzufügen. Diese Abschätzung ist jedoch nicht scharf, da zu vielen Punkte ohnehin schon Nachbarn im Gitter sind (je nach Funktion).

3.1.4 Aufstellung der rechten Seite des Gleichungssystems

Nachdem wir uns in den letzten Abschnitten mit der Multiplikation der Gleichungssystemmatrix mit einem Vektor zu gegebener rechten Seite $\bar{f}$ eingehend beschäftigt haben, wenden wir uns nun der Aufstellung der rechten Seite zu. Dabei ist sowohl die Auswertung aller Integrale

$$\int_{[0,1]^d} f(x)\psi_{\mathbf{k},\mathbf{j}}(x)dx \tag{3.16}$$

[2] Juni 2008: Leider stellte sich heraus, dass die hier vorgeschlagene Modifikation noch nicht ausreichend ist. Zusätzlich müssen auch diagonale Nachbarn einschliesslich derer Vorfahren eingefügt werden. Ich bedanke mich bei Andreas Zeiser für diesen Hinweis.

als auch die Berücksichtigung von inhomogenen Randbedingungen, die die rechte Seite modifizieren, zu beachten (siehe Seite 27).

Eine geeignete Methode, die Integrale (3.16) alle auf einmal zu bestimmen, erhält man durch folgende Vorgehensweise. Wir approximieren (3.16) mittels dem Integral über dem Interpolanten von f, $I_{\tilde{G}}f$, auf einem geeigneten dünnen Gitter $\tilde{G}$. Damit erhalten wir

$$\begin{aligned}\int_{[0,1]^d} f(x)\psi_{\mathbf{k},\mathbf{j}}(x)dx &\approx \int_{[0,1]^d} I_{\tilde{G}}f(x)\psi_{\mathbf{k},\mathbf{j}}(x)dx \\ &= \sum_{(\mathbf{l},\mathbf{i})\in\tilde{G}} f_{\mathbf{l},\mathbf{i}} \int_{[0,1]^d} \psi_{\mathbf{l},\mathbf{i}}(x)\psi_{\mathbf{k},\mathbf{j}}(x)dx \\ &= (M\cdot\bar{F})_{\mathbf{k},\mathbf{j}},\end{aligned}$$

wobei $\bar{F}$ die Koeffizienten des Interpolanten enthalte. Somit reduziert sich die näherungsweise Auswertung aller Integrale (3.16) auf die Interpolation von f auf ein dünnes Gitter mit anschliessender Multiplikation des zugehörigen Koeffizientenvektors mit der Massenmatrix – wobei eine wichtige Besonderheit gegenüber der Multiplikationen in den vorigen Abschnitten besteht. Ist nämlich $f|_\Gamma \neq 0$, so benötigt der Dünngitterinterpolant Randpunkte und die notwendige Massenmatrix muss auch Beiträge zwischen inneren Punkten und Randpunkten berücksichtigen. Somit stellt sich die Frage, ob die Matrix-Vektor-Multiplikation noch entsprechend den Resultaten des vorigen Abschnittes effizient durchführbar ist, denn die Orthogonalitätseigenschaft der Prewaveletbasis besteht nicht zwischen Randfunktionen und Funktionen im Inneren. Die entscheidende Beobachtung ist nun, daß die Integrale (3.16) nur für Punkte $(\mathbf{k},\mathbf{j})$ *im Innern* des Gebiets benötigt werden – die Matrix M hat in diesem Fall also die rechteckige Struktur $M \in \mathbb{R}^{N_I\times N_R}$, wobei N_I die Zahl der inneren Punkte in $\tilde{G}$ bezeichnet und N_R die Gesamtgröße des Gitters, $N_R = |\tilde{G}|$. Durch diese rechteckige Struktur besteht der eindimensionale Algorithmus, welcher auch Massenanteile zwischen Rand- und inneren Punkten berücksichtigt, nur aus *einem* Teilalgorithmus, denn TopDown und BottomUp bearbeiten jeweils den Teil ober- beziehungsweise unterhalb der Matrixdiagonale. Sobald nur ein eindimensionaler Algorithmus notwendig ist, vermeidet man die teure Rekursion und die Operation ist durchführbar, wobei beim Durchlauf nun auch Randpunkte zu traversieren sind. Damit ist die Laufzeit dieser Art von Integration für die rechte Seite trotz fehlender Orthogonalität von der Komplexität $O(d\cdot|\tilde{G}|)$. Ist die Anzahl der Punkte in $\tilde{G}$ größenordnungsmäßig gleich der Zahl der Punkte in G, so entspricht dies den Kosten einer einzigen Multiplikation mit der Massenmatrix.

In [14] wird gezeigt, daß bei geeignetem $\tilde{G}$ diese Technik die diskrete Lösung des Galerkinverfahrens nach dem ersten Lemma von Strang nur in der Größenordnung des Diskretisierungsfehlers verschlechtert. Dazu muss f die Glattheitsanforderungen an die Interpolation erfüllen, also $f \in X_0(\Omega)$ (siehe Abschnitt 2.2). Für $\tilde{G}$ erscheint als optimale Wahl ein adaptiv an f angepasstes Gitter. In der Praxis reicht es meist jedoch aus, dasselbe Gitter wie zur Diskretisierung der Lösung zu verwenden (oder dieses zu verfeinern, siehe [8]).

Eine sehr ähnliche Methode lässt sich zur Berücksichtigung von Dirichletrandbedingungen anwenden. Der kontinuierlichen Argumentation für Dirichletrandbedingungen auf

Seite 27 folgend, diskretisieren wir unsere Randbedingungen mittels der Randpunkte des dünnen Gitters. Die diskrete Lösung des Systems ist dann

$$u = u_I + u_R,$$

wobei u_R durch die Dirichletvorgaben fixiert ist und durch Beschränkung der Hierarchisierungsoperationen auf den Rand effizient ermittelt werden kann. Die zu inneren Punkten gehörenden Koeffizienten von u_R verschwinden entsprechend, daher kann u_R als ein diskreter Fortsetzungsoperator vom Rand ins Innere gesehen werden. Von der rechten Seite muß dann der Anteil $a(u_R, \psi_{\mathbf{k},\mathbf{j}})$ für alle $(\mathbf{k}, \mathbf{j})$ im Innern des Gitters subtrahiert werden. Für diesen gilt aber

$$a(u_R, \psi_{\mathbf{k},\mathbf{j}}) = (\tilde{A}^T \cdot \bar{u}_R)_{\mathbf{k},\mathbf{j}}$$

mit einer um die Randinteraktionen erweiterten Steifigkeitsmatrix $\tilde{A}$. Diese ist wiederum rechteckig, was mit derselben Argumentation wie für die Integration der rechten Seite die Laufzeit von $O(\text{Poly}(d) \cdot N_R)$ ermöglicht. Da die Diskretisierung der Dirichletrandbedingungen einen Fehler maximal von der Ordnung des Interpolanten $I_{G^d}\mathbf{u}$ der kontinuierlichen Lösung einführt, bleibt auch hier die Diskretisierungsordnung des Verfahrens erhalten.

3.2 Eindimensionale Prewaveletalgorithmen

Mit dem im vorigen Abschnitt gewonnenen „Unidirektionalen Prinzip“ ist es uns nun möglich, die notwendigen Matrix-Vektor-Produkte nur durch Verwendung eindimensionaler Algorithmen durchzuführen. Inhalt dieses Abschnitts ist die Entwicklung dieser eindimensionalen Verfahren für die hierarchische Prewaveletbasis aus Abschnitt 2.2.2.

Zur Lösung einer linearen elliptischen Differentialgleichung zweiter Ordnung mit konstanten Koeffizienten benötigen wir Algorithmen für die Multiplikation der eindimensionalen Massenmatrix M, Konvektionsmatrix C und Laplacematrix S mit dem Koeffizientenvektor einer Dünngitterfunktion $u = \sum_{(l,i)\in G_n^1} u_{l,i}\psi_{l,i}$. Dabei ist

$$\begin{aligned}
\int_0^1 u(x)\psi_{k,j}(x)dx &= \sum_{(l,i)\in G_n^1} m_{(l,i),(k,j)} \cdot u_{l,i} &&= (M^T\bar{u})_{k,j},\\
\int_0^1 u'(x)\psi_{k,j}(x)dx &= \sum_{(l,i)\in G_n^1} c_{(l,i),(k,j)} \cdot u_{l,i} &&= (C^T\bar{u})_{k,j},\\
\int_0^1 u'(x)\psi'_{k,j}(x)dx &= \sum_{(l,i)\in G_n^1} s_{(l,i),(k,j)} \cdot u_{l,i} &&= (S^T\bar{u})_{k,j}
\end{aligned}$$

mit den entsprechenden Matrixeinträgen $m_{(l,i),(k,j)}$, $c_{(l,i),(k,j)}$ und $s_{(l,i),(k,j)}$. Da die Matrizen M und S symmetrisch sind, spielt die Transposition nur bei der Matrix C eine Rolle. Die Matrix C ist antisymmetrisch, wie man durch partielle Integration verifiziert. Daher hat man mit der Matrix-Vektor-Multiplikation $C^T\bar{u}$ auch die Multiplikation mit C implementiert. Man benötigt sowohl C als auch C^T, um im höherdimensionalen gemischte zweite Ableitungen in der schwachen Form zu diskretisieren.

3.2.1 Massenanteile

Aufgrund der Semi-Orthogonalität ist die Durchführung der Multiplikation $M\bar{u}$ sehr einfach. Aus [31] entnehmen wir die noch zu beachtenden Anteile auf einem Level. Zu Berücksichtigen sind die Anteile

$$m_{(1,1),(1,1)} = \frac{1}{3}, \qquad m_{(l,1),(l,1)} = m_{(l,2^l-1),(l,2^l-1)} = \frac{44}{75} \cdot h_l,$$
$$m_{(2,1),(2,3)} = \frac{1}{25}, \qquad m_{(l,1),(l,3)} = m_{(l,2^l-1),(l,2^l-3)} = \frac{11}{75} \cdot h_l,$$
$$m_{(l,i),(l,i)} = \frac{18}{25} \cdot h_l, \qquad m_{(l,i),(l,i\pm 2)} = \frac{2}{15} \cdot h_l,$$
$$m_{(l,i),(l,i\pm 4)} = -\frac{1}{75} \cdot h_l.$$

Zusätzlich benötigt man zur Diskretisierung der rechten Seite auch Randanteile. Wir erhalten in diesem Fall

$$m_{(0,0),(l,1)} = \frac{4}{10} \cdot h_l, \qquad m_{(0,1),(l,2^l-1)} = \frac{4}{10} \cdot h_l.$$

Alle nicht angegebenen Matrixeinträge verschwinden.

3.2.2 Laplaceteil

Zur Multiplikation $S \cdot \bar{u}$ benötigen wir eine Aufteilung in TopDown und BottomUp. Aufgrund des breiten Trägers der Prewavelets gilt auch hier, daß Einträge von benachbarten Punkten auf einem Level zu berücksichtigen sind, also $s_{(l,i),(l,i\pm r\cdot 2)} \neq 0$. Dies vererbt sich auch auf die hierarchischen Nachfahren von sowohl (l,i) als auch den benachbarten Knoten $(l, i \pm r \cdot 2)$. Um alle diese Bestandteile zu berücksichtigen, betrachten wir das Integral über die Funktionen der *nodalen Hutbasis* auf allen Leveln und setzen die Prewaveletintegrale durch Ausnutzung des Sterns

$$\begin{bmatrix} \frac{1}{10} & -\frac{6}{10} & 1 & -\frac{6}{10} & \frac{1}{10} \end{bmatrix} \tag{3.17}$$

(beziehungsweise den entsprechenden Randmodifikationen) zusammen. Dazu bezeichnen wir die entsprechenden Integralbeiträge in der Hutbasis mit

$$\tilde{s}_{(l,i),(k,j)} := \int_0^1 \phi'_{l,i}(x)\phi'_{k,j}(x)dx,$$

wobei wir hier auch Beiträge von Erzeugendensystemfunktionen einbeziehen (d. h. i und j dürfen gerade sein). Diese sind nun leicht zu errechnen. Da S symmetrisch ist, nehmen wir ohne Einschränkung $k \leq l$ an. Dann gilt

$$\tilde{s}_{(l,i),(k,j)} = \begin{cases} 2 \cdot \frac{1}{h_k} & \text{falls } x_{l,i} = x_{k,j} \\ -\frac{1}{h_k} & \text{falls } x_{k,j} \pm h_k = x_{l,i} \\ 0 & \text{sonst.} \end{cases} \tag{3.18}$$

Die gesuchten Integrale hängen also immer nur von der *größeren* Maschenweite h_k ab. Basierend auf dieser Beobachtung können wir nun die Multiplikation mit der Prewaveletmatrix S^T realisieren. Dazu teilen wir in TopDown - und BottomUp-Anteile auf mittels

$$S^T =: S_t + S_b,$$

wobei

$$(S_t)_{(l,i),(k,j)} = \begin{cases} s_{(k,j),(l,i)} & \text{falls } (l \leq k), \\ 0 & \text{sonst} \end{cases}$$

eine untere Dreiecksmatrix ist und S_b entsprechend mit der Bedingung $l > k$ definiert wird.

TopDown-Teil

Wir entwickeln zunächst einen Algorithmus zur Errechnung des Produktes $\bar{r} := S_t \cdot \bar{u}$. Es gilt

$$r_{k,j} = (S_t \cdot \bar{u})_{k,j} = \sum_{\substack{(l,i)\in G_n^1, \\ l\leq k}} s_{(l,i),(k,j)} u_{l,i},$$

wobei wir algorithmisch nur die nicht-verschwindenden Einträge berücksichtigen. Da die Integrale über den nodalen Hutfunktionen immer nur von den größeren Trägern abhängen, ist die Idee, von oben (also große Träger, Level 1) nach unten (Level n) durch den Baum levelweise zu traversieren und dabei immer nur Faktoren $\frac{1}{h_l}$ an die größeren Levels weiterzureichen. Bei dem „weiterreichen“ werden jeweils die Prewaveletgewichte (3.17) berücksichtigt. Anschliessend kombiniert man schließlich die von niedrigen Levels akkumulierten Werte gemäß (3.18). Für die Einbeziehung der Diagonalwerte (d. h. $l = k$) in dem TopDown-Verfahren führen wir dies auch für Nachbarn auf einem Level aus.

Da bei Multiplikation zweier Prewavelets je 5 Hutfunktionen eingehen, sind pro Matrixeintrag ≈ 25 Werte richtig zu berücksichtigen (mit Modifikationen für randnahe Prewavelets). An dieser Stelle sei aufgrund der aufwändigen Rechnung nur das Resultat dieser Vorgehensweise festgehalten. Dazu führen wir zunächst zu jedem Knoten aus dem eindimensionalen Erzeugendensystem E_n^1 Hilfsknoten $t_{k,j}$ ein, die die Akkumulation der inversen Maschenweiten zusammen mit Prewaveletgewichten realisieren. Diese erhalten je nach Position im Baum die Werte

$$t_{k,j} := \begin{cases} u_{k,j}\frac{32}{10}\frac{1}{h_k} + u_{k,j\pm 2}\frac{8}{10}\frac{1}{h_k} & (k,j) \in G_n^1 \\ u_{k,j}\frac{24}{10}\frac{1}{h_k} + u_{k,j\pm 2}\frac{8}{10}\frac{1}{h_k} & (k,j) \in G_n^1 \text{ und randnah} \\ t_{k-1,\frac{j}{2}} - u_{k,j\pm 1}\cdot\frac{23}{10}\frac{1}{h_k} - u_{k,j\pm 3}\frac{1}{10}\frac{1}{h_k} & (k,j) \notin G_n^1 \\ t_{k-1,\frac{j}{2}} - u_{k,j\pm 1}\cdot\frac{22}{10}\frac{1}{h_k} - u_{k,j\pm 3}\frac{1}{10}\frac{1}{h_k} & (k,j) \notin G_n^1 \text{ und } (k, j\pm 1) \text{ randnah.} \end{cases}$$

Dabei bezeichnen wir mit „randnah“ den Fall $j \in \{1, 2^k - 1\}$. Falls ein Multiindex $(k, j \pm r)$ nicht im Gitter enthalten ist, entfällt der entsprechende Anteil. Mithilfe dieser temporärer Knoten ermitteln wir nun während einer levelweisen Baumtraversierung die Werte $r_{k,j}$

mittels

$$
\begin{aligned}
r_{k,j} = &-\frac{6}{10} t_{k-1,\frac{j\pm1}{2}} \\
&+ \frac{1}{h_k} u_{k,j} \cdot \begin{cases} \frac{612}{100} & \text{falls } (k,j) \text{ randfern} \\ \frac{356}{100} & \text{falls } (k,j) \text{ randnah} \end{cases} \\
&+ \frac{1}{h_k} u_{k,j\pm2} \cdot \begin{cases} \frac{256}{100} & \text{falls } (k,j) \text{ und } (k,j\pm2) \text{ randfern} \\ \frac{242}{100} & \text{falls } (k,j) \text{ randfern und } (k,j\pm2) \text{ randnah} \\ \frac{242}{100} & \text{falls } (k,j) \text{ randnah} \\ \frac{228}{100} & \text{falls } k = 2 \end{cases} \\
&+ \frac{1}{h_k} u_{k,j\pm4} \frac{14}{100} \\
r_{1,1} = &\frac{2}{h_1} = 4.
\end{aligned}
$$

Die „levelweise“ Baumtraversierung entspricht dabei lediglich einer Breitensuche in dem Binärbaum.

BottomUp-Teil

Wir wenden uns nun der Multiplikation $\bar{r} := S_b \cdot \bar{u}$ zu. Analog zu dem Vorgehen bei der Entwicklung von $S_t \cdot \bar{u}$ schreiben wir zunächst

$$r_{k,j} = (S_t \cdot \bar{u})_{k,j} = \sum_{\substack{(l,i)\in G_n^1, \\ l>k}} s_{(l,i),(k,j)} u_{l,i}.$$

Da die Matrix S symmetrisch ist, könnten wir $S_b \cdot \bar{u}$ durch Ausnutzung von $S_b = S_t^T$ (bis auf die Diagonale) ausdrücken. Dazu müsste man den zu $S_t \cdot \bar{u}$ gehörenden Algorithmus gewissermaßen umkehren. Aufgrund der vielen Fallunterscheidungen ist dies jedoch deutlich komplizierter als die folgende Vorgehensweise.

Wir nutzen erneut aus, daß die Integrale über Hutfunktionen gemäß (3.18) nur von der größeren Maschenweite abhängen. Wenn wir nun durch unseren Binärbaum traversieren und den Anteil $r_{k,j}$ ermitteln wollen, benötigen wir von den Knoten (l,i) mit $l > k$ nichts anderes als die Prewaveletgewichte, die zu $(l, i\pm r), r \in 0,1,2$ assoziiert sind. Wir definieren daher wiederum temporäre Hilfsknoten $t_{k,j}$, die alle Prewaveletgewichte aufsummieren, deren Hutfunktion an der Stelle $x_{k,j}$ zentriert ist. Dann müssen wir zur Ermittlung von $r_{k,j}$ gemäß (3.18) nur noch entscheiden, ob mit $\frac{2}{h_k}$ oder $-\frac{1}{h_k}$ multipliziert werden muß, sowie die Prewaveletgewichte von $\psi_{k,j}$ berücksichtigen.

Wir fassen das Resultat im Folgenden zusammen. Die temporären Hilfsknoten werden nur an Punkten benötigt, die im Erzeugendensystem, aber nicht in der Basis vorkommen. Wir definieren für $(k,j) \in E_n^1 \setminus G_n^1$

$$t_{k,j} := -\frac{6}{10} \cdot u_{k,j\pm1} + t_{k+1,2\cdot j}.$$

Damit erhalten wir schließlich für randferne $(k,j) \in G_n^1$ das Resultat

$$\begin{aligned} r_{k,j} = &\left(\frac{2}{h_k} t_{k+1,2i} - \frac{1}{h_k} t_{k+1,2(j\pm1)}\right) \\ &- \frac{6}{10}\left(-\frac{1}{h_k} t_{k+1,2(j\pm2)} + \frac{2}{h_k} t_{k+1,2(j\pm1)} - 2\frac{1}{h_k} t_{k+1,2i}\right) \\ &+ \frac{1}{10}\left(-\frac{1}{h_k} t_{k+1,2(j\pm1)} + \frac{2}{h_k} t_{k+1,2(j\pm2)} - \frac{1}{h_k} t_{k+1,2(j\pm3)}\right). \end{aligned}$$

Für randnahe $(k,j) \in G_n^1$ modifizieren wir die Gewichte leicht und erhalten

$$\begin{aligned} r_{k,j} = &\frac{9}{10} \cdot \left(\frac{2}{h_k} t_{k+1,2i} - \frac{1}{h_k} t_{k+1,2(j\pm1)}\right) \\ &- \frac{6}{10}\left(-\frac{1}{h_k} t_{k+1,2(j\pm2)} + \frac{2}{h_k} t_{k+1,2(j\pm1)} - 1\frac{1}{h_k} t_{k+1,2i}\right) \\ &+ \frac{1}{10}\left(-\frac{1}{h_k} t_{k+1,2(j\pm1)} + \frac{2}{h_k} t_{k+1,2(j\pm2)} - \frac{1}{h_k} t_{k+1,2(j\pm3)}\right). \end{aligned}$$

Um die Ergebnisse während eines Baumdurchlaufes zu akkumulieren, muß man levelweise von dem höchsten zum niedrigsten Level traversieren, d. h. von n bis 1. Dazu führt man eine Breitensuche durch und traversiert anschliessend die besuchten Punkte vom höchstens zum niedrigsten Level. Wie bereits bei der Einführung des TopDown-Teils sind Beiträge ausserhalb des adaptiven Gitters als 0 anzunehmen. Damit ist auch BottomUp in Zeit $O(|G_n^1|)$ durchführbar.

3.2.3 Konvektionsteil

Wir implementieren hier die Matrix-Vektor-Multiplikation $C^T \bar{u}$ wie sie zur Auswertung der Form $\int_0^1 u'(x)\psi_{k,j}dx$ benötigt wird. Der Vorgehensweise für den Laplaceteil folgend teilen wir in TopDown- und BottomUp-Anteile auf,

$$C^T =: C_t + C_b,$$

mit $C = (c_{(l,i),(k,j)})$ und $c_{(l,i),(k,j)} = \int_0^1 \psi'_{l,i}(x)\psi'_{k,j}(x)dx$ und ermitteln jeweils geeignete Algorithmen.

TopDown-Teil

Wir wenden uns zunächst der Errechnung der Multiplikation $\bar{r} := C_t\bar{u}$ mit

$$r_{k,j} = (C_t \cdot \bar{u})_{k,j} = \sum_{\substack{(l,i)\in G_n^1,\\ l\le k}} c_{(l,i),(k,j)} u_{l,i}$$

zu. Wegen $l \le k$ ist $\psi'_{l,i}$ auf jedem Teilintervall auf Level k konstant. Damit setzen sich die einzelnen Matrixeinträge aus Werten von Einzelintervallen zusammen, auf denen jeweils

eine Konstante mit dem Integral über $\psi_{k,j}$ auf dem Intervall multipliziert wird. Dazu schreiben wir

$$\int_0^1 \psi'_{l,i}(x)\psi_{k,j}(x)dx = \sum_{m=0}^{5} c_{k,j-3+m} I^m_{k,j}, \tag{3.19}$$

wobei wir jedem der 2^k Intervalle auf Level k je einen Wert $c_{k,j}, j \in \{0,\dots,2^k-1\}$ zuweisen. Der Index (k,j) bezeichnet damit den *linken* Endpunkt des Intervalls $[x_{k,j}, x_{k,j+1}]$. Der Wert $I^m_{k,j}$ beschreibt dabei das Integral über $\psi_{k,j}$ über genau einem der 6 Teilintervalle, auf denen $\psi_{k,j}$ nicht verschwindet. Wir definieren entsprechend für $m \in \{0,\dots,5\}$

$$I^m_{k,j} := \int_{x_{k,j}-3h_k+m\cdot h_k}^{x_{k,j}-3h_k+(m+1)\cdot h_k} \psi_{k,j}(x)dx.$$

Somit bezeichnet m für ein randfernes Prewavelet $\psi_{k,j}$ das zugehörige Intervall $[x_{k,j}-3h_k+m\cdot h_k, x_{k,j}-3h_k+(m+1)\cdot h_k]$. Für einen solchen Punkt errechnet man die Integralwerte $I^m := I^m_{k,j}$ zu

$$\begin{aligned} I^0 = I^5 &= \frac{1}{20}h_k, \\ I^1 = I^4 &= -\frac{5}{20}h_k, \\ I^2 = I^3 &= \frac{4}{20}h_k. \end{aligned}$$

Für randnahe Punkte ändert sich der Wert für das randnächste Intervall zu $\frac{9}{20}h_k$, der Wert zu dem dazu benachbarten Intervall erhält den Wert $-\frac{3}{20}h_k$.

Die Idee ist nun, die Argumentation auf *alle* Matrixeinträge (l,i) mit $l \le k$ durchzuführen und lediglich die Konstanten $c_{k,j}$ während eines Baumdurchlaufes zu akkumulieren. Dies ist möglich, da es sich immer um Konstanten handelt, die aus dem Matrixeintrag $c_{(l,i),(k,j)}$ herausgezogen werden. Dieser Argumentation folgend lassen sich die Intervallwerte $c_{k,j}$ für alle $j = 0,\dots,2^k-1$ durch eine levelweise Traversierung vom niedrigsten Level zum höchsten (TopDown) wie folgt ermitteln. Wir erhalten

$$\begin{aligned} c_{k,j} = c_{k-1,j/2} &+ \frac{1}{h_k}\cdot \\ &\cdot \begin{cases} -\frac{16}{10}u_{k,j} - \frac{1}{10}u_{k,j-2} - \frac{7}{10}u_{k,j+2} & \text{falls } (k,j) \in G^1_n \\ -\frac{7}{10}u_{k,j-1} + \frac{16}{10}u_{k,j+1} + \frac{1}{10}u_{k,j+3} & \text{falls } (k,j+1) \in G^1_n, \end{cases} \end{aligned}$$

wobei wir daran erinnern, daß (k,j) den linken und $(k,j+1)$ den rechten Endpunkt des Intervalles beschreibt. Die Koeffizienten vor den betrachteten hierarchischen Überschüssen $u_{k,\tilde{j}}$ ergeben sich dabei einfach als Steigung von $\psi_{k,\tilde{j}}$ auf dem gerade betrachteten Intervall zu $c_{k,j}$. Falls $(k,\tilde{j})$ randnah ist, sind Modifikationen an die Steigungen vorzunehmen.

Damit lassen sich die gesuchten Werte für den TopDown-Teil errechnen zu

$$r_{k,j} = \sum_{m=0}^{5} c_{k,j-3+m} I^m_{k,j}.$$

BottomUp-Teil

Zur Ermittlung eines Algorithmus für den Anteil $\bar{r} := C_b \cdot \bar{u}$,

$$r_{k,j} = (C_b \cdot \bar{u})_{k,j} = \sum_{\substack{(l,i)\in G_n^1,\\ l>k}} c_{(l,i),(k,j)} u_{l,i},$$

verfolgen wir hier einen ähnlichen Ansatz wie für den Laplaceteil. Wir kombinieren Integrale über Hutfunktionen aus dem Erzeugendensystem,

$$\tilde{c}_{(l,i),(k,j)} := \int_0^1 \phi'_{l,i}(x)\phi_{k,j}(x)dx.$$

In dem hier betrachteten Fall $l > k$ gibt es drei verschiedene Werte für $\tilde{c}_{(l,i),(k,j)}$. Bezeichne dazu $K_{k,j}^L := [x_{k,j-1}, x_{k,j}]$ den linken Teil des Trägers von $\phi_{k,j}$ und $K_{k,j}^R := [x_{k,j}, x_{k,j+1}]$ entsprechend den rechten Teil. Damit erhalten wir

$$\tilde{c}_{(l,i),(k,j)} = \begin{cases} 0 & \text{falls } x_{l,i} = x_{k,j}, \\ \pm\frac{h_l}{h_k} & \text{falls } \operatorname{supp}\phi_{l,i} \subset K_{k,j}^L \text{ oder } \operatorname{supp}\phi_{l,i} \subset K_{k,j}^R, \\ \pm\frac{h_l}{2h_k} & \text{falls } x_{l,i} = x_{k,j+1} \text{ oder } x_{l,i} = x_{k,j-1}. \end{cases}$$

Die Vorzeichen bestimmen sich dabei anhand der Koordinaten $x_{l,i}$ und $x_{k,j}$: für $x_{l,i} < x_{k,j}$ ist das Vorzeichen negativ, ansonsten positiv.

Ein einzelnes Integral $\int_0^1 \psi'_{l,i}(x)\psi_{k,j}(x)dx$ lässt sich mit diesem Wissen zusammentragen: Da immer eines der Hut-Integrale verschwindet, hat man $5 \cdot 4 = 20$ Integrale zu untersuchen. Ein Beispiel: für $(l,i) = (4,9)$ und $(k,j) = (3,5)$ und somit $h_k = \frac{1}{2^3}, h_l = \frac{1}{2^4}$ erhalten wir

$$\int_0^1 \psi'_{4,9}(x) \cdot \psi_{3,5}(x) = \begin{bmatrix} \frac{1}{10} & -\frac{6}{10} & 1 & -\frac{6}{10} & \frac{1}{10} \end{bmatrix} \cdot \begin{bmatrix} \frac{1}{10}\frac{h_l}{h_k} - \frac{6}{10}\frac{h_l}{2h_k} \\ -\frac{1}{10}\frac{h_l}{h_k} + \frac{h_l}{h_k} - \frac{6}{10}\frac{h_l}{2h_k} \\ +\frac{6}{10}\frac{h_l}{2h_k} - \frac{h_l}{h_k} + \frac{1}{10}\frac{h_l}{h_k} \\ +\frac{6}{10}\frac{h_l}{2h_k} - \frac{1}{10}\frac{h_l}{h_k} \\ 0 \end{bmatrix}.$$

Nun ist unser Ziel die Errechnung aller Komponenten der Matrix-Vektor-Multiplikation, daher akkumulieren wir die Hut-Integrale während eines Bottom-Up Durchlaufes durch die Datenstruktur. Dazu klammern wir jeweils $h_l \cdot \frac{1}{h_k}$ aus und müssen letztlich in richtiger Weise die Masken

$$\begin{bmatrix} \frac{1}{10} & -\frac{6}{10} & 1 & -\frac{6}{10} & \frac{1}{10} \end{bmatrix}, \begin{bmatrix} -\frac{1}{2} & -1 & 0 & 1 & \frac{1}{2} \end{bmatrix}$$

anwenden. Zur Akkumulation der Hutintegrale $\tilde{s}_{(l,i),(k,j)}$ definieren wir wieder Hilfsknoten aus dem Erzeugendensystem,

$$t_{k,j} = \begin{cases} h_l \cdot \left(u_{k,j} + \frac{1}{10}u_{k,j\pm 2}\right) + t_{k+1,2j\pm 1} + t_{k+1,2j} & \text{falls } (k,j) \in G_n^1 \\ -h_l \cdot \frac{6}{10}u_{k,j\pm 1} + t_{k+1,2j} & \text{sonst.} \end{cases}$$

Die Konstanten vor den einzelnen Überschüssen müssen für randnahe Knoten gemäß den Prewaveletgewichten angepasst werden. Der Gesamtalgorithmus lässt sich dann mittels

$$r_{k,j} = \frac{1}{h_k} \begin{bmatrix} \frac{1}{10} \\ -\frac{6}{10} \\ 1 \\ -\frac{6}{10} \\ \frac{1}{10} \end{bmatrix}^T \cdot \begin{bmatrix} t_{k+1,2(j-3)} & t_{.,.+1} & t_{.,.+2} & t_{.,.+3} & t_{.,.+4} \\ t_{k+1,2(j-2)} & \cdots & \cdots & \cdots & t_{.,.+4} \\ t_{k+1,2(j-1)} & \cdots & \cdots & \cdots & t_{.,.+4} \\ t_{k+1,2j} & \cdots & \cdots & \cdots & t_{.,.+4} \\ t_{k+1,2(j+1} & \cdots & \cdots & \cdots & t_{.,.+4} \end{bmatrix} \cdot \begin{bmatrix} -\frac{1}{2} \\ -1 \\ 0 \\ 1 \\ \frac{1}{2} \end{bmatrix}$$

und dem Sonderfall

$$r_{1,1} = 2 \cdot (-t_{2,1} + t_{2,3})$$

ausdrücken. Für randnahe Prewavelets sind dazu Anpassungen in den Masken sowie Indizes vorzunehmen. Die Werte der kleinen Matrixmultiplikation in dieser Darstellung lassen sich dabei mehrfach auf einem Level verwenden.

3.3 Zusammenfassung und Bemerkungen

Im letzten Abschnitt haben wir basierend auf einer Analyse des Unidirektionalen Prinzips aus [1, 4, 9] eine signifikante Verbesserung der Geschwindigkeit für die Multiplikation mit der Steifigkeitsmatrix ermöglicht. Durch Einsatz von L_2-semiorthogonalen Ansatzfunktionen und damit verbundenen Erweiterungen des Unidirektionalen Prinzips erreichen wir eine Reduktion der Laufzeit von $O(2^d \cdot N)$ auf $O(d \cdot N)$. Gleichzeitig reduziert sich der Speicherverbrauch von $O(d \cdot N)$ auf eine konstante Zahl, also $O(N)$. Die Allgemeinheit des Verfahrens bleibt davon unberührt.

An dieser Stelle fügen wir einige abschließende Bemerkungen über die Zusammensetzung der Steifigkeitsmatrizen durch mehrere Anwendungen des Unidirektionalen Prinzips sowie Optimierungsmöglichkeiten und alternative Beschreibungen an.

Während wir in obiger Einführung des Unidirektionalen Prinzips insbesondere die Gewährleistung des Datentransports und eine präzise algorithmische Formulierung im Blick hatten, ist auch eine andere Sichtweise denkbar, die wir aus [4] entnehmen. Wir erinnern uns, daß das Matrix-Vektor-Produkt aus Summanden der Form

$$\langle u, \phi_{\mathbf{k},\mathbf{j}} \rangle = \int\limits_{[0,1]^d} D^\alpha u(x) D^\beta \phi_{\mathbf{k},\mathbf{j}}(x) dx$$

zusammengesetzt werden kann, vgl. (3.3). Dies entspricht der Multiplikation mit einer bestimmten Matrix A, sodaß $(A \cdot \bar{u})_{k,j} = \langle u, \phi_{k,j} \rangle$. Im eindimensionalen Fall können wir die Unbekannten levelweise anordnen. Damit befindet sich beispielsweise der Matrixeintrag $a_{(l,i),(l+1,2i+1)}$ *oberhalb* der Diagonale von A, weil (l, i) einen kleineren Index hat als sein Sohn $(l + 1, 2i + 1)$. Betrachten wir, wie zu Anfang dieses Kapitels,

$$\langle \underline{u}, \phi_{k,j} \rangle = \sum_{(l,i),l>k} a_{(k,j),(l,i)} u_{l,i},$$

so entspricht dies exakt der Durchführung von BottomUp. Damit ist die eingangs durchgeführte Zerlegung in hierarchische Vor- und Nachfahren nichts anderes als eine Zerlegung von A in obere und untere Dreiecksmatrizen,

$$A = L + U,$$

vgl. [4]. Im Mehrdimensionalen gibt es nun für jede Richtung eine Matrix $A_i = L_i + U_i$. Diese Matrizen bestehen aus Integralen in nur eine Richtung entlang Gitterlinien. Damit ist $A = A_1 \cdots A_d$. Entscheidend ist, dass die Reihenfolge der Multiplikation willkürlich ist – die Integration in eine Richtung ist aufgrund der Produktstruktur unabhängig von den anderen und kann daher vertauscht werden. Das Unidirektionale Prinzip nutzt dies zum Datentransport aus, indem das Splitting

$$A = \prod_{m=1}^{d} A_m = \prod_{m=1}^{d-1} A_m \cdot U_d + L_d \cdot \prod_{m=1}^{d-1} A_m \tag{3.20}$$

rekursiv zur Hintereinanderausführung entlang von Gitterlinien genutzt wird (vergleiche mit der algorithmischen Beschreibung in Schema 3.1.2, siehe auch [4]).

Die in dieser Arbeit vorgestellte Verbesserung basiert nun darauf, daß nur für *konstant* viele Richtungen m ein Splitting $A_m = L_m + U_m$ notwendig ist. Alle anderen Teilmatrizen haben lediglich Diagonalgestalt (oder zumindest nahezu), wodurch der wesentliche Teil der Rekursion wegfällt. Unsere Untersuchung zeigt, daß durch leichte Modifikation der Gitterstruktur und Anpassung des algorithmischen Prinzips der wichtige Datentransport auch mit den breiten Trägern durchführbar ist.

Durch Permutation der Reihenfolge lassen sich Optimierungen durchführen. Dies führen wir exemplarisch am Beispiel der Laplacematrix vor. Die Laplacematrix besteht aus d Summanden, von denen jeder wiederum aus $(d-1)$ Massenanteilen M_k und einem Diffusionsteil $A_m = L_m + U_m$ besteht, also

$$R := \sum_{m=1}^{d} \Big(\prod_{k=1}^{m-1} M_k \cdot A_m \cdot \prod_{k=m+1}^{d} M_k \Big),$$

vgl. Abschnitt 3.1.2. Durch d-fache Anwendung des Unidirektionalen Prinzips ist also die Multiplikation $R \cdot \bar{u}$ in Zeit $O(d^2 N)$ möglich.

Für jeden Summand kann aber zunächst ein Faktor 2 in der Laufzeit einfach durch Verschieben von A_m an das Ende der Bearbeitungskette eingespart werden, denn

$$\prod_{k=1}^{m-1} M_k \cdot A_m \cdot \prod_{k=m+1}^{d} M_k \cdot \bar{u} = (L_m + U_m) \cdot \prod_{k \neq m} M_k \cdot \bar{u}.$$

Errechnen wir einmal $t := \prod_{k \neq m} M_k \cdot \bar{u}$ in $(d-1)N$ Durchläufen und dann $A_m \cdot t$, so beträgt die Gesamtzeit $(d-1)N + 2N = (d+1)N$. Wäre A_m die erste durchzuführende Operation, so müsste die Rekursion durchgeführt werden. Gemäß (3.20) erfordert dies eine Gesamtzeit von $(d-1)N + N + N + (d-1)N = 2dN$ Durchläufen.

Neben einem Faktor 2 ist mit diesem Argument aber ein weiterer Faktor d bei der Gesamtmultiplikation $R \cdot \bar{u}$ möglich. Mit obigem Argument kann in einem Schritt $v :=$

$\prod_{k=1}^{m} M_k \cdot \bar{u}$ in $d \cdot N$ Operationen errechnet werden. Das Ergebnis der Matrix-Vektor-Multiplikation $R \cdot \bar{u}$ lässt sich bedingt durch die Kommutativität dieser Matrizen darstellen als

$$R \cdot \bar{u} = \sum_{m=1}^{d} A_m \cdot M_m^{-1} v.$$

Statt $O(d^2 N)$ Operationen durch d-fache Anwendung des Unidirektionalen Prinzips sind so lediglich $d \cdot N + d \cdot (N + N) = O(d \cdot N)$ Operationen notwendig. Dazu muß noch die Multiplikation mit der inversen Massenmatrix, $M_m^{-1} \cdot v$, implementiert werden. Bei homogenen Randbedingungen erfordert dies lediglich die Lösung eines pentadiagonalen Gleichungssystems auf jedem Level, was in Zeit $O(N)$ möglich ist, siehe [34]. Damit ist die Durchführung der Iterationen beim iterativen Lösen des Gleichungssystems effizient. Allerdings erscheint die Implementierung von $M_m^{-1} \cdot v$ zur Durchführung der Randexpansion auf Seite 27 teurer als der Nutzen. Damit ist die Modifikation der rechten Seite zur Berücksichtigung inhomogener Dirichletrandwerte um einen Faktor d teurer als die sonst übliche Matrix-Vektor-Multiplikation.

Beide Argumente lassen sich ohne weiteres auf die Behandlung konvektiver Terme oder gemischter zweiter Ableitungen übertragen. Die teuren Operationen müssen lediglich an das Ende der Bearbeitungskette verschoben werden. Auch eindimensionale Koeffizienten der Art

$$\langle u, \phi_{\mathbf{k},\mathbf{j}} \rangle := \int_{[0,1]^d} c(x_r) D^\alpha u(x) D^\beta \phi_{\mathbf{k},\mathbf{j}}(x) dx$$

können mit lediglich einem *konstanten* Zusatzfaktor an Laufzeit implementiert werden. Eindimensionale Verfahren zu einer solchen Multiplikation in Zeit $O(N)$ sind für die hierarchische Hutbasis bekannt (siehe [21]) und können durch Transformationen angewendet werden. Ein solches Verfahren werden wir in Abschnitt 5.3 einsetzen.

Damit beträgt die Laufzeit $O(dN)$ für den Helmholtzoperator oder konvektive Terme sowie $O(d^2 N)$ bei gemischten zweiten Ableitungen. Der Speicherverbrauch beträgt jeweils $O(N)$. Im Gegensatz zu den bisher bekannten Verfahren, welche durch mehrmalige Anwendung des Unidirektionalen Prinzips entweder $O(d2^d N)$ oder $O(d^2 2^d N)$ Operationen benötigen, sind so die Kosten für die Behandlung hochdimensionaler partieller Differentialgleichungen auf dünnen Gittern sehr niedrig. Um nicht zwischen einem d^2 und d unterscheiden zu müssen, sprechen wir im Folgenden der Einfachheit halber auch von $O(\mathrm{Poly}(d)N)$ Operationen.

Kapitel 4

Datenstruktur

Zur Durchführung von Dünngitteralgorithmen ist es notwendig, zu Gitterpunkten $(\mathbf{l}, \mathbf{i})$ Zugriff auf Koeffizienten $u_{\mathbf{l},\mathbf{i}}$ bereitzustellen. Dies ist Aufgabe effizienter Datenstrukturen, die sowohl die logische Gitterstruktur als auch die Durchführung der Algorithmen realisieren sollen.

Dünne Gitter sind gewissermaßen auf d Dimensionen verallgemeinerte Binärbäume, deren Tiefe in jeder Richtung beschränkt ist. Entsprechend liegt es nahe, auch die für Binärbäume bekannten Datenstrukturen zu verallgemeinern. Eine naheliegende Lösung ist, zu jedem Knoten $(\mathbf{l}, \mathbf{i})$ in einem dünnen Gitter G^d, adaptiv oder regulär, Zeiger auf seine $2 \cdot d$ Söhne sowie auf seine d Väter zu speichern. Der Koeffizientenwert $u_{\mathbf{l},\mathbf{i}}$ würde dann in dem Knoten gespeichert. Jedoch ist es bei diesem Vorgehen sehr schwierig, die Zeiger auf die Väter beim Aufbau der Datenstruktur sowie bei adaptiver Verfeinerung zu setzen, vgl. [1] und die darin enthaltenen Zitate.

Bedingt durch diese Schwierigkeit verwendet man andere Ansätze, um die komplexe Struktur dünner Gitter widerzuspiegeln. In diesem Kapitel betrachten wir die aus der Literatur bekannten Datenstrukturen und untersuchen sie insbesondere im Hinblick auf die Anwendbarkeit in hohen Dimensionen. Relevant ist dabei die Zugriffsgeschwindigkeit und vor allem der Speicherverbrauch in Bezug auf die Dimension. Denn trotz der von der Ordnung her geringen Zahl von Punkten in einem dünnen Gitter wächst die Gesamtzahl von Punkten bei hohen Dimensionen bei zunehmendem Level schnell an. Hat man dann wie in obigem Beispiel einen Speicherverbrauch von $O(d)$ pro Gitterpunkt, so erreicht man schnell Grenzen des verfügbaren Speichers. Dies ist insbesondere wichtig bei der Verwaltung von Funktionen, die nicht auf dem Rand verschwinden, was für die Aufstellung der rechten Seite des Gleichungssystems sowie die Behandlung von Randbedingungen eine wichtige Rolle spielt (vgl. Kapitel 3). Neben der Geschwindigkeit hochdimensionaler Löser ist es daher Ziel dieser Arbeit, speichersparende Datenstrukturen zur Verwaltung der Dünngitterstruktur heranzuziehen.

Zur Durchführung der in Kapitel 3 eingeführten Algorithmen fassen wir die notwendigen Anforderungen an eine Datenstruktur zusammen.

- Jeder Algorithmus benötigt für eine beliebige Richtung $m \in \{1, \ldots, d\}$ die Traversierung aller Punkte auf Linien in dieser Richtung. Eine eindimensionale Linie ist

dabei nichts anderes als ein Binärbaum; innerhalb eines solchen Gitterelements benötigen wir Preorder- und Postorderdurchläufe sowie eine Breitensuche. Dies entspricht den uns bekannten TopDown- und BottomUp-Durchläufen sowie einer levelweisen Linientraversierung. Somit reicht es für Datenstrukturen aus, Zugriff auf Punkte *linienweise* bereitzustellen.

- Eine Datenstruktur muss lokale Verfeinerung bereitstellen können. Dabei ist es nicht erlaubt, Punkte ohne deren hierarchische Väter in jeder Richtung einzufügen.

- Während des Durchlaufes einer Linie werden häufig Punkte aus dem Erzeugendensystem mit assoziierten Koeffizienten benötigt (insbesondere für die Prewavelet-Verfahren). Diese können – je nach Datenstruktur – direkt in die Datenstruktur eingefügt werden oder aber als separate, *eindimensionale* Strukturen (Binärbäume) innerhalb der Algorithmen als temporäre Knoten verwendet werden. Letzteres basiert darauf, daß die Algorithmik Knoten des Erzeugendensystems nur innerhalb einer Linie benötigt.

- Will man mit mehreren unterschiedlichen Gittern arbeiten, so ist zusätzlich zu Linientraversierungen der Zugriff auf einen Koeffizienten $u_{\mathbf{l},\mathbf{i}}$ zu beliebigem $(\mathbf{l},\mathbf{i}) \in G^d$ ohne Kenntnis von hierarchischen Vorfahren notwendig.

Da man üblicherweise mehrere Dünngitterfunktionen (beziehungsweise Koeffizientenvektoren) während der Algorithmik benötigt, verwaltet man alle Koeffizientenvektoren gleichzeitig. Dazu stellt die Datenstruktur zu einem Punkt $(\mathbf{l},\mathbf{i}) \in G^d$ nicht direkt den Koeffizienten $u_{\mathbf{l},\mathbf{i}}$ zur Verfügung, sondern den Index in dem zugehörigen Koeffizientenvektor.

4.1 Eine Übersicht verwendeter Ansätze

Bei der Behandlung dünner Gitter für partielle Differentialgleichungen finden unterschiedliche Datenstrukturen Anwendung, die sich in mehrere Klassen einteilen lassen.

Eine erste Möglichkeit, dünne Gitter zu verwalten, besteht in der Ausnutzung von rekursiven Strukturen in dünnen Gittern. Im eindimensionalen Fall verwaltet man ein dünnes Gitter wie einen normalen Binärbaum. Beim Übergang ins Mehrdimensionale verwendet man in eine Richtung, beispielsweise die erste, einen Binärbaum, dessen Knoten eine $(d-1)$-dimensionale Scheibe des Gitters repräsentiert. Diese ist ein dünnes Gitter der Dimension $(d-1)$, also wird der Ansatz rekursiv fortgesetzt. In der letzten Richtung schließlich befindet sich die Information zu den eigentlichen Koeffizienten $u_{\mathbf{l},\mathbf{i}}$. Die Randknoten werden dabei gesondert behandelt, beide bilden die Väter des Mittelpunktes. Abbildung 4.1 illustriert die Struktur für $d = 2$. Rekursive Datenstrukturen finden Verwendung in u. a. [1], [4], [9], [31]. Durch diese Rekursion umgeht man die Schwierigkeiten der multidimensionalen Binärbaumstruktur und hat letztlich nur eindimensionale Bäume zu verwalten, was speichersparend möglich ist. So kann man Adaptivität auf natürliche Weise einbinden, indem man schlichtweg Teilbäume einfügt beziehungsweise entfernt. Jedoch ist so der Zugriff auf Linien lediglich in der Richtung d möglich, denn nur dort liegen die notwendigen Informationen dicht im Speicher. Die Algorithmen sind somit nur in diese

Richtung anwendbar. Als Abhilfe erweitert man die Algorithmen derart, daß sie nicht auf einzelnen Punkten, sondern immer auf ganzen Gitterscheiben arbeiten, vgl. [1]. Da die Zahl der notwendigen Gitterscheiben für Prewavelets bedingt durch die Breite der Träger deutlich höher ist als bei der hierarchischen Hutbasis ist ein großer Aufwand zur Entwicklung geeigneter Algorithmen notwendig, die durch das Ablegen von Gitterscheiben auf dem Stack zusätzlich viel Speicher verbraucht. Durch das Abarbeiten von ganzen Gitterscheiben sind u.U. notwendige Erzeugendensystemknoten für jede Scheibe nötig, was den Speicherverbrauch nicht unerheblich erhöht. Zudem ist die in Abschnitt 3.3 durchgeführte Permutation der Traversierungsreihenfolge hier nicht möglich.

Als eine Alternative zu rekursiven Datenstrukturen kann die eindimensionale Binärbaumstruktur *in jeder Richtung* gespeichert werden. Dabei wird ein Container verwendet, der bezüglich einer ausgezeichneten Richtung *alle* Linien des Gitters explizit enthält. Mithilfe dieses Containers können Linientraversierungen hocheffizient durchgeführt werden – ohne weiteren Aufwand. Um dieselbe Funktionalität für alle Richtungen bereitzustellen, wird pro Richtung ein solcher Container eingeführt; die Datenstruktur enthält also die Verwaltung für jeden Koeffizienten $u_{\mathbf{l},\mathbf{i}}$ genau d mal. Zur Auflösung dieser Redundanz müssen bei jeder Änderung des Gitters alle d Container synchronisiert werden. Der Vorteil dieser Struktur ist die hocheffiziente Traversierung von Linien in beliebiger Richtung. Diese ist möglich in Zeit $O(N)$, unabhängig von der Dimension. Eine derartige Datenstruktur wird in [28] eingesetzt. Innerhalb eines solchen Containers müssen neben den Linien aber auch die Position der Linien, d. h. $(d-1)$-dimensionale Indizes für jede Linie, gespeichert werden. Die Gesamtzahl aller Linien in eine Richtung m ist die Zahl der Punkte in der Gitterscheibe mit der Koordinate $x_m = \frac{1}{2}$. Im Fall eines regulären dünnen Gitters sind dies $\tilde{N}_m = |G_n^{d-1}|$ viele; im adaptiven Fall hängt die Anzahl stark von dem d-dimensionalen Gitter ab. Da es in jede Richtung solch einen Container gibt, betragen die Speicherkosten im worst-case also $O(dN + d(d-1)\tilde{N}_{\tilde{m}})$, wobei $\tilde{m}$ die Richtung mit der höchsten Anzahl von Linien bezeichne. Falls das Gitter (nahezu) regulär ist, können die Linienindizes einmal für alle Container zusammen gespeichert werden. In diesem Fall reduziert sich der Speicheraufwand auf $(dN + (d-1)\tilde{N}_{\tilde{m}}) = O(dN)$. Um die verschiedenen Container konsistent zu halten, modifiziert man üblicherweise nur eine Struktur und kopiert diese auf die anderen. Diese Kopieroperation erfordert es, einen gegebenen Punkt $(\mathbf{l}, \mathbf{i})$ aus der einen Struktur in die andere einzufügen – was für die andere einen Zugriff ohne Bezug auf eine Linie bedeutet und daher möglicherweise teuer ist. Sind $Q(d, N)$ die Kosten für einen solchen Zugriff, so kostet die gesamte Kopieroperation $O(d \cdot Q(d, N) \cdot N)$ Schritte. Organisiert man diesen Zugriff beispielsweise mittels Hashtabellen, so ist $Q(d, N) = O(d)$ erreichbar (s.u.) und man erhält eine konsistente Datenstruktur in Zeit $O(d^2 N)$ – was von der gleichen Ordnung wie die Durchführung der Matrix-Vektor-Multiplikation mit der Steifigkeitsmatrix ist. Bei geringer Dimension ist diese Technik empfehlenswert, denn die Geschwindigkeit von Linientraversierungen ist optimal. Für die von uns erstrebten hochdimensionale Anwendungen ist der hohe Speicherverbrauch von $O(dN)$ für reguläre dünne Gitter nicht vertretbar; von den Kosten für adaptive Verfeinerung, die neben der d-fachen Redundanz auch quadratisch in der Dimension skalierende Speicherkosten für spatiale Gitter verwalten müssen, ganz zu schweigen.

Eine weitere verbreitete Technik ist die Ausnutzung der Multiindizes $(\mathbf{l}, \mathbf{i})$ zur Git-

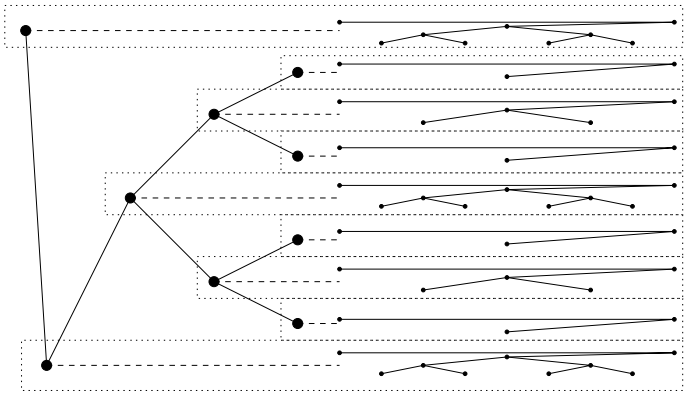

Abbildung 4.1: Diese Abbildung zeigt ein Beispiel einer rekursiven Datenstruktur für $d = 2$. In Richtung 2 erhält man einen Binärbaum, dessen Knoten wiederum Binärbäume für Richtung 1 enthalten.

terverwaltung. Diese enthalten die gesamte Information über Väter, Söhne, Koordinaten etc. Somit lässt sich die Algorithmik komplett formulieren mithilfe der Multiindizes; eine Linientraversierung ist dann eine Vorschrift, die Schritt für Schritt jede Linie in eine Richtung besucht und dann die Multiindizes auf dieser Linie in einer gewünschten Reihenfolge produziert. Der eigentliche Verwaltungsakt, die Zuordnung zu Koeffizienten, wird über einen assoziativen Container durchgeführt, der zu jedem $(\mathbf{l}, \mathbf{i}) \in G^d$ den zugehörigen Koeffizienten $u_{\mathbf{l},\mathbf{i}}$ (beziehungsweise dessen Index in einem Array) enthält. Ein Beispiel dafür sind Hashmaps, die mittels einer Hashfunktion $q(\mathbf{l}, \mathbf{i})$ alle vorkommenden Indizes $(\mathbf{l}, \mathbf{i})$ gleichmäßig (nicht unbedingt injektiv) auf eine Zahlenmenge $(1, 2, 3, \ldots, m)$ abbilden. Kollisionen werden beispielsweise mittels Listen aufgelöst. Hashtabellen sind Standardcontainer; Implementierungen sind frei verfügbar. Eine solche Datenstruktur findet beispielsweise Anwendung in [10] sowie [36]. Hashbasierte Datenstrukturen zeichnen sich durch eine höchstmögliche Abstraktion der Algorithmen aus, denn die Gitterstruktur wird genau wie in der Theorie mittels Multiindizes verwaltet. Adaptivität ist schlichtweg durch das Einfügen der entsprechenden Zuordnungen in dem Container realisierbar. Die Zugriffszeit des Containers wird bestimmt durch die Auswertung der Hashfunktion, die ca. $2 \cdot d$ betragen. Zur Auflösung der Kollision sind zusätzlich $\approx k \cdot 2 \cdot d$ Schritte notwendig, wobei k die Anzahl der Kollisionen ist. Da Hashtabellen ihre Daten immer so verwalten, dass im Mittel $k \leq \text{const}$ gilt, erhalten wir mittlere Zugriffszeiten von $O(d)$. Zur Auflösung der Kollisionen ist es notwendig, alle Koordinaten für die Multiindizes explizit zu speichern. Somit erfordert diese Technik für die Traversierung aller Linien in eine beliebige Richtung $O(d \cdot N)$ Operationen und ebensoviel Speicher zur Verwaltung des Gitters.

Wie wir sehen, ist die Durchführung von Linientraversierungen in einer beliebigen Richtung durch dünne Gitter meist mit Speicherkosten von mindesten $O(dN)$ verbunden. Die Bereitstellung der Traversierungsmethoden in nur eine Richtung ist bedingt durch die Breite der Ansatzfunktionen sehr aufwändig und kostet ebenfalls viel Speicher.

Speichertechnisch optimal ist ohne Zweifel ein asymptotisches Verhalten von $O(N)$, d. h. unabhängig von der Dimension. Im folgenden Abschnitt entwickeln wir eine neue

Datenstruktur, die in Bezug auf den Speicher optimale Eigenschaften hat und dennoch schnellen Zugriff gewährleistet.

4.2 Eine Arraybasierte Datenstruktur

Um eine Datenstruktur zu entwickeln, die ein dünnes Gitter in beliebiger Dimension verwalten kann und deren Speicherverbrauch dennoch unabhängig von der Dimension ist, verfolgen wir die Idee der logischen Gitterverwaltung mittels Multiindizes weiter. Wie wir bereits sahen, enthalten Multiindizes die gesamte Information des Gitters. Eine Möglichkeit des Zugriffes auf Koeffizienten besteht nun darin, zu gegebenem Multiindex direkt in einem Array von Gleitpunktzahlen an der „richtigen“ Stelle nachzusehen. Dazu müsste der Zugriff auf $u_{\mathbf{l},\mathbf{i}}$ durch Zugriff auf $U[r(\mathbf{l},\mathbf{i})]$ erfolgen, wobei U ein Array von Gleitpunktzahlen und $r(\mathbf{l},\mathbf{i}) \in \mathbb{N}$ ein zu $(\mathbf{l},\mathbf{i})$ *eindeutig* zugewiesener Index ist.

Die Idee ist nun folgende: für reguläre dünne Gitter entwickeln wir eine skalare Indexabbildung $r : \{(\mathbf{l},\mathbf{i})\} \to \{0,1,2,3,\ldots,\infty\}$, die eindeutig Multiindizes mit zugehörigen skalaren Indizes identifiziert. Fordert man, dass die Indexabbildung lückenlos abbildet, so können reguläre dünne Gitter ohne zusätzlichen Verwaltungsaufwand ausschliesslich durch Arrays von Koeffizientenvektoren verwaltet werden; der zusätzliche Speicheraufwand für die Gitterverwaltung beträgt $O(1)$. Zugriffe auf $u_{\mathbf{l},\mathbf{i}}$ würden dann zu $U[r(\mathbf{l},\mathbf{i})]$, die Zeit für einen beliebigen Zugriff wäre $O\big(T(r)\big)$, wobei $T(r)$ die maximale Zeit zur Ermittlung eines skalaren Index bezeichne.

Obwohl eine Identifikation von $(\mathbf{l},\mathbf{i})$ mit einem eindeutigen $r(l,i) \in \mathbb{N}$ statisch sein muß, lässt sich die Idee leicht auf adaptive Gitter übertragen, sobald man eine entsprechende reguläre Indexabbildung $r(\cdot,\cdot)$ gefunden hat. Dazu betrachten wir ein adaptives dünnes Gitter G^d in Bezug auf die Abbildung $r(\cdot,\cdot)$. Da G^d nicht alle Punkte eines regulären Gitters enthält, ist die Menge $r(G^d)$ entsprechend lückenhaft. Die Koeffizientenvektoren aber haben die Länge $N = |G^d|$, weshalb man noch die Möglichkeit benötigt, zu jedem $j \in r(G^d)$ den richtigen Offset in einem Koeffizientenvektor $U[\cdot]$ zu finden. Diese Zuordnung ist dynamisch und repräsentiert die Adaptivität – jedoch nur auf Basis von Zahlen $j \in r(G^d) \subset \mathbb{N}$ und damit unabhängig von der Dimension, denn die Zuordnung $r(\cdot,\cdot)$ ist eindeutig. Dies lässt sich leicht erreichen: beim Aufbau der adaptiven Struktur ist für jeden neu einzufügenden Punkt $(\mathbf{l},\mathbf{i})$ der skalare Index $r(\mathbf{l},\mathbf{i})$ eindeutig berechenbar, außerdem vergrößert man den alten Koeffizientenvektor $U[\cdot]$ um 1 auf Größe $|U|$ und merkt sich die Zuordnung

$$r(\mathbf{l},\mathbf{i}) = j \mapsto |U| - 1.$$

Eine geeignete Datenstruktur zum „merken“ solcher Zuordnungen ist wiederum eine Hashmap, die im Mittel in Zeit $O(1)$ zu gegebenem $j \in \mathbb{N}$ den assoziierten Wert aufsuchen kann. Damit kostet ein Zugriff auf einen Multiindex $(\mathbf{l},\mathbf{i}) \in G^d$ wie für reguläre Gitter Zeit $O\big(T(r)\big)$. Der Speicherverbrauch ist wiederum $O(N)$, wobei die Konstante leicht schlechter als im regulären Fall ist.

Dieses Vorgehen für die Realisierung von Adaptivität könnte man nutzen, um die Anforderungen an die reguläre Indexabbildung $r(\cdot,\cdot)$ zu reduzieren. Fordert man lediglich, dass $r(\mathbf{l},\mathbf{i})$ injektiv auf eine Menge $r(G^d)$ abbildet, so ist das Verfahren immer noch durchführbar, wobei – wie im adaptiven Fall – die Menge $r(G^d)$ der skalaren Indizes noch

assoziiert werden muss mit Offsets in Arrays. Die Verwendung von Hashing liefert dann dieselben Kosten in Zeit und Speicher wie oben. Ein Beispiel einer solchen Abbildung mit Lücken zwischen den skalaren Indizes erhält man, wenn man einem Dünngitterpunkt denselben Index zuweist, den man ihm in einem vollen Gitter zuweisen würde. Derartige Zuweisungen sind Standard und lassen sich in Zeit $T(r) = O(d)$ ermitteln. Jedoch ist ein herkömmlicher 32-Bit Rechner mit einer größten natürlichen Zahl von 2^{32} nur eingeschränkt geeignet für einen solchen Ansatz. Ein volles Gitter, das in jede Richtung $\approx 2^n \pm 1$ Punkte enthält, hat in Dimension d ca. $N = 2^{nd}$ Elemente. Somit lassen sich – bei Verwendung der effizienten, eingebauten Datentypen eines 32-Bit Rechners – mit dieser Methode nur dünne Gitter mit Level n verwalten, für die gilt $n \cdot d \leq 32$. Die maximale Dimension liegt daher für diese Methode bei $d = 32$ bei $n = 1$, analog bei 64-Bit Rechnern.

Aufgrund dieser Einschränkung wird es im folgenden das Ziel sein, eine lückenlose Indexfunktion $r : \{(\mathbf{l}, \mathbf{i})\} \to \mathbb{N}$ zu finden, die effizient in maximal $T(r) = O(d)$ Schritten errechenbar ist. Folgende Anforderungen stellen wir an eine solche Abbildung:

- Um bei einer Hinzunahme neuer Punkte auf höherem Level (beispielsweise bei adaptiver Verfeinerung) nicht alle bisherigen Indizes $r(\mathbf{l}, \mathbf{i})$ auf den niedrigeren Levels neu errechnen zu müssen, fordern wir für Levelindizes $\mathbf{l}, \mathbf{k}$:

$$n(\mathbf{l}) \leq n(\mathbf{k}) \Rightarrow r(\mathbf{l}, \cdot) \leq r(\mathbf{k}, \cdot). \tag{4.1}$$

- Um – wie eingangs erwähnt – bei Bedarf Speicher für Randpunkte getrennt von dem Speicher für innere Punkte verwalten zu können, suchen wir getrennt voneinander eine Abbildung für innere Punkte und eine für Randpunkte,

$$r_I \colon G_n^d \longrightarrow \{0, 1, 2, 3, 4, \ldots\},$$
$$r_B \colon G_{\mathrm{Rand},n}^d \longrightarrow \{0, 1, 2, 3, 4, \ldots\}.$$

  Diese Vorgehensweise ist nicht notwendig für Indexabbildungen, erweist sich in der Praxis aber als ausgesprochen hilfreich.

- Die Aufzählung braucht Knoten des Erzeugendensystems nicht zu berücksichtigen, diese stellen wir speichersparend mittels eindimensionaler Spezialstrukturen zur Verfügung.

4.2.1 Entwicklung einer Indexfunktion für innere Punkte

In diesem Abschnitt werden wir die Indexfunktion r_I entwickeln. Forderung (4.1) führt uns zu einer levelweise durchzuführenden Indizierung. Dazu greifen wir zurück auf die bereits in Kapitel 2 eingeführte Handhabung dünner Gitter als Vereinigung von Elementen hierarchischer Überschussräume,

$$G_n^d = \bigcup_{n(\mathbf{l}) \leq n} W_{\mathbf{l}},$$

mit

$$W_{\mathbf{l}} = \{\mathbf{i} = (i_1, \ldots, i_d) | i_1 \in \{1, 3, 5, \ldots, 2^{l_1} - 1\}, \ldots, i_d \in \{1, 3, 5, \ldots, 2^{l_d} - 1\}\},$$

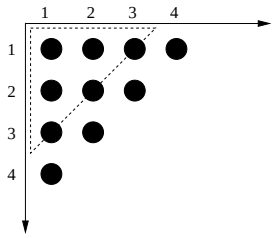

Abbildung 4.2: Darstellung der Levelkomponenten im zweidimensionalen als ganzahlige Punkte auf einem Simplex.

vgl. Kapitel 2. Innerhalb eines gegebenen $W_{\mathbf{l}}$ können sofort alle Punkte aufgezählt werden, denn letztlich entspricht dies der Aufzählung innerhalb eines vollen Gitters. Wenn wir nur die Levelkomponenten $\mathbf{l}$ betrachten, entspricht dies der Behandlung von Punkten auf einem Simplex, dessen Spitze im Koordinatenursprung und dessen Seiten die Koordinatenachsen darstellen, vgl. Abbildung 4.2. Die Koordinaten der betrachteten Punkte haben daher ganzzahlige Werte. Durch die Forderung nach levelweiser Aufzählung hat der Punkt mit dem *ersten* Index auf Level $n := n(\mathbf{l})$ daher den Index $P^{<n} := |G^d_{n-1}|$ (wir beginnen die Indizierung mit $r_I(\mathbf{1}, \mathbf{1}) := 0$). Die eigentlich Schwierigkeit des gesamten Ansatzes ist es, $\mathbf{l}$ einen eindeutig errechenbaren Index unter all denen Levelindizes $\mathbf{k}$ mit $n(\mathbf{k}) = n(\mathbf{l})$ zuzuweisen. Hat man eine solche Vorschrift gefunden und bezeichnet $r_n(\mathbf{l})$ diesen Index unter all denen Levelindizes auf Level n, so ist eine Lösung unseres Problems gefunden mit

$$r_I(\mathbf{l}, \mathbf{i}) := P^{<n(\mathbf{l})} + O_{\mathbf{l}} + r_{\mathbf{l}}(\mathbf{i}) \tag{4.2}$$

und einigen Hilfsbezeichnungen. Jeder $W_{\mathbf{l}}$ auf Level n hat genau 2^{n-1} Punkte, daher bezeichnet $O_{\mathbf{l}} := 2^{n-1} \cdot r_n(\mathbf{l})$ den Index des ersten Punktes in $W_{\mathbf{l}}$ unter allen $W_{\mathbf{k}}$ mit $n(\mathbf{k}) = n$. Mit $r_{\mathbf{l}}(\mathbf{i})$ bezeichnen wir nun noch den Index von $\mathbf{i}$ *innerhalb* von $W_{\mathbf{l}}$. Wie bei einer Indizierung voller Gitter schreiben wir

$$r_{\mathbf{l}}(\mathbf{i}) := \sum_{j=1}^{d} \frac{i_j}{2} \prod_{k<j} 2^{l_k - 1}.$$

Dabei ist zu bemerken, daß bei Division natürlicher Zahlen auch die Ganzzahldivision ohne Rest gemeint ist.

Es gilt nun, eine (beliebige, aber effiziente) Indizierung $r_n(\mathbf{l})$ zu finden. Hat man eine solche, so erfüllt (4.2) die Anforderungen an eine Indizierung der inneren Punkte eines dünnen Gitters.

Dazu betrachten wir ein äquivalentes Ersatzproblem. Es gilt, eine beliebige Indizierung derjenigen Seitenfläche unseres Simplex zu finden, welche Level n in unserem dünnen Gitter entspricht. Wir bezeichnen diese Seitenfläche mit

$$K^d_n := \Big\{ \mathbf{k} = (k_1, \ldots, k_d) \big| n(\mathbf{k}) = \sum_m k_m - d + 1 = n, \mathbf{k} \in \mathbb{N}^d_{>0} \Big\}.$$

Wir suchen nun zu gegebenem $\mathbf{l} \in K^d_n$ den Index $r_n(\mathbf{l}) =: r^d_n(\mathbf{l})$. K^d_n ist ein Teil einer

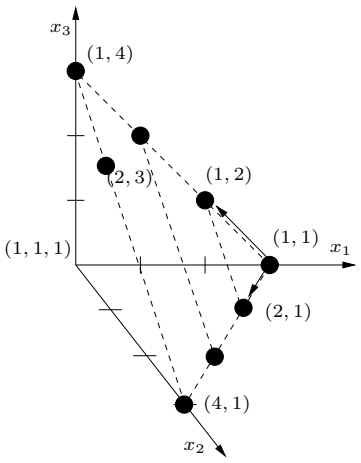

Abbildung 4.3: Alle Punkte auf der Seitenfläche $K_n^d = K_4^3$ bilden wieder einen parametrisierbaren Simplex. Eingezeichnet ist die Parameterdarstellung innerhalb des Simplex.

Hyperebene in $\mathbb{N}_{>0}^d$. Es lässt sich daher die Parameterdarstellung

$$K_n^d = \left\{ \mathbf{l} = \begin{pmatrix} n+d-1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} + \sum_{m=2}^{d} a_m \begin{pmatrix} -1 \\ e_{m-1} \end{pmatrix}, a_m > 0, m = 2, \ldots, d \text{ sodaß } \mathbf{l} \in \mathbb{N}_{>0}^d \right\}$$

angeben, wobei e_m den m-ten Einheitsvektor in $\mathbb{N}^{d-1}$ bezeichne. Zu gegebenem $\mathbf{l} \in K_n^d$ erhält man die Parameterdarstellung $\tilde{\mathbf{l}} = (l_2, \ldots, l_d) \in \mathbb{N}_{>0}^{d-1}$. Alle zulässigen Parameterdarstellungen (d. h. mit ganzzahligen, nichtverschwindenden Koordinaten) aber bilden wieder einen Simplex, jedoch in Dimension $(d-1)$. Dies veranschaulichen wir anhand von Abbildung 4.3. Alle Punkte auf Level $n = 4$ im dreidimensionalen Raum bilden einen zweidimensionalen Simplex, den wir mithilfe unserer Parameterdarstellung wieder mit ganzzahligen Koordinaten bezeichnen. Der Punkt $(1, 1)$ in Simplexkoordinaten entspricht dem Punkt $(4, 1, 1)$ im Gesamtraum. Die Indizierungsaufgabe lässt sich so völlig analog für das $(d-1)$-dimensionale Problem stellen: gegeben eine Parameterdarstellung $\tilde{\mathbf{l}}$, welchen Index weisen wir dieser zu? Zur Verfügung steht wieder das Level innerhalb des $(d-1)$-dimensionalen Simplex, $n(\tilde{\mathbf{l}}) = n(\mathbf{l}) - (l_1 - 1)$. In dem betrachteten dreidimensionalen Beispiel könnten wir einem Punkt $\mathbf{l}$ aus der Seitenfläche K_n^d mittels einer Funktion $r_n^3(\cdot)$ einen Index zuweisen, die innerhalb der Parameterdarstellung die Zahl der Punkte bis einschliesslich Level $n(\tilde{\mathbf{l}}) - 1$ enthält, zuzüglich einem Index auf der Diagonalen $n(\tilde{\mathbf{k}}) = n(\tilde{\mathbf{l}})$.

Diese rekursive Idee führen wir fort auf beliebige Dimensionen. Wir bezeichnen dazu mit $P^{n,d}$ die Zahl der Punkte in Dimension d mit Level kleiner oder gleich n,

$$P^{n,d} := \left| \{ \mathbf{l} \in \mathbb{N}_{>0}^d | n(\mathbf{l}) \leq n \} \right|.$$

So erhalten wir die rekursive Darstellung unserer gesuchten Funktion

$$r_n^d(\mathbf{l}) = P^{n(\tilde{\mathbf{l}})-1,d-1} + r_{n(\tilde{\mathbf{l}})}^{d-1}(\tilde{\mathbf{l}}).$$

Um die Rekursion zu beenden, definieren wir $r_n^1(\mathbf{l}) = 0$. Zur Veranschaulichung betrachten wir erneut Abbildung 4.3. Unter allen dargestellten Levelindizes $\mathbf{k} \in K_4^3$ hat $\mathbf{l} = (1,2,3)$ den Index $r_4^3(\mathbf{l}) = 7$, denn wir zählen mit $\tilde{\mathbf{l}} = (2,3)$ genau $P^{3,2} = 6$ Punkte $\tilde{\mathbf{k}}$, die innerhalb des Simplex vor Level $n(\tilde{\mathbf{l}}) = 4$ kommen. Die Rekursion ergibt $r_4^2((2,3)) = 2+0$, da $\tilde{\tilde{\mathbf{l}}} = 3$ und damit $P^{2,1} = 2$ viele Punkte auf der Linie von $(1,1,4)$ bis $(1,4,1)$ vor $(1,2,3)$ kommen. Setzt man die Rekursionen wieder ein und berücksichtigt dabei, daß in obigem Ansatz $n(\tilde{\mathbf{l}}) = n(\mathbf{l}) - (l_1 - 1)$, so erhält man schließlich

$$r_n(\mathbf{l}) := \sum_{m=2}^{d} P^{n_m(\mathbf{l})-1, d-(m-1)} \tag{4.3}$$

mit

$$n_m(\mathbf{l}) := \sum_{k=m}^{d} (l_k - 1) + 1 = n(\mathbf{l}) - \sum_{k<m} (l_k - 1).$$

Für die Werte $P^{n,d}$ gilt die Rekursionsformel $P^{n,d} = \sum_{m=1}^{n} P^{m,d-1}$ mit $P^{n,0} = 0$ sowie $P^{n,1} = n$, wie man durch Induktion über d verifiziert. Will man eine derartige Dünngitterindizierung für Dimension d und Level n vorbereiten, so lassen sich in einem einfach Setupschritt die notwendigen Werte $|G_k^m|$, $P^{k,m}$, $k = 0,\ldots,n$; $m = 1,\ldots,d$ in Zeit $O(d \cdot n^2)$ in Arrays mit Speicherverbrauch $O(d \cdot n)$ vorberechnen.[1] Fasst man alle Komponenten der Indexabbildung $r_I(\mathbf{l},\mathbf{i})$ zusammen, so erhält man Algorithmus 4.2.1. Die Laufzeit beträgt $O(d)$. Abbildung 4.4 zeigt ein reguläres dünnes Gitter in $d = 2$, $n = 4$ mit den mittels $r_I(\mathbf{l},\mathbf{i})$ zugewiesenen eindeutigen Indizes.

Algorithmus 4.2.1 Berechnung der Funktion $r_I(\mathbf{l},\mathbf{i})$

Erfordert: $(\mathbf{l},\mathbf{i})$ ist innerer, zu Basis gehörender Punkt (d. h. $\forall m\colon l_m \neq 0, i_m$ ungerade)

```
r_n(l) := 0
r_l(i) := 0
n := 1
d̂ := 0
for m = d; m ≥ 2; m = m − 1 do
    d̂ := d̂ + 1
    n := n + l_m − 1
    r_n(l) := r_n(l) + P^{n−1,d̂}
    r_l(i) := i_m/2 + 2^{l_m−1} · r_l(i) {Aufsplittung in Teleskopsumme}
end for
r_l(i) := i_1/2 + 2^{l_1−1} · r_l(i)
n := n + l_1 − 1
r_I(l,i) := |G_d^{n−1}| + 2^{n−1} · r_n(l) + r_l(i)
```

[1] Für $|G_k^m|$ kann man die bereits in Kapitel 2 eingeführten Rekursionsformeln programmieren, was bei Zwischenspeicherung von Resultaten obige Laufzeit ergibt. Dasselbe gilt auch für die Anzahl der Randknoten später.

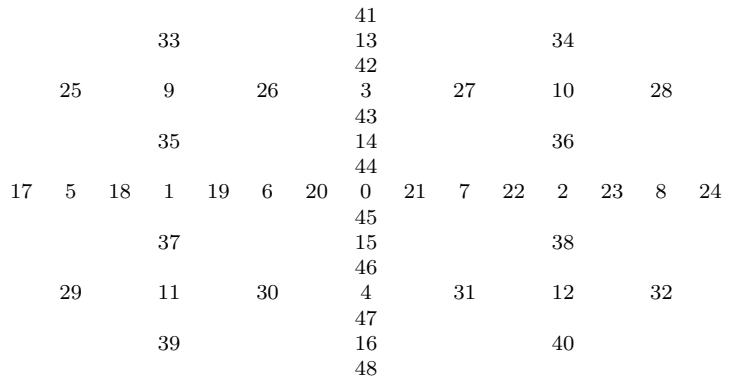

Abbildung 4.4: Ein Beispiel für die Anwendung der Indexabbildung $r_I(\cdot,\cdot)$ auf ein reguläres dünnes Gitter in $d = 2$, $n = 4$.

4.2.2 Entwicklung einer Indexfunktion für Randpunkte

Da wir für innere Punkte eine effiziente Routine zur Indizierung über das Splitting in Überschussräume entwickeln konnten, stellt sich die Frage, ob dies durch vergleichbare Überlegungen auch für Randpunkte erreicht werden kann. Wie wir jedoch bei der Einführung dünner Gitter sahen, werden dünne Gitter auf dem Rand rekursiv definiert. Jeder der $2 \cdot d$ Ränder des dünnen Gitters wird definiert als ein dünnes Gitter selben Levels in Dimension $(d-1)$; deren Ränder bilden die Kanten des d-dimensionalen Gitters und sind ihrerseits dünne Gitter der Dimension $d-2$, siehe Kapitel 2. Damit ist es nicht möglich, die Randkomponenten kompakt durch ein Splitting wie bei inneren Punkten zu erfassen. Eine Aufzählung von Randpunkten hat also die schwierige Aufgabe, die rekursive Struktur effizient zu erfassen.

In diesem Abschnitt werden wir eine entsprechende Indexfunktion $r_B(\mathbf{l}, \mathbf{i})$ entwickeln. Die grundsätzliche Idee dabei ist, die Ränder nacheinander aufzuzählen und diese einzelnen Aufzählungen wohldefiniert aneinanderzureihen. Im Zweidimensionalen werden wir so vorgehen, daß den vier Ecken die ersten Indizes zugewiesen werden, darauf folgend alle verbleibenden Punkte mit $x_2 = 0$, darauf alle mit $x_2 = 1$, dann $x_1 = 0$ sowie schließlich $x_1 = 1$, siehe dazu Abbildung 4.5. Da wir jedoch auch auf dem Rand levelweise aufzählen wollen, ist weitere Sorgfalt nötig: man kann beispielsweise nicht erst alle Punkte eines Randes A aufzählen und anschliessend Rand B, sondern es müssen gleichmäßig erst alle Punkte auf Level 1 aus A und B, dann alle auf Level 2 aus A und B usw. aufgezählt werden – wobei für ein festes Level die oben motivierte Reihenfolge eingehalten werden soll. Dieses Schema wollen wir auch für $d > 2$ fortsetzen, wobei dann weitere m-dimensionale Untermannigfaltigkeiten zu berücksichtigen sind. Die Reihenfolge wählen wir so, daß zuerst Untermannigfaltigkeiten geringer Dimension und anschliessend diejenigen höherer Dimension indiziert werden, d. h. im Falle $d = 3$ erst Ecken, dann Kanten, dann Seitenflächen. Um dann alle Punkte im Innern einer Untermannigfaltigkeit (beispielsweise einer Seitenfläche im Falle $d = 3$) zu indizieren, verwenden wir die bereits bekannte Abbildung $r_I(\cdot,\cdot)$.

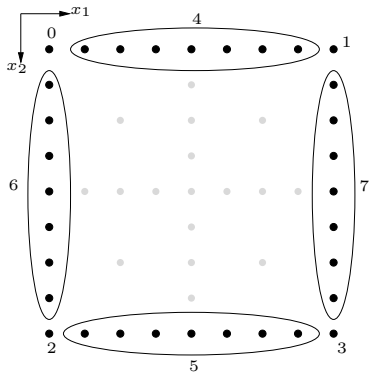

Abbildung 4.5: Schematische Darstellung der Reihenfolge der Randindizierung für $d = 2$.

Die Reihenfolge der Randstrukturen mit gleicher Dimensionalität bestimmen wir in Abhängigkeit von denjenigen Koordinaten, die für alle Punkte einer Randstruktur gleich sind. Entscheidend ist dabei zum Einen die *Position* von Nullen in dem Levelindex $\mathbf{l}$ eines Randpunktes $(\mathbf{l}, \mathbf{i})$ sowie ob für den Fall $l_m = 0$ der Ortsindex $i_m = 0$ oder $i_m = 1$ ist.

Diese grobe Skizze der Idee präzisieren wir im Folgenden. Um zu festem $(\mathbf{l}, \mathbf{i})$ auf dem Rand eines dünnen Gitters den Index $r_B(\mathbf{l}, \mathbf{i})$ zu ermitteln, untersuchen wir systematisch die Zahl der bereits vergebenen Indizes *vor* $(\mathbf{l}, \mathbf{i})$. Wir bezeichnen zunächst die Anzahl der verschwindenden Komponenten eines Levelindex $\mathbf{l}$ mit $K(\mathbf{l})$. Bekannt ist dann, daß alle Indizes $(\mathbf{k}, \mathbf{j})$ auf dem Rand mit $n(\mathbf{k}) < n(\mathbf{l}) =: n$ bereits vergeben sind – dies sind genau $P^{<n} := |G_R^{n-1,d}|$, wobei $G_R^{n,d}$ hier nur die Randpunkte eines regulären dünnen Gitters in Dimension d mit Level n enthalte. Weiterhin entscheidend ist die Zahl der Punkte in Randstrukturen mit geringerer Dimension, welche kleinere Indizes als $(\mathbf{l}, \mathbf{i})$ bekommen. Dies hängt wesentlich ab von $K(\mathbf{l})$, denn $d - K(\mathbf{l})$ ist die Dimension der Untermannigfaltigkeit in welcher $(\mathbf{l}, \mathbf{i})$ liegt. Wir bezeichnen mit $P(d, k, n)$ die Anzahl aller Randpunkte *auf* Level n in Dimension d, deren Levelindex genau k Nullen enthält. Dann können wir bereits festhalten, daß die Zahl der Indizes kleiner als $r_B(\mathbf{l}, \mathbf{i})$ mindestens

$$P^{<n(\mathbf{l})} + \sum_{k=K(\mathbf{l})+1}^{d-1} P\big(d, k, n(\mathbf{l})\big)$$

beträgt. Dies entspricht der Idee, das zuerst die Untermannigfaltigkeiten geringer Dimension (hohes K) indiziert werden. Damit sind noch nicht diejenigen Punkte berücksichtigt, die vor $(\mathbf{l}, \mathbf{i})$ indiziert werden sollen und in einer Untermannigfaltigkeit liegen, die ebenfalls $K(\mathbf{l})$ verschwindende Levelindizes haben. An dieser Stelle ist die eigentliche Arbeit zu tun, denn hier muss eine Reihenfolge definiert werden.

Dazu betrachten wir die Position der Nullen in dem Levelindex $\mathbf{l}$ sowie den Wert der zugehörigen Ortsindizes $i_m \in \{0, 1\}$. Wir definieren

$$\mathrm{BIN}(\mathbf{l}) := (b_1, \ldots, b_d), b_m := \begin{cases} 0 & l_m = 0 \\ 1 & l_m \neq 0 \end{cases}.$$

Sei jetzt $r_K\colon \{0,1\}^d \to \mathbb{N}$ eine Abbildung, die BIN($\mathbf{l}$) auf einen eindeutigen Index in der Menge

$$\{a = (a_1, \ldots, a_d) \in \mathbb{N}^d | K(a) = K(\mathbf{l}) = K\}$$

abbildet, also eine beliebige Reihenfolge festlegt. Eine solche werden wir weiter unten entwickeln. Wegen $l_m = 0 \Rightarrow i_m \in \{0,1\}$ existieren bei $K(\mathbf{l})$ verschwindenden Einträgen in $\mathbf{l}$ genau $2^{K(\mathbf{l})}$ viele Ränder, auf denen Punkte mit Levelindex $\mathbf{l}$ existieren. Damit können wir aussagen, daß vor $(\mathbf{l}, \mathbf{i})$

$$P^{<n(\mathbf{l})} + \sum_{k=K(\mathbf{l})+1}^{d-1} P\big(d, k, n(\mathbf{l})\big) + 2^{K(\mathbf{l})} \cdot r_K\big(\mathrm{BIN}(\mathbf{l})\big) \cdot I_n^{d-K(\mathbf{l})}$$

viele Indizes vergeben werden müssen, wobei wir mit I_n^d abkürzend die Zahl der Punkte *auf* Level n eines regulären dünnen Gitters im Innern bezeichnen,

$$I_n^d := |G_I^{n,d}| - |G_I^{n-1,d}|.$$

Hinzu kommt, wie bereits erläutert, noch die Abhängigkeit von den Ortsindizes in denjenigen Komponenten von $(\mathbf{l}, \mathbf{i})$, in denen $\mathbf{l}$ verschwindet. Bis hierher haben wir lediglich diejenigen Punkte vor $(\mathbf{l}, \mathbf{i})$ berücksichtigt, deren Ränder schon allein anhand von BIN($\mathbf{l}$), also nur vom Levelindex *vor* $(\mathbf{l}, \mathbf{i})$ indiziert werden. Zur Indizierung der Randpunkte, die alle die gleiche Null/Nicht-Null Belegung im Levelindex haben, definieren wir zu einer Null/Nicht-Null Belegung $z \in (0,1)^d$ den Index[2]

$$R(z, \mathbf{i}) := \sum_{\substack{m=1,\ldots,d \\ z_m=0}} i_m \prod_{\substack{k=1,\ldots,m \\ z_k=0}} 2.$$

Die Zahl der hier noch zu berücksichtigenden Punkte ist $R\big(\mathrm{BIN}(\mathbf{l}), \mathbf{i}\big) \cdot I_n^{d-K(\mathbf{l})}$. Zuletzt ist die Anzahl der Punkte vor $(\mathbf{l}, \mathbf{i})$ zu berücksichtigen, die sich in den Komponenten $m \in \{1, \ldots, d\}$ mit $l_m \neq 0$ von $(\mathbf{l}, \mathbf{i})$ unterscheiden. Für diese verwenden wir einfach die bereits bekannte Indizierung für innere Punkte, $r_I(\bar{\mathbf{l}}, \bar{\mathbf{i}})$, wobei $(\bar{\mathbf{l}}, \bar{\mathbf{i}})$ genau die Einträge von $(\mathbf{l}, \mathbf{i})$ enthält mit $l_m \neq 0$. Wir erhalten so die finale Darstellung

$$\begin{aligned} r_B(\mathbf{l}, \mathbf{i}) = P^{<n(\mathbf{l})} + \sum_{k=K(\mathbf{l})+1}^{d-1} & P\big(d, k, n(\mathbf{l})\big) \\ & + \Big(2^{K(\mathbf{l})} \cdot r_K\big(\mathrm{BIN}(\mathbf{l})\big) + R\big(\mathrm{BIN}(\mathbf{l}), \mathbf{i}\big)\Big) \cdot I_n^{d-K(\mathbf{l})} + r_I(\bar{\mathbf{l}}, \bar{\mathbf{i}}), \end{aligned} \quad (4.4)$$

wobei wir für $\mathbf{l} = \mathbf{0}$ die Sonderbehandlung

$$r_B\big((0, \ldots, 0), \mathbf{i}\big) := \sum_{m=1}^{d} i_m \prod_{k<m} 2 \quad (4.5)$$

[2] $R(z, \mathbf{i})$ entsteht dadurch, daß wir uns auf die Komponenten von $\mathbf{i}$ mit $z_m = 0$ beschränken und diese wie in einem vollen Gitter durchindizieren. Dieses „volle“ Gitter hat zwei Punkte in jeder infrage kommenden Richtung und die Koordinaten sind wahlweise 0 oder 1.

einführen. Zum Errechnen der Formel fehlen jedoch die entscheidenden Teile $P(d,k,n)$ und $r_K(\cdot)$. Um die für $P(d,k,n)$ benötigte Zahl der Punkte in Rändern mit k Nullen auf Level n zu bestimmen, können wir uns r_K zunutze machen. r_K indiziert alle diese Ränder, sodaß der größte Index plus eins die Anzahl solcher Ränder charakterisiert. Unter Einbeziehung der variierenden Ortskoordinaten enthalten wir so

$$2^k \cdot (\max_{|z|_1=k} r_K(z) + 1)$$

viele Ränder, da pro Levelindex $\mathbf{l}$ mit $K(\mathbf{l}) = k$ wie oben beschrieben 2^k Ränder existieren. Damit erhält man

$$P(d,k,n) := I_n^{d-k} \cdot 2^k \cdot (\max_{|z|_1=k} r_K(z) + 1).$$

Für das letzte fehlende Aufzählungselement $r_K(z)$ betrachten wir folgende äquivalente Problemstellung:

Gegeben eine *natürliche Zahl* A in Binärdarstellung mit genau $|A| = d$ Ziffern (führende Nullen erlaubt). Was ist der Index von A gemäß einer beliebigen Indizierung der Elemente in

$$Q(A) := \{B \in \mathbb{N} \big| |B| = d, \text{Quersumme von } B \text{ ist gleich der von } A\}?$$

Die Antwort mit einer entsprechenden Aufzählung lässt sich hier leicht angeben. Der Index ist genau die Anzahl der $B \in Q(A)$ mit $B < A$.

Algorithmus 4.2.2 Berechnet für alle $z \in (0,1)^d, |z|_1 \in \{1,\ldots,d\}$ den Bestandteil $r_K(z)$ und speichert die Anwort in einem Array A der Größe 2^d. Damit ist $r_K(z) = A[z]$, wobei z als natürliche Zahl interpretiert wird.

Erfordert: $d \leq$Wortbreite des Systems, üblicherweise 32
  resize(A, 2^d)
  resize(C, $d+1$) {$C[k]$ ist der aktuell höchste Index für Zahlen mit k Nullen}
  for all $k = 0,\ldots,d$ **do**
    $C[k] := 0$
  end for
  for $j := 0; j < 2^d; j := j+1$ **do**
    $n :=$ zähle Nullen in Binärdarstellung von j
    $A[j] := C[n]$
    $C[n] := C[n] + 1$
  end for

Wenn wir also $r_K(z)$ bestimmen wollen, so müssen wir lediglich $z \in (0,1)^d$ als Binärzahl mit u.U. führenden Nullen interpretieren und erhalten so die Äquivalenz der Problemstellungen. Wollen wir zu gegebenem z den Index $r_K(z)$ errechnen, so ermitteln wir die Antwort von obigem Ersatzproblem, also den Index j von $A_z = \sum_{i=0}^{d} z_i 2^i \in \mathbb{N}$ in der Menge $Q(A_z)$ und setzen $r_K(z) := j$. Algorithmus 4.2.2 zeigt eine Routine, die in Zeit $O(d \cdot 2^d)$ und Speicher $O(2^d)$ die gewünschten Resultate für alle $z \in (0,1)^d, |z|_1 \in \{1,\ldots,d\}$ vorberechnet, sodass $r_K(z)$ in Zeit $O(1)$ in einem Array unter dem Index mit Binärrepräsentation z zur Verfügung steht. Damit nun lässt sich auch $P(d,k,n)$ leicht vorberechnen,

0	32	16	33	8	34	17	35	4	36	18	37	9	38	19	39	1
48																56
24																28
49																57
12																14
50																58
25																29
51																59
6																7
52																60
26																30
53																61
13																15
54																62
27																31
55																63
2	40	20	41	10	42	21	43	5	44	22	45	11	46	23	47	3

Abbildung 4.6: Ein Beispiel für die Anwendung der Indexabbildung $r_B(\cdot,\cdot)$ auf die Randpunkte eines regulären dünnen Gitters in $d = 2$, $n = 4$.

da alle Werte von $r_K(z)$ tabellarisch vorliegen und $\max_{|z|_1=k} r_K(z)$ nur abgelesen werden braucht. Mit diesem Wissen wiederum kann man in einem Array der Größe $d \cdot n$ die Summen

$$P^{<m} + \sum_{k=K+1}^{d-1} P(d,k,m)$$

für alle $K \in \{1,\ldots,d\}, m \in \{1,\ldots,n\}$ leicht vorberechnen und in Zeit $O(1)$ verfügbar machen. Berechnet man auch I_n^d für die notwendigen Werte vor, so ist für die Ermittlung von $r_B(\mathbf{l},\mathbf{i})$ für beliebiges $(\mathbf{l},\mathbf{i})$ in Dimension d und Level n eines dünnen Gitters nur noch $\mathrm{BIN}(\mathbf{l}), r_z(\mathbf{i})$ und $r(\overline{\mathbf{l}},\overline{\mathbf{i}})$ auszurechnen. Dies lässt sich in einer einzigen FOR-Schleife effizient erledigen, insgesamt erhalten wir r_B mit folgenden Eigenschaften:

- Ermittlung für beliebige Randindizes $(\mathbf{l},\mathbf{i})$ in Dimension d, Level n in Zeit $O(d)$ in einer effizienten FOR-Schleife,
- einmaliges Aufstellen aller notwendigen vorzuberechnenden Werte in einer Laufzeit von $O\big(\max(d2^d, d\cdot n^2)\big)$ sowie einem Speicher von $O\big(\max(2^d, d\cdot n)\big)$. Dies macht in der Praxis für die in Frage kommenden n selbst für hohe Dimensionen weniger als 100KB aus.

Mit $r_B(\cdot,\cdot)$ ist es damit möglich, dünne Gitter auf dem Rand zu verwalten. Ein statischer Speicherverbrauch von 2^d Integerzahlen ist dabei keinesfalls eine Einschränkung. Dünne Gitter enthalten bereits 2^d Eckpunkte (d. h. Level $\mathbf{0}$). Schon bei geringen Levels ist die absolute Zahl von Punkten derart hoch, daß die Verwaltung dünner Gitter mit Randpunkten nur bis $d \approx 13$ machbar ist. In dieser Größenordnung benötigen 2^d Integerzahlen jedoch lediglich 32KB. Die Abbildung $r_B(\cdot,\cdot)$ ist ausgelegt für diejenigen Dimensionen, in denen auch die Verwaltung von Dünngitterpunkten auf dem Rand Sinn macht[3]. Abbildung 4.6

[3]Man beachte in diesem Zusammenhang, dass die Interpretation von $\mathrm{BIN}(\mathbf{l})$ als Zahl $A \in \mathbb{N}$ mit eingebauten Datentypen einer 32Bit Architektur nur möglich ist für $d \leq 32$, analog für 64Bit-Architekturen.

zeigt abschließend Randpunkte eines reguläres dünnen Gitters in $d = 2$, $n = 4$ mit den zu $r_B(\cdot, \cdot)$ gehörenden Indizes.

Algorithmus 4.2.3 TraversiereRand: Besucht rekursiv erst Randpunkte, dann innere Punkte eines dünnen Gitters.

Erfordert: Startknoten $(\mathbf{l}, \mathbf{i}) \in G^d$, Richtung m
  if $m = d$ **then**
    besuche $(\mathbf{l}, \mathbf{i})$
    besuche Bruder von $(\mathbf{l}, \mathbf{i})$, d. h. $i_m = 1$
  else
    rufe TraversiereRand auf mit $(\mathbf{l}, \mathbf{i})$ und Richtung $m + 1$
    rufe TraversiereRand auf für Bruder von $(\mathbf{l}, \mathbf{i})$ und Richtung $m + 1$
  end if
  Rufe TiefenSucheInnere auf für den Punkt mit $l_m = 1$, $i_m = 1$

Algorithmus 4.2.4 TiefenSucheInnere: führt Tiefensuche in Richtung m durch, ruft rekursiv Traversierung für die Knoten auf, die Bäume niederer Dimension enthalten

Erfordert: Startknoten $(\mathbf{l}, \mathbf{i})$, Richtung m
  stoppe, falls $(\mathbf{l}, \mathbf{i}) \notin G^d$
  if $m = d$ **then**
    besuche $(\mathbf{l}, \mathbf{i})$
  else
    rufe TraversiereRand auf mit $(\mathbf{l}, \mathbf{i})$ und Richtung $m + 1$
  end if
  rufe Tiefensuche auf für linken Sohn, d. h. mit $\tilde{l}_m = l_m + 1$, $\tilde{i}_m = 2i_m - 1$
  rufe Tiefensuche auf für rechten Sohn, d. h. mit $\tilde{l}_m = l_m + 1$, $\tilde{i}_m = 2i_m + 1$

4.2.3 Traversierung von Linien

Mit den in den vorigen Unterabschnitten entwickelten Indexabbildungen $r_I(\cdot, \cdot)$ sowie $r_B(\cdot, \cdot)$ ist es nun möglich, zu jedem Punkt in Multiindexdarstellung $(\mathbf{l}, \mathbf{i})$ eines regulären dünnen Gitters G_n^d einen eindeutigen skalaren Index in $O(d)$ Schritten zu ermitteln. Wie zu Anfang dieses Kapitels motiviert, ist somit die Verwaltung von d-dimensionalen Funktionen, dargestellt mittels hierarchischen Basen auf dünnen Gittern *ohne* einen Einfluss der Dimension auf den Speicherverbrauch möglich. Der Speicherverbrauch ist ausschliesslich bestimmt durch die Länge des Koeffizientenvektors aller $u_{\mathbf{l},\mathbf{i}}$. Da wir uns entschieden hatten, innere Punkte und Randpunkte getrennt zu behandeln, verwenden wir zwei Koeffizientenvektoren $U_I[\cdot]$ und $U_R[\cdot]$, die innere beziehungsweise Randkoeffizienten enthalten. Will man den Koeffizienten $u_{\mathbf{l},\mathbf{i}}$ wissen, so entscheidet man zunächst ob $(\mathbf{l}, \mathbf{i})$ ein Randpunkt ist oder nicht, errechnet dementsprechend $r_I(\mathbf{l}, \mathbf{i})$ beziehungsweise $r_B(\mathbf{l}, \mathbf{i})$ und schaut in dem entsprechenden Array nach. Wie bereits besprochen ist Adaptivität ebenso möglich mit Speicherverbrauch $O(N)$ und Zugriffskosten von $O(d)$ pro Knoten, siehe Seite 57.

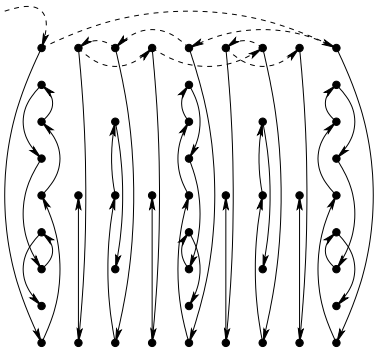

Abbildung 4.7: Reihenfolge der Traversierung TraversiereRand für das Gitter G_3^2.

Es verbleibt zu klären, wie man die in Kapitel 3 benötigten Traversierungen aller Linien in einer bestimmten Richtung oder aber das Berichten aller Dünngitterpunkte umsetzt. Da die gesamte Struktur dünner Gitter ausschliesslich logisch in Form der Multiindizes vorhanden ist, können Traversierungen nur mithilfe einer Vorschrift vorgenommen werden, die nach und nach alle Multiindizes eines (möglicherweise) adaptiven dünnen Gitters besucht und damit schließlich mittels $r_I(\cdot,\cdot)$ beziehungsweise $r_B(\cdot,\cdot)$ Zugriff auf die entsprechenden Koeffizienten erlangt. Derartige abstrakte Algorithmen, die lediglich Multiindizes besuchen, werden wir in diesem Unterabschnitt kennenlernen.

Besuchen aller Punkte

Um alle Punkte eines dünnen Gitters in beliebiger Reihenfolge zu besuchen, bedienen wir uns der zu Anfang dieses Kapitels beschriebenen Idee bei rekursiven Datenstrukturen. Dem folgend betrachten wir ein dünnes Gitter als einen Binärbaum in Richtung x_1, dessen Knoten wiederum Binärbäume der Dimension $(d-1)$ enthalten. Dies können wir durch eine rekursive Traversierungsmethode ausnutzen, die in x_1 Richtung schlichtweg eine Standard-Binärbaumtraversierung durchführt und jeweils rekursiv die $(d-1)$ dimensionalen, zu den Knoten gehörenden Unterstrukturen besucht. Dies realisieren wir mittels einem Aufruf von Algorithmus 4.2.3 mit den Parametern $(\mathbf{l},\mathbf{i}) = (\mathbf{0},\mathbf{0})$ und $m = 1$. Die Reihenfolge ist exemplarisch für ein reguläres Gitter in Dimension 2, Level 3 in Abbildung 4.7 dargestellt. Dabei bezeichnen die gestrichelten Linien die äußere Traversierung in Richtung x_1, durchgezogene Linien die innere in Richtung x_2.

Traversierung aller Linien in Richtung m

Eine wichtige Rolle spielt die Traversierung aller Linien des Gitters in Richtung m. Eine Linie beschreiben wir dabei immer durch einen Repräsentanten $(\mathbf{l},\mathbf{i})$ auf der Linie. Dazu verwenden wir den Punkt, dessen m.te Koordinate auf dem höchsten Level ist, also $l_m = 1$, $i_m = 1$ (dieser existiert aufgrund der Konstruktion immer). Die Repräsentanten können nacheinander durch Anwendung von Algorithmus 4.2.3 auf die Gitterscheibe mit $l_m = 1$, $i_m = 1$ besucht werden. Von einem Repräsentanten ausgehend kann die gewünschte eindimensionale Baumtraversierung auf der Linie in Richtung m durchgeführt werden.

Kapitel 5

Numerische Ergebnisse

Im Folgenden wenden wir uns der Untersuchung der Leistungsfähigkeit der in den letzten Kapiteln gewonnenen Algorithmen und Datenstrukturen zur Lösung einer elliptischen Differentialgleichung zu. Dabei befassen wir uns zunächst mit der Konditionierung des in Kapitel 3 eingeführten Gleichungssystems, die die Anzahl der benötigten Iterationen zur Lösung des Systems charakterisiert.

Daraufhin betrachten wir in Abschnitt 5.2 die Approximationsgüte der Diskretisierung. Wir bestätigen die asymptotischen Konvergenzaussagen für elliptische Differentialgleichungen bei fester Dimension d bezüglich $n \to \infty$ aus [1, 8, 9, 14] auch für höhere Dimensionen als $d = 2, 3$. Weiterhin untersuchen wir die Grenzen der Anwendbarkeit der Algorithmik in Bezug auf die Dimension und betrachten das Verhalten der Diskretisierung auch in Bezug auf variable Dimension d.

5.1 Kondition der Steifigkeitsmatrix

Bereits in Kapitel 3 haben wir uns mit der Entwicklung von effizienten Matrix-Vektor-Algorithmen als Bestandteil von Lösern des aus der hochdimensionalen Galerkindiskretisierung resultierenden linearen Gleichungssystems befasst. Bei Einsatz von Krylov-Verfahren können wir damit jede Iteration in Zeit $O(\text{Poly}(d)N)$ und Speicher $O(N)$ abarbeiten und es verbleibt, die Zahl der Iterationen zu beschränken. Ist die Steifigkeitsmatrix A symmetrisch (wie beispielsweise beim Helmholtzoperator), so bietet sich die Verwendung des CG-Verfahrens an. Bei nicht-symmetrischen Matrizen verwenden wir hier das BiCGStab-Verfahren (siehe [6]). Dieses resultiert für die Lösung von symmetrischen, positiv definiten Gleichungssystemen in derselben Zahl von Iterationen wie das CG-Verfahren, allerdings mit doppelten Kosten pro Iteration (vgl. [6]).

Für die Anzahl der Iterationen des CG-Verfahrens ist die Kondition

$$\kappa_A(d, n) = \frac{\lambda_{\max}(A)}{\lambda_{\min}(A)}$$

entscheidend. Die zur Reduktion des Anfangsfehlers um einen Faktor ϵ benötigte Zahl von Iterationen ist proportional zu $\sqrt{\kappa_A(d, n)}$, vgl. [6]. Dabei ist $\lambda_{\max}$ der größte und $\lambda_{\min}$ der kleinste Eigenwert von A. Bei den Experimenten in dieser Arbeit beobachten wir

$n\downarrow$ / $d\longrightarrow$	1	2	3	4	5	6	7	8	9	10	11
1	1	1	1	1	1	1	1	1	1	1	1
2	5.207	4.549	4.108	3.791	3.553	3.366	3.216	3.092	2.989	2.901	2.826
3	7.83	8.392	8.019	7.763	7.502	7.284	7.023	6.868	6.682	6.508	6.349
4	10.1	12.52	14.32	13.86	13.42	13.02	12.65	12.19	11.92	11.7	11.44
5	12.04	16.15	21.02	24.89	24.19	23.66	23.12	22.54	22.07	21.68	21.26
6	13.11	19.69	27.06	35.72	43.32	42.52	41.58	40.67	39.94	39.39	38.72
7	13.35	23.1	33.64	46.21	61.14	75.62	74.28	73.11	71.78	70.83	-
8	14.68	25.49	39.99	55.61	75.51	105.1	132.9	130.5	128.3	-	-
9	15.24	26.92	45.13	68.9	98.69	129.2	181.1	-	-	-	-

Tabelle 5.1: Die Konditionszahlen $\kappa(d,n)$ der vorkonditionierten Laplace-Steifigkeitsmatrix für $n=1,\ldots,9$ und $d=1,\ldots,11$.

auch für unsymmetrische Systeme vergleichbare Iterationszahlen wie bei symmetrischen Systemen. Dieses Verhalten ist bekannt, siehe [6]. Man vermutet auch für das BiCGStab-Verfahren einen Zusammenhang zwischen der Kondition und der Zahl der Iterationen. Durch Verwendung von Vorkonditionierern ist oftmals die Reduktion der Kondition und damit eine schnellere Lösung des Systems möglich.

Die Kondition der hier verwendeten Steifigkeitsmatrizen steigt bei fester Dimension mit dem Level der Diskretisierung an (vgl. [23]). Daher wird in [23] ein prewaveletbasierter Vorkonditionierer vorgestellt, der für die Steifigkeitsmatrix des Helmholtzoperators

$$L\mathbf{u} = -\sum_{m=1}^{d} a_m \partial^{mm}\mathbf{u} + b\mathbf{u}$$

mit homogenen Dirichletrandbedingungen und konstanten Koeffizienten eine levelunabhängige Kondition erzielt. Bei beliebiger, aber fester Dimension d ist die Kondition lediglich durch eine Konstante beschränkt,

$$\kappa_{\tilde{A}}(d,n) \leq c = \text{const.}$$

Dabei ist

$$\tilde{A} := D_A^{-1} \cdot A$$

die diagonal vorkonditionierte Matrix A, D_A ist eine Diagonalmatrix mit den Einträgen von A. In Bezug auf die Dimension ist i. a. anzunehmen, daß bei einem Tensorproduktansatz wie bei uns vorgenommen die Kondition exponentiell in der Dimension steigt, d. h.

$$\kappa_{\tilde{A}}(d,n) = O(\tilde{c}^d)$$

mit einer Konstanten $\tilde{c} > 1$ (siehe die theoretische Untersuchung dazu in [23]). Allerdings liegen keine genauen Untersuchungen bezüglich dieser Abhängigkeit der Kondition von der Dimension bei Dünngitterverfahren vor.

Da für die Lösung hochdimensionaler Probleme die Dimension einen entscheidenden Einfluss hat, ist es Inhalt dieses Abschnittes, die Dimensionsabhängigkeit $c = c(d)$ zu analysieren. Wir betrachten dazu die Vorkonditionierung der zu dem Laplaceoperator $L\mathbf{u} = -\Delta\mathbf{u}$ gehörenden Prewaveletsteifigkeitsmatrix $\tilde{A} := D_A^{-1} \cdot A$. Die Diskretisierung

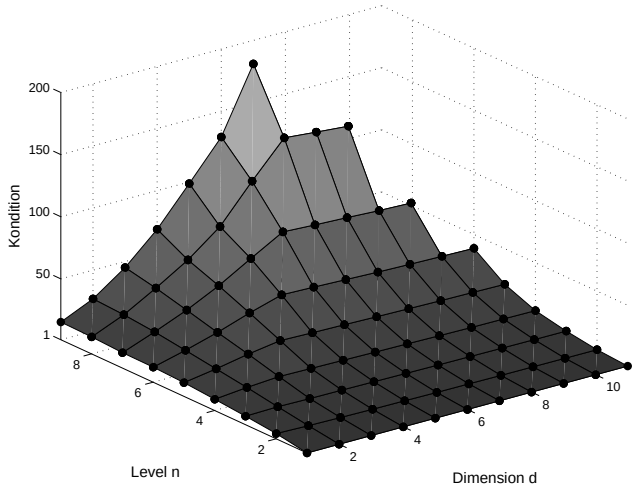

Abbildung 5.1: Visualisierung der Konditionszahlen $\kappa_{\tilde{A}}(d,n)$ der vorkonditionierten Laplace-Steifigkeitsmatrix. Auf der x_1-Achse eingetragen ist die Dimension, auf der x_2-Achse das Level sowie auf der x_3-Achse die Kondition.

führen wir dabei mit regulären dünnen Gittern des Levels n durch[1].Tabelle 5.1 zeigt die resultierenden Näherungen der Kondition $\kappa_{\tilde{A}}(d,n)$ für diverse Werte von n und d. Abbildung 5.1 visualisiert diese als diskrete Funktion mit zwei Variablen. Wir betrachten zunächst das Verhalten bei konstanter Dimension d. Für $d < 3$ sehen wir das aus [23] zu erwartende Konvergenzverhalten für steigendes n: in Dimension $d = 2$ steigt die Kondition zwar zunächst recht stark, so z. B. von $\kappa_{\tilde{A}}(2,2) = 4.549$ für $n = 2$ auf $\kappa_{\tilde{A}}(2,3) = 8.392$, bleibt aber beschränkt durch eine Konstante die vermutlich bei $c(2) \approx 27$ liegt. Für höhere Dimensionen ist diese Asymptotik aufgrund des durch Rechnerressourcen beschränkten Levels nicht zu sehen. So verdoppeln sich die Konditionszahlen für feste Dimension $d = 10$ bei $n = 6$ von $\kappa_{\tilde{A}}(10,6) = 39.39$ auf $\kappa_{\tilde{A}}(10,7) = 70.83$.

In Bezug auf die Dimension (d. h. bei festem Level) beobachten wir anhand von Abbildung 5.1 zweierlei Verhalten. Bei konstantem Level $n = 6$ entnehmen wir Tabelle 5.1, daß die Kondition $\kappa_{\tilde{A}}(1,n) = 13.11$ ansteigt auf $\kappa_{\tilde{A}}(2,n) = 19.69$ bis hin zu $\kappa_{\tilde{A}}(5,n) = 43.32$. Dann aber nimmt die Kondition sukzessive leicht ab auf $\kappa_{\tilde{A}}(6,n) = 42.52$ bis hin zu $\kappa_{\tilde{A}}(11,n) = 38.72$. Diese Eigenschaft beobachten wir auch bei den anderen Levels. Das Verhalten ändert sich dabei immer ab Dimension $d = n - 1$. Wir betrachten zunächst den Fall $d \geq n$ näher. Vernachlässigen wir den leichten Abfall der Konditionszahlen mit ansteigender Dimension, so entspricht die Kondition des d-dimensionalen Problems weitestgehend demjenigen des $\tilde{d} := (n-1)$-dimensionalen. Damit benötigt die Lösung eines Problems in Dimension $d = 10$ oder höher mit Level $n = 4$ annähernd genausoviele Iterationen wie ein Problem in Dimension $d = 3$ und Level $n = 4$. In diesem Fall ist also eine

[1]Damit ist auch der adaptive Fall erschlossen: bei Diskretisierung auf einem adaptiven Gitter, das Punkte mit Level $n(\mathbf{l}) \leq n$ hat, ist die Steifigkeitsmatrix eine Untermatrix von derjenigen einer regulären Dünngitterdiskretisierung mit Level n. Damit ist aufgrund von Minorargumenten auch das Spektrum – beziehungsweise die Kondition – abschätzbar durch den Fall regulärer dünner Gitter.

$n \downarrow$	$d \longrightarrow$ 1	2	3	4	5	6	7	8	9	10
1	1	1	1	1	1	1	1	1	1	1
2	1.75	1.75	1.75	1.75	1.75	1.75	1.75	1.75	1.75	1.75
3	2.04	3.06	3.06	3.06	3.06	3.06	3.06	3.06	3.06	3.06
4	2.17	3.58	5.36	5.36	5.36	5.36	5.36	5.36	5.36	5.36
5	2.22	4.18	6.26	9.38	9.38	9.38	9.38	9.38	9.38	9.38
6	2.23	4.44	7.32	11	16.4	16.4	16.4	16.4	16.4	16.4
7	2.21	4.52	8.55	12.8	19.2	28.7	28.7	28.7	28.7	28.7
8	2.23	4.76	9.06	14.9	22.4	33.6	50.3	50.3	50.3	50.3

Tabelle 5.2: Die Konditionszahlen $\kappa(d, n)$ der diagonalskalierten Massenmatrix für $n = 1, \ldots, 8$ und $d = 1, \ldots, 10$.

effiziente Lösung des Systems möglich.

Es stellt sich jedoch die Frage, wie groß die Kondition der Matrix in der Grenzdimension $\tilde{d} = n - 1$ ist. Wir betrachten dazu bei festem Level n und Dimension $d > 1$ den Quotient

$$K_n(d) := \frac{\kappa_{\tilde{A}}(d, n)}{\kappa_{\tilde{A}}(d-1, n)}$$

für den Fall $d < n$. Wir erhalten so beispielsweise für $n = 7$ die Quotienten $K_7(2) = 1.7$, weiterhin $K_7(3) = 1.46$ bis zu $K_7(5) = 1.32$. Für $K_7(6) = 1.23$ ist der Quotient sehr viel geringer, ab $d = n$ bleibt die Konditionszahl weitgehend gleich. Diese Messung ist mit vergleichbaren Resultaten auch für andere Level durchführbar. Dies deutet darauf hin, daß von Dimension $d = 1$ bis zu Dimension $\tilde{d} = n - 1$ ein exponentieller Anstieg $\kappa_{\tilde{A}}(d, n) = O(c^d)$ der Kondition vorliegt. Zusammen mit unseren Beobachtungen für den Fall $d \geq n$ vermuten wir deshalb einen Gesamtzusammenhang von

$$\kappa_{\tilde{A}}(d, n) = O(c^{\min\{d,n\}})$$

mit einer Konstante $c > 1$ und beliebiger Dimension d sowie Level n. Dies ist deutlich geringer als die von der Theorie zu erwartende Abhängigkeit $O(c^d)$, zumal der Fall $n < d$ in hohen Dimensionen aufgrund der hohen Zahl von Unbekannten eher realisierbar ist als der Fall $n \geq d$.

Der Grund für das exponentielle Verhalten liegt nach [23] in der Tensorproduktstruktur des diskreten Operators. Dadurch multiplizieren sich die Konditionen auf, was das exponentielle Verhalten erklärt. Während die auf den Prewaveletkoeffizienten basierende Vorkonditionierung zu einer levelunabhängigen Kondition führt, erhält man durch die $(d - 1)$ Massenanteile der Laplacematrix ein exponentiell dimensionsabhängiges Verhalten. Dies ist ersichtlich anhand der Kondition der Massenmatrix. Tabelle 5.2 zeigt die zugehörigen Werte. Hier ist deutlich das oben beobachtete exponentielle Verhalten im Fall $n > d$ zu sehen. Gleichzeitig wird ersichtlich, daß bei konstantem Level n und Dimension $d \geq n - 1$ die Kondition *konstant* bleibt. Dieses Verhalten scheint die Konditionierung der Laplacematrix in dem Tensorprodukt zu verursachen. Ein möglicher Grund liegt in der Gitterstruktur.

Dazu betrachten wir die Verteilung der Gitterpunkte im Raum. Mit $r_\mathbf{l} := |\{m | l_m = 1\}|$

erhalten wir für jeden beliebiger Multiindex $(\mathbf{l}, \mathbf{i})$ im Innern des dünnen Gitters G_n^d

$$\begin{aligned} n(\mathbf{l}) = \sum_{m=1}^{d} (l_m - 1) + 1 &\leq n \\ \Rightarrow d - r_{\mathbf{l}} + 1 \leq n(\mathbf{l}) &\leq n \\ \Rightarrow r_{\mathbf{l}} &\geq d - n + 1. \end{aligned}$$

Daher sind für jeden Dünngitterpunkt $x_{\mathbf{l},\mathbf{i}}$ mindestens $(d - n + 1)$ Komponenten in der Mitte des Gitters, also $x_{l_m,i_m} = x_{1,1} = \frac{1}{2}$. Wenn also nur $O(n)$ viele Komponenten in das Tensorprodukt eingehen, erscheint dies als Ursache für die Eigenschaft $\kappa_{\tilde{A}}(d, n) = O(c^n)$ im Fall $n < d$ plausibel. Gleichzeitig läßt unsere Untersuchung vermuten, daß durch Verwendung eines Vorkonditionierers, der zu der optimalen Kondition 1 der eindimensionalen *Masse* führt, die exponentielle Dimensionsabhängigkeit vermeidbar ist. Eine Möglichkeit wäre die Verwendung einer vollständig L_2-orthonormalen Basis; aber auch auf dem Erzeugendensystem basierende Techniken wie in [21] für den Diffusionsteil vorgestellt sind denkbar. Weitere Untersuchungen mögen hier in der Zukunft Klarheit bringen.

5.2 Untersuchung der Diskretisierung

Im Folgenden betrachten wir die Diskretisierungseigenschaften des entwickelten Verfahrens. Nach dem Lemma von Céa kann der Diskretisierungsfehler in der durch den Operator induzierten Energienorm gegen den Interpolationsfehler abgeschätzt werden (vgl. [12]). Dies führt zu der Ordnung 1 bei H_2-Regularität der Lösung. Für klassische Finite Elemente lässt sich der Fehler in der L_2-Norm mithilfe des Lemmas von Aubin/Nitsche (siehe [12]) abschätzen, die Ordnung des Verfahrens ist damit wie bei der Interpolation 2. Die nach Frehse/Rannacher bekannte Abschätzung in der stärkeren L_∞-Norm weist eine um einen $|\log h|$ verringerte Ordnung 2 auf, siehe [15].

Für Dünngitterverfahren konnten die Fehlerabschätzungen in L_2- und L_∞-Norm bislang nicht erbracht werden. Dennoch weisen zahlreiche numerische Experimente mit Galerkinverfahren in Dimensionen $d = 2, 3$ u. a. in [9, 11, 14] darauf hin, daß sich der Diskretisierungsfehler durch den Interpolationsfehler abschätzen lässt.

In diesem Abschnitt führen wir diese Untersuchung der L_2- und L_∞-Fehler auch für höhere Dimensionen fort. Dazu betrachten wir zunächst analog zu den genannten Vorarbeiten die Dimension d als konstant und untersuchen die Asymptotik $n \to \infty$. Im nächsten Schritt untersuchen wir den bislang unbekannten Einfluss der Dimension auf die Fehlerabschätzungen, indem wir das Level festhalten und die Dimension variieren.

Allgemein ist dabei aufgrund der begrenzten Ressourcen eines Rechners oftmals nur die Präasymptotik erreichbar; das Level n ist dann deutlich kleiner als die Dimension. Dadurch enthält die diskrete Approximation an eine Funktion keinerlei Information ausserhalb der Mittelkreuzes des Gitters (dies sind die Punkte mit $x_m = \frac{1}{2}$ für mindestens eine Koordinate), siehe die Diskussion auf Seite 72. In Abhängigkeit von der darzustellenden Funktion ist hier ggf. durch adaptive Verfeinerung eine passende Darstellung zu ermitteln.

5.2.1 Messverfahren

Zur Ermittlung des Fehlers $\mathbf{u} - u$ einer kontinuierlichen Funktion $\mathbf{u}$ gegen eine Dünngitterapproximation $u = \sum_{(\mathbf{l},\mathbf{i})\in G^d} u_{\mathbf{l},\mathbf{i}}\psi_{\mathbf{l},\mathbf{i}}$ an $\mathbf{u}$ stellt sich vor allem die Frage, wie die Fehlernormen ausgewertet werden können. Uns interessieren die L_2 Norm, die L_∞ Norm sowie die Energienorm des Fehlers. Da diese jedoch alle nur approximativ ermittelbar sind, ist eine für verlässliche Aussagen hinreichend hohe Genauigkeit sicherzustellen. Optimal wäre die Interpolation von sowohl der Referenzlösung als auch der diskreten Lösung auf ein sehr feines volles Gitter mit anschließender Normauswertung. Dies erlaubt jedoch aufgrund des exponentiellen Wachstums der Anzahl der Punkte eines vollen Gitters lediglich die Auswertung bis Dimension $d = 3$. Alternativ ist die Interpolation mit anschließender Fehlerauswertung auf einem feinen *dünnen* Gitter möglich. Dies jedoch führt zusätzliche Fehler durch den logarithmischen Term bei der Interpolation ein und erschwert insbesondere die Messung von Konvergenzraten, für die wiederum logarithmische Terme eingehen und genau gemessen werden müssen. Ein Vergleich dieser beiden Messmethoden wird in [14] für $d \leq 3$ durchgeführt, wobei beides akzeptable Genauigkeit lieferte. Da zudem die gitterbasierte Fehlerauswertung einer Galerkinlösung algorithmisch einfach ist (im wesentlichen eine Multiplikation mit der Massen- beziehungsweise Steifigkeitsmatrix) wird der Fehler häufig auf dünnen Gittern gemessen (so etwa in [1, 8, 9, 11]). Für Dimensionen $d > 3$ sind volle Gitter nicht mehr machbar, und bei dünnen Gitter wird die Messung mit steigender Dimension immer stärker durch den logarithmischen Strafterm bei der Interpolation beeinträchtigt. Numerische Tests zeigen, daß damit selbst die nach dem Lemma von Céa theoretische klare Abschätzung in der Energienorm (vgl. [12]) nicht nachgewiesen werden kann. Aus diesem Grund verwenden wir hier alternative Messverfahren, um auch in hohen Dimensionen verlässliche Messwerte zu erhalten. Für die L_∞-Norm erreichen wir dies, indem man den maximalen Fehler nicht nur an Punkten eines feinen dünnen Gitters ermittelt, sondern auch an (konstant vielen) Punkten außerhalb des Gitters. Da meist nur Punkte des Mittelkreuzes zur Darstellung von Funktionen möglich sind verwenden hier die Punkte entlang einer Diagonalen durch den Einheitswürfel,

$$\{(\frac{\pi}{8},\ldots,\frac{\pi}{8}), \{r \cdot \frac{1}{15} \cdot \mathbf{1} | r = 1,\ldots,14\} \subset \mathbb{R}^d.$$

Durch $\frac{\pi}{8} \approx 0.39$ berücksichtigen wir einen Punkt, der für kein n in einem dünnen Gitter enthalten ist, die restlichen Punkte dienen zur Abtastung entlang einer Diagonalen. Aufgrund von Symmetrie der weiter unten vorgestellen Modellprobleme ist so eine hinreichend hohe Genauigkeit möglich.

Um genaue L_2-Normauswertungen durchführen zu können, bietet sich die Ausnutzung von numerischen Quadraturmethoden an. Da jedoch die Basisfunktionen unserer Dünngitterapproximation stückweise linear sind, ist die Anwendung von Verfahren mit hohen Glattheitsanforderungen nicht möglich. Eine Auswertung mit Monte-Carlo-Verfahren ist möglich aber sehr aufwändig. Hat die Lösung $\mathbf{u}$ wie in unserem Fall eine Produktstruktur

$$\mathbf{u}(x) = \prod_{m=1}^{d} g_m(x_m),$$

so ist die Auswertung der L_2-Norm des Fehler möglich mithilfe von *eindimensionalen* numerischen Quadraturformeln. Dazu betrachten wir

$$\|\mathbf{u} - u\|_{L_2}^2 = \|\mathbf{u}\|_{L_2}^2 - 2 \cdot \sum_{(\mathbf{l},\mathbf{i}) \in G^d} u_{\mathbf{l},\mathbf{i}} \int_\Omega \mathbf{u}(x) \psi_{\mathbf{l},\mathbf{i}}(x) dx + \|u\|_{L_2}^2.$$

Den Anteil $\|u\|_{L_2}^2$ können wir exakt mittels

$$\|u\|_{L_2} = \sqrt{\bar{u}^T M \bar{u}}$$

mithilfe der in Kapitel 3 entwickelten Matrix-Vektor-Algorithmen errechnen. Das Integral $\int_\Omega \mathbf{u}(x)^2 dx$ ist bekannt und für

$$\int_\Omega \mathbf{u}(x) \psi_{\mathbf{l},\mathbf{i}}(x) dx = \prod_{m=1}^{d} \int_0^1 g_m(x) \psi_{l_m,i_m}(x_m) dx_m$$

wenden wir *stückweise* eindimensionale Gaussquadratur auf all den Intervallen an, auf denen ψ_{l_m,i_m} linear ist. [2] Damit erhalten wir auch in hohen Dimensionen für derartige Funktionen $\mathbf{u}$ eine genaue Methode zur Ermittlung von L_2-Fehlern. Eine weitere Einschränkung an $\mathbf{u}$ erhalten wir jedoch durch das Laufzeitverhalten der Multiplikation $M \cdot \bar{u}$. Wie in Kapitel 3 diskutiert, ist eine Laufzeit von $O(\text{Poly}(d) \cdot N)$ möglich, wenn die Orthogonalität der Ansatzfunktionen geeignet ausgenutzt wird. Für die Aufstellung der rechten Seite ist dies auch noch bei Berücksichtigung von nicht-verschwindenden Randanteilen möglich (vgl. 3.1.4), bei der L_2-Norm beträgt die Laufzeit aber $O(2^d \cdot \text{Poly}(d) \cdot N)$. Aus diesem Grund beschränken wir uns zur Untersuchung von Konvergenzraten auf Modellprobleme, deren Lösungen homogene Dirichletrandbedingungen haben.

Den Fehler in der Energienorm schließlich werden wir nicht näher untersuchen, da das Galerkinverfahren die Bestapproximation bezüglich der Bilinearform $a(\cdot,\cdot)$ darstellt und der Diskretisierungsfehler somit schon von der Theorie durch den Interpolationsfehler abgeschätzt werden kann (siehe [12]).

5.2.2 Untersuchung der Asymptotik $n \to \infty$

Aus den numerischen Ergebnissen in [9, 14] u. a. erwarten wir, daß der Fehler der mittels dem in Kapitel 3 entwickelten Galerkinverfahren für elliptische partielle Differentialgleichungen gewonnenen diskreten Lösung u_n^d gegen $\mathbf{u}$ sich durch den Interpolationsfehler $\|\mathbf{u} - I_{G_n^d}\mathbf{u}\|$ nach oben abschätzen lässt. Wie in Kapitel 2 angegeben, gelten für den Interpolationsfehler $\mathbf{e}_n^d := \mathbf{u} - I_{G_n^d}\mathbf{u}$ in der L_2- und L_∞-Norm die oberen Schranken

$$\|\mathbf{u} - I_{G_n^d}\mathbf{u}\|_{L_\infty} \leq \frac{2 \cdot |\mathbf{u}|_{\mathbf{2},\infty}}{8^d} \cdot 2^{-2n} \cdot A(d,n), \tag{5.1}$$

$$\|\mathbf{u} - I_{G_n^d}\mathbf{u}\|_{L_2} \leq \frac{2 \cdot |\mathbf{u}|_{\mathbf{2},2}}{12^d} \cdot 2^{-2n} \cdot A(d,n), \tag{5.2}$$

$$\tag{5.3}$$

[2] Eine entsprechende Technik läßt sich für rechte Seiten mit Produktstruktur verwenden, wo ähnliche Integrale auftreten.

mit der Hilfsgröße

$$A(d,n) := \sum_{m=0}^{d-1} \binom{n+d-1}{k},$$

siehe Abschnitt 2.2.1 auf Seite 14. Bei fester Dimension d lassen sich alle nur von d abhängigen Koeffizienten (sowie die Norm der gemischten zweiten Ableitungen von $\mathbf{u}$) als konstant ansehen. Wir erhalten dann unter Vernachlässigung von Termen niedriger Ordnung für den Übergang von Level $(n-1)$ auf n für den Interpolationsfehler

$$\|\mathbf{e}_n^d\|_{L2,\infty} \approx \frac{1}{4\left(\frac{n}{n+1}\right)^{d-1}} \cdot \|\mathbf{e}_{n-1}^d\|_{L2,\infty}, \tag{5.4}$$

wie durch Umformung von (2.12) und (2.13) erreicht werden kann. Dementsprechend erwarten wir für den Diskretisierungsfehler

$$e_n^d := \mathbf{u} - u_n^d$$

ein ähnliches Konvergenzverhalten. Um dies zu verifizieren, untersuchen wir die Reduktionsfaktoren für $M \in \{L_2, L_\infty\}$,

$$\rho_{n,M}^d := \frac{\|e_{n-1}^d\|_M}{\|e_n^d\|_M}.$$

Die Diskretisierung führen wir, wie im letzten Kapitel besprochen, mit dem Unidirektionalen Prinzip durch. Die rechte Seite f diskretisieren wir durch Interpolation von f auf das Diskretisierungsgitter und anschließender Multiplikation mit der Massenmatrix, vgl. Abschnitt 3.1.4. Nach dem Lemma von Strang führt diese „numerische Quadratur“ schlimmstenfalls einen Fehler von der Ordnung des Interpolationsfehlers in die Diskretisierung ein, vgl. [14].

Konvektion/Diffusion Wir betrachten nun exemplarisch das Modellproblem

$$-\Delta\mathbf{u}(x) + \sum_{m=1}^{d} \partial^m \mathbf{u}(x) = f(x), \tag{5.5}$$

für $x \in [0,1]^d$ und $\mathbf{u}(x) = 0$ für $x \in \partial[0,1]^d$. Die rechte Seite f ist dabei so gewählt, daß

$$\mathbf{u}(x) = \prod_{m=1}^{d} \sin \pi \cdot x_m$$

die Lösung der Differentialgleichung ist.

Um an dieser Stelle das vorteilhafte Kosten/Nutzen-Verhältnis von dünnen Gittern im Gegensatz zu vollen Gittern darzustellen, führen wir die Diskretisierung in Dimension $d = 3$ mithilfe regulärer voller und regulären dünnen Gittern durch und geben jeweils den L_2-Fehler sowie die dazu notwendigen Punkte an. Abbildung 5.2 zeigt die zur Lösung notwendige Anzahl Unbekannte auf der x_1-Achse gegen den L_2-Fehler auf der x_2-Achse

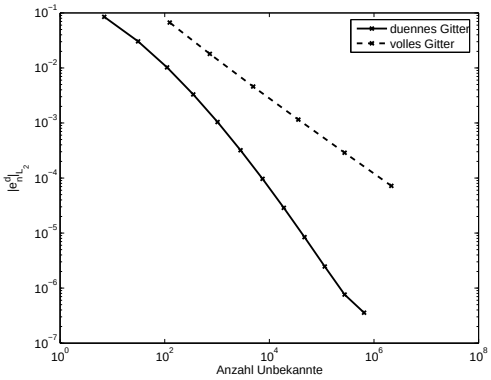

Abbildung 5.2: Gegenüberstellung der Vollgitterdiskretisierung von (5.5) und der Dünngitterdiskretisierung in $d = 3$. Dargestellt ist der L_2-Fehler gegen die Zahl der Unbekannten in doppelt-logarithmischer Skala.

aufgetragen, beides in logarithmischer Skala. Die Dünngitterdiskretisierung benötigt für einen vorgegebenen Fehler signifikant weniger Unbekannte als die Vollgitterdiskretisierung. So erreicht die Dünngitterdiskretisierung mit 2815 Punkten einen Fehler von $\|e_n^d\|_{L_2} = 3.2 \cdot 10^{-4}$, während für eine vergleichbare Genauigkeit bereits 274625 Vollgitterpunkte notwendig sind (bei einem L_2 Fehler von $2.88 \cdot 10^{-4}$). Wir betrachten die Fehlerentwicklung im Verhältnis zur Zahl der Unbekannten,

$$\|e_n^d\| = O(N^{-\alpha}).$$

In Abbildung 5.2 entspricht $-\alpha$ der (asymptotischen) Steigung der Funktionsgraphen in doppelt-logarithmischer Skala. Für die Vollgitterdiskretisierung messen wir $\alpha \approx 0.7$ während die Dünngitterdiskretisierung eine Ordnung $\alpha \approx 1.34$ aufweist. Tatsächlich degeneriert das Kosten/Nutzenverhältnis für volle Gitter wegen

$$O(2^{-2n}) = O(2^{-d \cdot n \cdot \alpha}) = O(N^{-\frac{2}{d}})$$

exponentiell in der Dimension, siehe die Diskussion in Abschnitt 2.2.1 auf Seite 16. Dünne Gitter weisen ein vorteilhafteres Kosten/Nutzen-Verhältnis auf. Bei $N = |G_n^d|$ vielen Unbekannten beträgt der Interpolationsfehler

$$\|\mathbf{e}_n^d\| = O(N^{-2} \cdot |\log_2 N|^{3 \cdot (d-1)}),$$

siehe Seite 16. Der Fluch der Dimensionen ist daher auf den Logarithmus beschränkt. Diese Diskussion für den Interpolanten lässt sich auch für den Fehler in L_∞-Norm, Energienorm sowie allgemeinere Ansatzfunktionen führen (siehe [9, 11]). Wir wenden uns jetzt jedoch der Untersuchung der genannten Vermutung zu, daß der Fehler der Galerkindiskretisierung maximal von der Größenordnung des Interpolationsfehlers ist.

Dazu untersuchen wir die Fehlerentwicklung bei der Dünngitterdiskretisierung von (5.5) im Falle $d \geq 2$. Wir fassen die Messergebnisse zusammen in Abbildung 5.3 für die Dimensionen $d = 2, \ldots, 8$. Dargestellt sind die Diskretisierungsfehler in L_2- und L_∞-Norm,

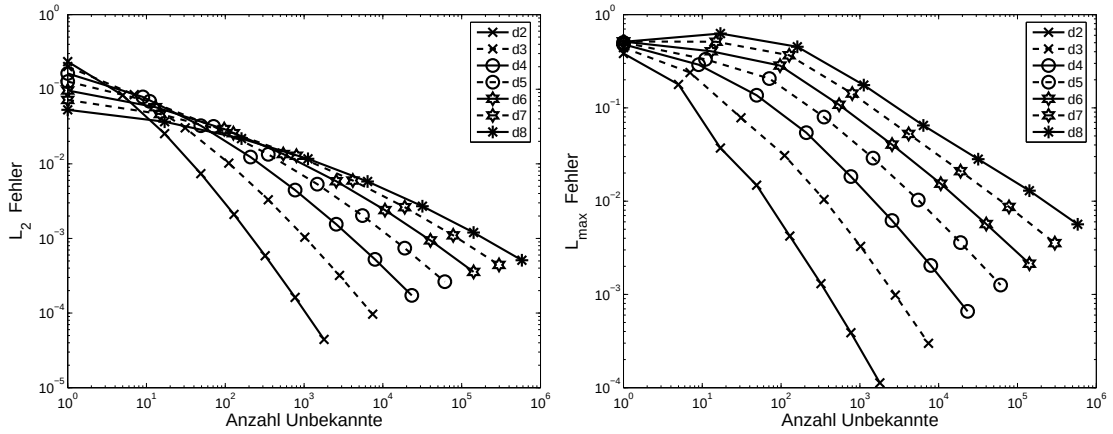

Abbildung 5.3: Dargestellt sind die Konvergenzgraphen für (5.5), links in L_2-Norm und rechts in L_∞-Norm. Aufgetragen sind auf der x-Achse die Zahl der Unbekannten und auf der y-Achse die entsprechenden Fehlermessungen.

	$d=3$	$d=4$	$d=5$	$d=6$	$d=7$	$d=8$
$4 \cdot \left(\frac{n-1}{n}\right)^{d-1}$	3.06	2.68	2.34	2.05	1.8	1.57
ρ^d_{n,L_2}	3.31	3.04	2.82	2.64	2.49	2.36
ρ^d_{n,L_∞}	3.29	3.11	2.85	2.66	2.45	2.3

Tabelle 5.3: Reduktionsfaktoren der Dünngitterdiskretisierung von (5.5) in verschiedenen Dimensionen.

jeweils die Zahl der Unbekannten gegen den Fehler in doppelt-logarithmischer Darstellung. Uns interessiert dabei sowohl der Zusammenhang zum Interpolationsfehler als auch ein gutes Kosten/Nutzen-Verhältnis der Diskretisierung.

Zu festem $n = 8$ messen wir die in Tabelle 5.3 dargestellten Werte. Gegenübergestellt sind die Reduktionsfaktoren in L_2- und L_∞-Norm mit dem aus der Interpolationsfehlerabschätzung (5.4) zu erwartenden Faktor $4 \cdot \left(\frac{n-1}{n}\right)^{d-1}$. Die Messwerte zeigen eine stärkere Reduktion des Diskretisierungsfehlers als von der Interpolationsfehlerschranke zu erwarten. So ist für $d = 3$ der erwartete Faktor von 3.06 den Messwerten $\rho^d_{n,L_2} = 3.31$ und $\rho^d_{n,L_\infty} = 3.29$ gegenübergestellt; in $d = 8$ entsprechend der erwarteten Faktor von 1.57 den Messwerten $\rho^d_{n,L_2} = 2.36$ und $\rho^d_{n,L_\infty} = 2.3$.

Dies bestätigt die bereits in [9, 11, 14] für Dimension $d = 2, 3$ geäußerte Vermutung, daß sich der Diskretisierungsfehler bei fester Dimension in manchen Fällen sogar besser verhält als aus der Interpolationsfehlerschranke zu erwarten. In [22] wird für ein dünngitterbasiertes Finite Differenzen Verfahren in Dimension $d = 2$ exemplarisch eine Konvergenzordnung von 2 wie bei vollen Gittern gemessen. Um dies für unser Galerkinverfahren zu untersuchen, ermitteln wir an einem Beispiel die resultierenden Diskretisierungsfehler, wenn man die rechte Seite durch numerische Quadraturmethoden anstelle der diskutierten Interpolationstechnik diskretisiert. Dann entfällt der nach dem Lemma von Strang einfließende logarithmische Term durch die Interpolation und wir messen nur den Diskretisierungsfehler. Dazu bezeichnen wir die mithilfe der Interpolationstechnik diskretisierten

	mit $\tilde{f}_{\mathbf{k},\mathbf{j}}$		mit $\hat{f}_{\mathbf{k},\mathbf{j}}$	
n	$\|e\|_{L_2}$	ρ_{L_2}	$\|e\|_{L_2}$	ρ_{L_2}
2	5.75e-02	-	1.25e-02	-
3	2.93e-02	1.97	3.46e-03	3.6
4	1.35e-02	2.16	9.36e-04	3.69
5	5.85e-03	2.31	2.52e-04	3.72
6	2.40e-03	2.44	6.74e-05	3.73
7	9.42e-04	2.55	1.80e-05	3.74
8	3.57e-04	2.64	4.82e-06	3.74
9	1.31e-04	2.73	1.29e-06	3.75
10	4.67e-05	2.8	3.42e-07	3.76

Tabelle 5.4: Der L_2-Diskretisierungsfehler des Modellproblems (5.5) in Dimension $d = 6$, linkerhand bei Diskretisierung der rechten Seite mithilfe von Interpolation gemäß Abschnitt 3.1.4 und rechterhand durch Verwendung von eindimensionalen Quadraturformeln.

rechte Seite mit $\tilde{f}_{\mathbf{k},\mathbf{j}}$, siehe Abschnitt 3.1.4. Aufgrund der Produktstruktur der Lösung hat auch die rechte Seite Produktstruktur. Damit ist die Anwendung von eindimensionalen numerischen Quadraturformeln $Q(\cdot)$ zur Aufstellung der rechten Seite möglich (vgl. mit der Messtechnik in Unterabschnitt 5.2.1). Die so gewonnenen Werte für die rechte Seite bezeichnen wir mit $\hat{f}_{\mathbf{k},\mathbf{j}}$. Tabelle 5.4 zeigt exemplarisch für Dimension $d = 6$ in der linken Spalte die L_2-Reduktionsfaktoren für die Diskretisierung mit $\tilde{f}_{\mathbf{k},\mathbf{j}}$ und in der rechten Spalte die entsprechenden Messungen mit $\hat{f}_{\mathbf{k},\mathbf{j}}$. Wie wir sehen, ist die Reduktion bei genauerer Integration der rechten Seite deutlich näher an 4 als bei der bisher verwendeten Technik. Jedoch deutet der Faktor für $n = 10$, $\rho^d_{10,L_2} = 3.76$ darauf hin, daß auch ohne den Einfluss der Diskretisierung der rechten Seite keine volle Ordnung 2 erreicht werden kann. Im allgemeinen ist vermutlich ein Einfluss von logarithmischen Reduktionsfaktoren zu erwarten.

Abbildung 5.3 zeigt auch die Kosten der Diskretisierung im Verhältnis zum Nutzen auf. Wie in dem bereits oben betrachteten Fall $d = 3$ verwenden wir die Zahl der Unbekannten (d. h. die Zahl der inneren Gitterpunkte) als Kostenmaß und den damit zu erzielenden Fehler als Nutzen. Für höhere Dimension sehen wir im Vergleich zum Fall $d = 3$ deutlich den Einfluss des logarithmischen Strafterms in den Fehlerabschätzungen, der eine Auffächerung der Konvergenzgraphen bewirkt. Das Kosten/Nutzen-Verhältnis bleibt dabei jedoch ebenso wie oben für $d = 3$ dargestellt günstig. Durch die in Kapitel 3 entwickelten Algorithmen sowie den in Abschnitt 5.1 vorgestellten Vorkonditionierer gibt die Zahl der Unbekannten auch ein gutes Maß für die Laufzeit an, da $O(K\text{Poly}(d)N)$ Rechenschritte zur Lösung des Gleichungssystems benötigt werden, wobei K die Zahl der Iterationen bezeichnet. Letztere unterliegt den den in Abschnitt 5.1 beobachteten Einschränkungen. Jedoch entstehen weitere Zeitkosten für die Diskretisierung der rechten Seite (i. a. auch der Behandlung von Dirichletrandbedingungen). Die dafür notwendigen Randpunkte des dünnen Gitters haben zwar asymptotisch für $n \to \infty$ und konstante Dimension dieselbe Größenordnung wie die Zahl der Unbekannten, jedoch mit einem dimensionsabhängigen großen Koeffizienten. Die genaue Zahl der dafür notwendigen Punkte ist in Abbildung 5.4 für die Dimensionen $d \in \{6, 8, 10, 12, 14\}$ dargestellt (vgl. (2.6) auf Seite 11). Die x_1-Achse zeigt das Level des Gitters, die x_2-Achse die Zahl der Randpunkte in logarithmischer Ska-

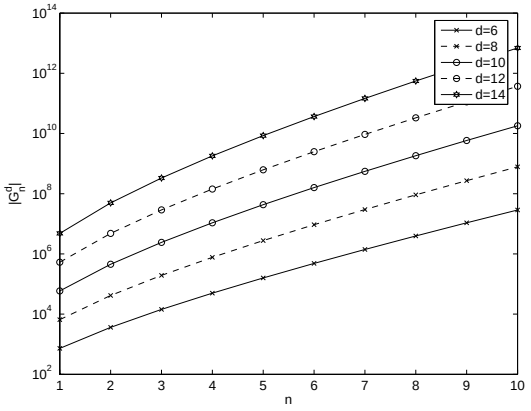

Abbildung 5.4: Die Zahl der Gitterpunkte inklusive Rand für einige ausgewählte Dimensionen, siehe (2.6) auf Seite 11.

	$d=2$	$d=3$	$d=4$	$d=5$	$d=6$	$d=7$	$d=8$
$4 \cdot \left(\frac{n-1}{n}\right)^{d-1}$	3.5	3.06	2.68	2.34	2.05	1.8	1.57
ρ^d_{n,L_2}	3.91	3.61	3.29	3.02	3.80	2.62	2.45
ρ^d_{n,L_∞}	3.47	3.5	3.56	3.17	2.92	2.72	2.57

Tabelle 5.5: Reduktionsfaktoren der Dünngitterdiskretisierung von (5.6) in verschiedenen Dimensionen.

la. Dabei wird deutlich, daß mit steigender Dimension die Behandlung der Randpunkte die Zeit- und Speicherkosten dominiert. So gibt es beispielsweise für $d = 8$ und $n = 8$ eine Zahl von ca. 580.000 Unbekannten im Gegensatz zu ca. 91.000.000 Randpunkten. An dieser Stelle bewährt sich die in Kapitel 4 entwickelte Datenstruktur, da damit die rechte Seite mit Speicherkosten $c \cdot |G_n^d|$ [3] errechnet werden kann. Dennoch stellt dieser Teil der Diskretisierung technisch die Grenze der behandelbaren Probleme dar. Alternative Algorithmen zur Behandlung der rechten Seite beziehungsweise der Randwerte sind derzeit nur für Spezialfälle (z. B. Ausnutzung von vorhandener Produktstruktur) bekannt, sodaß i. a. die technische Grenze für dieses Verfahren bei Dimension $d \approx 13$ liegt. Spezialfälle werden wir im nächsten Unterabschnitt kennenlernen.

Gemischte zweite Ableitungen. Wir untersuchen nun ein weiteres Modellproblem im Hinblick auf die Konvergenzraten in hohen Dimensionen. Wir betrachten den Diskretisierungsfehler von

$$-\nabla \mathbf{u}^T(x) C \nabla \mathbf{u}(x) + 3\mathbf{u}(x) = f(x) \tag{5.6}$$

für $x \in (0,1)^d$ mit der homogenen Randbedingung $\mathbf{u}|_\Gamma = 0$. Die positiv definite, symme-

[3] Die Konstante beträgt für Kopien der Koeffizienten je nach verwendetem Differentialoperator ca. $c \approx 4,5$. Für Rechner mit geringem Hauptspeicher können wir in der im Rahmen dieser Arbeit entwickelten Software momentan ungenutzte Koeffizientenvektoren auf die Festplatte auslagern und so die Konstante auf $c \approx 3$ drücken.

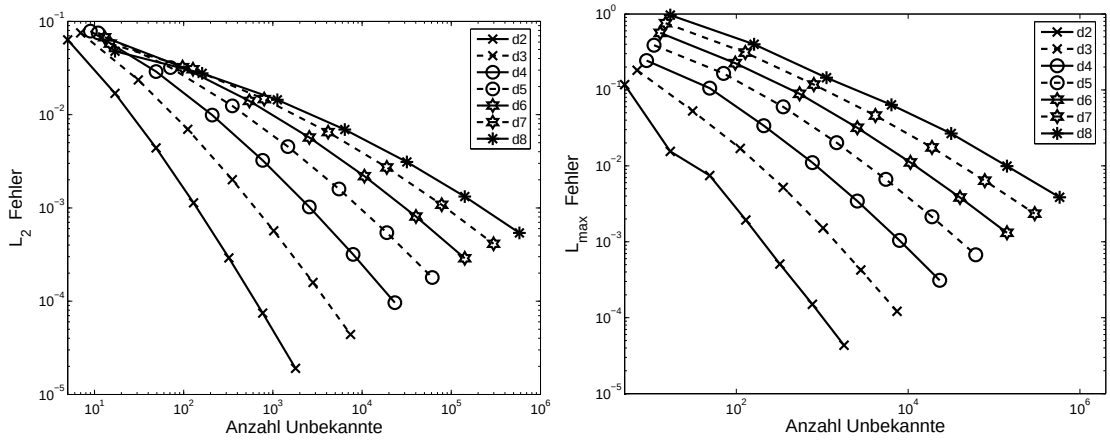

Abbildung 5.5: Dargestellt sind die Konvergenzgraphen für (5.6), links in L_2-Norm und rechts in L_∞-Norm. Aufgetragen sind auf der x-Achse die Zahl der Unbekannten und auf der y-Achse die entsprechenden Fehlermessungen.

trische Matrix C ist definiert durch

$$C_{mr} = \begin{cases} 2 & m = r \\ 1 & m \neq r. \end{cases}$$

Die rechte Seite wählen wir so, daß

$$\mathbf{u}(x) = \prod_{m=1}^{d} 4 \cdot x_m \cdot (1 - x_m)$$

die Lösung von (5.6) ist. Die Messergebnisse sind in Abbildung 5.5 dargestellt. Wir berechnen wiederum die Fehlerquotienten $\rho^d_{n,M}$ für $M \in \{L_\infty, L_2\}$ für das Level $n = 8$ und erhalten so die in Tabelle 5.5 dargestellten Werte. Auch hier lässt sich also der Diskretisierungsfehler durch den Interpolationsfehler bei fester Dimension abschätzen: für $d = 8$ messen wir eine Fehlerreduktion von $\rho^d_{n,L_2} = 2.45$ sowie $\rho^d_{n,L_\infty} = 2.57$ gegenüber einem erwarteten Wert von 1.57.

Adaptiv Die bislang für glatte Lösungen ermittelten Konvergenzraten lassen sich auch für singuläre Lösungen von elliptischen Gleichungen mittels Adaptivität erreichen. Die entsprechenden Techniken sind bereits aus der Literatur bekannt (siehe [11] und die darin enthaltenen Zitate) und lassen sich auch für höhere Dimensionen führen, daher beschränken wir uns lediglich auf ein Beispiel. Wir betrachten das Modellproblem

$$-\Delta \mathbf{u}(x_1, x_2) + 3\mathbf{u}(x_1, x_2) = f(x_1, x_2), \tag{5.7}$$

wobei wir die Randbedingung und rechte Seite so wählen, daß

$$\mathbf{u}(x_1, x_2) = 4 \cdot x_1 \cdot (1 - x_1) \cdot \sqrt{x_2}$$

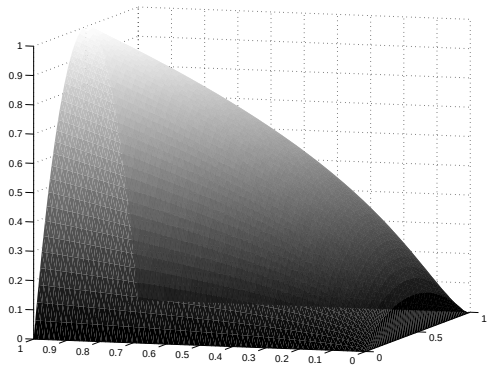

Abbildung 5.6: Die Funktion $\mathbf{u}(x_1, x_2) = 4 \cdot x_1 \cdot (1 - x_1) \cdot \sqrt{x_2}$

die starke Lösung ist, siehe Abbildung 5.6. An der Kante $x_2 = 0$ sind die ersten beiden Ableitungen in Richtung x_2 singulär.

Zur Diskretisierung der rechten Seite können wir daher nicht wie vorher den Interpolanten von f verwenden, da für diesen eine Auswertung an Randpunkten notwendig wäre. Um dennoch Aussagen über das Konvergenzverhalten machen zu können, integrieren wir die rechte Seite durch Ausnutzung der Produktstruktur mittels eindimensionaler stückweiser Gaussquadratur (vergleiche mit ähnlichen Methoden in Abschnitt 5.2.1). Für die adaptive Diskretisierung gehen wir wie folgt vor.

1. Wir lösen (5.7) auf einem groben *regulären* dünnen Gitter.

2. Wir verfeinern das Gitter mithilfe des in Abschnitt 2.2.1 auf Seite 16 eingeführten Fehlerindikators zu fest gegebenem $\epsilon > 0$. Je nach Messung gewichten wir bei der Fehlerschätzung die Koeffizienten $u_{\mathbf{l},\mathbf{i}}$ mit der Maximumsnorm $\|\psi_{\mathbf{l},\mathbf{i}}\|_{L_\infty}$ oder der L_2-Norm $\|\psi_{\mathbf{l},\mathbf{i}}\|_{L_2}$.

3. Falls neue Punkte eingefügt wurden, lösen wir (5.7) erneut auf dem verfeinerten Gitter und machen anschliessend weiter bei Schritt 2.

In Abbildung 5.7 sind die Messungen für die verschiedenen Diskretisierung dargestellt. Zu sehen ist die Zahl der Unbekannten gegen den Fehler in L_2- beziehungsweise L_∞-Norm für reguläre dünne Gitter, adaptive dünne Gitter und volle Gitter. Erwartungsgemäß sind im Fall voller Gitter ebenso wie bei Verwendung regulärer dünner Gitter enorm viele Unbekannte notwendig, um den Diskretisierungsfehler zu reduzieren. Da die Lösung der Gleichung (5.7) jedoch bis auf die Kante $x_2 = 0$ glatt ist, erzielt die adaptive Diskretisierung ein weit günstigeres Kosten/Nutzen-Verhältnis. Es entspricht demselben Kosten/Nutzen-Verhältnis wie die Diskretisierung mit regulären dünnen Gittern für das oben betrachtete glatte Modellproblem (5.5). Um dies zu beurteilen betrachten wir die Fehlerordnung α_M,

$$\|e_n^d\|_M = O(N^{-\alpha_M}),$$

in den Normen $M \in \{L_\infty, L_2\}$ jeweils für die in Abbildung 5.7 dargestellte adaptive Diskretisierung und die bereits zu Anfang dieses Abschnitts untersuchten Resultate für

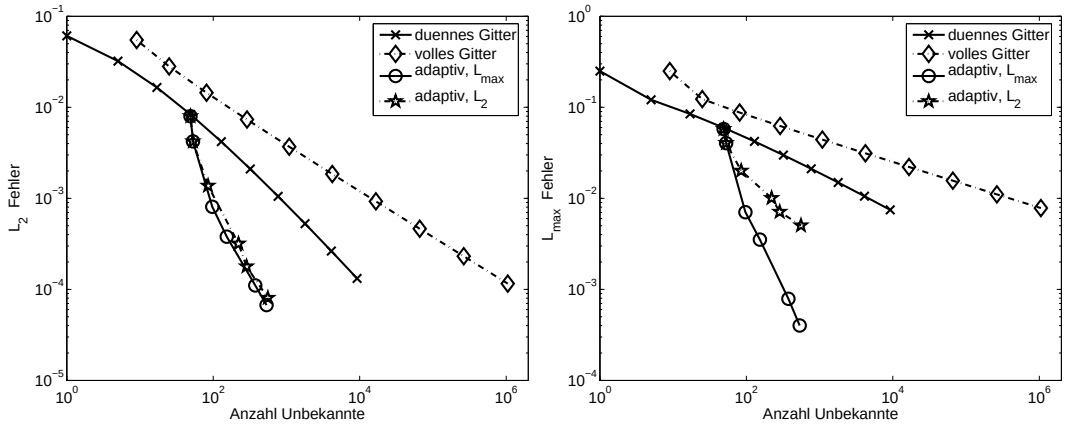

Abbildung 5.7: Dargestellt ist der Diskretisierungsfehler von (5.7) gegen die Zahl der Unbekannten. Die linke Abbildung zeigt den L_2-Fehler und die rechte den L_∞-Fehler, jeweils für die Diskretisierung mit vollen, regulären dünnen und adaptiven Gittern.

(5.5) in Abbildung 5.3. Zur Generierung des adaptiven Gitters verwenden wir dabei einmal den auf der Maximumsnorm basierenden Fehlerschätzer aus Abschnitt (2.2.1) und einmal den L_2-Norm basierten Schätzer. Damit messen wir die folgenden Raten.

	α_{L_2}	α_{L_∞}
Das glatte Problem (5.5)	1.53	1.43
Problem (5.7) mit L_∞-Fehlerschätzer	1.44	1.68
Problem (5.7) mit L_2-Fehlerschätzer	1.54	0.7

Es zeigt sich, daß bei Verwendung des passenden Fehlerschätzers die Raten der adaptiven Diskretisierung für (5.7) mindestens so gut sind wie die reguläre Diskretisierung für (5.5). Die Raten der auf dem L_∞-basierten adaptiven Diskretisierung zeigen zudem auch in L_2-Norm gemessen gute Resultate, während der L_2-basierte Fehlerschätzer nur den L_2-Fehler gut reduziert und in Bezug auf den L_∞-Fehler lediglich Ordnung 0.7 aufweist.

Damit entspricht die adaptive Diskretisierungsordnung für die singuläre Lösung (5.7) der regulären Diskretisierungsordnung für die glatte Problemstellung (5.5), wenn für die Gittergenerierung dieselbe Norm wie zur Messung der Ordnung verwendet wird. Diese Diskussion lässt sich analog für höherdimensionale Problemstellungen führen, ist aber nicht Thema dieser Arbeit.

5.2.3 Konvergenzanalyse für $d \to \infty$

Nachdem wir im letzten Unterabschnitt die Vermutung

$$\|\mathbf{u} - u\|_M \leq c\|\mathbf{u} - I_{G_n^d}\mathbf{u}\|_M \tag{5.8}$$

für feste Dimension d und die Normen $M \in \{L_2, L_\infty\}$ experimentell gestützt haben, wenden wir uns nun dem Einfluss der Dimension auf den Fehler zu. Interessant ist dabei, wie groß der Fehler absolut in Abhängigkeit der Dimension werden kann und durch welche Einflüsse diese Abhängigkeiten bestimmt sind. Um dies besser zu verstehen, betrachten wir

die dimensionsabhängigen Koeffizienten in den Interpolationsfehlerabschätzungen (2.10) und (2.9) aus Kapitel 2. Neben dem Interpolationsfehler spielt auch die Dimensionsabhängigkeit des Koeffizienten $c = c(d)$ in (5.8) für die betrachteten Normen eine entscheidende Rolle. Beide Aspekte werden wir in diesem Abschnitt zum besseren Verständnis hochdimensionaler Dünngitterapproximation beleuchten.

Für das Fehlerverhalten in der L_2-Norm können wir Aussagen mithilfe der vom Problem induzierten Energienorm treffen. Nach dem Lemma von Céa ist die Lösung des Galerkinverfahrens die Bestapproximation bezüglich der Energienorm. Damit gilt wegen der Interpolationsfehlerabschätzung (2.11) für Funktionen mit verschwindenden Randwerten

$$\|\mathbf{u} - u\|_a \leq \frac{d \cdot |\mathbf{u}|_{\mathbf{2},\infty}}{2 \cdot 3^{(d-1)/2} \cdot 4^{d-1}} \cdot 2^{-n}.$$

Mit der Poincaré-Ungleichung (siehe [38]) gilt dies auch mit einer dimensionsunabhängigen[4] Konstanten $\tilde{c}$ für die L_2-Norm des Fehlers, also

$$\|\mathbf{u} - u\|_{L_2} \leq \tilde{c} \cdot \frac{d \cdot |\mathbf{u}|_{\mathbf{2},\infty}}{2 \cdot 3^{(d-1)/2} \cdot 4^{d-1}} \cdot 2^{-n}.$$

Diese lässt zum einen darauf schließen, daß der L_2-Fehler nicht schlechter als der Fehler in der Energienorm sein wird. Allerdings kann $|\mathbf{u}|_{\mathbf{2},\infty}$ exponentiell von der Dimension abhängen. Dies lässt vermuten, daß auch der Koeffizient $c(d)$ in der schärferen Abschätzung (5.8) für $M = L_2$ nicht exponentiell von der Dimension abhängt.

Bevor wir konkrete Messungen zur Untersuchung von $c(d)$ für die $M \in \{L_2, L_\infty\}$ durchführen, betrachten wir den Einfluss der d-abhängigen Koeffizienten in den Interpolationsfehlerabschätzungen der L_2-Norm,

$$\|\mathbf{u} - I_{G_n^d}\mathbf{u}\|_{L_2} \leq \frac{2 \cdot |\mathbf{u}|_{\mathbf{2},2}}{12^d} \cdot 2^{-2n} \cdot A(d,n)$$

(wiedergegeben nach (2.10)). Numerische Experimente zeigen, dass der Koeffizient $A(\cdot,\cdot)$ die Abschätzung sehr unscharf macht. Eine Verschärfung der Abschätzung auf

$$\|\mathbf{u} - I_{G_n^d}\mathbf{u}\|_{L_2} \leq \frac{|\mathbf{u}|_{\mathbf{2},2}}{12^d} \cdot 2^{-2n} \cdot B(d,n)$$

mit der Hilfsgröße

$$B(d,n) := \sum_{i=0}^{\infty} 2^{-2i} \cdot \binom{n+i+d-1}{d-1}$$

aus [11, S. 29] resultierte bei Beispielen in Vorhersagen, die den Messwerten des Interpolationsfehlers gut entsprechen. Da die dimensionsabhängigen Koeffizienten nicht leicht zusammengefasst werden können, sind die Größen

$$S(d,n) := \frac{B(d,n)}{12^d \cdot 2^{2n}} \tag{5.9}$$

[4] Die Konstante hängt vom Durchmesser des Gebiets ab, im Fall des Einheitsquadrats ist sie unabhängig von der Dimension (vgl. [38, Seite 120]).

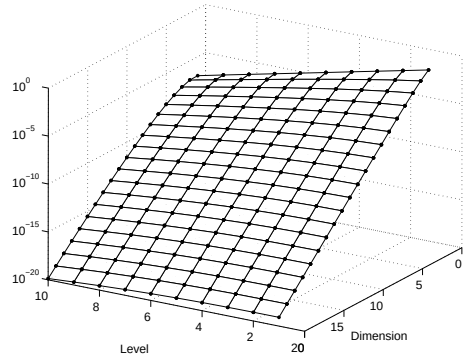

Abbildung 5.8: Darstellung der Koeffizienten (5.9) der Interpolationsfehlerschranke in Abhängigkeit von Dimension und Level. Die x_3-Achse zeigt die Werte $S(d, n)$ in logarithmischer Skala.

wiedergegeben. Abbildung 5.8 zeigt die Werte für $d = 1, \ldots, 20$ und $n = 1, \ldots, 10$ in logarithmischer Skala. Wie bereits im letzten Unterabschnitt betrachtet, sinkt der Koeffizient (und damit der Fehler) in Bezug auf n mit der Ordnung $O(2^{-2n} n^{d-1})$. Jedoch beobachten wir auch in Bezug auf die Dimension einen exponentiellen Abfall. Ob ein tendenziell hoher absoluter Fehler in Abhängigkeit der Dimension erwartet werden kann oder nicht hängt also entscheidend von der Seminorm der gemischten zweiten Ableitung von $\mathbf{u}$ ab, genauer: dem Verhältnis

$$\frac{|\mathbf{u}|_{\mathbf{2},2}}{12^d} \cdot B(d, n).$$

Daher ist für Funktionen, deren Seminorm $|\cdot|_{\mathbf{2},2}$ exponentiell mit der Dimension ansteigt mit Schwierigkeiten bei der Dünngitterinterpolation (und damit auch bei der Galerkindiskretisierung) zu rechnen – selbst, wenn die Norm der Funktion, $\|\mathbf{u}\|_{L_2}$, mit der Dimension klein wird. Dieselbe Überlegung lässt sich analog mit der L_∞ Interpolationsfehlerschranke durchführen. Hier ist anstelle von 12^d im Nenner der Wert 8^d zu verwenden, siehe Anfang dieses Kapitels.

Basierend auf diesen Vorüberlegungen untersuchen wir im Folgenden zwei Modellprobleme. In einer ersten Untersuchung betrachten wir erneut das Modellproblem (5.5) mit Lösung

$$\mathbf{u}(x) = \prod_{m=1}^{d} \sin \pi x_i$$

in Bezug auf den L_2- und L_∞ Diskretisierungsfehler im Vergleich zum Interpolationsfehler. Die Seminormen werden gemäß

$$|\mathbf{u}|_{\mathbf{2},2} = \left(\frac{\pi^2}{\sqrt{2}}\right)^d, \quad |\mathbf{u}|_{\mathbf{2},\infty} = \pi^{2d}$$

immer größer, während die Norm der Funktion, $\|\mathbf{u}\|_{L_2} = \sqrt{2^{-d}}$, immer kleiner wird. Algorithmisch verwenden wir das im letzten Abschnitt angesprochene, auf der Produktstruktur

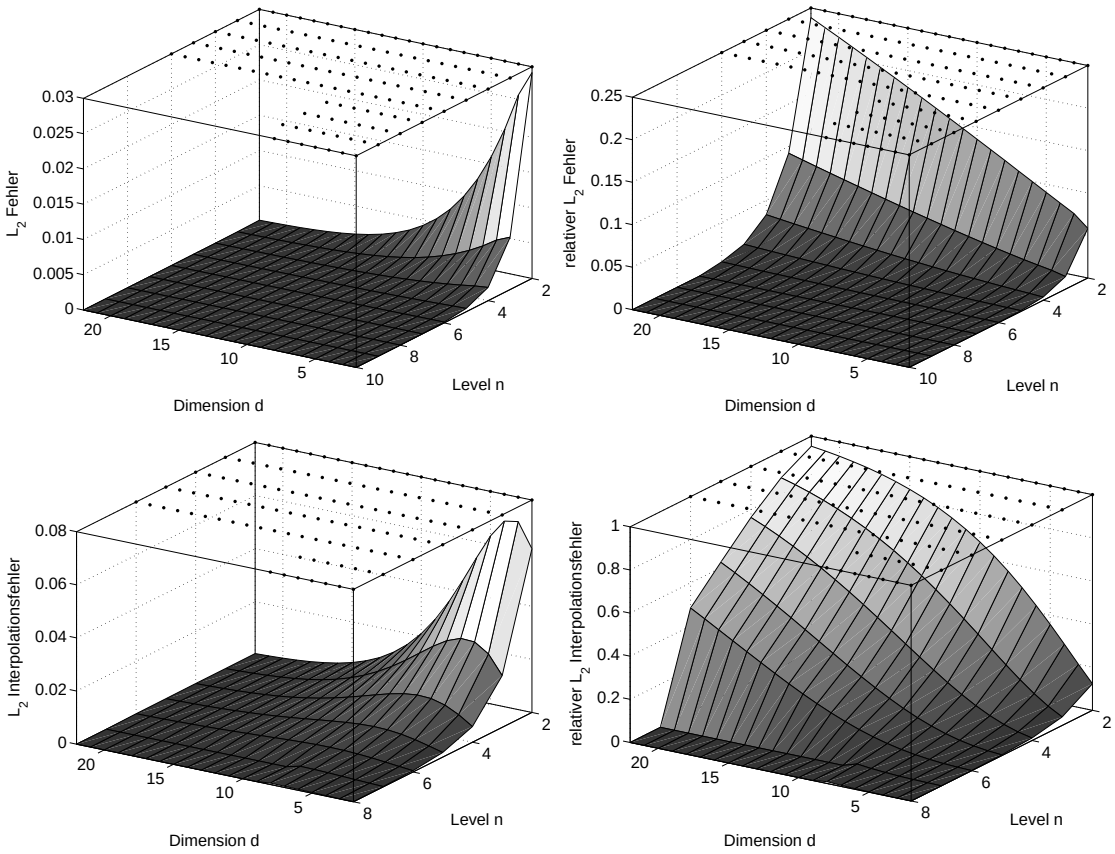

Abbildung 5.9: Messergebnisse zu Modellproblem (5.5) für den Diskretisierungsfehler in L_2-Norm (oben) und den Interpolationsfehler in L_2-Norm (unten). Linkerhand ist der absolute Fehler dargestellt, rechterhand der relative. Die x_1-Achse bezeichnet die Dimension, die x_2-Achse das Gitterlevel n sowie auf die x_3-Achse den Fehler in L_2-Norm.

basierende Verfahren zur Integration der rechten Seite. Damit ist es nicht notwendig, Randpunkte zu behandeln. Abbildung 5.9 stellt den Diskretisierungsfehler (oben) gemessen in der L_2-Norm dem Interpolationsfehler (unten) gegenüber. Linkerhand sind absolute Fehler abgebildet, rechterhand der Fehler relativ zu $\|\mathbf{u}\|_{L_2}$. Auf der x-Achse ist die Dimension d eingetragen, auf der y-Achse das Level und dazu jeweils die entsprechenden Messwerte des Fehlers. Die am oberen Rand eingezeichneten Punkte markieren die tatsächlichen Messungen, alle anderen werden als 0 angenommen. Das Verhalten für konstante Dimension und steigendes Level entspricht dem bereits in Abbildung 5.3 auf Seite 78 dargestelltem Verhalten. In Bezug auf die Dimension beobachten wir einen starken Abfall des absoluten L_2-Fehlers. Der Diskretisierungsfehler zeigt dabei qualitativ dasselbe Verhalten wie der Interpolationsfehler, was die Annahme bestätigt, dass der Diskretisierungsfehler abschätzbar ist durch den Interpolationsfehler, also

$$\|e_n^d\|_{L_2} \leq c\|\mathbf{u} - I_{G_n^d}\mathbf{u}\|_{L_2}$$

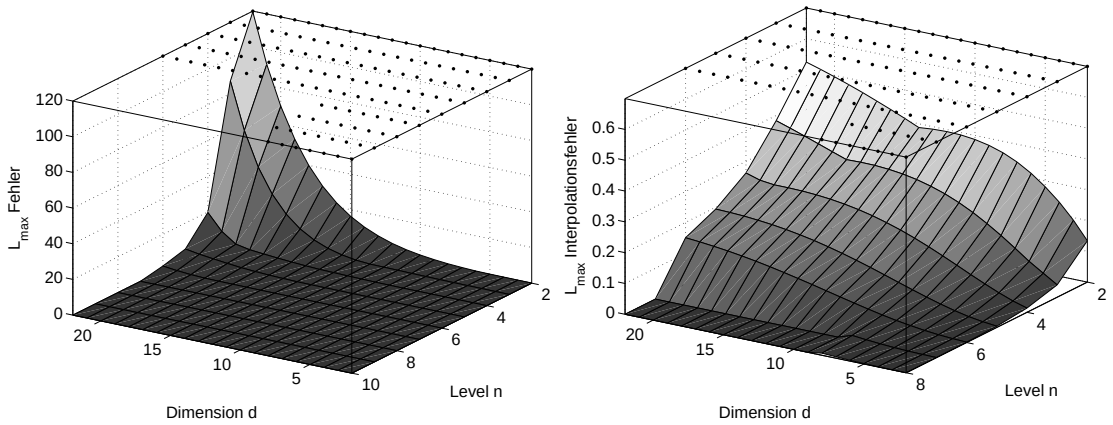

Abbildung 5.10: Messergebnisse zu Modellproblem (5.5) für den Diskretisierungsfehler in L_∞-Norm (links) und den Interpolationsfehler in L_∞-Norm (rechts).

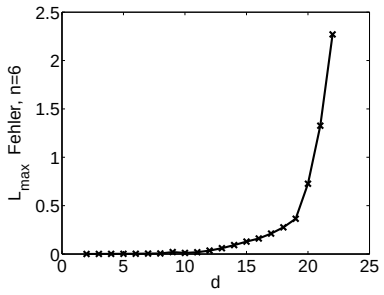

Abbildung 5.11: Der Diskretisierungsfehler von Modellproblem (5.5) bei konstantem Level $n = 6$. Auf der x_1-Achse ist die Dimension aufgetragen, auf der x_2 Achse der L_∞-Fehler.

mit einer dimensions- und levelunabhängigen Konstanten. Da die L_2-Norm mit der Dimension gegen 0 konvergiert ist der relative Fehler jedoch hoch. Dies ist zurückzuführen darauf, daß die Seminorm nicht wesentlich kleiner ist als die restlichen dimensionsabhängigen Koeffizienten: $|\mathbf{u}|_{\mathbf{2},2} < S(d, n)$.

Wir wenden uns nun dem Verhalten des Diskretisierungsfehlers von Modellproblem (5.5) in der Maximumsnorm zu. Abbildung 5.10 zeigt oben wiederum den Diskretisierungsfehler in L_∞-Norm und unten den Interpolationsfehler. An dieser Stelle werden Unterschiede zwischen Interpolation und Galerkinapproximation sichtbar: während der Interpolant ebenso wie die Funktion $\mathbf{u}$ eine L_∞-Norm von 1 hat und so auch bei steigender Dimension einen geringen Fehler aufweist, steigt der Fehler der Galerkindiskretisierung mit der Dimension stark an. Während – wie im letzten Abschnitt untersucht – der Diskretisierungsfehler bei fester Dimension in Bezug auf $n \to \infty$ wie der Interpolationsfehler konvergiert, ist dies für festes n und variierender Dimension aufgrund der Messung fraglich. [5] In unserem Fall beobachten wir einen exponentiellen Anstieg des Fehlers mit der Dimen-

[5] An dieser Stelle ist zu bemerken, daß die L_∞-Fehlerschranke des Interpolationsfehlers noch deutlich stärker mit der Dimension ansteigt als der in Abbildung 5.10 dargestellte Diskretisierungsfehler in L_∞-

sion, wobei aufgrund von $\|\mathbf{u}\|_{L_\infty} = 1$ der absolute- und relative Fehler übereinstimmen. Dazu greifen wir exemplarisch die Werte für das Level $n = 6$ heraus, siehe Abbildung 5.11. Für $d = 2, \ldots, 22$ erhalten wir genähert

$$\|e_n^d\|_{L_\infty} \approx \frac{3}{2}\|e_n^{d-1}\|_{L_\infty},$$

was sich in der Abbildung widerspiegelt. Eine genaue Betrachtung des Messverfahrens ergab, daß der Fehler an den Gitterpunkten maximal ist. Dies lässt sich analog für die anderen Werte für n nachmessen. Wir vermuten daher, daß der Diskretisierungsfehler sich mit einem exponentiell dimensionsabhängigen Koeffizienten durch den Interpolationsfehler abschätzen lässt, also

$$\|e_n^d\|_{L_\infty} \leq c_e^d \|\mathbf{u} - I_{G_n^d}\mathbf{u}\|_{L_\infty}$$

mit einer Konstanten $c_e > 1$.

Um die Untersuchung über das hochdimensionale Verhalten der Dünngitterdiskretisierung zu stützen betrachten wir ein weiteres Modellproblem. Dabei verwenden wir eine Funktion, bei der der Einfluss der Variablen x_m mit m abfällt. Dies erfüllt das Modellproblem

$$-\Delta\mathbf{u} = f \tag{5.10}$$

mit derart gewählten Randbedingungen und rechter Seite, daß

$$\mathbf{u}(x) := \prod_{m=1}^{d} \left(1 - 4 \cdot x_m \cdot (1 - x_m)\frac{1}{m}\right) \tag{5.11}$$

die Lösung der Differentialgleichung ist. Damit gilt

$$D^{\mathbf{2}}\mathbf{u}(x) = \frac{8^d}{d!}.$$

Die Funktion (5.11) ist für $d = 2$ in Abbildung 5.12 dargestellt. Für diese Funktion ist die Messung des L_2-Fehlers gemäß den Überlegungen in Abschnitt 5.2.1 nur sehr ungenau oder aber mit exponentieller Zeit in der Dimension möglich; aus diesem Grund beschränken wir uns auf die Analyse des L_∞-Fehlers. Abbildung 5.13 zeigt oben die Messung des Diskretisierungsfehlers in der L_∞-Norm und unten die Messung des Interpolationsfehlers. Wie wir sehen ändert sich der Interpolationsfehler wie erwartet nur nur sehr gering mit steigender Dimension. Demgegenüber steigt der Diskretisierungsfehler mit der Dimension stark an. Analog zu der obigen Untersuchung lässt sich auch hier ein Zusammenhang

$$\|e_n^d\|_{L_\infty} \approx c^d \|e_n^{d-1}\|_{L_\infty}$$

mit einer Konstanten $c > 1$ messen. Dies bestärkt die Vermutung, daß für den Fehler der Galerkindiskretisierung und dem Interpolationsfehler ein Zusammenhang der Art

$$\|e_n^d\|_{L_\infty} = O(c^d \|\mathbf{u} - I_{G_n^d}\mathbf{u}\|_{L_\infty})$$

gilt.

Norm.

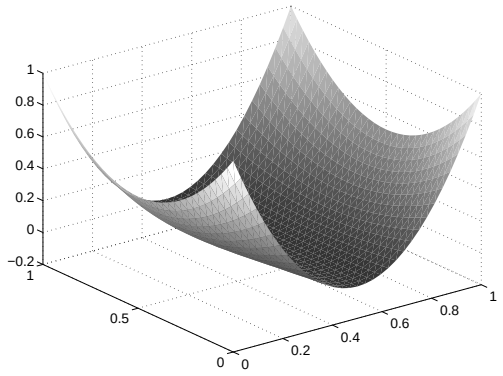

Abbildung 5.12: Die Funktion (5.11) in $d = 2$.

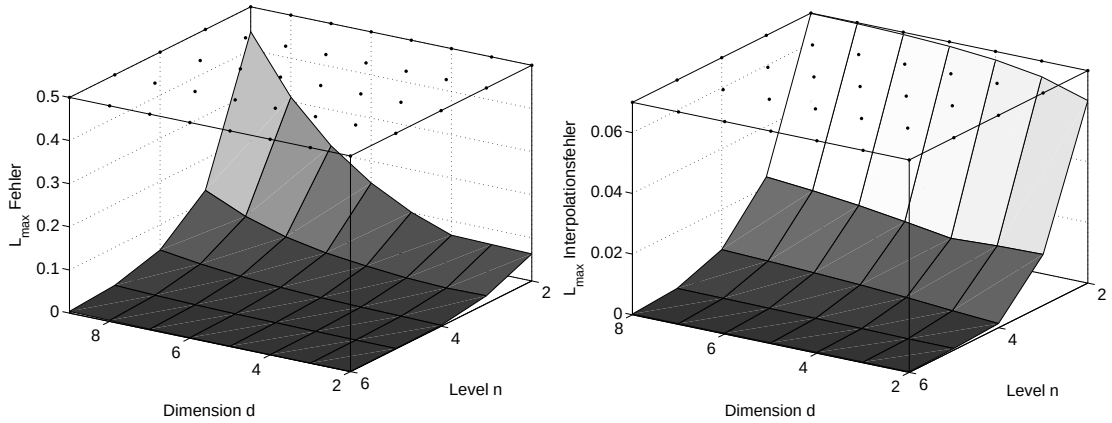

Abbildung 5.13: Messergebnisse zu Modellproblem (5.10) für den Diskretisierungsfehler in L_∞-Norm (links) und den Interpolationsfehler in L_∞-Norm (rechts).

5.3 Ein Anwendungsbeispiel

In diesem Abschnitt betrachten wir ein Anwendungsbeispiel. Viele mechanische Systeme unterliegen Umwelteinflüssen, die beim Entwurf und der Konstruktion zu beachten sind. Beispiele sind Wind - und Wellenbelastung bei Hochseebauten oder der Einfluß von elektromagnetischen Feldern beziehungsweise Rauschen auf geladene Partikel. Ein weiteres Beispiel sind zufällig schwingende Systeme, die von der Vibration von Satelliten aufgrund der Bewegung der eingebauten Technik bis hin zu seismischer Anregung von Gebäuden reichen. Oftmals können derartige Einflüsse als hochdimensionale Zufallsprozesse modelliert werden, bei denen statistische Eigenschaften Aussagen über das konkrete Ingenieurssystem erlauben, siehe [26, 29, 39, 42]. Haben diese Zufallsprozesse einen Markovcharakter, so kann die zeitliche Entwicklung anhand einer Übergangswahrscheinlichkeitsdichte erfaßt werden. Dies erlaubt die Lösung des Problems mithilfe der Fokker-Planck-Gleichung (auch Vorwärts-Kolmogorovgleichung genannt), einer parabolischen partiellen Differenti-

algleichung, die in der Regel nur numerisch gelöst werden kann. Eine klassische Diskretisierungstechnik dafür ist die Linienmethode (siehe [25]), bei der zu jedem diskreten Zeitpunkt eine elliptische partielle Differentialgleichung gelöst werden muß. Charakteristisch für die dabei auftretenden Gleichungen ist die Schwierigkeit, Peaks einer hochdimensionalen Wahrscheinlichkeitsdichte aufzulösen. Dies begrenzt die Anwendung von klassischen Finite Elemente oder Finite Differenzen Techniken auf maximal drei- bis vierdimensionale Probleme. So wird beispielsweise in [39] ein klassischer Finite-Element-Löser vorgestellt, der ein derartiges vierdimensionales Problem mit 41^4 Unbekannten löst. Die Kovarianz der gesuchten Wahrscheinlichkeitsdichte ist dabei auf 1-2 Stellen genau.

Unser Dünngitterverfahren stellt aufgrund des vorteilhaften Kosten / Nutzen Verhältnisses ein Ansatz dar, der eine höhere Skalierbarkeit der Genauigkeit auch noch in höheren Dimensionen erreicht. Dies zeigen wir anhand von zwei Beispielen. Wir betrachten zunächst die fünfdimensionale Fokker-Planck-Gleichung

$$\frac{\partial}{\partial t}p(x,t) - \Delta p(x,t) + \mathbf{b}(x,t) \cdot \nabla p(x,t) + \lambda(x,t) \cdot p(x,t) = 0 \tag{5.12}$$

für $x \in \mathbb{R}^5$ und $t \in T := (0, 0.03)$. Da die Gleichung eine Wahrscheinlichkeitsdichte beschreibt, die über $\mathbb{R}^5$ das Integral 1 haben muß, ist das Verschwinden der Funktion im Unendlichen notwendig. Die Randbedingungen sind daher

$$\lim_{\|x\| \to \infty} p(x,t) = 0 \text{ für } t \in T$$

mit einer beliebigen Norm. Die Anfangsbedingung $p(x, t_0) = N(0, \Sigma(t_0))$ ist eine fünfdimensionale Normalverteilung mit zentriertem Erwartungswert $\mu = 0$ und diagonaler Kovarianzmatrix mit den Werten $\left(\frac{1}{3} \quad \frac{1}{2} \quad \frac{1}{3} \quad \frac{1}{2} \quad \frac{1}{3}\right)$. Die Konvektionskoeffizienten sind dabei

$$\begin{aligned} b_1(x,t) &= x_1 \cdot \frac{4t + t^3 + 48t^2}{(t^2-4)^2}, \\ b_2(x,t) &= b_4(x,t) = b_5(x,t) = 0, \\ b_3(x,t) &:= 2x_1 \cdot \frac{4 + t^2 + 48t}{(t^2-4)^2}; \end{aligned}$$

der Reaktionkoeffizient beträgt

$$\lambda(x,t) := \frac{-4t^2 + t + 40}{t^2 - 4} + (x_1^2 + x_3^2)12\frac{t + 3t^2 + 12}{(t^2-4)^2} + 4(x_2^2 + x_4^2) + 9x_5^2 - 3.$$

Wir diskretisieren die Randbedingungen, indem wir das Gebiet auf $\Omega := [-4, 4]^5$ beschränken und homogene Dirichletbedingungen $p(x,t) = 0$ für $x \in \partial\Omega$ verwenden. Aufgrund der Normierungsbedingung für Wahrscheinlichkeitsdichten kann das unbeschränkte Gebiet immer so diskretisiert werden, daß der Fehler vernachlässigbar klein ist (vgl. [26, 39]). Die Zeitdiskretisierung führen wir mithilfe des Crank-Nicolson-Verfahrens durch. Für jeden Zeitschritt wird dabei die Gleichung

$$A^{t+1}u^{t+1} + \frac{2}{\delta t}Mu^{t+1} = (\frac{2}{\delta t}M - A^{t+1})u^t + 2b^{t+h/2}$$

gelöst, vgl. [25]. Dabei ist A^t die auf den Zeitpunkt t fixierte Steifigkeitsmatrix zu (5.12), M die Massenmatrix und u^t die diskrete Lösung zum Zeitpunkt t. Da die rechte Seite von (5.12) verschwindet, gilt auch $b^{t+h/2} = 0$ (vgl. [25]). Für volle Gitter konvergiert das Verfahren in zweiter Ordnung in Ort- und Zeitschrittweite, daher erwarten wir – bis auf einen Logarithmus – ähnliche Resultate für dünne Gitter. Die eindimensionalen Koeffizienten vor dem konvektiven Term und der Reaktion diskretisieren wir wie in Abschnitt 3.3 angedeuted.

Zur Diskretisierung im Ort verwenden wir ansonsten die in Kapitel 3 entwickelte Algorithmik auf einem regulären dünnen Gitter des Levels $n = 11$ mit 1579007 Unbekannten. Als Zeitschrittweite verwenden wir $\delta t = 0.0005$. Wir messen den Fehler nach 60 Zeitschritten zum Endzeitpunkt $t_{\text{end}} = 0.03$. Die Lösung für (5.12) ist die Gaussche Normalverteilung

$$\tilde{p}(x,t) = N(0, \Sigma(t))$$

mit verschwindendem Erwartungswert und Kovarianzmatrix

$$\Sigma(t) = \begin{bmatrix} \frac{1}{3} & 0 & \frac{t}{6} & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 & 0 \\ \frac{t}{6} & 0 & \frac{1}{3} & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{3} \end{bmatrix}.$$

Wir messen die diskrete Kovarianzmatrix[6] zum Zeitpunkt t_{end},

$$\Sigma_h(t_{\text{end}}) = \begin{bmatrix} 0.3330 & 0 & 0.0042 & 0 & 0 \\ 0 & 0.4994 & 0 & 0 & 0 \\ 0.0042 & 0 & 0.3330 & 0 & 0 \\ 0 & 0 & 0 & 0.4994 & 0 \\ 0 & 0 & 0 & 0 & 0.3330 \end{bmatrix},$$

was einer Genauigkeit von zwei bis drei Stellen entspricht. Wie in Abschnitt 5.2.1 diskutiert, messen wir den Maximumsfehler

$$\|\tilde{p}(x, t_{\text{end}}) - p_h(x, t_{\text{end}})\|_{L_\infty} = 2.198 \cdot 10^{-3}.$$

In unserem zweiten Beispiel messen wir den Fehler der Kovarianzmatrix bei einem sechsdimensionalen Beispiel. Dazu betrachten wir

$$\frac{\partial}{\partial t} p(x,t) - \Delta p(x,t) + \tilde{\mathbf{b}}(x,t) \cdot \nabla p(x,t) + \tilde{\lambda}(x,t) \cdot p(x,t) = 0 \tag{5.13}$$

[6] Zum Errechnen der Kovarianzmatrix $\Sigma_h = (\sigma_{n,m})_{n,m=1,\ldots,d}$ und Erwartungswerte $E[x_n]$ anhand einer diskret gegebenen Wahrscheinlichkeitsdichte $p_h(x) = \sum_{(\mathbf{l},\mathbf{i}) \in G^d} p_{\mathbf{l},\mathbf{i}} \phi_{\mathbf{l},\mathbf{i}}(x)$ nutzen wir

$$\sigma_{n,m} = E[x_n x_m] - E[x_n]E[x_m]$$

sowie

$$E[a(x_n)b(x_m)] = \int_\Omega a(x_n)b(x_m)p_h(x)dx$$

aus. Dies kann aufgrund der Tensorproduktbasis durch die Verwendung von eindimensionaler stückweiser Gaussquadratur mit einer Laufzeit von $O(|G|)$ angenähert werden.

für $x \in \mathbb{R}^6$ und $t \in T := (0, 0.015)$. Die Koeffizienten sind dabei von (5.12) verallgemeinert zu

$$\begin{aligned}\tilde{\mathbf{b}}(x_1, \ldots, x_6) &:= \mathbf{b}(x_1, \ldots, x_5),\\ \tilde{\lambda}(x_1, \ldots, x_6, t) &:= \lambda(x_1, \ldots, x_5, t) + 9x_6^2 - 3.\end{aligned}$$

Die Randbedingungen sind wie bei (5.12) und werden entsprechend durch homogene Dirichletrandbedingungen auf dem Gebiet $\tilde{\Omega} := [-4, 4]^6$ approximiert. Die Anfangsbedingung ist $p(x, 0) = N(0, \Sigma(t_0))$, wobei die Kovarianzmatrix $\Sigma(t_0)$ wie im fünfdimensionalen Fall Diagonalgestalt hat. Die Werte auf der Diagonale betragen $\left(\frac{1}{3} \quad \frac{1}{2} \quad \frac{1}{3} \quad \frac{1}{2} \quad \frac{1}{3} \quad \frac{1}{3}\right)$. Die Durchführung des oben skizzierten Crank-Nicolson-Verfahrens mit $\delta_t = 0.0005$ sowie einem sechsdimensionalen regulären dünnen Gitters mit $n = 11$ und 4571137 Freiheitsgraden resultiert zum Zeitpunkt $t_{\text{end}} = 0.015$ in einer Wahrscheinlichkeitsdichte mit verschwindendem Erwartungswert und einer Kovarianzmatrix

$$\Sigma_h(t_{\text{end}}) = \begin{bmatrix} 0.3243 & 0 & 0.0023 & 0 & 0 & 0 \\ 0 & 0.4850 & 0 & 0 & 0 & 0 \\ 0.0023 & 0 & 0.3243 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.4850 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.3243 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.3243 \end{bmatrix}.$$

Verglichen mit der analytischen Lösung $\tilde{p}(x, t) = N(0, \Sigma(t))$,

$$\Sigma(t) = \begin{bmatrix} \frac{1}{3} & 0 & \frac{t}{6} & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 & 0 & 0 \\ \frac{t}{6} & 0 & \frac{1}{3} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{3} & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{3} \end{bmatrix},$$

ist also das diskrete Verfahren in Dimension $d = 6$ immer noch auf ein- bis zwei Stellen genau.

Damit ist unser Dünngitterverfahren in der Lage, auch in höheren Dimensionen feine Peaks aufzulösen. Eine deutliche Verbesserung ist bei Verwendung von adaptiver Gitterverfeinerung zu erwarten, was Gegenstand von Folgearbeiten sein wird. Dabei sind neben der hier angesprochenen Anwendung aus dem Bereich der stochastischen Modellierung wie in [26, 29, 39, 42] oder in [37] auch andere, ähnlich modellierte Anwendungen von Interesse. Dazu gehört insbesondere die Bewertung von Aktienoptionen mithilfe der Black-Scholes-Gleichung (siehe [19, 30, 35]) oder die numerische Lösung der Schrödingergleichung in der Quantenmechanik, siehe [16, 17, 24, 40].

Kapitel 6

Fazit

In dieser Arbeit war es unser Ziel, die Kosten zur Durchführung einer Galerkindiskretisierung für hochdimensionale, elliptische, partielle Differentialgleichungen tragbar zu machen. Bisher kostete die Durchführung der Kernalgorithmik in Dimension d, der Multiplikation der Steifigkeitsmatrix mit einem Koeffizientenvektor, $O(d^2 2^d N)$ Operationen bei einem Speicherverbrauch von $O(dN)$. In dieser Arbeit haben wir ein Verfahren entwickelt, das durch Ausnutzung struktureller Massenanteile mithilfe der L_2-semiorthogonalen Prewaveletbasis lediglich $O(d^2 N)$ Operationen zur Durchführung der Multiplikation benötigt; in dem häufigen Fall von Konvektions-Diffusionsproblemen ohne gemischte zweite Ableitungen sogar nur $O(dN)$. Gleichzeitig reduziert sich der für die Algorithmik benötigte Speicherverbrauch auf $O(N)$. Lokal adaptive Gitterverfeinerung ist mit Zusatzkosten in Laufzeit und Speicherverbrauch von nur einem Faktor d möglich.

Damit ist nun die Laufzeit jeder Iteration der zur Lösung des linearen Gleichungssystems verwendeten CG- bzw. BiCGStab-Verfahren wie in früheren Arbeiten linear in der Zahl der Unbekannten, aber nur polynomiell anstatt exponentiell in der Dimension beschränkt – der in Bezug auf die Algorithmik bislang vorhandene „Fluch der Dimensionen“ konnte gebrochen werden. Basierend auf numerischen Experimenten konnten wir erstmals Aussagen über die Dimensionsabhängigkeit der für die Anzahl der Iterationen ausschlaggebenden Kondition $\kappa_A(d, n)$ machen. Während theoretische Abschätzungen in [23] ein exponentiell dimensionsabhängiges Verhalten $\kappa_A(d, n) = O(c^d)$ erwarten lassen, weisen unsere numerischen Ergebnisse des prewaveletbasierten Vorkonditionierers auf einen günstigeren Zusammenhang $\kappa_A(d, n) = O(c^{\min\{d,n\}})$ hin. Daher benötigt eine Diskretisierung beispielsweise in Dimension $d = 10$ und Level $n = 4$ so viele Iterationen wie eine ähnliche Diskretisierung in Dimension $\tilde{d} = 4, n = 4$. Somit ist das Verfahren auch in hohen Dimensionen durchführbar, in denen aufgrund der hohen Zahl an Unbekannten das Level ohnehin kleiner als die Dimension ist. Die algorithmische Formulierung behält dabei denselben Grad an Allgemeinheit wie in den Vorarbeiten (vgl. [4, 9]) und erlaubt auch die in [1, 14] vorgeschlagene Erweiterung auf allgemeine Koeffizienten.

Zur Bewältigung der hohen Zahl an Unbekannten für hochdimensionale Probleme ist eine speichersparende Verwaltung notwendig. Durch eine rein logische Gitterverwaltung konnten wir eine Datenstruktur mit optimalem, dimensionsunabhängigem Speicherverbrauch und dennoch geringer Zugriffszeit von $O(d)$ entwickeln. So ist ein reguläres dünnes

Gitter mit nur $O(1)$ Speicherkosten verwaltbar; der adaptive Fall mittels Hashing-Techniken mit lediglich zwei Integerzahlen pro Gitterpunkt.

Dies ermöglicht die Lösung von inhomogenen Dirichletrandwertproblemen in Dimensionen $d = 10$ bis ca. $d = 15$, für die eine sparsame Verwaltung der vielen benötigten Punkte auf dem Gebietsrand unerläßlich ist. Obowohl die für die geringe Laufzeit entscheidende Orthogonalität der Prewavelets nicht für Randpunkte gilt, konnten wir die Laufzeit zur Diskretisierung der Randbedingungen mithilfe unserer Algorithmik ebenfalls von $O(d^2 2^d \tilde{N})$ auf $O(d^3 \tilde{N})$ reduzieren, ohne gemischte zweite Ableitungen auf $O(d^2 \tilde{N})$. Analog erreichten wir auch für die Diskretisierung der rechten Seite eine Laufzeit von $O(d\tilde{N})$ anstelle von $O(2^d \tilde{N})$. Die technische Grenze wird dabei durch die Anzahl der Punkte auf dem Rand ($\tilde{N}$) bestimmt. Bei homogenen Dirichletrandwertproblemen, deren rechte Seite zum Rand hin verschwindet oder aber Produktstruktur aufweist, ist mit unserem Verfahren auch noch in Dimensionen jenseits von $d = 100$ eine diskrete Lösung ermittelbar.

Die mit dem Dünngitter-Galerkinverfahren erzielbare Genauigkeit wurde in den Vorarbeiten für feste Dimension $d \leq 3$ anhand numerischer Experimente durch die Genauigkeit der Interpolation charakterisiert. Dies verifizierten wir experimentell in Dimensionen bis zu $d = 8$. Erstmalig konnten wir zudem durch Untersuchungen in verschiedenen Dimensionen $d = 2, \ldots, 22$ die sonst nicht erkennbaren dimensionsabhängigen Koeffizienten aufweisen. Die Untersuchung ergab, daß die Approximation in der L_2- oder Energienorm nur durch den Interpolationsfehler und nicht durch weitere dimensionsabhängige Koeffizienten bestimmt ist. In Bezug auf die Approximation in der L_∞-Norm weisen unsere Experimente jedoch darauf hin, daß der Fehler der Galerkindiskretisierung um einen exponentiell dimensionsabhängigen Faktor schlechter als der Interpolationsfehler ist.

Die genauen Hintergründe des hochdimensionalen Approximationsverhaltens, auch die Verallgemeinerung der Ergebnisse auf andere Funktionenklassen, konnten im Rahmen dieser Arbeit nicht untersucht werden und mögen Bestandteil von Folgearbeiten bilden. Offen bleibt auch, inwieweit die in dieser Arbeit aufgewiesene Dimensionsabhängigkeit der Kondition durch andere Vorkonditionierer verbessert werden kann. Denkbar ist, daß bei Ausnutzung der überwiegenden Massenanteilen der tensorproduktartigen Matrizen – beispielsweise durch vollständig L_2-orthogonale Basen oder andere Techniken mit einer Massenkondition von 1 – die exponentielle Abhängigkeit beseitigen werden kann. Hier könnten die in [3] vorgestellten Multiwavelets hilfreich sein. Eine weitere offene Frage ist, ob die Diskretisierung der Randbedingungen und der rechten Seite ohne Verwendung der teuren Randpunkte vorgenommen werden kann.

Literaturverzeichnis

[1] Achatz, S.: *Adaptive finite Dünngitter-Elemente höherer Ordnung für elliptische partielle Differentialgleichungen mit variablen Koeffizienten.* Doktorarbeit, Technische Universität München, 2003.

[2] Adams, R.A.: *Sobolev Spaces.* Academic Press, New York, 1975.

[3] Alpert, B.K.: *A Class of Bases in L^2 for the Sparse Representation of Integral Operators.* SIAM Journal on Mathematical Analysis, 24:246–262, 1993.

[4] Balder, R.: *Adaptive Verfahren für elliptische und parabolische Differentialgleichungen auf dünnen Gittern.* Doktorarbeit, Technische Universität München, 1994.

[5] Balder, R. und C. Zenger: *The d-dimensional Helmholtz equation on sparse grids.* Technische Universität München, SFB-Bericht Nr. 342/21/92 A, 1992.

[6] Barrett, R., M. Berry, T.F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine und H. Van der Vors: *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition.* SIAM, auch online unter http://netlib2.cs.utk.edu/linalg/html_templates/Templates.html, 1994.

[7] Bungartz, H.-J.: *An adaptive Poisson solver using hierarchical bases and sparse grids.* In: Beauwens, R. (Herausgeber): *Iterative Methods in Linear Algebra*, Seiten 293–310. North–Holland, Amsterdam, 1992.

[8] Bungartz, H.-J.: *Dünne Gitter und deren Anwendung bei der adaptiven Lösung der dreidimensionalen Poisson-Gleichung.* Doktorarbeit, Technische Universität München, 1992.

[9] Bungartz, H.-J.: *Finite Elements of Higher Order on Sparse Grids.* Habilitationsschrift, Technische Universität München, 1998.

[10] Bungartz, H.J.: *A Unidirectional Approach for d-Dimensional Finite Element Methods of Higher Order on Sparse Grids.* Copper Mountain Conference on Iterative Methods, Copper Mountain, CO, 8.-13. 4., 1996.

[11] Bungartz, H.J. und M. Griebel: *Sparse grids.* Acta Numerica, pp. 1-123, Cambridge University Press, 2004.

[12] Ciarlet, P.G.: *The Finite Element Method for Elliptic Problems.* Classics in Applied Mathematics. SIAM, 2002.

[13] Dirnstorfer, S.: *Numerical quadrature on sparse grids.* Diplomarbeit, Technische Universität München, 2000.

[14] Dornseifer, T.: *Diskretisierung allgemeiner elliptischer Differentialgleichungen in krummlinigen Koordinatensystemen auf dünnen Gittern.* Doktorarbeit, Technische Universität München, 1997.

[15] Frehse, J. und R. Rannacher: *Asymptotic L^∞-error estimates for linear finite element approximations of quasilinear boundary value problems.* SIAM J. Numer. Anal., 15(2):418–431, 1978.

[16] Garcke, J.: *Berechnung von Eigenwerten der stationären Schrödingergleichung mit der Kombinationstechnik.* Diplomarbeit, Universität Bonn, 1998.

[17] Garcke, J. und M. Griebel: *On the computation of the eigenproblems of hydrogen and helium in strong magnetic and electric fields with the sparse grid combination technique.* Journal of Computational Physics, 165:694–716, 2000.

[18] Gerstner, T. und M. Griebel: *Numerical integration using sparse grids.* Numer. Algorithms, 18:209–232, 1998.

[19] Gerstner, T. und M. Griebel: *Dimension-adaptive tensor-product quadrature.* Computing, 71:65–87, 2003.

[20] Griebel, M.: *Parallel multigrid methods on sparse grids.* In: *Multigrid methods III*, ISNM 98, Seiten 211–221. Birkhäuser, Basel, 1991.

[21] Griebel, M.: *Multilevelmethoden als Iterationsverfahren über Erzeugendensystemen.* Teubner Skripten zur Numerik. Teubner, Stuttgart, 1994.

[22] Griebel, M.: *Adaptive Sparse Grid Multilevel Methods for elliptic PDEs based on finite differences.* Computing, 61(2):151–179, 1998.

[23] Griebel, M. und P. Oswald: *Tensor Product Type Subspace Splitting and Multilevel Iterative Methods for Anisotropic Problems.* Adv. Comput. Math., 4:171–206, 1995.

[24] Hackbusch, W.: *The efficient computation of certain determinants arising in the treatment of Schrödinger's equations.* Computing, 67:35–56, 2001.

[25] Hanke-Bourgeois, M.: *Grundlagen der Numerischen Mathematik und des Wissenschaftlichen Rechnens.* Teubner, 2002.

[26] Johnson, E.A., S.F. Wojtkiewicz, L.A. Bergman und B.F. Spencer: *Finite element and finite difference solutions to the transient Fokker Planck Equation.* In: *Proceedings of a Workshop: Nonlinear and Stochastic Beam Dynamics in Accelerators A Challenge to Theoretical and Computational Physics*, Seiten 290–306, Lüneburg, Germany, 1997.

[27] Knapek, S.: *Approximation und Kompression mit Tensorprodukt–Multiskalen– Approximationsräumen.* Doktorarbeit, Universität Bonn, Institut für Angewandte Mathematik, 2000.

[28] Koster, F.: *Multiskalen-basierte Finite-Differenzen-Verfahren auf adaptiven dünnen Gittern.* Doktorarbeit, Rheinische Friedrich-Wilhelms-Universität Bonn, 2001.

[29] McWilliam, S., D.J. Knappett und C.H.J. Fox: *Numerical Solution of the stationary FPK Equation using Shannon Wavelets.* Journal of Sound and Vibration, 232(2):405–430, 2000.

[30] Mertens, T.: *Optionspreisbewertung mit dünnen Gittern.* Diplomarbeit, Rheinische Friedrich-Wilhelms-Universität Bonn, 2005.

[31] Niedermeier, A.: *Lösung von Differentialgleichungen durch Prewavelets.* Diplomarbeit, Technische Universität München, 1998.

[32] Petersdorff, T. und C. Schwab: *Numerical solution of parabolic equations in high dimensions.* M2AN, 38(1):93–127, 2003.

[33] Pfaffinger, A.: *Funktionale Beschreibung und Parallelisierung von Algorithmen auf dünnen Gittern.* Doktorarbeit, Technische Universität München, 1997.

[34] Press, W.H., B.P. Flannery, S.A. Teukolsky und W.T. Vetterling: *Numerical recipes in C.* Cambridge University Press, auch online unter http://www.library.cornell.edu/nr/bookcpdf.html, 1992.

[35] Reisinger, C.: *Numerische Methoden für hochdimensionale parabolische Gleichungen am Beispiel von Optionspreisaufgaben.* Doktorarbeit, Universität Heidelberg, 2004.

[36] Schiekofer, T.: *Die Methode der Finiten Differenzen auf dünnen Gittern zur Lösung elliptischer und parabolischer partieller Differentialgleichungen.* Doktorarbeit, Universität Bonn, 1998.

[37] Schwab, C. und R. Todor: *Sparse finite elements for stochastic elliptic problems: Higher order moments.* Computing, 71:43–63, 2003.

[38] Wloka, J.: *Partielle Differentialgleichungen.* Teubner, 1982.

[39] Wojtkiewicz, S.F. und L.A. Bergman: *Numerical Solution of High Dimensional Fokker - Planck - Equations.* Specialty Conference on Probabilistic Mechanics and Structural Reliability, 8, 2000.

[40] Yserentant, H.: *On the regularity of the electronic Schrödinger equation in Hilbert spaces of mixed derivatives.* Numer. Math., 2004. Derzeit im Druck.

[41] Zenger, C.: *Sparse Grids.* In: Hackbusch, W. (Herausgeber): *Parallel Algorithms for Partial Differential Equations*, Notes on Numerical Fluid Mechanics 31, Seiten 241–251. Vieweg, Braunschweig, 1991.

[42] Zorzano, M.P., H. Mais und L. Vazquez: *Numerical solution of Fokker–Planck Equations in accelerators.* Physica D, 113:379–381, 1998.

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbständig angefertigt, keine anderen als die angegebenen Hilfsmittel benutzt, sowie Zitate kenntlich gemacht habe.

Bonn, den 25. April 2005