Institut für Parallele und Verteilte Systeme

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit Nr. 162

Dünngitter-Diskretisierungen für Probleme mit variablen Koeffizienten

Constantin Schreiber

Studiengang:	Informatik
Prüfer/in:	JP Dr. Dirk Pflüger
Betreuer/in:	Dr. Stefan Zimmer
Beginn am:	21. Juli 2014
Beendet am:	20. Januar 2015
CR-Nummer:	G.1.8

Abstract

Mit Hilfe von Dünnen Gittern können Funktionen in mehreren Veränderlichen mit einer Anzahl von Freiheitsgraden, die deutlich langsamer wächst als mit „vollen“ Gittern, diskretisiert werden. In der Simulation ist es ein häufiges Problem, partielle Differentialgleichungen lösen zu müssen, die mit der Finiten-Elemente-Methode diskretisiert wurden. Hierbei liegt die Herausforderung vor allem in der Berechnung des Residuums, also hauptsächlich der Anwendung der Steifigkeitsmatrix. Existierende Verfahren lassen sich nur auf Probleme anwenden, die eine Tensorproduktdarstellung besitzen, also nicht auf Probleme mit über dem Gebiet variablen Koeffizienten. In dieser Arbeit soll der Ansatz aus der Bachelorarbeit von S. Hirschmann [Hir], welcher auf der levelbasierten Darstellung nach B. Peherstorfer und Ch. Zenger basiert, so erweitert werden, dass er auch zur Lösung solcher Fälle verwendet werden kann. Hierzu wird an einem Modellproblem mit stückweise konstanten Ansatzfunktionen ein Algorithmus demonstriert, welcher die Multiplikation mit der Steifigkeitsmatrix auch mit variablen Koeffizienten berechnen kann. Diese Lösung könnte in der Zukunft die Simulation von mehrdimensionalen Problemen erleichtern, da sie eine flexiblere Vorgehensweise erlaubt, welche auch in mehreren Dimensionen dank der Dünnen Gitter nicht zu teuer wird.

Inhaltsverzeichnis

1 Einleitung **4**

2 Mathematische Grundlagen **5**
2.1 Dünne Gitter 5
2.2 Finite Elemente Verfahren 7
2.3 Levelbasierte Verfahren 8

3 Verfahren **9**
3.1 Operatoren 9
3.2 Basis-Algorithmus 11
3.3 Variable Koeffizienten 13

4 Numerische Resultate **17**
4.1 Implementiertechnisches 17
4.2 Konstante Koeffizienten 17
4.3 Variable Koeffizienten 18

5 Ausblick **23**

Literaturverzeichnis **24**

1 Einleitung

Der in der Simulation häufig verwendete Finite-Elemente Ansatz liefert ein Gleichungssystem, welches sehr viele Freiheitsgrade haben kann. Mehrdimensionale Probleme besitzen bei vollbesetzten Gittern Gitterpunkte in der Größenordnung $O(N^d)$, wobei N die Anzahl der Gitterpunkte pro Koordinatenrichtung und d die Anzahl der Dimensionen darstellt. Dies stellt bei Problemen mit einer hohen Zahl an Dimensionen ein großes Hindernis dar. Um diese Probleme effizienter lösen zu können, werden sogenannte dünne Gitter verwendet, welche die Anzahl der Gitterpunkte auf $O(N(\log N)^{d-1})$ reduzieren. Für eine genauere Einführung in die dünnen Gitter siehe [BG04].

In dieser Arbeit soll der levelbasierte Ansatz aus [PZZB] und [Hir] auf die Lösung von Problemen mit variablen Koeffizienten erweitert werden. Es gilt hierbei, mit der Steifigkeitsmatrix zu multiplizieren um ein Residuum zu berechnen. Mit diesem wird dann auf jedem Level eine verbesserte Lösung berechnet. Für den vorgestellten Algorithmus wird als Steifigkeitsmatrix die Identität verwendet, das bedeutet das zu lösende Problem ist $u = f$ im Sinne einer L^2-Bestapproximation. Mit variablen Koeffizienten w ergibt sich $wu = f$.

Zu Beginn werden einige mathematische Grundlagen erklärt, welche für das Verständnis des Algorithmus notwendig sind. Anschließend wird der Algorithmus für konstante Koeffizienten hergeleitet. Dieser wird dann an variable Koeffizienten angepasst. Zum Abschluss werden noch numerische Resultate des Algorithmus, sowohl mit konstanten als auch mit variablen Koeffizienten vorgestellt, sowie ein Ausblick auf mögliche Erweiterungen des Verfahrens gegeben.

2 Mathematische Grundlagen

2.1 Dünne Gitter

In dieser Arbeit werden aus Gründen der Einfachheit stückweise konstante Basisfunktionen verwendet. Im 1-D sehen diese Funktionen auf Level l wie folgt aus:

$$\phi_{l,i}(x) = \begin{cases} 1 & \text{wenn } 2^{1-l}(i-1) \leq x 2^{1-l} i \\ 0 & \text{sonst} \end{cases}$$

Damit ergibt sich als Basis:

(2.1) $B_l = \{\phi_{l,i} \mid i \in \{1, \ldots, 2^{l-1}\}\}$

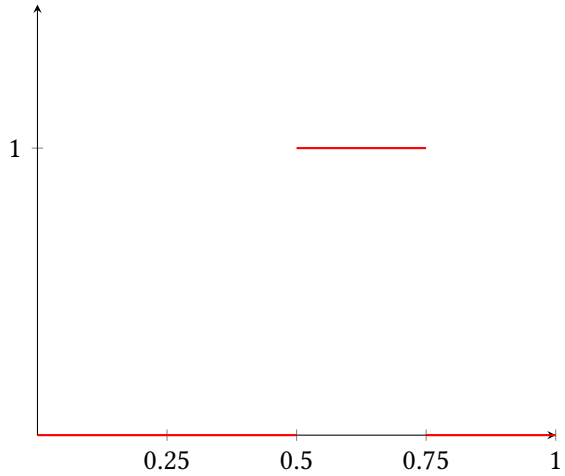

Abbildung 2.1: $\phi_{3,3}(x)$

Level l hat also 2^{l-1} verschiedene Basisfunktionen.
Die Basis im Mehrdimensionalen wird mithilfe eines Tensorproduktansatzes aus der 1-D Basis erstellt:

$$\phi_{(\underline{l},\underline{i})}(x) = \prod_k \phi_{(\underline{l}_k,\underline{i}_k)}(x_k)$$

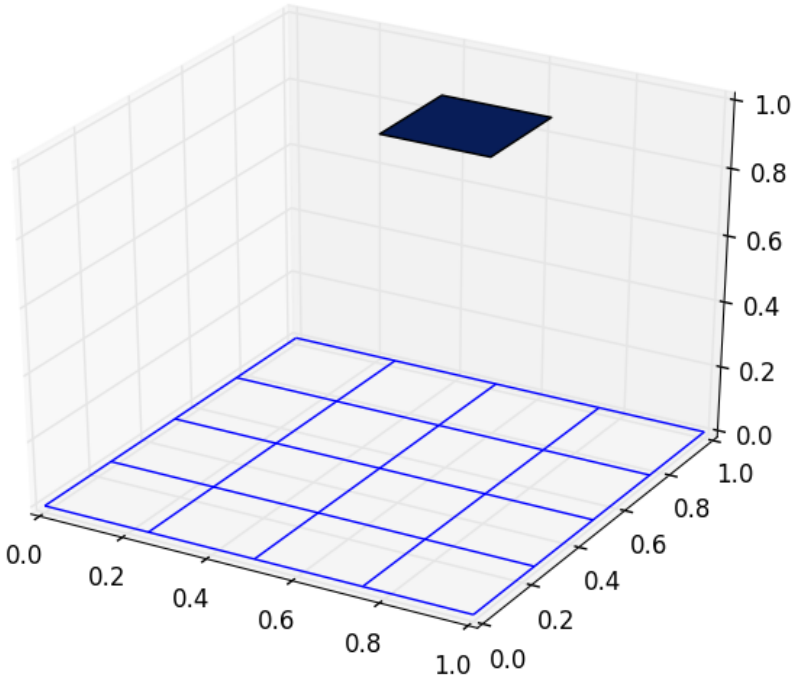

Abbildung 2.2: $\phi_{(3,3),(3,3)}(x)$

wobei $\underline{l}$ und $\underline{i}$ Multiindizes mit Anzahl der Dimensionen darstellen.
Die Basismenge definiert sich dann als:

$$B_{\underline{l}} = \{\phi_{\underline{l},\underline{i}}(x) \mid i_k \in \{1, \ldots, 2^{l_k - 1}\}\}$$

Wobei d die Anzahl der Dimensionen beschreibt. Diese sogenannte Knotenbasis spannt einen Raum $V_{\underline{l}}$=span{$B_{\underline{l}}$} auf.

Im 2-D lassen sich Gitter somit als Paar von Indizes repräsentieren, wobei der erste die Ordnung in x-Richtung angibt, der zweite die in y-Richtung. Bild 2.3 zeigt eine Anordnung solcher Gitter, wobei diese aufsteigend nach Index sortiert sind. Nur Level, die auf oder unter der Diagonalen liegen werden betrachtet, da sonst der Vorteil des geringeren Rechenaufwandes bei Dünnen Gittern verloren geht. Gitter (6,6) wäre in diesem Beispiel das volle Gitter, auf welchem das Problem exakt gelöst werden kann. Das Lösen findet also nicht in $< B_{(m,m)} >$ statt, sondern in

$$V_m^1 =< \cup_{|\underline{l}| \leq m+d-1} B_{\underline{l}} >$$

Der im Folgenden vorgestellte Algorithmus verwendet sogar nur die auf der Diagonalen liegenden Gitter, also $V_m^1 =< \cup_{|\underline{l}|=m+d-1} B_{\underline{l}} >$.

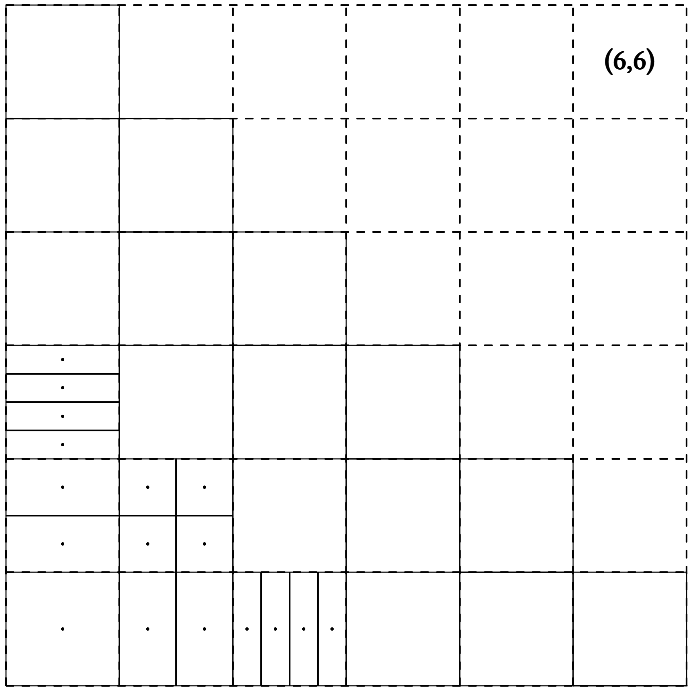

Abbildung 2.3: Anordnung der Gitter

2.2 Finite Elemente Verfahren

Um partielle Differentialgleichungen numerisch lösen zu können, wird in der Simulation häufig das Finite Elemente Verfahren verwendet. Mit Ritz-Galerkin-Ansatz und der Knotenbasis B_l ergibt sich als Anforderung im 1-D nach [Hac96]:

$$a(u, \phi_{l,i}) = b(\phi_{l,i}) \ \ \forall \phi_{l,i} \in B_l$$

Wobei a(u,v) eine Bilinearform ist. Die rechte Seite b(v) stellt für uns kein Problem dar und wird deswegen als 0 angenommen. Die Herausforderung liegt darin, die Bilinearform auszuwerten. Die Form von a hängt von der partiellen Differentialgleichung ab, für unsere Vorgaben (Identität als Operator und L^2-Bestapproximation) wählt man als Bilinearform das L^2-Skalarprodukt:

$$a(u, v) = \int u(x)v(x)\mathrm{d}x$$

beziehungsweise mit variablen Koeffizienten:

$$a(u, v) = \int u(x)w(x)v(x)\mathrm{d}x$$

Wenn man das entstehende Gleichungssystem als Matrixmultiplikation schreibt, erhält man die Matrix $A = (a(\phi_{l,j}, \phi_{l,i}))_{i,j}$, welche als Steifigkeitsmatrix bezeichnet wird. Diese Steifigkeitsmatrix enthält Blöcke von Einträgen für jedes Level. Auf diese wird später in unserem Algorithmus in Blockzeilen zugegriffen. Wir berechnen $r = Au$, was das Residuum im Fall $f = 0$ ist. Ein Großteil der vorangegangenen Einführung wurde bereits in [Hir] gegeben.

2.3 Levelbasierte Verfahren

Levelbasierte Verfahren beschreiben Algorithmen für den levelweisen Durchlauf (im Gegensatz zu baumorientierten Verfahren [BG04] [Zei11]) von Gittern, basierend auf den Verfahren aus [PZZB] und [Hir], welche für nichtkonstante Koeffizienten erweitert werden sollen. Bei diesen Verfahren muss dafür gesorgt werden, dass auf dem betrachteten Level alle Teile des Residuums bereitgestellt sind. Dies entspricht der Multiplikation mit einer Blockzeile der Steifigkeitsmatrix A. Mit diesem Residuum wird dann eine verbesserte Lösung auf dem Level berechnet.

Der hier vorgestellte Algorithmus beschränkt sich auf diejenigen Level, welche in Bild 2.3 auf der Diagonalen liegen. Also die Gitter für die gilt: $l_1 + l_2 = m + 1$ wobei m so gewählt wird, dass die gewünschte Genauigkeit bei akzeptablem Rechenaufwand erreicht wird. Der Algorithmus durchläuft nacheinander die Level in eine Diagonalrichtung, wobei auf jedem Level das Residuum und damit eine verbesserte Lösung berechnet wird. Anschließend geschieht dasselbe in die andere Diagonalrichtung.

Die Werte und Koeffizienten sind hierbei auf die einzelnen Level verteilt: $u_{\underline{l}} \in V_{\underline{l}}$ und $w_{\underline{l}} \in V_{\underline{l}}$. Diese Darstellung ist nicht eindeutig. Die Gesamtwerte und -koeffizienten können durch Summieren der Werte der einzelnen Level berechnet werden:

$$u = \sum u_{\underline{l}} \text{ und } w = \sum w_{\underline{l}}$$

Gesucht ist nun das Residuum $r = Au$, welches aus der Multiplikation der Steifigkeitsmatrix mit den Werten entsteht, A entspricht hierbei $\int u(x)v(x)w(x)$. Im folgenden werden Werte wie $u_{\underline{l}}$ als Funktionen mit einem Koeffizientenvektor bezüglich $B_{\underline{l}}$ identifiziert.

3 Verfahren

Die Beschränkung auf die Diagonale im 2-D hat den Vorteil, dass es nur zwei Richtungen ($\searrow$ und $\nwarrow$) gibt, in welche Informationen zwischen den verschiedenen Leveln bewegt werden (Anstatt vier Richtungen ohne diese Einschränkung).

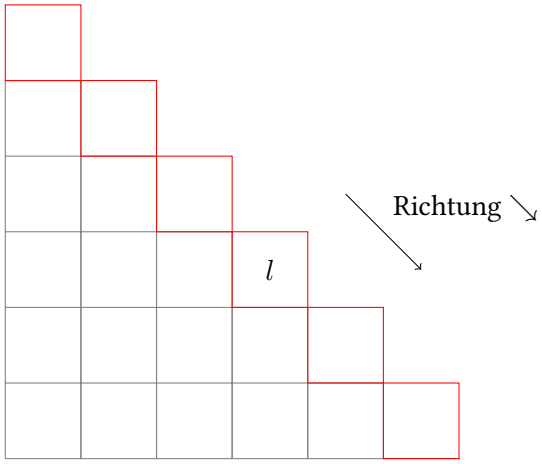

Abbildung 3.1: Verwendete Level und Richtungen

Auch wenn es hiermit möglich wäre, ein Level durch einen Index darzustellen, werden zum besseren Verständnis weiterhin Paare von Indizes für jedes Level verwendet. Im Folgenden wird zwischen den Richtungen 1,2 und $\searrow$,$\nwarrow$ unterschieden. Erstere beschreiben die Richtung der Koordinatenachsen, während Zweitere die Diagonalrichtung darstellen.

3.1 Operatoren

Für den Transport des Residuums werden einige Operatoren benötigt, welche im Folgenden vorgestellt werden. Diese Operatoren können als zeilen- beziehungsweise spaltenweise Anwendung einer Matrix, die einen eindimensionalen Operator repräsentiert, auf die Werte eines Gitters verstanden werden. Sie haben die Eigenschaft, dass Operatoren in verschiedene Richtungen (x- bzw. y-Richtung) beliebig vertauschbar sind, für Operatoren in dieselbe Richtung gilt dies nicht.

Um von einem groben Gitter l_1 auf ein um eins feineres Gitter l_2 zu kommen wird ein Prolongationsoperator verwendet. Dieser kann ohne Informationsverlust angewendet werden, denn jede Funktion

die auf einem groben Gitter dargestellt werden kann, kann auch auf dem feineren Gitter repräsentiert werden. Für unsere Wahl der Basis sieht der eindimensionale Operator wie folgt aus:

$$\text{Prolongationsoperator } P^r_{l_1+1\leftarrow l_1} \;=\; \begin{pmatrix}1\\1\end{pmatrix} \quad (r \in \{1,2\})$$

Dies meint eine "Diagonalmatrix"mit Einsen auf der Hauptdiagonalen und der unteren Nebendiagonale.

Das Gegenstück zum Prolongationsoperator, der Restriktionsoperator, dient zum Transport von feineren auf gröbere Gitter. Als Restriktionsoperator wählen wir das Transponierte des Prolongationsoperators.

$$\text{Restriktionsoperator } (P^r_{l_1\leftarrow l_1-1})^T$$

Diese Operatoren lassen sich sehr einfach auch für den Fall erweitern, dass die beiden Gitter nicht direkt nebeneinander liegen. Hierbei wird der Transport in Schritten über die dazwischenliegenden Level durchgeführt und die einzelnen Operatoren durch Multiplikation aneinandergefügt.

$$\text{Prolongationsoperator } P^r_{l_1+k\leftarrow l_1} = \prod_{i=k}^{1} P^r_{l_1+i\leftarrow l_1+(i-1)}$$

$$\text{Restriktionsoperator } (P^r_{l_1\leftarrow l_1-k})^T = \prod_{i=1}^{k} (P^r_{l_1-(i-1)\leftarrow l_1-i})^T$$

Bisher arbeiten die Operatoren jedoch nur in Richtungen 1 und 2, da wir uns jedoch auf die Gitter auf der Diagonalen beschränken, können diese beiden Operatoren nun zu einem neuen Operator, dem Transportoperator, zusammengefügt werden. Dieser übernimmt den Transport von einem Gitter auf das diagonal darunter bzw. darüber liegende.

(3.1) Transportoperator $T^{\searrow}_{(l_1+k,l_2-k)\leftarrow(l_1,l_2)} = P^1_{l_1+k\leftarrow l_1}(P^2_{l_2\leftarrow l_2-k})^T$

(3.2) Transportoperator $T^{\nwarrow}_{(l_1-k,l_2+k)\leftarrow(l_1,l_2)} = P^2_{l_2+k\leftarrow l_2}(P^1_{l_1\leftarrow l_1-k})^T$

Nun fehlt noch die Steifigkeitsmatrix, welche das zu lösende Gleichungssystem beschreibt. Diese ergibt sich mit dem L^2-Skalarprodukt und den stückweise konstanten Basisfunktionen als

$$\text{Steifigkeitsmatrix } A^r_l \;=\; \Big(1\Big) * h \;\; (r \in \{1,2\})$$

Wobei $h = 2^{1-l}$ die Breite eines Teilintervalls bezeichnet. Die Steifigkeitsmatrix ist konsistent in dem Sinne, dass Prolongation auf ein feineres Gitter, dortige Auswertung und anschließender Rücktransport dasselbe Ergebnis liefern, wie die direkte Auswertung auf dem aktuellen Gitter.

(3.3) $A_l = P^T_{k\leftarrow l} A_k P_{k\leftarrow l}$ mit k feiner als l

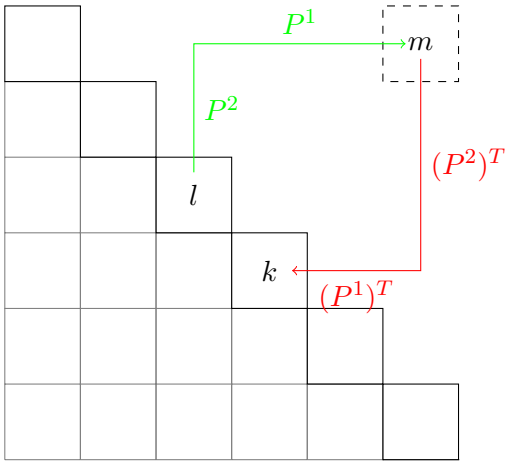

Abbildung 3.2: Grundidee mit maximalem Gitter

3.2 Basis-Algorithmus

Vorerst nehmen wir an, dass gilt: $w \equiv 1$, also dass die Koeffizienten keine Rolle im Algorithmus spielen. Die aktuelle Approximation an die Lösung des Problems, die Werte auf jedem Level l, werden als

Werte $u_l \in V_l$

bezeichnet, aus diesen kann nach Ablauf des Algorithmus das Ergebnis zusammengesetzt werden.

Angenommen, das Gitter m, welches in beide Richtungen maximale Feinheit besitzt wäre in der Menge der Levels enthalten. Dann kann der Beitrag eines Levels k zum Residuum auf Level l berechnen, indem zuerst die Werte von Level k auf Level m prolongiert werden. Auf dem maximalen Level würde dann die Steifigkeitsmatrix angewendet und die Ergebnisse auf Level l restringiert.

$$r_l^{ges} = \sum_k (P^{1,2}_{m\leftarrow l})^T A^{1,2}_m P^{1,2}_{m\leftarrow k} u_k$$

Die Operatoren in mehrere Richtungen lassen sich in die Anwendung der Operatoren in die einzelnen Richtungen nacheinander aufteilen:

$$r_l^{ges} = \sum_k (P^1_{m\leftarrow l} P^2_{m\leftarrow l})^T A^1_m A^2_m P^1_{m\leftarrow k} P^2_{m\leftarrow k} u_k$$

Für Operatoren in verschiedene Richtungen gilt das Kommutativgesetz, damit lässt sich der Term in die Anteile, die in Richtung 1 wirken, und die, die in Richtung 2 wirken, aufteilen:

$$r_l^{ges} = \sum_k (P^1_{m\leftarrow l})^T A^1_m P^1_{m\leftarrow k} (P^2_{m\leftarrow l})^T A^2_m P^2_{m\leftarrow k} u_k$$

Die Summe über alle Level kann aufgeteilt werden in den Anteil der Level, der links von Level l liegt (das heißt $k_1 < l_1$), den der rechts davon liegt, und das Level l selbst. Außerdem lassen sich die Prolongationsoperatoren mit Gleichung 3.3 in die Steifigkeitsmatrix hereinziehen.

$$r_l^{ges} = \sum_{k_1<l_1} A^1_l P^1_{l\leftarrow k} (P^2_{k\leftarrow l})^T A^2_k u_k + \sum_{k_1>l_1} (P^1_{k\leftarrow l})^T A^1_k A^2_l P^2_{l\leftarrow k} u_k + A^1 A^2 u_l$$

$$r_l^{ges} = A_l^1 \sum_{k_1<l_1} P^1_{l\leftarrow k}(P^2_{k\leftarrow l})^T A_k^2 u_k + A_l^2 \sum_{k_1>l_1} (P^1_{k\leftarrow l})^T P^2_{l\leftarrow k} A_k^1 u_k + A^1 A^2 u_l$$

Nun lassen sich noch die verbleibenden Prolongationen nach Gleichungen 3.1 und 3.2 als Transportoperatoren ausdrücken.

$$r_l^{ges} = A_l^1 \sum_{k_1<l_1} T^{\searrow}_{l\leftarrow k} A_k^2 u_k + A_l^2 \sum_{k_1>l_1} T^{\nwarrow}_{l\leftarrow k} A_k^1 u_k + A^1 A^2 u_l \tag{3.4}$$

Das bedeutet also, dass um den Beitrag eines Levels k zum Residuum auf Level l zu berechnen, zuerst die Steifigkeitsmatrix in die Richtung angewendet wird, in welcher k feiner ist als l. Da in die andere Richtung nur prolongiert wird, kann die Steifigkeitsmatrix auch erst auf Level l selbst angewendet werden.

Nun zum Algorithmus, der dieses Ergebnis liefern soll. Jedes Level erhält drei Residuums-Bestandteile, eines für jede der beiden Richtungen, sowie eines für das Level selbst. Letzteres wäre nicht nötig, vereinfacht aber die Notation, da nun nur noch mit Residuen gearbeitet wird. Diese Residuen sind dazu gedacht, dass in ihnen das aktuelle Residuum in die entsprechende Richtung (ohne Multiplikation mit der zweiten Steifigkeitsmatrix) gespeichert ist.

Residuum $r_l = u_l$

Residuum $r_l^{\searrow} = \sum_{\hat{l}_1<l_1} T^{\searrow}_{l\leftarrow\hat{l}} A^2_{\hat{l}} u_{\hat{l}}$

Residuum $r_l^{\nwarrow} = \sum_{\hat{l}_2<l_2} T^{\nwarrow}_{l\leftarrow\hat{l}} A^1_{\hat{l}} u_{\hat{l}}$

Wenn diese Werte korrekt berechnet vorhanden sind, kann man aus ihnen das Residuum berechnen, welches benötigt wird, um die Lösung auf dem Gitter zu verbessern. Dies geschieht, indem man die Residuen mit der Steifigkeitsmatrix in die Richtungen multipliziert, in welche sie noch nicht multipliziert wurden. Anschließend werden die entstandenen Werte addiert um das Gesamtresiduum zu erhalten.

Der Algorithmus durchläuft immer nacheinander alle Level, erst in $\nwarrow$-Richtung, anschließend in $\searrow$-Richtung. Hierbei wird bei jedem Übergang zwischen 2 Leveln das Residuum in Bewegungsrichtung aktualisiert. Anschließend wird mithilfe dieser Residuen das Gesamtresiduum berechnet und eine verbesserte Lösung für das Gitter erstellt. Diese wird in das Residuum des Levels geschrieben und der Algorithmus wandert zum nächsten Gitter.
Ein Schritt in Richtung $\searrow$ ($l = (l_1, l_2) \to \hat{l} = (l_1 + 1, l_2 - 1)$) sieht damit wie folgt aus:

$$r_{\hat{l}}^{\searrow} = T^{\searrow}_{\hat{l}\leftarrow l}(r_l^{\searrow} + A_l^2 r_l)$$

Um zu zeigen, dass diese Übergangsvorschrift tatsächlich das gewünschte Ergebnis hat, setzt man in die Gleichung ein:

$$r_{\hat{l}}^{\searrow} = T^{\searrow}_{\hat{l}\leftarrow l}(T^{\searrow}_{l\leftarrow k}(r_k^{\searrow} + A_k^2 r_k) + A_l^2 r_l) = T^{\searrow}_{\hat{l}\leftarrow k}(r_k^{\searrow} + A_k^2 r_k) + T^{\searrow}_{\hat{l}\leftarrow l} A_l^2 r_l$$

Weiteres Einsetzen ergibt das gewünschte Ergebnis für $r_{\hat{l}}^{\searrow}$. Aus den drei Residuen eines Levels berechnet man das Gesamtresiduum nach:

$$r_l^{ges} = A_l^1 r_l^{\searrow} + A_l^2 r_l^{\nwarrow} + A_l^1 A_l^2 r_l$$

Was Gleichung 3.4 entspricht wenn man die Residuen durch ihre Werte ersetzt.

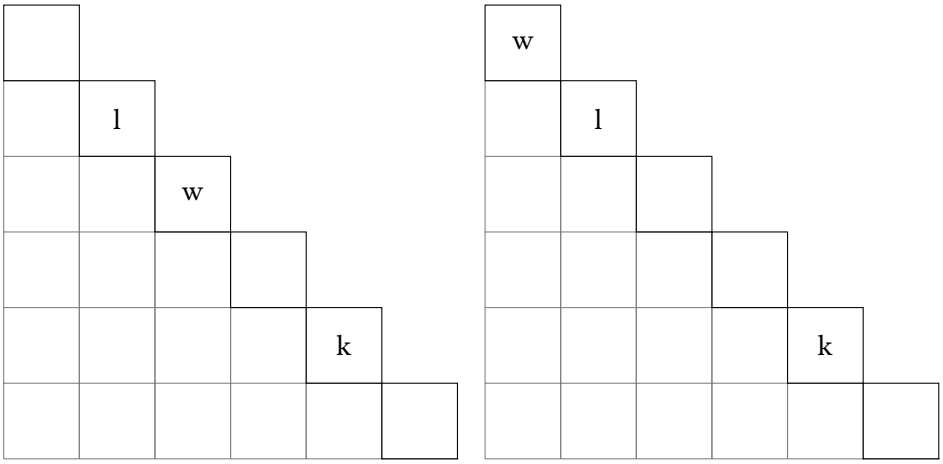

Abbildung 3.3: Verschiedene Positionen der w erfordern unterschiedliche Behandlung

3.3 Variable Koeffizienten

Der bisherige Algorithmus löst Probleme, welche keine variablen Koeffizienten besitzen. Bisherige Ansätze stoßen an Probleme, wenn man die Gleichung

$w(x)u(x) = f(x)$

lösen will. Dies ergibt den Ansatz

$$\int w(x)u(x)\phi(x)\mathrm{d}x = \int f(x)\phi(x)\mathrm{d}x \tag{3.5}$$

Hierbei besitzt jedes Level, ähnlich wie bei den Werten u_l, einen Anteil der Koeffizienten w_l.

Gesamtkoeffizienten $w = \sum_l w_l$

Multiplikation mit diesen Koeffizienten erfolgt komponentenweise, was durch eine Diagonalmatrix mit den Einträgen von w geschieht. Dies ist möglich, da die Einträge von w nicht miteinander interagieren, sondern auf separate Bereiche wirken. Die Diagonalmatrix wird durch $D(w)$ beschrieben.

Diagonalmatrix mit Einträgen von w: $D(w_{\mathrm{l}})$

Wir betrachten vier Fälle:

1. Betrachtet man den einfachsten Fall zuerst:

   l,w,k fest $\qquad \mathrm{l}_1 < \mathrm{w}_1 < \mathrm{k}_1 \qquad w = w_{\mathrm{w}}\,,\, u = u_{\mathrm{l}} = r_{\mathrm{l}}$

   Es gibt also genau ein Level w auf welchem es Koeffizienten gibt und ein Level l, auf welchem es Werte gibt. Das Residuum soll auf Level k berechnet werden. Hierbei liegt Level w zwischen k und l, wie in Abbildung 3.3 links dargestellt. Da u auf dem Weg nach Level k an Level w vorbeikommt, kann dort mit der Diagonalmatrix der w multipliziert werden. Damit ergibt sich für das Residuum auf Level k:

   Residuum $r_{\mathrm{k}}^{\mathrm{ges}} = A_{\mathrm{k}}^{1} T_{\mathrm{k}\leftarrow\mathrm{w}}^{\searrow} D(w_{\mathrm{w}}) T_{\mathrm{w}\leftarrow\mathrm{l}}^{\searrow} A_{\mathrm{l}}^{2} r_{\mathrm{l}}$

2. Wenn man nun das Szenario erweitert, so dass es mehrere Gitter gibt, welche Werte beinhalten, die jedoch immer noch von k aus gesehen "hinter" w liegen, muss einfach über alle diese Werte summiert werden.

   $$\text{Residuum } r_{\mathrm{k}}^{\mathrm{ges}} = \sum_{\mathrm{l}_1<\mathrm{w}_1} A_{\mathrm{k}}^{1} T_{\mathrm{k}\leftarrow\mathrm{w}}^{\searrow} D(w_{\mathrm{w}}) T_{\mathrm{w}\leftarrow\mathrm{l}}^{\searrow} A_{\mathrm{l}}^{2} r_{\mathrm{l}}$$

   Hier kann man jedoch, dank der Linearität, geschickt das Residuum in eine Richtung nutzen, um die Summe zu beseitigen, l ist hierbei das Level, welches w am nächsten ist:

   $$\text{Residuum } r_{\mathrm{k}}^{\mathrm{ges}} = A_{\mathrm{k}}^{1} T_{\mathrm{k}\leftarrow\mathrm{w}}^{\searrow} D(w_{\mathrm{w}}) T_{\mathrm{w}\leftarrow\mathrm{l}}^{\searrow} r_{\mathrm{l}}^{\searrow}$$

3. Gibt es nun mehrere Gitter mit Koeffizienten w_l, gilt also:

   $$\text{Gesamtkoeffizienten } w = \sum_{\mathrm{l}_1<\mathrm{w}_1<\mathrm{k}_1} P_{\mathrm{m}\leftarrow\mathrm{l}}^{1,2} w_{\mathrm{w}}$$

   So ergibt sich das Residuum auf Level k aufgrund der Linearität von w als:

   $$\text{Residuum } r_{k}^{\mathrm{ges}} = \sum_{\mathrm{l}_1<\mathrm{w}_1<\mathrm{k}_1} A_{\mathrm{k}}^{1} T_{\mathrm{k}\leftarrow\mathrm{w}}^{\searrow} D(w_{\mathrm{w}}) T_{\mathrm{w}\leftarrow\mathrm{l}}^{\searrow} r_{\mathrm{l}}^{\searrow}$$

   Es wird nun eine weitere Variable benötigt, um im Algorithmus die bereits multiplizierten Anteile des Residuums zu transportieren. Diese bezeichnen wir als multipliziertes Residuum m. Es hat, wie die Residuen r für jedes Level einen Anteil in jede der Richtungen. Mit der gegebenen Konstellation von Koeffizienten ergibt sich:

   $$\text{multipliziertes Residuum } m_{\mathrm{k}}^{\searrow} = \sum_{\mathrm{l}_1<\mathrm{w}_1<\mathrm{k}_1} T_{\mathrm{k}\leftarrow\mathrm{w}}^{\searrow} D(w_{\mathrm{w}}) T_{\mathrm{w}\leftarrow\mathrm{l}}^{\searrow} r_{\mathrm{l}}^{\searrow}$$

   Dies lässt sich einfach in den Algorithmus integrieren, indem man für den Übergang in Richtung $\searrow$ $(l = (l_1, l_2) \to \hat{l} = (l_1 + 1, l_2 - 1))$ die Übergangsvorschrift

   $$\text{multipliziertes Residuum } m_{\hat{l}}^{\searrow} = T_{\hat{l}\leftarrow l}^{\searrow} (m_l^{\searrow} + D(w_l) r_l^{\searrow}) \tag{3.6}$$

   Es werden also für jedes Level die Koeffizienten multipliziert mit dem Residuum auf m addiert.

4. Dieser Algorithmus funktioniert jedoch nur für den oben bestimmten Fall, in dem die Koeffizienten zwischen Werten und Auswertungslevel liegen. Um den Algorithmus auch für allgemeine Fälle wie in Abbildung 3.3 rechts anzupassen, betrachte man Gleichung 3.5. Es lässt sich erkennen, dass das Integral symmetrisch ist in w und u. Daraus folgt, dass anstelle der Werte auch die Koeffizienten summiert werden können. Hierzu führen wir, äquivalent zum Residuum, Koeffizienten w_l^r in die beiden Richtungen ein.

   $$\text{Vorberechnung der } w_{(l_1,l_2)}^{\searrow} = \sum_{k_1<l_1} T_{(l_1,l_2)\leftarrow(k_1,D-1-k_1)}^{\searrow} A_{(k_1,D-1-k_1)}^{2} w_{(k_1,D-1-k_1)}$$

   $$\text{Vorberechnung der } w_{(l_1,l_2)}^{\nwarrow} = \sum_{k_1>l_1} T_{(l_1,l_2)\leftarrow(k_1,D-1-k_1)}^{\nwarrow} A_{(k_1,D-1-k_1)}^{1} w_{(k_1,D-1-k_1)}$$

   Der große Vorteil hierbei ist, dass sich die Koeffizienten während des Algorithmus nicht verändern. Somit können die w_l^r zu Beginn einmal für jedes Gitter berechnet werden und müssen

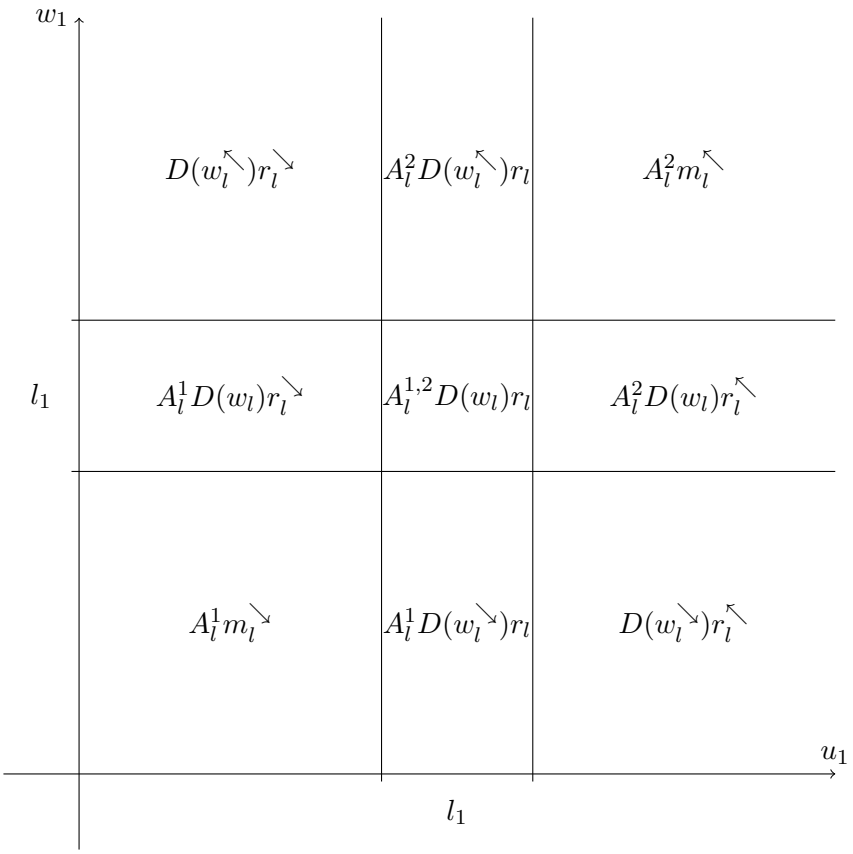

Abbildung 3.4: Zusammensetzung des Gesamtresiduums

danach nicht mehr bearbeitet werden. Mit diesen neuen Koeffizienten ändert sich dann auch der Übergang des multiplizierten Residuums, da auch der Fall bedacht werden muss, in dem das Level selbst Werte enthält.

$$m_{\hat{l}}^{\searrow} = T_{\hat{l}\leftarrow l}^{\searrow}(m_l^{\searrow} + D(w_l)(r_l^{\searrow} + r_l) + D(w_l^{\searrow})r_l)$$

Es ist klar zu sehen, dass sich für den Fall, in welchem die w und l getrennt sind, genau Gleichung 3.6 ergibt.

Um nun das Gesamtresiduum für Level l zu berechnen nachdem der Algorithmus dort angekommen ist, müssen die Hilfsvariablen richtig zusammengesetzt werden. Abbildung 3.4 zeigt die Terme, aus welchen das Residuum berechnet wird. Auf der Abszisse ist hierbei die erste Koordinate des Levels abgetragen, von welchem die Werte stammen. Auf der Ordinate findet sich die erste Koordinate des Gitters, auf welchem die Koeffizienten ihren Ursprung haben. Insgesamt gibt es neun Fälle, die betrachtet werden müssen, in jede Richtung kann die Koordinate entweder größer, gleich oder kleiner der ersten Koordinate des Levels l sein. Durch Summieren der verschiedenen Fälle ergibt sich als Formel für das gesamte Residuum auf Level l:

$$\begin{aligned} r_l^{ges} =& A_l^1 m_l^{\searrow} + D(w_l^{\nwarrow})r_l^{\searrow} + A_l^1 D(w_l)r_l^{\searrow} + A_l^2 m_l^{\nwarrow} + D(w_l^{\searrow})r_l^{\nwarrow} \\ &+ A_l^2 D(w_l)r_l^{\nwarrow} + A_l^2 A_l^1 D(w_l)r_l + A_l^2 D(w_l^{\nwarrow})r_l + A_l^1 D(w_l^{\searrow})r_l \end{aligned} \tag{3.7}$$

Der Algorithmus kann als Pesudocode folgendermaßen dargestellt werden:

```
für alle Level l:
  w_vorberechnen(l)
für i von 1 bis anzahl_iterationen:
  für l von 1 bis lmax:
    Übergang_r(l-1,l)
    Übergang_m(l-1,l)
    r = gesamtresiduum(l)
    u = u_neuberechnen(r,l)
    r_l = u
  für l von lmax bis 1:
    Übergang_r(l,l-1)
    Übergang_m(l,l-1)
    r = gesamtresiduum(l)
    u = u_neuberechnen(r,l)
    r_l = u
```

wobei die Subroutinen die entsprechenden Formeln implementieren. Die Laufzeit ist offensichtlich linear in der Anzahl der Gitterpunkte, da jedes Gitter nur einmal pro Diagonalendurchlauf besucht wird.

4 Numerische Resultate

4.1 Implementiertechnisches

Die Implementierung des vorgestellten Algorithmus wurde in Python vorgenommen. Werte und Residuen eines Gitters wurden als zweidimensionale numpy-Arrays implementiert, was die Multiplikation mit Matrizen in Zeilen- und Spaltenrichtung mithilfe der numpy-Funktion tensordot ermöglicht. Damit entfällt der Aufwand, selbst den Vektor auseinander zu bauen, den Operator anzuwenden, und die Werte wieder zusammenzufügen. Da bei der Implementierung in Python jedoch nicht auf Effizienz Wert gelegt wurde, wird hier auf eine exakte Laufzeitmessung verzichtet. Als Anhaltspunkt: für $l_1 + l_2$=12 ergibt sich für einen Durchlauf des Algorithmus mit der Gleichung $f(x, y) = sin(2\pi x)sin(2\pi y)$ mit unterschiedlichen Koeffizienten auf jedem Viertel des Berechnungsgebietes eine Laufzeit von 1,5 Sekunden. Das resultierende Bild wird im Folgenden noch gezeigt.

Für den Algorithmus ist es notwendig für ein gegebenes f die rechte Seite zu berechnen. Diese wird durch $\int \phi(x)f(x)$ beschrieben. Mit separierbarem f kann das Integral in die einzelnen Richtungen nacheinander berechnet und anschließend durch Multiplikation zusammengesetzt werden. Deshalb beschränken wir uns auf im Folgenden auf solche Funktionen, die sich als $f(x, y) = g(x) * h(y)$ ausdrücken lassen.

Die folgenden Bilder sind, soweit nicht anders angegeben alle mit einer maximalen Gittertiefe von 9 entstanden. Das bedeutet, das feinste Gitter in jede Richtung besitzt 256 Punkte. Um die verschiedenen Level zusammenzufügen wurde auf ein maximales 256x256 Gitter erweitert, wo die Werte addiert werden.

4.2 Konstante Koeffizienten

Zuerst ein paar Beispiele für den Algorithmus ohne variable Koeffizienten. Aufgrund der Verwendung der stückweise konstanten Basisfunktionen können stetige Funktionen nur dann exakt dargestellt werden, wenn sie konstant sind. Ein Beispiel hierfür zeigt Bild 4.1 mit der konstanten Funktion $f(x, y) = 1$.

Schon für lineare Funktionen treten jedoch bei der L^2-Approximation Fehler auf. Abbildung 4.2 zeigt das Resultat für die bilineare Funktion $f(x, y) = xy$, aufgrund der hohen Anzahl an Gitterpunkten lässt sich nicht direkt erkennen, dass die Funktion nicht exakt interpoliert werden kann. Dies wird allerdings sofort deutlich, wenn man nur die erste Zeile an Werten als 2-D Plot betrachtet, wie in Abbildung 4.3. Man erkennt die größeren Sprünge bei 0.25, 0.5 und 0.75, welche von den Basisfunktionen stammen, welche in x-Richtung gröber sind. Die feineren Basisfunktionen verbessern

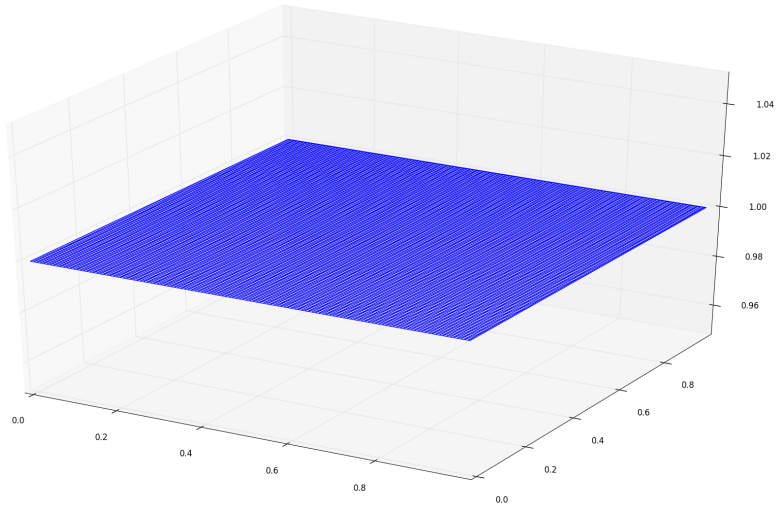

Abbildung 4.1: Konstante 1 Funktion

den Fehler, der von den groben Funktionen gemacht wird soweit, dass die Fläche unter der Kurve fast null ergibt. Mit linearen Basisfunktionen kämen diese Sprünge nicht vor, der Plot wäre einfach eine flache Linie, da die Funktion exakt durch die Basis dargestellt werden kann.

Nun noch in Bild 4.4 eine trigonometrische Funktion $f(x, y) = sin(\pi x)sin(\pi y)$, welche später mit variablen Koeffizienten verändert wird. Auch hier sind am Rand wieder die Sprünge zu erkennen.

4.3 Variable Koeffizienten

Abschließend noch ein paar Bilder zum Algorithmus mit variablem Koeffizienten, welcher in dieser Arbeit vorgestellt wurde. Ein Beispiel, an welchem sich die Koeffizienten gut erkennen lassen, ist die Sinusschwingung in beide Richtungen $f(x, y) = sin(2\pi x)sin(2\pi y)$, wenn man jeweils ein Viertel des Berechnungsgebietes mit einem Koeffizienten versieht. Abbildung 4.5 zeigt, dass die entsprechenden Amplituden des Sinus mit steigenden Koeffizienten abnehmen, wie man es erwartet. Das Viertel, in welchem der Koeffizient 0.5 beträgt, ist doppelt so hoch, während das mit 2 nur halb so hoch ist.
Da die Koeffizienten jedoch auch auf den dünnen Gittern angegeben werden müssen, was es erschwert komplexere Koeffizienten direkt anzugeben, kann man sich einen Trick zunutze machen. Zuerst wird der Algorithmus einmal ohne Koeffizienten ausgeführt, wobei die Zielfunktion f den gewünschten Koeffizienten entspricht. Anschließend wird das Ergebnis des ersten Durchlaufs als Koeffizienten w für einen zweiten Durchlauf des Algorithmus verwendet, wobei diesmal die Funktion, für welche das Problem gelöst werden soll als rechte Seite f verwendet wird. Abbildungen 4.6 und 4.7 zeigen eine Sinusschwingung $f(x, y) = sin(\pi x)sin(\pi y)$, allerdings wurden zuvor die Koeffizienten als

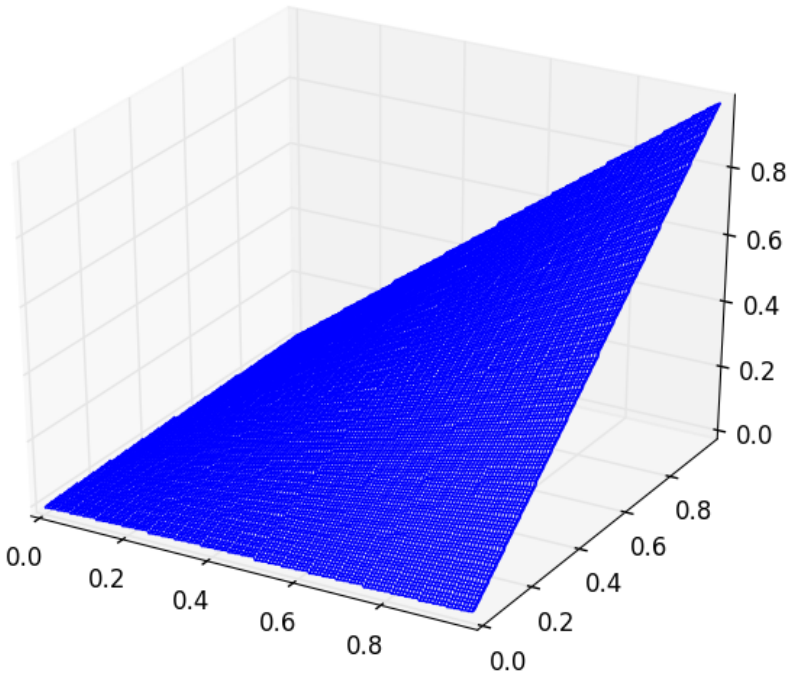

Abbildung 4.2: Lineare Funktion in beide Richtungen

$w(x, y) = (x + c)(y + c)$ berechnet (mit Addition einer Konstanten c, damit es im Bereich nahe der 0 nicht zu starken Ausbrüchen kommt). Die beiden Bilder zeigen das Ergebnis mit unterschiedlichen Werte für c.

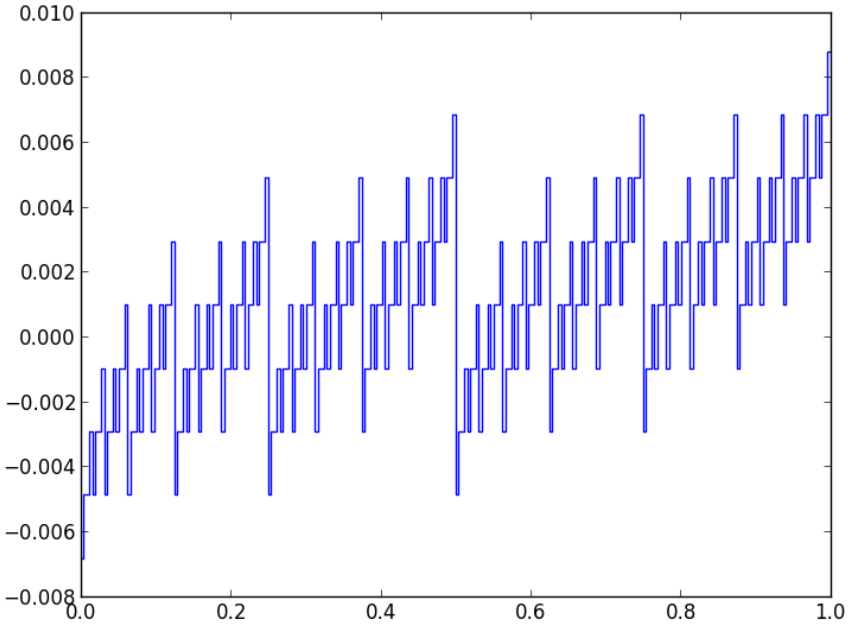

Abbildung 4.3: Erste Zeile der Werte für die Lineare Funktion in beide Richtungen

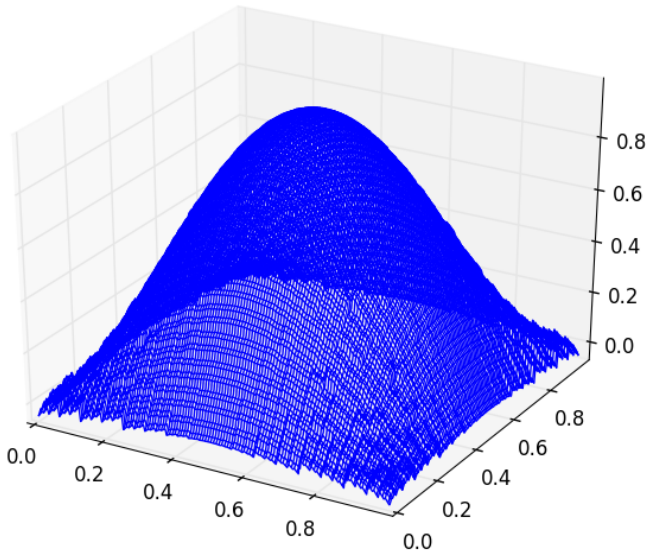

Abbildung 4.4: $f(x, y) = sin(\pi x)sin(\pi y)$

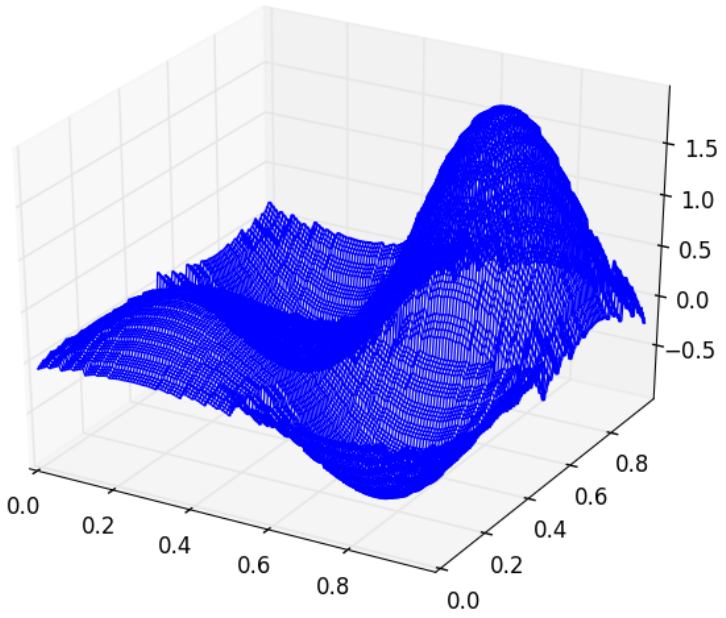

Abbildung 4.5: $f(x, y) = sin(2\pi x)sin(2\pi y)$ mit variablen Koeffizienten (0.5,1,1,2)

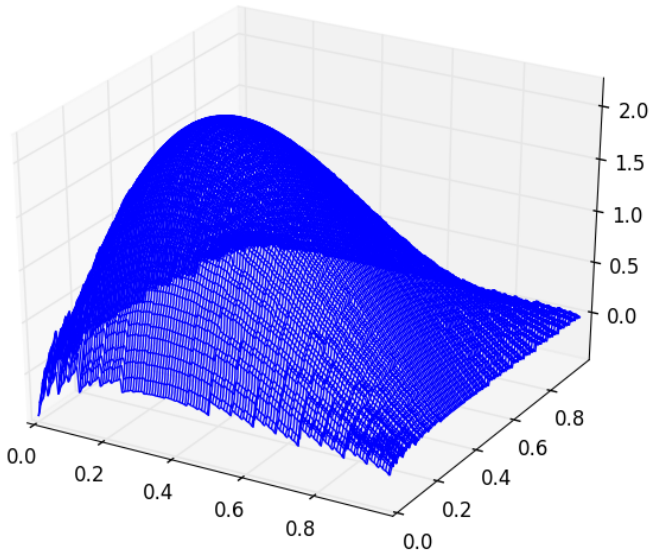

Abbildung 4.6: $f(x, y) = sin(\pi x)sin(\pi y)$ mit variablen Koeffizienten (x+0.25)*(y+0.25)

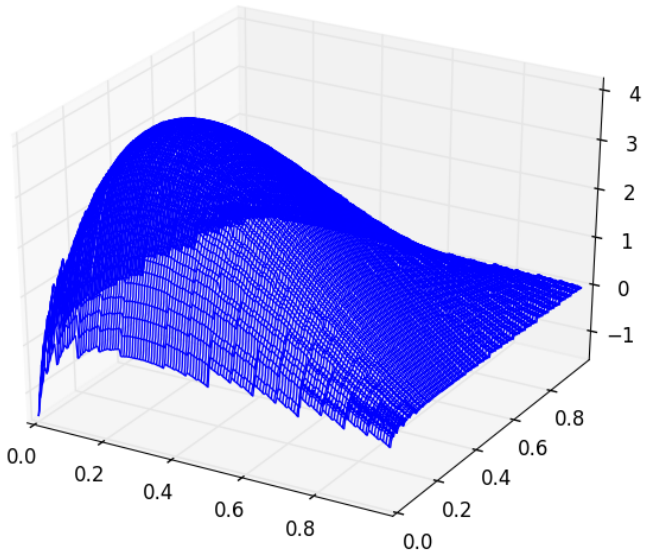

Abbildung 4.7: $f(x, y) = sin(\pi x)sin(\pi y)$ mit variablen Koeffizienten (x+0.1)*(y+0.1)

5 Ausblick

Das in dieser Arbeit vorgestellte Verfahren ermöglicht es Probleme mit variablen Koeffizienten auf dünnen Gittern zu lösen. Allerdings wurde die Einschränkung auf stückweise konstante Basisfunktionen gemacht. Eine Erweiterungsmöglichkeit wäre, den Algorithmus auch für andere Basisfunktionen zu entwickeln. Dies sollte kein großes Problem darstellen, da im Algorithmus selbst nur Operatoren verwendet werden, die aus der Basis entstanden sind. Durch Anpassen der Operatoren und der Auswertung der rechten Seite, sollte es möglich sein, den Algorithmus auf beliebige Basen zu erweitern.

Ebenso wurde in dieser Arbeit nur auf die Identität als Steifigkeitsmatrix mit einer L^2-Bestapproximation eingegangen. Hier kann untersucht werden, inwiefern sich das Verfahren auf komplexere Gleichungen als $u = f$ beziehungsweise $\int w(x)u(x) = \int f(x)$ erweitern lässt. Vermutlich ist auch dies in den Algorithmus integrierbar, ein Beweis hierfür bleibt jedoch aus.

Des Weiteren wäre eine Anwendung des Verfahrens in höherdimensionalen Problem zu untersuchen. Hier steht man allerdings vor dem Problem, dass die Einschränkung auf das zweidimensionale und dort auf die Diagonale eine Grundlage des Algorithmus bildet. Im Mehrdimensionalen ist es nicht ohne weiteres möglich, sich auf eine Diagonale zu beschränken.

Literaturverzeichnis

[BG04] H.-J. Bungartz, M. Griebel. Sparse grids. *Acta Numerica*, 13:1–123, 2004. (Zitiert auf den Seiten 4 und 8)

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

[Hir] S. Hirschmann. *Hierarchische Transformationen und der Q-Zyklus.* Bachelorarbeit, Universität Stuttgart. (Zitiert auf den Seiten 2, 4, 7 und 8)

[PZZB] B. Peherstorfer, C. Zenger, S. Zimmer, H.-J. Bungartz. A Multigrid Solver for Elliptic Partial Differential Equations on Spatially Adaptive Sparse Grids, Angenommen im SIAM Journal on Scientific Computing. (Zitiert auf den Seiten 4 und 8)

[Zei11] A. Zeiser. Fast Matrix-Vector Multiplication in the Sparse-Grid Galerkin Method. *Journal of Scientific Computing*, 47(3):328–346, 2011. doi:10.1007/s10915-010-9438-2. URL http://dx.doi.org/10.1007/s10915-010-9438-2. (Zitiert auf Seite 8)

Erklärung

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

Ort, Datum, Unterschrift