Rheinische Friedrich-Wilhelms-Universität Bonn
Mathematisch-Naturwissenschaftliche Fakultät

Diplomarbeit

Effiziente adaptive Lösung von Multilevelsystemen

Ralph Thesen

angefertigt am

Institut für Numerische Simulation

Bonn, 31. August 2010

Erklärung

Mit der Abgabe der Diplomarbeit versichere ich gemäß §20 Absatz 6 der Diplomprüfungsordnung vom 14. März 2003, dass ich die Arbeit selbstständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Bonn, den 31. August 2010

Inhaltsverzeichnis

1. Einleitung

Die Lösung von partiellen Differentialgleichungen ist ein elementarer Teil vieler wissenschaftlicher Problemstellungen, da fundamentale Naturphänomene durch diese beschrieben werden. Die Physik und das Ingenieurwesen liefern viele Aufgaben, deren Lösung auch die von partiellen Differentialgleichungen verlangt. Will man diese Aufgaben nun numerisch, also mit Hilfe von Computern, berechnen, ist es notwendig, die Aufgabenstellung mit endlich vielen Daten beschreiben zu können. Wenn dieser als Diskretisierung bezeichnete Prozess abgeschlossen ist, verbleibt es, ein System von Gleichungen zu lösen. Auf die effiziente Lösung dieser Gleichungssysteme richtet sich das Hauptaugenmerk dieser Arbeit.

Wie schnell sich die Systeme lösen lassen, ist zum einen von der Genauigkeit der Diskretisierung und zum anderen von den Parametern der Gleichung abhängig. Je genauer das Problem diskret beschrieben wird, desto länger dauert es, bis ein genügend genaues Ergebnis erreicht ist. Außerdem beeinflussen die Parameter der Differentialgleichung deren Eigenschaften, also auch den Schwierigkeitsgrad und damit die Geschwindigkeit der Lösungsverfahren. Es ist erstrebenswert, dass man von diesen beiden Effekten unabhängig wird.

Damit die Genauigkeit der Diskretisierung keinen Einfluss mehr auf die Konvergenzgeschwindigkeit der Lösungsverfahren nimmt, lassen sich Mehrgitter- oder Multilevelmethoden einsetzen. Mit Hilfe dieser Verfahren erreicht man Stabilität, also die Unabhängigkeit von der Auflösung der Diskretisierung.

Von den Parametern unabhängig zu werden, ist jedoch ungemein komplizierter. Neben verschiedenen problemabhängigen Ansätzen liegt eine Möglichkeit darin, ein Lösungsverfahren einzusetzen, welches sich adaptiv an die Problemstellung anpassen kann. Das heißt, der entsprechende Algorithmus sollte so beschaffen sein, dass er ohne zusätzliche Informationen auf die Eigenschaften des Systems und damit die Parameter der zugrunde liegenden Differentialgleichung reagieren kann. Das Ziel dabei ist es, die Abhängigkeit der Lösungsgeschwindigkeit von den Parametern zu verringern. Würde man vollständig unabhängig von diesen werden, hätte man sogenannte Robustheit erreicht.

Problemstellung

Existierende Iterationsverfahren sind im Allgemeinen nicht robust, insbesondere nicht für beliebige Parameter. Das heißt, dass die Geschwindigkeit, mit der eine Lösung erreicht werden kann, immer davon abhängt, wie die Parameter der zugrunde liegenden Differentialgleichung aussehen.

Diesem Problem wird in der Praxis derart begegnet, dass entweder speziell diskretisiert wird wie etwa in [Ape99], wo anisotrope Finite Elemente verwendet werden, das Lösungsverfahren an das Problem angepasst oder das Gleichungssystem entsprechend vorkonditioniert wird.

Diese Vorgehensweise hat den Nachteil, dass die Anpassungen stets a priori vorgenommen werden müssen. Sollten sich die Parameter der Gleichung ändern, müssen die entsprechenden

Maßnahmen erneut durchgeführt werden.

Das Problem wird umso schwieriger, wenn innerhalb des Rechengebiets unterschiedliche Werte für bestimmte Parameter der Differentialgleichung gelten. Ein Beispiel wäre hier die Strömung durch poröse Medien wie etwa bei der Simulation von Grundwasser oder Erdöl. Dabei müsste der Löser gebietsabhängig modifiziert werden, was in der Praxis häufig durch andere Modellierungstechniken wie Homogenisierung vollständig umgangen wird.

Eine andere Herangehensweise besteht darin, ein adaptives Lösungsverfahren zu verwenden, das sich an das Gleichungssystem und damit an die zugrunde liegende Differentialgleichung und deren Parameter dynamisch anpassen kann. Dafür muss es die nötigen Informationen allerdings auch erst gewinnen und verarbeiten. Daraus resultieren Kosten, die gegenüber anderen Verfahren abgewägt werden müssen.

Wir untersuchen in dieser Arbeit speziell das Gauß-Southwell-Verfahren. Dieses verwendet das Residuum, um die Durchlaufreihenfolge innerhalb des Gleichungssystems adaptiv festzulegen. Dazu ist es zum einen notwendig, stets das aktuelle Residuum zur Verfügung zu haben, und zum anderen muss darin immer der jeweils betragsgrößte Eintrag gefunden werden. Diese Kosten sind schwer zu reduzieren, was zur Folge hat, dass das Gauß-Southwell-Verfahren kaum verwendet wird.

In [Rü93] wird die Idee des Verfahrens aufgegriffen, jedoch stark modifiziert. Die sogenannte *active set*-Strategie sieht vor, nur solche Unbekannte zu behandeln, deren Residuenwerte eine Mindestgröße überschreiten. Nach einer Relaxation werden die davon betroffenen Punkte jeweils darauf untersucht, ob sie im nächsten Schritt in das *active set* einbezogen werden.

Unser Ziel ist es jedoch, den Gauß-Southwell an sich nicht zu verändern. Folglich ist es notwendig, die für die adaptive Durchlaufreihenfolge notwendigen Kosten zu senken, ohne diese an sich zu ändern oder den verwendeten Suchraum einzuschränken. Wenn die Kosten sich derart senken lassen, dass das Verfahren konkurrenzfähig zu Algorithmen mit statischer Durchlaufreihenfolge wird, muss das Konvergenzverhalten der Verfahren für verschiedene Problemstellungen untersucht werden. So lässt sich feststellen, ob sich der für die adaptive Durchlaufreihenfolge investierte Aufwand rentiert.

Um den Gauß-Southwell-Algorithmus darauf untersuchen zu können, wie er sich für das Lösen von partiellen Differentialgleichungen eignet, ist es notwendig, verschiedene Modellprobleme zu formulieren. Diese sollten zum einen so allgemein gewählt werden, dass sie Aufgabenstellungen aus der Praxis als Spezialfälle beinhalten. Außerdem ist zu überlegen, wie die Parameter Einfluss auf die Differentialgleichung nehmen sollen, um die Abhängigkeit des Gauß-Southwell und somit die Robustheit untersuchen zu können.

Dabei muss man beachten, dass speziell die Diffusion für das Gauß-Southwell-Verfahren ein schwieriges Problem ist. Im Gegensatz zu richtungsorientierten Operatoren, denen der Gauß-Southwell durch seine adaptiv gewählte Durchlaufreihenfolge leicht folgen kann, bewirkt die Diffusion, dass sich bei der Relaxation, also der Reduzierung des Fehlers in einem bestimmten Gitterpunkt, Fehleranteile isotrop auf die umliegenden Punkte verteilen.

Lösungsansätze

Zur Diskretisierung der Differentialgleichungen verwenden wir das Galerkin-Verfahren mit Finiten Elementen. Wir diskretisieren verschiedene Differentialgleichungen, die uns im weiteren Verlauf der Arbeit als Modellprobleme dienen. Die Konvergenzgeschwindigkeit der Iterationsverfahren, die die auf diese Weise aufgestellten Gleichungssysteme lösen sollen, sinkt allerdings

mit der Feinheit der Diskretisierung. Um diesem Problem zu begegnen, setzen wir Multilevelmethoden ein, die uns ein Verfahren liefern, die Gleichungssysteme in Multilevelsysteme zu transformieren. Die Lösung dieser so erweiterten Systeme ist stabil, d.h. die Konvergenzrate der Iterationsverfahren ist von der Maschenweite der Diskretisierung unabhängig.

Unser Ansatz, das Multilevelsystem effizient zu lösen, besteht darin, das Gauß-Southwell-Verfahren, welches sich adaptiv an ein Gleichungssystem anpassen kann, einzusetzen. Dabei wollen wir das Verfahren selbst nicht modifizieren, indem wir es nur Teile der Unbekannten oder des Residuums betrachten lassen. Die Kosten, die gegenüber einem Verfahren, das eine statische Durchlaufreihenfolge einsetzt, auftreten, müssen analysiert und so reduziert werden, dass das Gauß-Southwell-Verfahren konkurrenzfähig wird.

Dabei werden wir zunächst ausnutzen, dass sich nach einem Iterationsschritt das neue Residuum geschickt ausrechnen lässt, da nur die von der Relaxation betroffenen Elemente aktualisiert werden müssen. Der nächste Kostenfaktor, den es zu reduzieren gilt, ist der, stets den jeweils größten Eintrag im Residuenvektor kennen zu müssen. Zu diesem Zweck werden wir verschiedene Ansätze diskutieren und eine Datenstruktur suchen, die geeignet ist, das Residuum zu speichern.

Mit dem Binär-Heap finden wir eine Datenstruktur, die für diese Anwendung nützlich ist, und werden diese so erweitern, dass wir alle im Gauß-Southwell-Verfahren verwendeten Operationen kostengünstig in $\mathcal{O}(\log N)$ durchführen können.

Um den Gauß-Southwell-Algorithmus darauf untersuchen zu können, wie er sich für die Lösung verschiedener Differentialgleichungen in Multilevelsystemen eignet, formulieren wir mehrere Modellprobleme, die wesentliche Grundbausteine praxisrelevanter Problemstellungen sind. Für diese betrachten wir das Lösungsverhalten des adaptiven Algorithmus und vergleichen die erreichten Ergebnisse mit denen des Gauß-Seidel-Algorithmus. Um den Einfluss der Adaptivität auf die Robustheit des Verfahrens zu analysieren, verwenden wir für die Modellprobleme sowohl reine Operatoren als auch verschieden stark singulär gestörte. Dabei stellen wir fest, dass sich der für die adaptive Durchlaufreihenfolge benötigte Aufwand rentiert, so dass das Gauß-Southwell-Verfahren selbst für die *worst case*-Abschätzung der Kosten weniger aufwändig ist als das Gauß-Seidel-Verfahren mit einer statischen Durchlaufreihenfolge. Weiterhin beobachten wir, dass der Gauß-Southwell bei der Lösung einer anisotropen Diffusion vom Grade der Anisotropie deutlich weniger beeinflusst wird als der Gauß-Seidel. Folglich können wir mit der adaptiven Durchlaufreihenfolge die Robustheit des Verfahrens beachtlich erhöhen.

Zu diesem Zweck implementieren wir die Finite Elemente-Diskretisierung, die Multilevelmethoden und die Iterationsverfahren. Eine von uns entwickelte C++-Bibliothek ermöglicht es, verschiedene Modellprobleme zu konstruieren, den Einfluss von beliebigen Vorkonditionierungen und das Konvergenzverhalten von verschiedenen Iterationsverfahren für diese zu untersuchen. Zur Speicherung der dünn besiedelten Matrizen verwenden wir die entsprechenden Implementierungen aus den *Boost-uBLAS*-Bibliotheken [boo].

Eigene Beiträge

Fassen wir an dieser Stelle die eigenen Beiträge in dieser Arbeit zusammen.

- Ausführliche, praxisorientierte Darstellung des Galerkin-Verfahrens in Kombination mit Multilevelmethoden.
- Reduktion der Kosten des Gauß-Southwell-Verfahrens durch geschickte Berechnung der Elemente des Residuums, welches in einer speziellen Datenstruktur abgelegt wird. Da-

zu wird der bekannte Binär-Heap so erweitert, dass alle vom Gauß-Southwell benötigten Operationen in $\mathcal{O}(\log N)$ durchgeführt werden können.

- Erstmalig genauere Untersuchungen des Kosten-Nutzen-Verhältnisses des Gauß-Southwell-Verfahrens im Vergleich mit dem Gauß-Seidel-Verfahren. Zu diesem Zweck wird das Konvergenzverhalten beider Algorithmen im Multilevelsystem für verschiedene Modellprobleme untersucht.
- Vergleich des Lösungsverhaltens für das Modellproblem der anisotropen Diffusion der beiden Algorithmen im Nested-Iteration-Verfahren über Multilevelsystemen. Dabei wird die durch die Adaptivität gewonnene Verbesserung der Robustheit und die eindrucksvolle Verkürzung der Laufzeiten in der Praxis analysiert.

Aufbau der Arbeit

Die Arbeit ist wie folgt strukturiert. Wir beginnen in Kapitel 2 mit den für die Arbeit nötigen mathematischen Grundlagen, stellen die Problemstellung vor und behandeln erste Verfahren, mit denen diese gelöst werden kann.

In Kapitel 3 beschäftigen wir uns mit dem Galerkin-Verfahren und den Finiten Elementen. Wir leiten das Verfahren her und diskretisieren grundlegende Operatoren, aus denen Differentialgleichungen typischerweise bestehen. Diese dienen uns im weiteren Verlauf als Modellprobleme.

Daran schließen wir in Kapitel 4 mit Multilevelmethoden an, die wir herleiten und zum Aufstellen eines Multilevelsystems nutzen. Mit diesem Verfahren erreichen wir Stabilität, also die Unabhängigkeit der Konvergenzrate des Iterationsverfahrens von der Maschenweite der Diskretisierung.

In Kapitel 5 stellen wir das Gauß-Southwell-Verfahren vor, zeigen die Konvergenz des Verfahrens und zeichnen Möglichkeiten auf, die durch die Adaptivität entstehenden Kosten zu senken, indem wir das Residuum geschickt berechnen und in einer für diesen Zweck modifizierten Datenstruktur speichern.

Anschließend untersuchen wir in Kapitel 6, wie das Gauß-Southwell-Verfahren von der Adaptivität profitieren kann und vergleichen dessen Ergebnisse mit denen des Gauß-Seidel-Verfahrens. Wir betrachten dabei verschiedene Modellprobleme im Multilevelsystem und mit zusätzlichem Nested-Iteration-Verfahren. Wir analysieren den Gewinn des adaptiven Verfahrens mittels einer *worst case*-Abschätzung der Kosten und untersuchen die Laufzeit der Algorithmen in der Praxis. Schließlich zeigen wir auf, wie wir die Robustheit im Vergleich zum Gauß-Seidel-Verfahren verbessern konnten.

Eine Zusammenfassung und einen Ausblick auf weitere interessante Fragestellungen geben wir in Kapitel 7.

Danksagung

Bedanken möchte ich mich bei Prof. Dr. Michael Griebel für das interessante Thema, seine vielen Anregungen und die stete Unterstützung bei der Entwicklung dieser Arbeit. Prof. Dr. Michael Clausen danke ich für die Übernahme des Zweitgutachtens.

Meiner Betreuerin Jutta Adelsberger, die mich immer engagiert mit ihren Fachkenntnissen und Ideen unterstützte, möchte ich besonders danken. Ihre Ruhe und Geduld haben sehr zum Gelingen dieser Arbeit beigetragen.

Meinen Freunden und den Kollegen am Institut für Numerische Simulation danke ich für die vielen Diskussionen, ihre Hilfsbereitschaft und das Korrekturlesen.

Ich danke meinen Eltern für ihr Vertrauen, den moralischen Beistand und die Subventionen.

Schließlich gilt meine tiefe Dankbarkeit Karla Makarenko für ihre uneingeschränkte Unterstützung und den Rückhalt, den sie mir immer geboten hat.

2. Grundlagen

Zunächst befassen wir uns mit den mathematischen Grundlagen zu den in dieser Arbeit behandelten Problemen und Aufgabenstellungen. Wir beginnen damit, partielle Differentialgleichungen vorzustellen und betrachten Methoden, die angewendet werden können, um diese numerisch zu lösen. Dazu werfen wir einen ersten Blick auf Diskretisierungsverfahren, mit denen kontinuierliche Problemstellungen diskret formuliert werden können, so dass es möglich ist, eine Näherungslösung mit endlich vielen Daten zu beschreiben. Da sich diese als Gleichungssystem formulieren lassen, dessen Lösung gesucht wird, geben wir im Folgenden einen kurzen Überblick über einige Lösungsverfahren für diese Systeme.

2.1. Partielle Differentialgleichungen

Partielle Differentialgleichungen dienen der mathematischen Modellierung physikalischer Vorgänge. Allgemein formuliert ist eine partielle Differentialgleichung eine Gleichung, in der eine unbekannte Funktion u, die abhängig von mehreren voneinander unabhängigen Variablen ist, gesucht wird. Dabei werden in der Gleichung die unabhängigen Variablen, die abhängige Funktion u und deren partielle Ableitungen in Beziehung gesetzt [Str95].

Als Beispiel soll die Poisson-Gleichung dienen. Diese stellt eine elliptische partielle Differentialgleichung zweiter Ordnung dar und ist zu einem Standardproblem bei der Untersuchung partieller Differentialgleichungen und deren numerischer Lösung geworden. Sie tritt unter anderem bei der Modellierung reibungsbehafteter Strömungsfelder als Teil der inkompressiblen Navier-Stokes-Gleichungen auf und beschreibt die durch Reibung bedingte Diffusion.

Unter Verwendung des Laplace-Operators

$$\Delta = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$$

lässt sich die Poisson-Gleichung als Randwertproblem auf dem Gebiet $\Omega \subset \mathbb{R}^2$ in der Form

$$\begin{aligned} -\Delta u(x,y) &= f(x,y) \text{ für } (x,y) \in \Omega \subset \mathbb{R}^2, \\ u(x,y) &= g(x,y) \text{ für } (x,y) \in \partial\Omega \end{aligned} \tag{2.1}$$

formulieren, wobei $\partial\Omega$ den Rand des Gebiets Ω bezeichnet.

Partielle Differentialgleichungen lassen sich oft nur schwer analytisch lösen. Für den linearen Fall existieren Lösungen, für den nichtlinearen Fall gibt es noch keine abgeschlossene Theorie. Zur praktischen Berechnung werden in der Regel numerische Verfahren verwendet. Der erste Schritt hierbei ist es, von der kontinuierlichen Aufgabenstellung zu einer diskreten zu gelangen.

2.2. Diskretisierung

Diskretisierung ist der Übergang von einem kontinuierlichen Problem, wie etwa einer partiellen Differentialgleichung, zu einem diskreten Problem, dessen Lösung mit endlich vielen Daten beschrieben werden kann. Um dies zu erreichen, wird über das kontinuierliche Gebiet ein Gitter gelegt und das Problem für jeden Gitterpunkt diskret formuliert. Im Folgenden werden wir zunächst eine verbreitete Methode zur Diskretisierung behandeln, die Finiten Differenzen. Im weiteren Verlauf der Arbeit werden wir uns darüber hinaus auch mit dem Ritz-Galerkin-Verfahren genauer auseinandersetzen (siehe Kapitel 3).

2.2.1. Diskretisierungsfehler

Zunächst behandeln wir den Fehler, der bei einer Diskretisierung unweigerlich entsteht. Beschreibt man ein kontinuierliches Problem durch endlich viele Daten, so ist es nicht zu vermeiden, dass sich die diskrete Näherung von der korrekten Lösung unterscheidet.

Sei u die kontinuierliche Lösung und u_N die Näherung beruhend auf $N \in \mathbb{N}$ mit $N < \infty$ vielen Daten, dann ist der Diskretisierungsfehler definiert als

$$e_N := u - u_N. \tag{2.2}$$

Für ein konvergentes Diskretisierungsverfahren gilt

$$\lim_{N \to \infty} e_N = 0.$$

Demzufolge kann der Diskretisierungsfehler beliebig klein werden, wenn man ein konvergentes Verfahren einsetzt und die Auflösung des Gitters fein genug wählt. Die Abschätzung dieses Fehlers wird meistens in Verbindung mit der Maschenweite h angegeben. Je feiner die Maschenweite gewählt wird, desto größer ist die Anzahl der Daten und desto kleiner wird der Fehler, der durch den Übergang von der kontinuierlichen zur diskreten Lösung entsteht.

2.2.2. Finite Differenzen

Wir behandeln nun ein erstes numerisches Verfahren zur Lösung von partiellen Differentialgleichungen, das Verfahren der Finiten Differenzen. Wir folgen dabei der Notation von [Gri03]. Die Idee dieses Verfahrens ist es, die partielle Differentialgleichung in ein System von Differenzengleichungen umzuformen, welches dann mittels verschiedener Algorithmen gelöst werden kann.

Wir betrachten auf einem gegebenen Gebiet $\Omega \in \mathbb{R}^n$ mit einem Differentialoperator zweiter Ordnung L und einem festen $f \in C^0(\Omega)$ und $g \in C^0(\partial\Omega)$ die partielle Differentialgleichung

$$\begin{aligned} Lu &= f \quad \text{in } \Omega \\ u &= g \quad \text{auf } \partial\Omega. \end{aligned}$$

Wir suchen also eine Funktion $u \in C^2(\Omega)$, die diese Differentialgleichung erfüllt. Finite Differenzen basieren nun auf dem Prinzip, den Differentialoperator L durch sogenannte Differenzenoperatoren zu ersetzen.

Dazu betrachten wir anstatt des kontinuierlichen Gebiets $\Omega \in \mathbb{R}^n$ das Gitter Ω_h zu einer fest vorgegebenen Maschenweite $h > 0$

$$\Omega_h := \Omega \cap \{x = h \cdot z | z \in \mathbb{Z}^n\}$$

mit den Randpunkten

$$\partial\Omega_h := \partial\Omega \cap \{x = h \cdot z | z \in \mathbb{Z}^n\}$$

und definieren den approximierten Raum V_h der Gitterfunktionen

$$V_h(\Omega) := \{u_h : \Omega_h \mapsto \mathbb{R}\}.$$

Motiviert durch die Definition der Ableitung

$$\partial_x u := \lim_{h \to 0} \frac{u(x+h) - u(x)}{h}$$

ersetzen wir dann den kontinuierlichen Differentialoperator an der Stelle $x \in \Omega_h$ durch einen Differenzenoperator, indem wir alle auftretenden Differentiale durch entsprechende Differenzenquotienten ersetzen.

Definition 2.1 (Differenzenoperator)**.** *Sei u die zu diskretisierende Funktion, h die gewählte Maschenweite und e_i der i-te Einheitsvektor. Die wichtigsten Differenzenoperatoren sind die Vorwärtsdifferenz/rechtsseitige Differenz*

$$(\partial^+_{x_i} u)(x) := \frac{1}{h}[u(x + he_i) - u(x)], \tag{2.3}$$

die Rückwärtsdifferenz/linksseitige Differenz

$$(\partial^-_{x_i} u)(x) := \frac{1}{h}[u(x) - u(x - he_i)] \tag{2.4}$$

und die zentrale/symmetrische Differenz

$$(\partial^0_{x_i} u)(x) := \frac{1}{2h}[u(x + he_i) - u(x - he_i)]. \tag{2.5}$$

Die zweite partielle Ableitung $\partial^2_{x_i}$ wird durch

$$(\partial^+_{x_i}\partial^-_{x_i} u)(x) := \frac{1}{h^2}[u(x + he_i) - 2u(x) + u(x - he_i)] \tag{2.6}$$

approximiert.

Stellt man die Gleichung nun diskret in jedem Gitterpunkt auf, kann man das Problem als lineares Gleichungssystem formulieren. Weiterführende Analysen der Finiten Differenzen sowie Untersuchungen zur Konvergenz des Verfahrens und des Diskretisierungsfehlers finden sich in [Smi85].

Betrachten wir nun ein Beispiel für eine Finite Differenzen-Diskretisierung.

2.2.3. Eine Finite Differenzen-Diskretisierung für die Poisson-Gleichung

Wir diskretisieren die Poisson-Gleichung auf dem Einheitsquadrat $\Omega = (0,1) \times (0,1)$. Mittels einer Finite-Differenzen-Methode wird das Gebiet $\overline{\Omega} = \Omega \cup \partial\Omega$ mit einem Gitter Ω_h der Schrittweite $h = 1/(N+1)$ mit $N \in \mathbb{N}$ versehen.

Wir notieren

$$(x_i, y_j) = (ih, jh) \quad \text{für } i, j = 0, \ldots, N+1$$

sowie

$$u_{ij} = u(x_i, y_j) \quad \text{und} \quad f_{ij} = f(x_i, y_j) \quad \text{für } i, j = 0, \ldots, N+1.$$

Die zweite partielle Ableitung approximieren wir mit

$$\begin{aligned} \frac{\partial^2 u}{\partial x^2}(x_i, y_j) &\approx \frac{1}{h}\left(\frac{u_{i+1,j} - u_{i,j}}{h} - \frac{u_{i,j} - u_{i-1,j}}{h}\right) \\ &= \frac{1}{h^2}(u_{i+1,j} - 2u_{ij} + u_{i-1,j}). \end{aligned} \tag{2.7}$$

Für $\frac{\partial^2 u}{\partial y^2}$ gehen wir analog vor und erhalten so für den Laplace-Operator

$$-\Delta u(x_i, y_j) \approx \frac{1}{h^2}(4u_{ij} - u_{i-1,j} - u_{i+1,j} - u_{i,j-1} - u_{i,j+1}) \tag{2.8}$$

und damit die diskrete Form der Gleichung (2.1)

$$\begin{aligned} 4u_{i,j} - u_{i-1,j} - u_{i+1,j} - u_{i,j-1} - u_{i,j+1} = h^2 f_{ij} \qquad & \text{für } 1 \le i, j \le N, \\ u_{0,j} = g_{0,j}, \quad u_{N+1,j} = g_{N+1,j} \qquad & \text{für } j = 0, \ldots, N+1, \\ u_{i,0} = g_{i,0}, \quad u_{i,N+1} = g_{i,N+1} \qquad & \text{für } i = 0, \ldots, N+1. \end{aligned}$$

Um das Gleichungssystem aufzustellen, nummerieren wir u zeilenweise so um, dass sich

$$u_1 = u_{1,1},\ u_2 = u_{2,1},\ u_3 = u_{3,1},\ \ldots,\ u_N = u_{N,1},\ u_{N+1} = u_{1,2},\ \ldots,\ u_{N^2} = u_{N,N}$$

ergibt. Analog verfahren wir mit f und erhalten damit ein Gleichungssystem $Au = g$ für die Bestimmung des Lösungsvektors $u := (u_1, \ldots, u_{N^2})^T$. Hierbei gilt

$$A = \frac{1}{h^2}\begin{pmatrix} B & -I & & \\ -I & \ddots & \ddots & \\ & \ddots & \ddots & -I \\ & & -I & B \end{pmatrix} \in \mathbb{R}^{N^2 \times N^2} \tag{2.9}$$

mit $B = \begin{pmatrix} 4 & -1 & & \\ -1 & \ddots & \ddots & \\ & \ddots & \ddots & -1 \\ & & -1 & 4 \end{pmatrix} \in \mathbb{R}^{N \times N}$ und $I = Id \in \mathbb{R}^{N \times N}$. Die rechte Seite weist für den Spezialfall $g \equiv 0$ die Gestalt $g = h^2 (f_1, \ldots, f_{N^2})^T \in \mathbb{R}^{N^2}$ auf. Im Fall $g \neq 0$ müssen die entsprechenden Randwerte im Vektor der rechten Seite berücksichtigt werden.

Damit haben wir unser Problem in ein Gleichungssystem übertragen können, welches sich nun mit verschiedenen Verfahren lösen lässt. Nun betrachten wir eine Notation, mit der sich die Ergebnisse der Diskretisierung mit Finiten Differenzen und anderen Verfahren, die auf Linearkombinationen von benachbarten Punkten beruhen, einfach darstellen lassen.

2.3. Sternnotation

Auch andere Diskretisierungsverfahren lassen sich wie die Finiten Differenzen aus Definition 2.1 über Differenzen bzw. Linearkombinationen der Nachbarn eines jeden Punktes darstellen. Dies

führt zur Notation des Differenzensterns [Hac91], der sich für beliebige Dimensionen konstruieren lässt, den wir hier aber der Übersichtlichkeit halber in der zweiten Dimension definieren.

Wir folgen der Notation von 2.2.3.

Definition 2.2. *Ein Differenzenstern mit variablen Koeffizienten $a_{p,q} \in V,\ -\infty < p,q < \infty$ im Punkt $(ih, jh) \in \Omega_h$ ist gegeben durch*

$$\begin{bmatrix} & \vdots & \vdots & \vdots & \\ \dots & a_{-1,1} & a_{0,1} & a_{1,1} & \dots \\ \dots & a_{-1,0} & a_{0,0} & a_{1,0} & \dots \\ \dots & a_{-1,-1} & a_{0,-1} & a_{1,-1} & \dots \\ & \vdots & \vdots & \vdots & \end{bmatrix} u_{i,j} := \sum_{p,q=-\infty}^{\infty} a_{p,q} \cdot u_{i+p,j+q}. \tag{2.10}$$

Es werden nur Zeilen und Spalten notiert, die nicht-Null Koeffizienten enthalten. Einzelne Koeffizienten $a_\bullet$, die Null sind, werden nicht notiert.

2.3.1. Operationen auf Sternen

Jede Diskretisierung, die sich pro Gitterpunkt als Linearkombination seiner Nachbarn schreiben lässt, kann als Stern notiert werden. Durch diese Notation verlieren die Operatoren keine ihrer Eigenschaften. Es lassen sich die üblichen Rechenregeln auf Sterne anwenden, wobei diese nicht mit Operationen auf Matrizen zu verwechseln sind.

Es gilt für die Addition von zwei Sternen A, B mit den Koeffizienten $a_{i,j}, b_{i,j} \in V, -\infty < i,j < \infty$ und $c \in V$

$$\begin{aligned} c \cdot \begin{bmatrix} & \vdots & \vdots & \vdots & \\ \dots & a_{-1,1} & a_{0,1} & a_{1,1} & \dots \\ \dots & a_{-1,0} & a_{0,0} & a_{1,0} & \dots \\ \dots & a_{-1,-1} & a_{0,-1} & a_{1,-1} & \dots \\ & \vdots & \vdots & \vdots & \end{bmatrix} \pm \begin{bmatrix} & \vdots & \vdots & \vdots & \\ \dots & b_{-1,1} & b_{0,1} & b_{1,1} & \dots \\ \dots & b_{-1,0} & b_{0,0} & b_{1,0} & \dots \\ \dots & b_{-1,-1} & b_{0,-1} & b_{1,-1} & \dots \\ & \vdots & \vdots & \vdots & \end{bmatrix} \\ = \begin{bmatrix} & \vdots & \vdots & \vdots & \\ \dots & c \cdot a_{-1,1} \pm b_{-1,1} & c \cdot a_{0,1} \pm b_{0,1} & c \cdot a_{1,1} \pm b_{1,1} & \dots \\ \dots & c \cdot a_{-1,0} \pm b_{-1,0} & c \cdot a_{0,0} \pm b_{0,0} & c \cdot a_{1,0} \pm b_{1,0} & \dots \\ \dots & c \cdot a_{-1,-1} \pm b_{-1,-1} & c \cdot a_{0,-1} \pm b_{0,-1} & c \cdot a_{1,-1} \pm b_{1,-1} & \dots \\ & \vdots & \vdots & \vdots & \end{bmatrix}. \end{aligned}$$

Seien A, B, C Differenzensterne mit den Koeffizienten $a_{i,j}, b_{i,j}, c_{i,j} \in V$, $-\infty < i,j < \infty$. Dann gilt für die Multiplikation $A \cdot B = C$

$$c_{i,j} := \sum_{p,q=-\infty}^{\infty} a_{p,q} \cdot b_{i+p,j+q} \quad \forall i,j.$$

Diese Eigenschaften führen zusammen mit der guten Lesbarkeit der Sterne zu einer überschaubaren Notation.

2.3.2. Finite Differenzen in Sternnotation

Die Finite-Differenzen-Diskretisierungen aus Abschnitt 2.2.3 lassen sich nun auch mittels Differenzensternen darstellen. Für die zweiten partiellen Ableitungen (2.7) ergibt sich somit

$$\frac{\partial^2 u}{\partial x^2}(x_i, y_j) \approx \frac{1}{h^2}(u_{i+1,j} - 2u_{ij} + u_{i-1,j}) = \frac{1}{h^2}\begin{bmatrix}1 & -2 & 1\end{bmatrix} \tag{2.11}$$

$$\frac{\partial^2 u}{\partial y^2}(x_i, y_j) \approx \frac{1}{h^2}(u_{i,j+1} - 2u_{ij} + u_{i,j-1}) = \frac{1}{h^2}\begin{bmatrix}1\\-2\\1\end{bmatrix} \tag{2.12}$$

Analog verkürzt sich die Darstellung des diskreten Laplace-Operators (2.8) auf

$$-\Delta_h u(x_i, y_j) = \frac{1}{h^2}(4u_{ij} - u_{i-1,j} - u_{i+1,j} - u_{i,j-1} - u_{i,j+1}) = \frac{1}{h^2}\begin{bmatrix}&-1&\\-1&4&-1\\&-1&\end{bmatrix}.$$

Dieser Stern wird als *Fünf-Punkte-Stern* bezeichnet. Er ergibt sich per Definition aus den obigen zweiten Ableitungen (2.11, 2.12) und der Addition ihrer Sterne

$$-\Delta u = -\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) \approx -\left(\frac{1}{h^2}\begin{bmatrix}1 & -2 & 1\end{bmatrix} + \frac{1}{h^2}\begin{bmatrix}1\\-2\\1\end{bmatrix}\right) = \frac{1}{h^2}\begin{bmatrix}&-1&\\-1&4&-1\\&-1&\end{bmatrix}.$$

2.4. Lösung linearer Gleichungssysteme

2.4.1. Lineare Gleichungssysteme

Ein lineares Gleichungssystem bezeichnet ein System linearer Gleichungen, die mehrere unbekannte Variablen enthalten. Die Gleichungen modellieren Zusammenhänge zwischen diesen mit dem Ziel, die Unbekannten bestimmen zu können. Allgemein kann ein lineares Gleichungssystem mit m Gleichungen und n Unbekannten immer in folgender Form notiert werden:

$$\begin{aligned}
a_{11}x_1 + a_{12}x_2 \quad &+ \cdots + \quad a_{1n}x_n \quad &= \quad b_1\\
a_{21}x_1 + a_{22}x_2 \quad &+ \cdots + \quad a_{2n}x_n \quad &= \quad b_2\\
&&\vdots\\
a_{m1}x_1 + a_{m2}x_2 \quad &+ \cdots + \quad a_{mn}x_n \quad &= \quad b_m
\end{aligned}$$

Die Koeffizienten a_{ij} des Gleichungssystems lassen sich als Matrix A schreiben

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n}\\ a_{21} & a_{22} & \cdots & a_{2n}\\ \vdots & \vdots & \ddots & \vdots\\ a_{m1} & a_{m2} & \cdots & a_{mn}\end{pmatrix}$$

und die Unbekannte x sowie die rechte Seite b als Vektoren

$$x = \begin{pmatrix} x_1\\ x_2\\ \vdots\\ x_n\end{pmatrix} \quad \text{und} \quad b = \begin{pmatrix} b_1\\ b_2\\ \vdots\\ b_m\end{pmatrix}.$$

Das lineare Gleichungssystem kann also als

$$Ax = b \tag{2.13}$$

mit $A \in \mathbb{R}^{m \times n}, x \in \mathbb{R}^n, b \in \mathbb{R}^m$ dargestellt werden. Wir nehmen an, dass A und b reell sind, obwohl diese Einschränkung bei den meisten Verfahren unwesentlich ist.

Ein lineares Gleichungssystem $Ax = b$ ist genau dann lösbar, wenn der Rang der Matrix A gleich dem Rand der erweiterten Matrix

$$\text{rang}(A) = \text{rang}(A, b)$$

ist. Entspricht der Rang der erweiterten Matrix der Anzahl der Unbekannten n, so ist die Lösung eindeutig [Mei08].

Neben den oben genannten Verfahren zur Diskretisierung von partiellen Differentialgleichungen gibt es viele andere Verfahren, die zu einem linearen Gleichungssystem führen.

Die naive Lösung eines linearen Gleichungssystems (2.13) wäre die Multiplikation von links mit der Inversen der Koeffzientenmatrix A

$$\begin{aligned} & A^{-1}Ax = A^{-1}b \\ \Leftrightarrow \quad & x = A^{-1}b. \end{aligned}$$

Die Inverse einer Matrix zu berechnen, ist im Allgemeinen sehr aufwändig, fehlerbehaftet und auch häufig gar nicht möglich, da nur reguläre Matrizen invertierbar sind. Außerdem können lineare Gleichungssysteme für viele Anwendungen sehr groß sein, so dass diese Methode unpraktikabel ist. Daher wenden wir uns nun anderen Verfahren zu.

2.4.2. Klassische direkte Verfahren

Direkte Verfahren zur Lösung von linearen Gleichungssystemen sind Algorithmen, die in endlich vielen Schritten die exakte Lösung berechnen. Ihre Eigenschaften haben jedoch zur Folge, dass sie nur selten zum Lösen großer linearer Gleichungssysteme eingesetzt werden. Meist finden sie in einer unvollständigen Form eine Verwendung als Vorkonditionierer oder werden zur Lösung von Subproblemen eingesetzt. Insbesondere mit dünn besiedelten Matrizen kommen direkte Verfahren nicht gut zurecht, da diese die Gestalt der Matrizen nicht ausnutzen können und so vollbesetzte Zwischenmatrizen berechnet werden. Dies hat zur Folge, dass Speicher- und Rechenaufwand extrem ansteigen.

2.4.2.1. Gauß-Elimination

Die Grundidee des Gaußschen Eliminationsverfahrens liegt in der Transformation des Systems $Ax = b$ in ein äquivalentes System der Form

$$LRx = b$$

mit einer rechten oberen Dreiecksmatrix R und einer linken unteren Dreiecksmatrix L, welche durch einfaches Vorwärts- und anschließendes Rückwärtseinsetzen gelöst werden kann. Hierbei wird die Multiplikation $\hat{b} = L^{-1}b$ simultan mit der Berechnung der Matrix R durchgeführt.

Die führenden $k \times k$-Hauptabschnittsmatrizen $A[k]$ von $A \in \mathbb{R}^{n \times n}$ sind definiert über

$$A[k] := \begin{pmatrix} a_{11} & \cdots & a_{1k} \\ \vdots & \ddots & \vdots \\ a_{k1} & \cdots & a_{kk} \end{pmatrix} \in \mathbb{R}^{k \times k} \text{ für } k \in \{1, \ldots, n\}.$$

Die Gauß-Elimination setzt voraus, dass die Matrix $A \in \mathbb{R}^{n \times n}$ regulär ist, und für Determinanten der führenden $k \times k$-Hauptabschnittsmatrizen von A gilt

$$\det(A[k]) \neq 0.$$

Durch vollständige Pivotisierung, bei der die Zeilen und Spalten der Matrix A so vertauscht werden, dass auf der Diagonalen die betragsmäßig größten Elemente stehen, wird der Algorithmus stabil [Mei08]. Wir lassen hier diesen Vorverarbeitungsschritt aus und betrachten den Algorithmus ohne die vorher ausgeführte Pivotisierung unter der Voraussetzung, dass alle Diagonaleinträge der Matrix ungleich Null sind.

Algorithmus 1: Gauß-Elimination

Input : $n \times n$ Matrix A, rechte Seite b
Output : Lösungsvektor x
`// Berechnung der LR-Zerlegung`
for $k = 1, \ldots, n-1$ **do**
  for $i = k+1, \ldots, n$ **do**
    $a_{i,k} := \frac{a_{i,k}}{a_{k,k}}$
    for $j = k+1, \ldots, n$ **do**
      $a_{i,j} := a_{i,j} - a_{i,k} a_{k,j}$
    end
  end
end
`// Vorwärtselimination` $\hat{b} := L^{-1} b$
for $k = 2, \ldots, n$ **do**
  for $i = 1, \ldots, k-1$ **do**
    $b_k := b_k - a_{k,i} b_i$
  end
end
`// Rückwärtselimination`
for $k = n, \ldots, 1$ **do**
  for $i = k+1, \ldots, n$ **do**
    $b_k := b_k - a_{k,i} x_i$
  end
  $x_k := \frac{b_k}{a_{k,k}}$
end

Die Laufzeit der Gauß-Elimination beträgt $\mathcal{O}(n^3)$.

2.4.2.2. Cholesky-Zerlegung

Ist die gegebene Matrix A symmetrisch und positiv definit, kann genutzt werden, dass sich $A \in \mathbb{R}^{n\times n}$ in ein Produkt

$$A = LL^T$$

mit einer linken unteren Dreiecksmatrix $L \in \mathbb{R}^{n\times n}$ aufspalten lässt. Eine solche Zerlegung heißt *Cholesky-Zerlegung*. Der Rechenaufwand der Cholesky-Zerlegung kann gegenüber der LR-Zerlegung der Gauß-Elimination halbiert werden, der Aufwand der Vorwärts- und Rückwärtssubstitution bleibt gleich [Sto94].

2.4.3. Klassische iterative Verfahren

Iterative Verfahren ermitteln nicht direkt die Lösung eines Gleichungssystems, sondern nähern sich sukzessive an die Lösung an. Dabei wird eine feste Rechenvorschrift wiederholt ausgeführt. Da die Gleichungssysteme meist durch eine Diskretisierung der zugrundeliegenden Aufgabenstellung entstehen, stellt die exakte Lösung des Gleichungssystems nur eine Näherung an die gesuchte kontinuierliche Lösung dar. Demnach ist eine Näherungslösung mit einem Iterationsfehler in der Größe des Diskretisierungsfehlers ausreichend. Dies kann als Abbruchkriterium des Iterationsverfahrens genutzt werden.

Definition 2.3 (lineares Iterationsverfahren)**.** *Ein Iterationsverfahren ist gegeben durch eine Abbildung*

$$\phi : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^n$$

und heißt linear, falls Matrizen M,N $\in \mathbb{R}^n \times \mathbb{R}^n$ *derart existieren, dass*

$$\phi(x, b) = Mx + Nb$$

gilt. Die Matrix M wird als Iterationsmatrix der Iteration ϕ *bezeichnet.*

Bis die gewünschte Näherung der Lösung des Gleichungssystems erreicht ist, wird die vom Iterationsverfahren vorgeschriebene Rechenvorschrift

$$x^{(m+1)} = \phi(x^{(m)}, b) \text{ für } m = 0, 1, \ldots$$

wiederholt ausgeführt.

Bevor wir konkrete Beispiele betrachten, beschreiben wir einige Eigenschaften iterativer Verfahren.

Definition 2.4 (Fixpunkt)**.** *Einen Vektor* $x \in \mathbb{R}^n$ *bezeichnen wir als Fixpunkt des Iterationsverfahrens* $\phi : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^n$ *zu* $b \in \mathbb{R}^n$*, falls*

$$x = \phi\,(x, b)$$

gilt.

Definition 2.5 (konsistent, konvergent)**.** *Ein Iterationsverfahren* ϕ *heißt konsistent zur Matrix A, wenn für alle* $b \in \mathbb{R}^n$ *die Lösung* $A^{-1}b$ *ein Fixpunkt von* ϕ *zu b ist. Ein Iterationsverfahren* ϕ *heißt konvergent, wenn für alle* $b \in \mathbb{R}^n$ *und alle Startwerte* $x^{(0)} \in \mathbb{R}^n$ *ein vom Startwert unabhängiger Grenzwert*

$$\hat{x} = \lim_{m\to\infty} x^{(m)} = \lim_{m\to\infty} \phi\left(x^{(m-1)}, b\right)$$

existiert.

Die Konsistenz eines Verfahrens ist eine notwendige Bedingung, da mit ihr ein sinnvoller Zusammenhang zwischen der numerischen Methode und dem Gleichungssystem sichergestellt wird.

Um ein Iterationsverfahren zur Lösung eines Gleichungssystems einsetzen zu können, ist es notwendig, dass das Verfahren konvergiert. In [Mei08] und [Sto94] wird die Konvergenz von den hier vorgestellten linearen Verfahren anhand der Iterationsmatrix untersucht. Am Ende des Kapitels setzen wir uns aus praktischer Sicht mit der Konvergenz der Verfahren auseinander und betrachten diese ausgehend vom Iterationsfehler.

2.4.3.1. Jacobi-Verfahren

Das Jacobi-Verfahren gehört zu den Splitting-Methoden, deren Grundidee wir kurz vorstellen. Die Splitting-Methoden zur Lösung von Gleichungssystemen (2.13) basieren auf einer Aufteilung der Matrix A in der Form

$$A = B + (A - B), \quad B \in \mathbb{R}^{n \times n}, \tag{2.14}$$

so dass sich aus

$$Ax = b$$

das äquivalente System

$$Bx = (B - A)x + b$$

ergibt. Ist B zudem regulär, dann erhalten wir

$$x = B^{-1}(B - A)x + B^{-1}b$$

und definieren hierdurch das lineare Iterationsverfahren

$$x^{(m+1)} = \phi(x^{(m)}, b) = Mx^{(m)} + Nb \text{ für } m = 0, 1, \ldots$$

mit

$$M := B^{-1}(B - A)$$

und

$$N := B^{-1}.$$

Die verschiedenen Splitting-Methoden versuchen nun, durch die Matrix B die Matrix A möglichst gut zu approximieren, so dass man sich durch Iterationen immer weiter an die exakte Lösung $A^{-1}b$ annähert. Dabei muss darauf geachtet werden, dass die gewählte Matrix B möglichst einfach zu invertieren ist. Ohne diese Bedingung fiele die Wahl leicht, da A die bestmöglichste Wahl wäre.

Ist die Matrix B regulär, dann ist das lineare Iterationsverfahren zur Matrix A konsistent. Die Konvergenz der Splitting-Methoden ist nur vom Spektralradius der Iterationsmatrix M abhängig und kann durch jede beliebige Matrixnorm abgeschätzt werden [Mei08].

Das Jacobi-Verfahren setzt voraus, dass die reguläre Matrix A des linearen Gleichungssystems (2.13) nichtverschwindende Diagonalelemente $a_{ii} \neq 0$ für $i = 1, \ldots, n$ besitzt, so dass mit

$$D = \text{diag}\{a_{11}, \ldots, a_{nn}\}$$

eine reguläre Diagonalmatrix vorliegt. Das Jacobi-Verfahren wählt nun als Näherung an die Matrix A

$$B = D.$$

Die Matrix D ist regulär und durch ihre Diagonalgestalt leicht zu invertieren.

Der Grundidee der Splitting-Methoden folgend, schreiben wir nun das lineare Gleichungssystem $Ax = b$ in der äquivalenten Form

$$x = \underbrace{D^{-1}(D - A)}_{=:M_J} x + \underbrace{D^{-1}}_{=:N_J} b.$$

Das hiermit definierte lineare Iterationsverfahren

$$x^{(m+1)} = D^{-1}(D - A)x^{(m)} + D^{-1}b \quad \text{für } m = 0, 1, 2, \ldots \tag{2.15}$$

ist konsistent zur Matrix A, und wir erhalten die Komponentenschreibweise

$$x_i^{(m+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{\substack{j=1 \\ j \neq i}}^{n} a_{ij} x_j^{(m)} \right) \quad \text{für } i = 1, \ldots, n \text{ und } m = 0, 1, 2, \ldots$$

Da jede neue Iterierte $x^{(m+1)}$ ausschließlich mittels der alten Iterierten $x^{(m)}$ ermittelt wird, nennt man diese Methode auch *Gesamtschrittverfahren*.

Die Konvergenz des Jacobi-Verfahrens ist gesichert, wenn die reguläre Matrix $A \in \mathbb{R}^{n \times n}$ *irreduzibel* sowie *diagonaldominant* ist und $\forall\, i\ a_{ii} \neq 0$ gilt [Mei08].

Algorithmus 2: Jacobi-Algorithmus

Input : $n \times n$ Matrix A, rechte Seite b, Startvektor x^0
Output : Näherungslösung x^c

```
// Führe c Relaxierungen durch
for m = 1, ..., c do
    for i = 1, ..., n do
        x_i := 0
        for j = 1, ..., n do
            if j ≠ i then
                x_i = x_i + a_{i,j} x_j^{(m-1)}
            end
        end
        x_i = 1/a_{i,i} (b_i - x_i)
    end
    x^{(m)} = x;
end
```

2.4.3.2. Gauß-Seidel-Verfahren

Wie beim Jacobi-Verfahren wird beim Gauß-Seidel-Verfahren ein Gleichungssystem (2.13) mit einer regulären Matrix $A \in \mathbb{R}^{n \times n}$ betrachtet, die einen regulären Diagonalanteil

$$D = \operatorname{diag}\{a_{11}, \ldots, a_{nn}\}$$

aufweist. Im Gauß-Seidel-Verfahren wird nun A genauer approximiert als im Jacobi-Verfahren, doch bevor wir die Näherung betrachten, definieren wir zunächst strikte untere und obere Dreiecksmatrizen.

Definition 2.6. *Wir definieren die strikte untere Dreiecksmatrix*

$$L = (l_{ij})_{i,j=1,\dots,n} \text{ mit } l_{ij} = \begin{cases} a_{ij}, & i > j \\ 0, & \text{sonst} \end{cases}$$

und die strikte obere Dreiecksmatrix

$$R = (r_{ij})_{i,j=1,\dots,n} \text{ mit } r_{ij} = \begin{cases} a_{ij}, & i < j \\ 0, & \text{sonst.} \end{cases}$$

Wenn wir die Matrix A in die Diagonalmatrix, die strikte untere und obere Dreieicksmatrix aufsplitten, dann erhalten wir das zu $Ax = b$ äquivalente Gleichungssystem

$$\begin{aligned} & Ax &=& b \\ \Leftrightarrow \quad & (R + D + L)x &=& b \\ \Leftrightarrow \quad & (D + L)x + Rx &=& b \\ \Leftrightarrow \quad & (D + L)x &=& -Rx + b \end{aligned}$$

und damit

$$x = \underbrace{-(D + L)^{-1}R}_{=:M_{GS}} x + \underbrace{(D + L)^{-1}}_{=:N_{GS}} b.$$

Mit den so gewonnen Matrizen M_{GS} und N_{GS} können wir die Iterationsvorschrift des linearen Gauß-Seidel-Iterationsverfahrens

$$x^{(m+1)} = -(D + L)^{-1}Rx^{(m)} + (D + L)^{-1}b \quad \text{für } m = 0, 1, 2, \dots \tag{2.16}$$

definieren. Das Verfahren ist konsistent zur Matrix A [Mei08]. Es ist zu erwarten, dass die Konvergenzgeschwindigkeit des Verfahrens verglichen der des Jacobi-Verfahrens schneller ist, da Gauß-Seidel mit $D+L$ eine bessere Approximation der Matrix A verwendet, woraus ein kleinerer Spektralradius der Iterationsmatrix resultiert.

Wir betrachten die Komponentenschreibweise unter der Voraussetzung, dass $x_j^{(m+1)}$ für $j = 1, \dots, i-1$ bekannt sind, dann kann $x_i^{(m+1)}$ durch

$$x_i^{(m+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(m+1)} - \sum_{i=1+1}^{n} a_{ij} x_j^{(m)} \right) \quad \text{für } i = 1, \dots, n \text{ und } m = 0, 1, 2, \dots$$

berechnet werden. Hier wird deutlich, dass bei jeder berechneten Komponente zusätzlich zu den Komponenten des vorherigen Iterationsschritts die bereits berechneten Komponenten des aktuellen Iterationsschritts benötigt werden. Daher wird das Verfahren auch als *Einzelschritt-verfahren* bezeichnet.

Algorithmus 3: Gauß-Seidel-Algorithmus

Input : $n \times n$ Matrix A, rechte Seite b, Startvektor x^0
Output : Näherungslösung x^c

```
// Führe c Schritte durch
for m = 1, ..., c do
    for i = 1, ..., n do
        // Relaxiere einzelne Komponente x_i
        x_i := 0
        for j = 1, ..., i − 1 do
            x_i = x_i + a_{i,j} x_j^{(m+1)}
        end
        for j = i + 1, ..., n do
            x_i = x_i + a_{i,j} x_j^{(m)}
        end
        x_i = (1 / a_{i,i}) (b_i − x_i)
    end
    x^{(m)} = x
end
```

2.4.4. Konvergenzrate, Stabilität und Robustheit

Sei $x^{(m)}$ die Iterierte im Schritt m eines Iterationsverfahrens über einem Gleichungssystem $Ax = b$ und u die Lösung dieses Systems. Dann ist der Iterationsfehler $e^{(m)}$ definiert als

$$e^{(m)} := u - x^{(m)}.$$

Mit dem Iterationsfehler lässt sich die Konvergenzrate eines Iterationsverfahrens definieren. Die Konvergenzrate zwischen Schritt m und Schritt $m + k$ ist definiert als

$$\rho^{(m,m+k)}\left(x^{(0)}\right) := \left(\frac{\left|\left|e^{(m+k)}\right|\right|}{\left|\left|e^{(m)}\right|\right|}\right)^{\frac{1}{k}}, \text{ Startvektor } x^{(0)} \in \mathbb{R}^n.$$

Für die Konvergenzrate des gesamten Verfahrens folgt somit

$$\rho := \lim_{k\to\infty} \max\left\{\rho^{(m,m+k)}\left(x^{(0)}\right) \,\middle|\, x^{(0)} \in \mathbb{R}^n\right\} \quad \forall m \in \mathbb{N}.$$

Genau dann, wenn $\rho < 1$ gilt, konvergiert das Iterationsverfahren, da in diesem Fall der Fehler immer von Schritt zu Schritt kleiner wird.

Die Konvergenzgeschwindigkeit eines Verfahrens hängt im Allgemeinen von der Genauigkeit der Diskretisierung, sprich der Maschenweite h, und den Parametern der Differentialgleichung, die wir im Folgenden mit $c_i \; \forall\, i$ bezeichnen, ab, d.h.

$$\rho = \rho(h, c_i).$$

Ist ein Verfahren dennoch unabhängig von einem der beiden, dann lassen sich die Begriffe Stabilität und Robustheit definieren.

Definition 2.7 (Stablität)**.** *Ein Iterationsverfahren heißt* stabil*, wenn für die Konvergenzrate gilt, dass sie unabhängig von der Maschenweite der dem Gleichungssystem zugrunde liegenden Diskretisierung ist, d.h.*

$$\rho\left(h, c_i\right) = \rho\left(c_i\right).$$

Definition 2.8 (Robustheit)**.** *Wenn die Konvergenzgeschwindigkeit eines Iterationsverfahrens unabhängig von Änderungen der Parameter der Differentialgleichung ist, so heißt dieses Verfahren* robust*, und es gilt*

$$\rho\left(h, c_i\right) = \rho\left(h\right).$$

Damit haben wir die Grundlagen für die folgenden Kapitel erarbeitet. Unser Ziel ist es, ein Iterationsverfahren mit einer geeigneten Diskretisierung zu konstruieren, welches uns sowohl Stabilität als auch Robustheit in gewissem Sinne garantiert.

3. Galerkin-Verfahren

Das Galerkin-Verfahren ist ein numerisches Diskretisierungsverfahren zur Übertragung einer kontinuierlichen partiellen Differentialgleichungen in ein lineares Gleichungssystem. Die Idee des Verfahrens ist es, die schwache Form der Differentialgleichung mittels Multiplikation mit einer Testfunktion zu bilden und anschließend über dem gegebenen Gebiet zu integrieren.

3.1. Herleitung am Beispiel der Poisson-Gleichung

Wir betrachten die Formulierung der schwachen Form anhand des Poisson-Problems

$$\begin{aligned} -\Delta u &= f \text{ für } (x,y) \in \Omega \subset \mathbb{R}^2, \\ u &= 0 \text{ für } (x,y) \in \partial\Omega. \end{aligned} \tag{3.1}$$

Mit der Multiplikation der ersten Gleichung mit einer beliebigen Testfunktion $v \in C_0^\infty(\Omega)$ und der Integration über Ω erhalten wir die schwache Form

$$\begin{aligned} -\int_\Omega v\Delta u dx &:= \int_\Omega v f dx \text{ für alle } v \in C_0^\infty(\Omega) \\ \overset{\text{partielle Integration}}{\Longleftrightarrow} \int_\Omega \nabla v \nabla u dx &:= \int_\Omega f v dx \text{ für alle } v \in C_0^\infty(\Omega). \end{aligned}$$

In der *schwachen Formulierung* muss u nur noch einmal, nicht mehr zweimal differenzierbar sein. Der Lösungsraum wurde also erweitert.

Die schwache Formulierung induziert eine symmetrische Bilinearform $a(\cdot,\cdot)$ und ein lineares Funktional $l(\cdot)$

$$\begin{aligned} a(u,v) &:= \int_\Omega \nabla u \nabla v dx, \\ l(v) &:= \int_\Omega f v dx. \end{aligned} \tag{3.2}$$

Damit können wir die schwache Formulierung auch als

$$a(u,v) = l(v) \text{ für alle } v \in C_0^\infty(\Omega) \tag{3.3}$$

schreiben.

Die schwache Formulierung besitzt eine eindeutige Lösung, wenn die Bilinearform $a(\cdot,\cdot)$ die Bedingungen des Lemmas von Lax-Milgram erfüllt [Gri03].

Die dem Ritz-Galerkin-Verfahren zugrundeliegende Idee ist nun, durch das Lösen von (3.3) eine Näherung u_N an die Lösung u in einem endlichdimensionalen Teilraum V_N von V mit $\dim V_N = N < \infty$ zu berechnen. Dies führt uns zu dem diskreten Problem: Finde $u_N \in V_N$ mit

$$a(u_N, v_N) = l(v_N) \quad \forall\, v_N \in V_N \subset V. \tag{3.4}$$

Um die Diskretisierung $V \to V_N$ in der Praxis umsetzen zu können, wählen wir eine Basis $\{\varphi_i\}_{i=1\ldots N}$ des endlichdimensionalen Raumes V_N.

Da (3.4) für beliebige Elemente aus V_N gelten soll, muss es insbesondere für die Basis gelten, also

$$a(u_N, \varphi_j) = l(\varphi_j) \quad \forall\, j = 1, \ldots, N.$$

Da u_N ein Element von V_N ist und damit eine Darstellung in der Basis $\{\varphi_1, \ldots, \varphi_N\}$ besitzt, können wir u_N mit Hilfe von geeigneten Koeffizienten $u_{N,j} \in \mathbb{R}$ und der Basis darstellen

$$u_N = \sum_{i=1}^{N} u_{N,i}\varphi_i.$$

Wenn wir die schwache Formulierung diskretisieren, als Testfunktion v die Basiselemente wählen und u_N mittels der Basis darstellen, ergibt sich

$$\begin{aligned} & a(u_N, \varphi_j) = l(\varphi_j) && \forall j = 1, \ldots, N \\ \Leftrightarrow \quad & a\left(\sum_{i=1}^{N} u_{N,i}\varphi_i, \varphi_j\right) = l(\varphi_j) && \forall j \\ \Leftrightarrow \quad & \sum_{i=1}^{N} u_{N,i}\, a(\varphi_i, \varphi_j) = l(\varphi_j) && \forall j. \end{aligned} \tag{3.5}$$

Auf diese Weise erhalten wir ein lineares Gleichungssystem

$$\underbrace{\begin{pmatrix} a(\varphi_1,\varphi_1) & \cdots & a(\varphi_N,\varphi_1) \\ \vdots & & \vdots \\ a(\varphi_1,\varphi_N) & \cdots & a(\varphi_N,\varphi_N) \end{pmatrix}}_{=:L_N} \cdot \underbrace{\begin{pmatrix} u_{N,1} \\ \vdots \\ u_{N,N} \end{pmatrix}}_{=:u_N} = \underbrace{\begin{pmatrix} l(\varphi_1) \\ \vdots \\ l(\varphi_N) \end{pmatrix}}_{=:l_N}. \tag{3.6}$$

Die Matrix L_N bezeichnet man auch als Steifigkeitsmatrix und den Vektor l_N als Lastvektor. Die Eigenschaften der kontinuierlichen Bilinearform $a(\cdot,\cdot)$ übertragen sich auf die Matrix L_N. Ist $a(\cdot,\cdot)$ symmetrisch, so ist auch die Steifigkeitsmatrix symmetrisch, ist die V-Elliptizität aus dem Lemma von Lax-Milgram erfüllt, so ist die Matrix positiv definit und damit regulär.

Zur Abschätzung des Fehlers zwischen der kontinuierlichen Lösung der schwachen Formulierung und der Lösung des endlichdimensionalen Problems sei auf das Lemma von Céa verwiesen [Gri03].

Die Effizienz des Galerkin-Verfahrens ist stark von der Wahl des Raumes V_N und der entsprechenden Basis abhängig. Der Raum sollte die Lösung möglichst gut approximieren können, und die Basis sollte so gewählt werden, dass die Steifigkeitsmatrix leicht aufzustellen und zu berechnen ist.

Ebenso erleichtert es das Lösen des Gleichungssystems, wenn dieses dünn besiedelt ist, was von einer entsprechend gewählten Basis gewährleistet werden kann. Im Folgenden sei $L := L_N$. Stellt man die Steifigkeitsmatrix L mit dem Galerkin-Verfahren auf, kann sie aus bis zu N^2 vielen Integralen $a(\varphi_i, \varphi_j)$ bestehen. Für große N ist die dichtbesiedelte Matrix viel zu teuer aufzustellen, zu speichern und das resultierende Gleichungssystem viel zu aufwändig zu lösen. Die verwendete Basis sollte also so beschaffen sein, dass möglichst viele Einträge der Matrix $a(b_i, b_j) = 0$ sind. Dies erreicht man mit den sogenannten *Finiten Elementen*.

3.2. Finite Elemente

Unser Ziel ist es nun, durch eine geschickte Wahl der Basisfunktionen die Dünnbesiedeltheit der Steifigkeitsmatrix zu erreichen. In der Bilinearform $a(\cdot,\cdot)$ wird stets das Produkt von zwei Basisfunktionen gebildet. Dieses Produkt ist nur dann ungleich Null, wenn sich die Träger der beiden Basisfunktionen überlappen. Wählt man also die Basisfunktionen so, dass die Träger sehr klein sind, hat das zur Folge, dass $a(\cdot,\cdot)$ für viele Paare von Basisfunktionen Null ist. Das ist die Idee der Finiten Elemente.

Als einleitendes Beispiel betrachten wir die Finiten Elemente im Eindimensionalen. Wir zerlegen das Gebiet $\Omega = [a,b]$ in Teilintervalle $[x_i, x_{i+1}]$ mit $i = 0,\dots,N-1$ mit den Stützstellen $a = x_0 < x_1 < \cdots < x_N = b$. Wir diskretisieren den Raum V und definieren den endlichdimensionalen Teilraum

$$V_N = \{u \in C^0([a,b]) : u|_{[x_i,x_{i+1}]} \text{ ist linear }, 0 \le i < N, u(a) = u(b) = 0\}.$$

Man beachte, dass die Randbedingungen in den Raum aufgenommen worden sind. Als Basis für V_N wählen wir hier die sogenannte Knotenbasis

$$\varphi_i(x) := \begin{cases} \frac{x-x_{i-1}}{x_i-x_{i-1}} & x_{i-1} < x \le x_i \\ \frac{x_{i+1}-x}{x_{i+1}-x_i} & x_i \le x < x_{i+1} \\ 0 & \text{sonst.} \end{cases} \tag{3.7}$$

Dies entspricht Hütchen, die in Abbildung 3.1 anschaulich dargestellt.

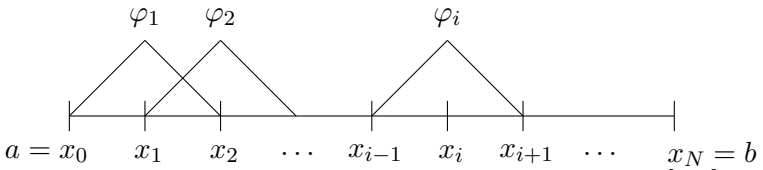


Abb. 3.1.: Knotenbasis im eindimensionalen Intervall $[a,b]$

Diese Basisfunktionen sind gerade so gewählt, dass für ihren Träger *supp*, also ihre nicht Nullstellenmenge,

$$\operatorname{supp}(\varphi_i) \cap \operatorname{supp}(\varphi_j) \begin{cases} \neq \emptyset & \text{falls } j \in \{i-1, i, i+1\} \\ = \emptyset & \text{sonst} \end{cases}$$

gilt. So erreichen wir die Dünnbesiedeltheit der Steifigkeitsmatrix L.

3.3. Diskretisierung von Modellproblemen

Wir betrachten nun die grundlegenden Komponenten Diffusion, Konvektion und Reaktion vieler physikalisch motivierter partiellen Differentialgleichungen. Diese werden wir auch im Kapitel 6 näher untersuchen. Dabei betrachten wir stets das Einheitsquadrat als zweidimensionales Gebiet, das wir auf einem uniformen Gitter mit einer quadratischen Triangulierung diskretisieren. So erhalten wir stets Diskretisierungen, die sich als Differenzensterne (siehe Abschnitt 2.3) notieren lassen. Weitere Beispiele für Finite-Elemente-Diskretisierungen finden sich in [Bra97].

Wir werden feststellen, dass wir durch Tensorproduktansätze nur noch eindimensionale Hütchen betrachten müssen, die miteinander multipliziert die zweidimensionale Darstellung ergeben.

3.3.1. Diffusion

Als erstes widmen wir uns dem Laplace im Zweidimensionalen. Nach der Anwendung des Galerkin-Verfahrens erhalten wir ein lineares Gleichungssystem (3.6). Um die Steifigkeitsmatrix L aufzustellen, müssen wir für alle Paare φ_i, φ_j die Bilinearform

$$a(\varphi_i, \varphi_j) = \int_\Omega \nabla\varphi_i \nabla\varphi_j \; dV$$

berechnen. Die zweidimensionale Basisfunktion $\varphi_\bullet(x, y)$ lässt sich als Produkt von zwei eindimensionalen Funktionen $\varphi_\bullet(x, y) = \varphi_\bullet(x)\varphi_\bullet(y)$ darstellen. Dadurch reduziert sich die Berechnung des zweidimensionalen Integrals mit Anwendung des Satzes von Fubini auf die Berechnung von eindimensionalen Integralen

$$\begin{aligned}
\int_\Omega \nabla\varphi_i \nabla\varphi_j \; dV &= \int_x \int_y \nabla(\varphi_i(x)\varphi_i(y)) \; \nabla(\varphi_j(x)\varphi_j(y)) \; dydx \\
&= \int_x \int_y \begin{pmatrix}\partial_x \\ \partial_y\end{pmatrix} (\varphi_i(x)\varphi_i(y)) \cdot \begin{pmatrix}\partial_x \\ \partial_y\end{pmatrix} (\varphi_j(x)\varphi_j(y)) \, dydx \\
&= \int_x \int_y \partial_x (\varphi_i(x)\varphi_i(y)) \, \partial_x (\varphi_j(x)\varphi_j(y)) + \partial_y (\varphi_i(x)\varphi_i(y)) \, \partial_y (\varphi_j(x)\varphi_j(y)) \, dydx \\
&= \int_x \int_y \varphi_i(y)\varphi_j(y)\partial_x\varphi_i(x)\partial_x\varphi_j(x) \; dydx + \int_x \int_y \varphi_i(x)\varphi_j(x)\partial_y\varphi_i(y)\partial_y\varphi_j(y) \; dydx \\
&= \underbrace{\int_y \varphi_i(y)\varphi_j(y)dy}_{M_y(i,j)} \underbrace{\int_x \partial_x\varphi_i(x)\partial_x\varphi_j(x)dx}_{D_x(i,j)} + \underbrace{\int_x \varphi_i(x)\varphi_j(x)dx}_{M_x(i,j)} \underbrace{\int_y \partial_y\varphi_i(y)\partial_y\varphi_j(y)dy}_{D_y(i,j)}.
\end{aligned} \tag{3.8}$$

Im letzten Schritt wurden die Integrale auseinandergezogen, so dass die Basisfunktionen mit den entsprechenden Integralen der Richtungen zusammenstehen. Die Integrale $M_\bullet$ und $D_\bullet$ sind nun nur noch eindimensional, d.h. M_x und M_y bzw. D_x und D_y sind äquivalent. Ohne Beschränkung der Allgemeinheit sei im folgenden $M = M_x$ und $D = D_x$.

Für alle Fälle außer $i = j$ und $i = j \pm 1$ überdecken sich die Träger der Funktionen nicht, folglich gilt für diese Fälle $M(i, j) = 0$. Betrachten wir nun die übrigen Fälle.

Wir beginnen mit dem Fall $i = j$ (siehe Abbildung 3.2), bei dem die beiden Hütchen deckungsgleich sind. Dazu betrachten wir die erste Hälfte der Hütchen im Intervall $[0, h]$ und integrieren das Produkt der Hütchen i und j in diesem Intervall

$$\begin{aligned}
\int_0^h \frac{1}{h^2} x^2 dx &= \frac{1}{3h^2} x^3 \Big|_0^h \\
&= \frac{1}{3h^2} h^3 - 0 \\
&= \frac{1}{3} h.
\end{aligned}$$

Analog verfahren wir im Intervall $[h, 2h]$ und erhalten das gleiche Ergebnis. Daraus folgt, dass die Fläche unter dem Produkt der beiden Hütchen

$$M(i, j) = \frac{2}{3} h \quad \text{für } i = j$$

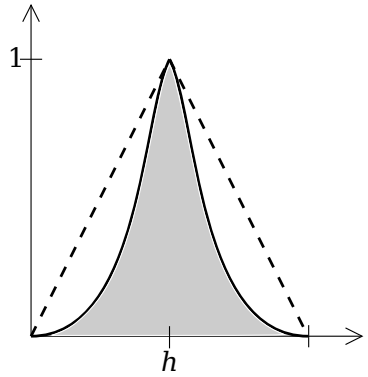


Abb. 3.2.: M mit $i = j$

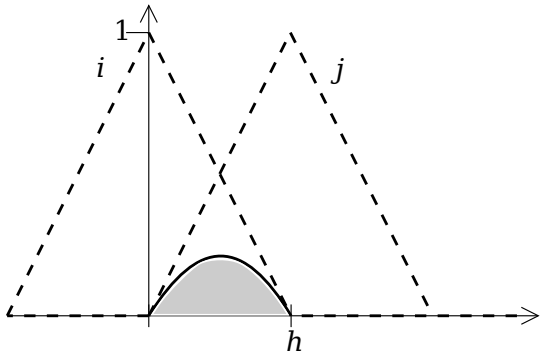


Abb. 3.3.: M mit $i = j \pm 1$

ist.

Wir betrachten nun den Fall $i = j \pm 1$ (siehe Abbildung 3.3). Die benachbarten Hütchen i und j überlappen sich im Intervall $[0, h]$, also integrieren wir dort das Produkt der beiden Hütchen.

$$\begin{aligned} M(i,j) = \int_0^h -\frac{1}{h^2}x^2 + \frac{1}{h}x \; dx &= -\frac{1}{3h^2}x^3 + \frac{1}{2h}x^2\Big|_0^h \\ &= -\frac{1}{3h^2}h^3 + \frac{1}{2h}h^2 \\ &= -\frac{2}{6} + \frac{3}{6}h \\ &= \frac{1}{6}h \quad \text{für } i = j \pm 1. \end{aligned}$$

Damit lässt sich M als eindimensionaler Stern

$$M = \frac{1}{6}h \begin{bmatrix} 1 & 4 & 1 \end{bmatrix} \tag{3.9}$$

schreiben.

Wir berechnen nun das Integral D in den beiden Fällen $i = j$ und $i = j \pm 1$. Für alle übrigen Fälle gilt $D = 0$, da sich die Träger der Funktionen nicht überlappen.

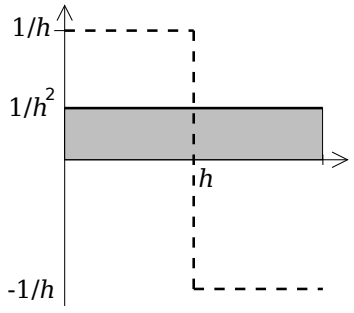


Abb. 3.4.: D mit $i = j$

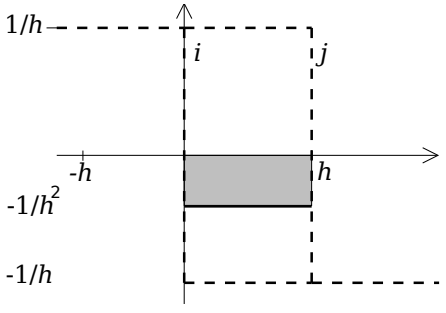


Abb. 3.5.: D mit $i = j \pm 1$

Für den Fall $i = j$ (siehe Abbildung 3.4) ist die Fläche unter dem Produkt der Ableitung der Hütchen

$$D(i,j) = 2h \cdot \frac{1}{h^2} = 2\frac{1}{h} \quad \text{für } i = j.$$

Für den Fall $i = j \pm 1$ ist die Fläche unter dem Produkt der Ableitung der Hütchen

$$D(i,j) = h \cdot - \left(\frac{1}{h^2}\right) = -\frac{1}{h} \quad \text{für } i = j \pm 1.$$

Damit lässt sich D als eindimensionaler Stern

$$D = \frac{1}{h}\begin{bmatrix} -1 & 2 & -1 \end{bmatrix} \tag{3.10}$$

schreiben.

Setzt man die Ergebnisse in (3.8) ein, so erhält man

$$\int_\Omega \nabla\varphi_i \nabla\varphi_j dV = \begin{cases} 2\left(\frac{2}{3}h \cdot 2\frac{1}{h}\right) = \frac{8}{3} & \text{für } i = j \\ 2\left(\frac{1}{6}h \cdot (-\frac{1}{h})\right) = -\frac{1}{3} & \text{für } i = j \pm 1 \\ 0 & \text{sonst} \end{cases}.$$

und kann damit für die Massenmatrix der Diskretisierung des zweidimensionalen Laplace den Stern

$$L = \frac{1}{3}\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

ablesen.

Diskretisiert man nun den Laplace auf dem Einheitsquadrat mit einen 3×3-Gitter, so ergibt sich beispielsweise die Steifigkeitsmatrix

$$L = \frac{1}{3}\begin{pmatrix} 8 & -1 & & -1 & -1 & & & & \\ -1 & 8 & -1 & -1 & -1 & -1 & & & \\ & -1 & 8 & & -1 & -1 & & & \\ -1 & -1 & & 8 & -1 & & -1 & -1 & \\ -1 & -1 & -1 & -1 & 8 & -1 & -1 & -1 & -1 \\ & -1 & -1 & & -1 & 8 & & -1 & -1 \\ & & & -1 & -1 & & 8 & -1 & \\ & & & -1 & -1 & -1 & -1 & 8 & -1 \\ & & & & -1 & -1 & & -1 & 8 \end{pmatrix}. \tag{3.11}$$

3.3.2. Anisotrope Diffusion

Betrachten wir nun den anisotropen Laplace. Zunächst multiplizieren wir die Laplace-Gleichung in einer Richtung

$$-\frac{d^2}{dx^2}u = f$$

mit einer beliebigen Testfunktion v, integrieren und stellen mit partieller Integration die schwache Form auf

$$\begin{aligned} -\int_\Omega \frac{d^2}{dx^2}uv \; dV &= \int_\Omega fv \; dV \quad \forall v \\ \Leftrightarrow \quad \underbrace{-\partial_x u \partial_x v\Big|_\Gamma}_{=0} + \underbrace{\int_\Omega \partial_x u \partial_x v \; dV}_{=:a(u,v)} &= \underbrace{\int_\Omega fv \; dV}_{=:l(v)} \quad \forall v \end{aligned}$$

Wir diskretisieren $V \to V_N$ und wählen die Basis $\{\varphi_i\}_{i=1\dots N}$ für V_N analog zu (3.5). Wir betrachten die Bilinearform

$$a(\varphi_i, \varphi_j) = \int_\Omega \partial_x \varphi_i \partial_x \varphi_j \; dV.$$

Mit $\varphi_i(x,y) = \varphi_i(x)\varphi_i(y)$ gilt

$$\begin{aligned} a(\varphi_i, \varphi_j) &= \int_y \int_x \partial_x\big(\varphi_i(x)\varphi_i(y)\big)\partial_x\big(\varphi_j(x)\varphi_j(y)\big) \; dxdy \\ &= \int_y \int_x \varphi_i(y)\varphi_j(y)\partial_x\varphi_i(x)\partial_x\varphi_j(x) \; dxdy \\ &= \underbrace{\int_y \varphi_i(y)\varphi_j(y) \; dy}_{M_y(i,j)} \underbrace{\int_x \partial_x\varphi_i(x)\partial_x\varphi_j(x) \; dx}_{D_x(i,j)}. \end{aligned}$$

Daraus folgt für $-\frac{d^2}{dx^2}u = f$ der Stern

$$L_x = \frac{1}{6}h \begin{bmatrix} 1 \\ 4 \\ 1 \end{bmatrix} \cdot \frac{1}{h} \begin{bmatrix} -1 & 2 & -1 \end{bmatrix} \quad = \quad \frac{1}{6} \begin{bmatrix} -1 & 2 & -1 \\ -4 & 8 & -4 \\ -1 & 2 & -1 \end{bmatrix} \quad = \quad \frac{1}{3} \begin{bmatrix} -\frac{1}{2} & 1 & -\frac{1}{2} \\ -2 & 4 & -2 \\ -\frac{1}{2} & 1 & -\frac{1}{2} \end{bmatrix}.$$

Analog lässt sich für $-\frac{d^2}{dy^2}u = f$ der Stern

$$L_y = \frac{1}{3} \begin{bmatrix} -\frac{1}{2} & -2 & -\frac{1}{2} \\ 1 & 4 & 1 \\ -\frac{1}{2} & -2 & -\frac{1}{2} \end{bmatrix}$$

berechnen.

Damit lautet die Sternnotation für den anisotropen Laplace $-(\alpha u_{xx} + \beta u_{yy}) = f$

$$\alpha L_x + \beta L_y = \alpha \cdot \frac{1}{3} \begin{bmatrix} -\frac{1}{2} & 1 & -\frac{1}{2} \\ -2 & 4 & -2 \\ -\frac{1}{2} & 1 & -\frac{1}{2} \end{bmatrix} + \beta \cdot \frac{1}{3} \begin{bmatrix} -\frac{1}{2} & -2 & -\frac{1}{2} \\ 1 & 4 & 1 \\ -\frac{1}{2} & -2 & -\frac{1}{2} \end{bmatrix}.$$

3.3.3. Reaktion

Nun betrachten wir die Reaktion

$$u = f,$$

multiplizieren sie mit einer beliebigen Testfunktion v und integrieren die Gleichung

$$\underbrace{\int_\Omega uv \; dV}_{=:a(u,v)} = \underbrace{\int_\Omega fv \; dV}_{=:l(v)}.$$

Wir diskretisieren $V \to V_N$ und wählen die Basis $\{\varphi_i\}_{i=1\dots N}$ für V_N analog zu (3.5). Wir betrachten nun die Bilinearform

$$a(\varphi_i, \varphi_j) = \int_\Omega \varphi_i \varphi_j \; dV.$$

Mit $\varphi_i(x,y) = \varphi_i(x)\varphi_i(y)$ gilt

$$\begin{aligned} a(\varphi_i,\varphi_j) &= \int_y \int_x \varphi_i(x)\varphi_i(y)\varphi_j(x)\varphi_j(y)\ dxdy \\ &= \underbrace{\int_y \varphi_i(y)\varphi_j(y)\ dy}_{M_y(i,j)} \underbrace{\int_x \varphi_i(x)\varphi_j(x)\ dx}_{M_x(i,j)}. \end{aligned}$$

Daraus folgt für die Reaktion der Stern

$$Id = \frac{1}{6}h \begin{bmatrix}1\\4\\1\end{bmatrix} \frac{1}{6}h \begin{bmatrix}1 & 4 & 1\end{bmatrix} = \frac{1}{36}h^2 \begin{bmatrix}1 & 4 & 1\\4 & 16 & 4\\1 & 4 & 1\end{bmatrix}.$$

3.3.4. Konvektion

Nun betrachten wir die Konvektionsgleichung

$$b \cdot \nabla u = f,$$

mit der festen Strömungsrichtung $b \in \mathbb{R}^2$, multiplizieren sie mit einer beliebigen Testfunktion v und integrieren die Gleichung

$$\underbrace{\int_\Omega b \cdot \nabla u v\ dV}_{=:a(u,v)} = \underbrace{\int_\Omega f v\ dV}_{=:l(v)}.$$

Analog zu (3.5) diskretisieren wir $V \to V_N$ und wählen die Basis $\{\varphi_i\}_{i=1\dots N}$ für V_N . Wir betrachten nun die Bilinearform

$$a(\varphi_i,\varphi_j) = \int_\Omega b \cdot \nabla \varphi_i \varphi_j\ dV.$$

Mit $\varphi_i(x,y) = \varphi_i(x)\varphi_i(y)$ gilt

$$\begin{aligned} a(\varphi_i,\varphi_j) &= \int_y \int_x \begin{pmatrix}b_x\\b_y\end{pmatrix} \cdot \begin{pmatrix}\partial_x\varphi_i(x)\varphi_i(y)\\ \partial_y\varphi_i(x)\varphi_i(y)\end{pmatrix} \varphi_j(x)\varphi_j(y)\ dxdy \\ &= \int_y \int_x \Big(b_x \partial_x\varphi_i(x)\varphi_i(y) + b_y\partial_y\varphi_i(x)\varphi_i(y)\Big)\varphi_j(x)\varphi_j(y)\ dxdy \\ &= \int_y \int_x b_x \left(\partial_x\varphi_i(x)\right)\varphi_i(y)\varphi_j(x)\varphi_j(y) + b_y\varphi_i(x)\left(\partial_y\varphi_i(y)\right)\varphi_j(x)\varphi_j(y)\ dxdy \\ &= \int_y \int_x b_x \left(\partial_x\varphi_i(x)\right)\varphi_i(y)\varphi_j(x)\varphi_j(y)\ dxdy + \int_x \int_y b_y\varphi_i(x)\left(\partial_y\varphi_i(y)\right)\varphi_j(x)\varphi_j(y)\ dydx \\ &= b_x \underbrace{\int_y \varphi_i(y)\varphi_j(y)\ dy}_{M_y} \underbrace{\int_x \varphi_j(x)\partial_x\varphi_i(x)\ dx}_{K_x} + b_y \underbrace{\int_x \varphi_i(x)\varphi_j(y)\ dx}_{M_x} \underbrace{\int_y \varphi_j(y)\partial_y\varphi_i(y)\ dy}_{K_y}. \end{aligned}$$

Die Integrale $M_\bullet$ und $K_\bullet$ sind nur noch eindimensional, $M_\bullet$ entspricht dem in (3.9) berechneten Integral.

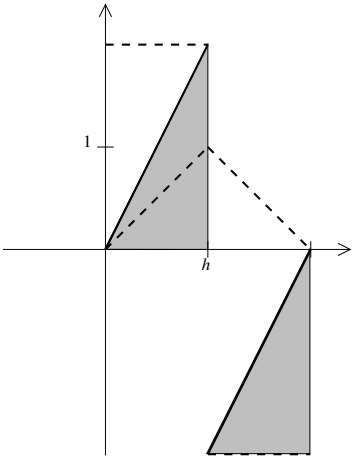


Abb. 3.6.: K mit $i = j$

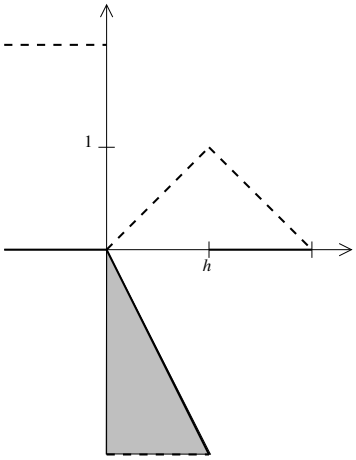


Abb. 3.7.: K mit $i = j + 1$

Wir berechnen nun K_x für die Fälle $i = j$, $i = j - 1$ und $i = j + 1$. Für alle übrigen Fälle ist $K_x = 0$, da sich die Träger der Funktionen nicht überlappen.

Wir betrachten den Fall $i = j$ (siehe Abbildung 3.6) und integrieren im Intervall $[0, h]$

$$
\begin{aligned}
K(i,j) = & \\
\int_0^h \frac{1}{h} \cdot \frac{1}{h} x \; dx + \int_h^{2h} -\frac{1}{h}\Big(-\frac{1}{h}x + 2\Big) \; dx &= \frac{1}{2}\frac{1}{h^2}x^2\Big|_0^h \; + \; \frac{1}{2}\frac{1}{h^2}x^2 - \frac{2}{h}x\Big|_h^{2h} \\
&= \frac{1}{2} + \left(\left(\frac{1}{2}\frac{1}{h^2}4h^2 - \frac{2}{h}2h\right) - \left(\frac{1}{2}\frac{1}{h^2}h^2 - \frac{2}{h}h\right)\right) \\
&= \frac{1}{2} - \frac{1}{2} = 0 \quad \text{für } i = j.
\end{aligned}
$$

Im Fall $i = j + 1$ (siehe Abbildung 3.7) integrieren wir ebenfalls im Intervall $[0, h]$

$$
\begin{aligned}
K(i,j) = \int_0^h -\frac{1}{h^2} x \; dx &= -\frac{1}{2}\frac{1}{h^2}x^2\Big|_0^h \\
&= -\frac{1}{2}\frac{1}{h^2}h^2 \\
&= -\frac{1}{2} \quad \text{für } i = j + 1
\end{aligned}
$$

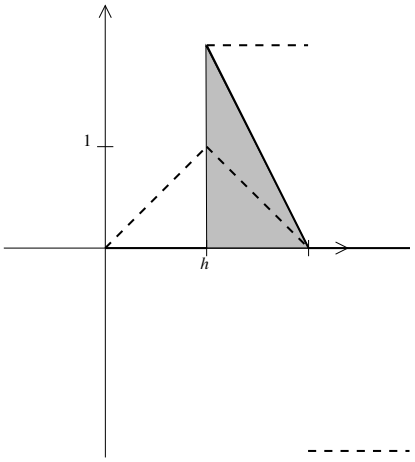


Abb. 3.8.: K mit $i = j - 1$

Im Fall $i = j - 1$ (siehe Abbildung 3.8) integrieren wir im Intervall $[h, 2h]$

$$\begin{aligned} K(i,j) = \int_h^{2h} \frac{1}{h}\left(-\frac{1}{h}x + 2\right)\, dx &= \int_h^{2h} -\frac{1}{h^2}x + \frac{2}{h}\, dx \\ &= -\frac{1}{h^2}\frac{1}{2}x^2 + \frac{2}{h}x\Big|_h^{2h} \\ &= \left(-\frac{1}{h^2}\frac{1}{2}4h^2 + \frac{2}{h}2h\right) - \left(-\frac{1}{h^2}\frac{1}{2}h^2 + \frac{2}{h}h\right) \\ &= 2 = \frac{3}{2} \\ &= \frac{1}{2} \quad \text{für } i = j - 1. \end{aligned}$$

Damit lassen sich K_x und K_y als eindimensionale Sterne

$$K_x = \frac{1}{2}\begin{bmatrix}-1 & 0 & 1\end{bmatrix} \quad \text{und} \quad K_y = \frac{1}{2}\begin{bmatrix}1\\0\\-1\end{bmatrix} \tag{3.12}$$

schreiben.

Setzen wir $M_\bullet$ und $K_\bullet$ in die Gleichung der Bilinearform ein, so ergibt sich

$$\begin{aligned} L &= b_x M_y K_x + b_y M_x K_y \\ &= b_x \frac{1}{6}h \begin{bmatrix}1\\4\\1\end{bmatrix} \frac{1}{2}\begin{bmatrix}-1 & 0 & 1\end{bmatrix} + b_y \frac{1}{6}h \begin{bmatrix}1 & 4 & 1\end{bmatrix} \frac{1}{2}\begin{bmatrix}1\\0\\-1\end{bmatrix} \\ &= b_x \frac{1}{3}h \begin{bmatrix}-\frac{1}{4} & 0 & \frac{1}{4}\\ -1 & 0 & 1\\ -\frac{1}{4} & 0 & \frac{1}{4}\end{bmatrix} + b_y \frac{1}{3}h \begin{bmatrix}\frac{1}{4} & 1 & \frac{1}{4}\\ 0 & 0 & 0\\ -\frac{1}{4} & -1 & -\frac{1}{4}\end{bmatrix}. \end{aligned}$$

3.4. Übersicht der Sterne

Wir haben die verschiedenen Modellprobleme diskretisiert und verschaffen uns nun eine Übersicht über die gewonnenen Ergebnisse. Die Diskretisierung durch die Methode der Finiten Elemente führt auf einem uniformen Gitter mit einer quadratischen Triangulierung für die verschiedenen zweidimensionalen Differentialoperatoren zu den Sternen in Tabelle 3.1.

2D Differentialoperator	Formel	Stern
Reaktion	Id	$\frac{1}{36}h^2 \begin{bmatrix} 1 & 4 & 1 \\ 4 & 16 & 4 \\ 1 & 4 & 1 \end{bmatrix}$
Konvektion	$\frac{\partial}{\partial x}$	$\frac{1}{3}h \begin{bmatrix} -\frac{1}{4} & 0 & \frac{1}{4} \\ -1 & 0 & 1 \\ -\frac{1}{4} & 0 & \frac{1}{4} \end{bmatrix}$
	$\frac{\partial}{\partial y}$	$\frac{1}{3}h \begin{bmatrix} \frac{1}{4} & 1 & \frac{1}{4} \\ 0 & 0 & 0 \\ -\frac{1}{4} & -1 & -\frac{1}{4} \end{bmatrix}$
anisotroper Laplace	$\frac{\partial^2}{\partial x^2}$	$\frac{1}{3} \begin{bmatrix} -\frac{1}{2} & 1 & -\frac{1}{2} \\ -2 & 4 & -2 \\ -\frac{1}{2} & 1 & -\frac{1}{2} \end{bmatrix}$
	$\frac{\partial^2}{\partial y^2}$	$\frac{1}{3} \begin{bmatrix} -\frac{1}{2} & -2 & -\frac{1}{2} \\ 1 & 4 & 1 \\ -\frac{1}{2} & -2 & -\frac{1}{2} \end{bmatrix}$
Laplace	$\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$	$\frac{1}{3} \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$

Tabelle 3.1.: Differentialoperatoren und ihre Sterne

Wir haben die Finiten Elemente mit variabler Maschenweite betrachtet und mittels dieser Methode einige Modellprobleme diskretisiert. Mit den gewonnenen Sternen können wir leicht Steifigkeitsmatrizen für verschieden feine Gitter konstruieren und somit die entsprechenden Gleichungssysteme aufstellen.

Im folgenden Kapitel führen wir das Multilevelverfahren ein. Mit diesem Verfahren schachteln wir die Basen von verschieden feinen Gittern ineinander und stellen mit diesem Erzeugendensystem ein erweitertes lineares Gleichungssystem auf. Das Ziel dieser Methode ist es, mit einem iterativen Verfahren über dem erweiterten Gleichungssystem Stabilität, also die Unabhängigkeit der Konvergenzrate von der Maschenweite, zu erreichen.

4. Multilevelmethoden

Diskretisiert man eine partielle Differentialgleichung, so spielt die gewählte Maschenweite eine entscheidende Rolle. Je feiner man das Gitter wählt, desto kleiner ist.

Löst man das resultierende Gleichungssystem mit einem klassischen Iterationsverfahren, stellt man fest, dass auch die Konvergenzgeschwindigkeit mit der Maschenweite der Diskretisierung zusammenhängt. Je feiner das Gitter, desto schlechter ist die Konvergenzrate des Iterationsverfahrens.

Multilevelmethoden bieten eine Möglichkeit, diesem Dilemma zu begegnen, da sie stabil sind, d.h. ein Iterationsverfahren über einem Multilevelsystem konvergiert unabhängig von der Maschenweite der Diskretisierung. Wir geben hier nur einen groben Überblick über dieses Verfahren, ausführliche Betrachtungen der Multilevelmethoden finden sich in [Gri94].

4.1. Mehrgitterverfahren

Zunächst erörtern wir die Idee des Mehrgitterverfahrens, die uns einen Zugang zu den Multilevelverfahren liefern wird. Man beobachtet bei klassischen Iterationsverfahren, dass der Fehler auf dem diskreten Gebiet nach wenigen Iterationsschritten geglättet ist. Danach sind allerdings viele Schritte nötig, um den Gesamtfehler genügend zu reduzieren (siehe Abbildung 4.1).

Generell gilt, dass sich die exakte Lösung eines Gleichungssystems als Summe aus der gewonnenen Näherung und dem zugehörigen Iterationsfehler ergibt. Ließe sich dieser Fehler bestimmen, so wäre das Problem gelöst. Die Idee des Mehrgitterverfahrens ist es nun, das klassische Iterationsverfahren nur als Glätter zu verwenden, d.h. mit diesem nur wenige Iterationsschritte durchzuführen. Anschließend ist der Fehler zwar noch groß aber glatt. Das Ziel ist es nun, diesen Fehler zu approximieren und als Korrekturterm auf die Iterierte zu addieren. Da er glatt ist, lässt sich eine solche Näherung kostengünstig auf einem gröberen Gitter durchführen. Die so gewonnene Approximation des Fehlers wird dann auf das feine Gitter interpoliert und zur aktuellen Iterieren addiert.

Die Glättung auf dem feinen Gitter und die Grobgitterkorrektur werden abgewechselt, bis der Fehler die gewünschte Größe erreicht hat. Die rekursive Anwendung dieser Schritte auf eine Hierarchie von Gittern liefert das Mehrgitterverfahren. Im Mehrgitterverfahren werden für die Reihenfolge, in der Glättungs- und Grobgitterkorrekturschritte durchgeführt werden, verschiedene Schemata verwendet. Als Beispiel sei hier der V-Zyklus genannt, der beginnend mit der Glättung auf dem feinsten Gitter bis zum gröbsten Gitter herabsteigt und dann wieder Gitter für Gitter bis zum feinsten aufsteigt und dabei auf jedem Gitter einen Glättungsschritt durchführt.

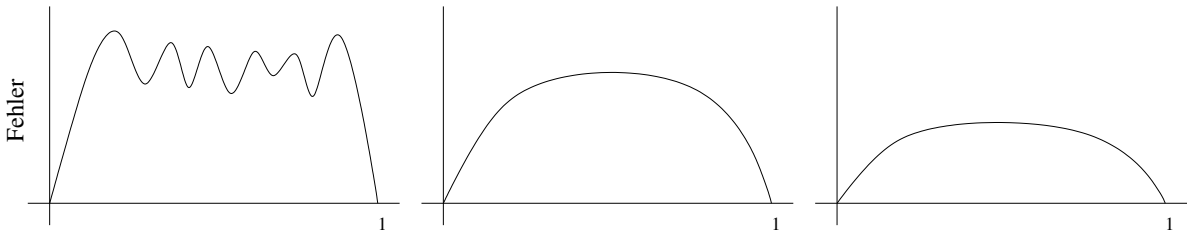


Abb. 4.1.: Der Fehler in einem ein-dimensionalen Gebiet nach 0, 5 und 100 Iterationen eines klassischen Iterationsverfahrens

4.2. Herleitung des Multilevelverfahrens

Im Multilevelverfahren werden ebenfalls verschiedene Maschenweiten betrachtet. Hier wird jedoch nicht für jedes Gitter ein eigenes Gleichungssystem aufgestellt, sondern alle Gitter werden gemeinsam in einem großen System dargestellt.

Das Mehrgitterverfahren bietet uns nicht nur einen Einstieg in das Multilevelverfahren, auch werden wir sehen, dass sich ein klassisches Iterationsverfahren über einem erweiterten linearen Gleichungssystem als Mehrgitterverfahren mit V-Zyklus interpretieren lässt.

4.2.1. Die Idee

Wir haben in Kapitel 3 das Galerkin-Verfahren behandelt, das ein festes Gebiet Ω mit einer Triangulierung $\mathcal{T}_h$ diskretisiert, mit V_N als Ansatzraum der stückweise d-linearen Funktionen aus dem $\mathbb{R}^d$ mit der Basis B_N, die aus den Hütchenfunktionen $\{b_i\}, i = 1, \ldots, N$ besteht.

Dieses Verfahren wollen wir nun erweitern, indem wir nicht ein Gitter, sondern eine Sequenz von Gittern betrachten.

Sei $\mathcal{T}_k$ eine Sequenz von Gittern mit

$$\mathcal{T}_1 \subset \mathcal{T}_2 \subset \cdots \subset \mathcal{T}_k$$

mit den Gitterweiten $h = 2^{-l}, l = 1, \ldots, k$ und den zugehörigen Teilräumen

$$V_1 \subset V_2 \subset \cdots \subset V_k$$

der Dimension $n_1, \ldots, n_k$. Wir wählen zu jedem Teilraum eine Basis

$$B_1 = \{\varphi_i^{B_1}\}_{i=1}^{n_1},\ B_2 = \{\varphi_i^{B_2}\}_{i=1}^{n_2},\ \ldots,\ B_k = \{\varphi_i^{B_k}\}_{i=1}^{n_k},$$

bestehend aus den Hütchenfunktionen $\{\varphi_i^{B_l}\}, i = 1, \ldots, n_k,\ l = 1, \ldots, k$.

Die Diskretisierung der Bilinearform $a(u, v) = f(v)$ lässt sich auf jedem Gitter l mit der Steifigkeitsmatrix A^{B_l} mit

$$(A^{B_l})_{i,j} = a(\varphi_i^{B_l}, \varphi_j^{B_l}),$$

der rechten Seite f^{B_l} mit

$$(f^{B_l})_j = f(\varphi_j^{B_l})$$

und der Funktion $u^{B_l} \in V_l$ mit

$$u^{B_l} = \sum_{i=1}^{n_l} u_i^{B_l} \varphi_i^{B_l}$$

darstellen.

4.2.2. Das Erzeugendensystem

Wir definieren ein Erzeugendensystem für V_k mit

$$E_k := \bigcup_{l=1}^{k} B_l.$$

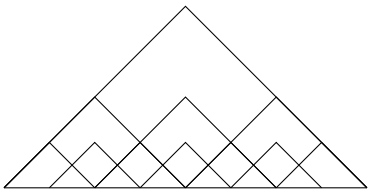


Abb. 4.2.: Basen des Erzeugendensystems

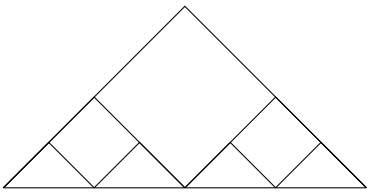


Abb. 4.3.: Ausgedünnte Basen

Die Elemente des Erzeugendensystems sind offensichtlich nicht linear unabhängig voneinander (siehe Abbildung 4.2), da die Räume ineinander geschachtelt sind. Um Eindeutigkeit zu erreichen, müssen wir die Basen ausdünnen (siehe Abbildung 4.3). Dazu nutzen wir die Prolongation [Gri90], die es ermöglicht, ein Element aus dem kleineren Raum in einem größeren Raum darzustellen.

Definition 4.1 (Prolongation, Restriktion)**.** *Sei $1 \leq l \leq k$ fest, mit den Stützstellen x_i^l, $1 \leq i \leq n_l$ auf dem Level l. Wir definieren die sogenannte Prolongation, $P_{l-1}^l \in \mathbb{R}^{n_l \times n_{l-1}}$ durch*

$$\left(P_{l-1}^l\right)_{ij} := \varphi_j^{B_{l-1}}\left(x_i^l\right).$$

Die Restriktion $R_l^{l-1} \in \mathbb{R}^{n_{l-1} \times n_l}$ ist definiert durch

$$R_l^{l-1} := \left(P_{l-1}^l\right)^T.$$

Für $l > n+1$ definieren wir

$$P_n^l := P_{l-1}^l \cdot P_{l-2}^{l-1} \cdot \ldots \cdot P_n^{n+1}$$
$$R_l^n := \left(P_n^l\right)^T.$$

Für den trivialen Fall gilt

$$P_l^l := I_l \text{ bzw. } R_l^l := I_l,$$

wobei I_l die Identität mit der Dimension n_l ist.

Haben wir eine kontinuierliche Funktion u in der diskretisierten Darstellung $u^{B_l} \in V_l$ mit

$$u^{B_l} = \sum_{i=1}^{n_l} u_i^{B_l} \varphi_i^{B_l}$$

gegeben, so lässt sich diese Funktion mit der Prolongation direkt im nächst feineren Raum durch

$$u^{B_{l+1}} = P_l^{l+1} u^{B_l}$$

darstellen.

Für den nächsten Schritt benötigen wir einen Einbettungsoperator, da wir Prolongationen verschiedenen Levels, folglich auch verschiedener Dimension, gemeinsam in eine Matrix einbetten werden.

Definition 4.2 (Einbettungsoperator)**.** *Wir definieren den Einbettungsoperator Λ_l^k mit $l \leq k$derart, dass ein Vektor u^{B_l} mit Dimension n_l in einen Vektor mit der Dimension des Multilevelvektors*

$$n_k^E := \sum_{i=1}^{k} n_i$$

transformiert wird und alle Einträge, die nicht dem Level l entsprechen, mit Null aufgefüllt werden

$$\Lambda_l^k : \mathbb{R}^{n_l} \to \mathbb{R}^{n_k^E}, \quad \Lambda_l^k \in \mathbb{R}^{n_k^E \times n_l}$$
$$\left(\Lambda_l^k\right)_{ij} := \begin{cases} 1, & \textit{falls } j = i - \left(\sum_{m=1}^{l-1} n_m\right) \\ 0, & \textit{sonst.} \end{cases}$$

Die Rücktransformation $\left(\Lambda_l^k\right)^T : \mathbb{R}^{n_k^E} \to \mathbb{R}^{n_l}$ schneidet den Vektor mit dem Level l aus dem Multilevelvektor aus.

Mit Hilfe der Prolongation erhalten wir auch auf einfache Weise eine Transformation der Darstellung $u^{E_k} \in V_k$ von u im Erzeugendensystem E_k in die eindeutige Darstellung u^{B_k} bezüglich der Basis B_k. Diese lineare, surjektive Transformation bezeichnen wir im Folgenden mit $S^{E_k B_k} : \mathbb{R}^{n_k^E} \to \mathbb{R}^{n_k}$, die die Gestalt

$$S^{E_k B_k} := \sum_{l=1}^{k} P_l^k {\Lambda_l^k}^T = \left[P_1^k, P_2^k, \ldots, P_{k-1}^k, I_k\right] \tag{4.1}$$

hat. Für ${S^{E_k B_k}}^T : \mathbb{R}^{n_k} \to \mathbb{R}^{n_k^E}$ gilt

$${S^{E_k B_k}}^T := \begin{bmatrix} {P_1^k}^T \\ {P_2^k}^T \\ \vdots \\ {P_{k-1}^k}^T \\ I_k \end{bmatrix} = \begin{bmatrix} R_k^1 \\ R_k^2 \\ \vdots \\ R_k^{k-1} \\ I_k \end{bmatrix}.$$

Wir betrachten nun einen eindimensionalen Raum mit homogenen Dirichlet-Randbedingungen. Wenn wir für dieses Problem eine Galerkin-Diskretisierung mittels E_k durchführen, ergibt sich die Suche nach dem Lösungsvektor u^{E_k} durch

$$a(u^{E_k}, \varphi_i) = (f, \varphi_i) \quad \forall\, \varphi_i \in E_k$$

und somit das erweiterte lineare Gleichungssystem

$$A^{E_k} u^{E_k} = f^{E_k}, \tag{4.2}$$
$$(A^{E_k})_{ij} := a(\varphi_i, \varphi_j)$$

und

$$(f^{E_k})_j := a(f, \varphi_j)$$

mit $\varphi_i, \varphi_j \in E_k$, $i, j = 1, \ldots, n^{E_k}$.

Man beachte, dass A^{E_k} semidefinit ist,

$$\operatorname{rang}(A^{E_k}) = \operatorname{rang}(A^{B_k})$$

und

$$\operatorname{rang}(A^{E_k}|f^{E_k}) = \operatorname{rang}(A^{E_k}).$$

Daraus folgt, dass das Gleichungssystem lösbar ist, aber unendlich viele Lösungen existieren. Es genügt aber mittels eines iterativen Verfahrens eine beliebige Lösung u^{E_k} zu berechnen, da dann durch

$$u^{B_k} = S^{E_k B_k} u^{E_k}$$

die eindeutige Lösung des ursprünglichen Problems gegeben ist. Der Vektor u^{E_k} hat folglich eine Gestalt, in der verschiedene Bereiche, die zu den im Multilevelsystem vereinigten Leveln korrespondieren und jeweils einen Anteil zur eindeutigen Lösung u^{B_k} liefern.

Das erweiterte lineare Gleichungssystem (4.2) lässt sich auch durch

$$S^{E_k B_k}{}^T A^{B_k} S^{E_k B_k} u^{E_k} = S^{E_k B_k}{}^T f^{B_k}$$

darstellen. Wir haben somit einen symmetrischen Vorkonditionierer für

$$A^{B_k} u^{B_k} = f^{B_k}$$

erhalten.

Wir betrachten nun die Gestalt des erweiterten Gleichungssystems am Beispiel einer Drei-Level-Zerlegung. Die Matrix A^{E_3} wird durch die Multiplikation mit der Transformation $S^{E_3 B_3}{}^T$ von links und $S^{E_3 B_3}$ von rechts gebildet.

$$\begin{aligned}
A^{E_3} :=& S^{E_3 B_3}{}^T A^{B_3} S^{E_3 B_3} \\
=& \begin{bmatrix} R_3^1 \\ R_3^2 \\ I_3 \end{bmatrix} A^{B_3} \begin{bmatrix} P_1^3, P_2^3, I_3 \end{bmatrix} \\
=& \begin{pmatrix} \underbrace{R_3^1 A^{B_3} P_1^3}_{=A^{B_1}} & R_3^1 A_{B_3} P_2^3 & R_3^1 A_{B_3} \\ R_3^2 A^{B_3} P_1^3 & \underbrace{R_3^2 A^{B_3} P_2^3}_{=A^{B_2}} & R_3^2 A^{B_3} \\ A^{B_3} P_1^3 & A^{B_3} P_2^3 & A^{B_3} \end{pmatrix}
\end{aligned}$$

Die Blöcke auf den Diagonalen sind gerade die Galerkin-Matrizen A^{B_l} auf dem Level l.

4.2.3. Beispiele für Prolongation und Restriktion

Wir haben die Prolongation und die Restriktion in Definition 4.1 eingeführt und werden diese beiden Transformationen nun mittels der linearen Interpolation, beginnend mit dem eindimensionalen Fall, konkret herleiten. Wir wollen das uniforme Gitter $\mathcal{T}_l$ mit der Maschenweite $2h$ auf das nächst feinere Gitter $\mathcal{T}_{l+1}$ mit der Maschenweite h prolongieren und interpolieren zu diesem Zweck linear zwischen den Grobgitterhütchen (Abbildung 4.4).

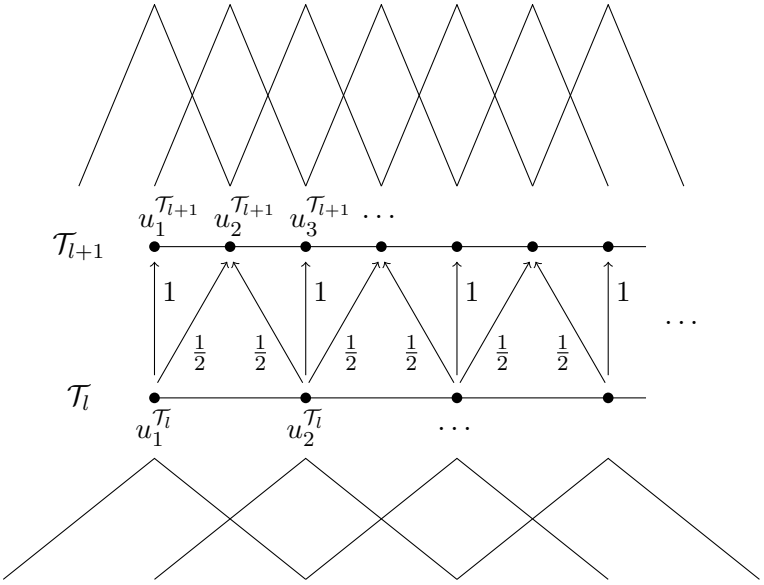


Abb. 4.4.: Prolongation im Eindimensionalen

Für den Fall, dass der zu interpolierende Punkt mit einem Grobgitterpunkt übereinstimmt, gilt

$$u_i^{\mathcal{T}_{l+1}} = u_{\frac{i+1}{2}}^{\mathcal{T}_l},$$

andernfalls liegt der zu interpolierende Punkt zwischen Grobgitterpunkten, und es gilt

$$u_i^{\mathcal{T}_{l+1}} = \frac{1}{2} u_{\frac{i}{2}}^{\mathcal{T}_l} + \frac{1}{2} u_{\frac{i}{2}+1}^{\mathcal{T}_l}.$$

Daraus lässt sich leicht die Sternnotation für die Prolongation im eindimensionalen Fall ablesen

$$P_l^{l+1} \;=\; \frac{1}{2}\begin{bmatrix}1 & 2 & 1\end{bmatrix}. \tag{4.3}$$

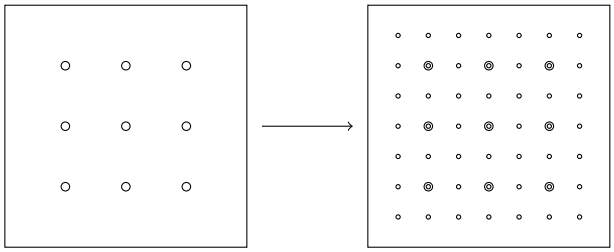
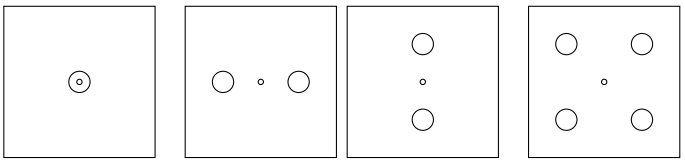


Abb. 4.5.: Grobes und nächst feineres Gitter

Will man im zweidimensionalen Fall ein grobes Gitter auf ein feines prolongieren (Abbildung 4.5), lässt sich der Prolongationsoperator aus dem Operator im Eindimensionalen errechnen, indem der eindimensionale Operator in x- mit dem in y-Richtung multipliziert wird

$$P_l^{l+1} \;=\; \frac{1}{2}\begin{bmatrix}1 & 2 & 1\end{bmatrix} \cdot \frac{1}{2}\begin{bmatrix}1\\2\\1\end{bmatrix} = \frac{1}{4}\begin{bmatrix}1 & 2 & 1\\2 & 4 & 2\\1 & 2 & 1\end{bmatrix}. \tag{4.4}$$

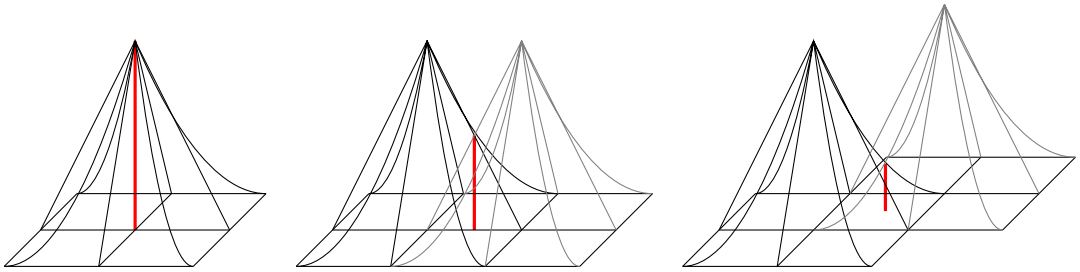
Der Prolongationsstern (4.4) bildet die vier verschiedenen Fälle von Nachbarschaften zu Grobgitterpunkten, die bei der Interpolation im zweidimensionalen Fall auftreten (siehe Abbildung 4.6), ab.



(a) (b) (c)

Abb. 4.6.: Fälle von Grobgitter-Nachbarschaften zu einem Feingitterpunkt

Anschaulich lässt sich der Stern (4.4) auch von den Hütchen im Zweidimensionalen ablesen (siehe Abbildung 4.7). Man beachte hierbei, dass die unter Abbildung 4.6 (b) aufgeführten Fälle äquivalent zueinander sind. Im Fall (c) der Abbildung 4.7 sind der Übersichtlichkeit halber nur zwei der vier sich überschneidenen Hütchen abgebildet.



(a) (b) (c)

Abb. 4.7.: Interpolation der Feingitterpunkte mittels Grobgitter-Hütchen für verschiedene Fälle von Grobgitter-Nachbarschaften.

Wenden wir uns nun der Restriktion zu. In Definition 4.1 wird die Restriktion als Transponierte der Prolongation eingeführt. Wir betrachten wie oben zwei Gitter im Eindimensionalen, das Gitter $\mathcal{T}_l$ mit der Maschenweite h und das nächst gröbere Gitter $\mathcal{T}_{l-1}$ mit der Maschenweite $2h$. An jedem Punkt des gröberen Gitters werden die Hütchen des feineren Gitters, die sich mit dem Hütchen des Grobgitterpunktes überdecken, linear interpoliert.

Für alle Punkte $u_i^{\mathcal{T}_{l-1}}$ des gröberen Gitters gilt

$$u_i^{\mathcal{T}_{l-1}} := \frac{1}{2}u_{2i-2}^{\mathcal{T}_l} + u_{2i-1}^{\mathcal{T}_l} + \frac{1}{2}u_{2i}^{\mathcal{T}_l},$$

woraus sich der Stern

$$R_l^{l-1} = \frac{1}{2}\begin{bmatrix}1\\2\\1\end{bmatrix}. \tag{4.5}$$

für die Restriktion im Eindimensionalen ablesen lässt. Für den zweidimensionalen Fall verfährt man analog zur Prolongation und erhält auch hier den Stern

$$R_l^{l-1} = \frac{1}{4}\begin{bmatrix}1 & 2 & 1\\2 & 4 & 2\\1 & 2 & 1\end{bmatrix}.$$

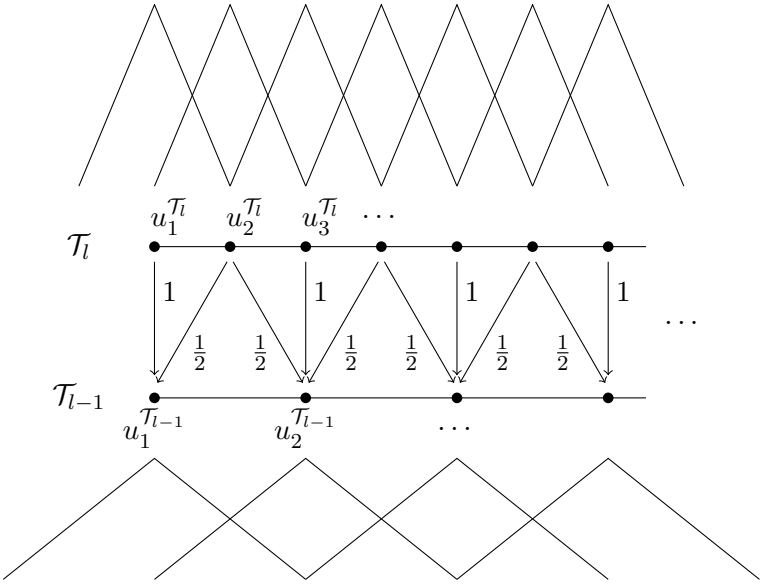


Abb. 4.8.: Restriktion im Eindimensionalen

Mit Hilfe dieser Sterne können die Matrizen für die Prolongation und die Restriktion und damit die Transformationsmatrix $S^{E_k B_k}$ aufgestellt werden, mit der wir ein lineares Gleichungssystem in ein erweitertes lineares Gleichungssystem (4.2) überführen können.

Um eine Prolongationsmatrix P_{l-1}^{l} mittels eines gegebenen Sterns aufzustellen, gehen wir wie folgt vor. Die Gitterpunkte des groben Gitters mit dem Level $l-1$ seien mit $i = 1, \ldots, n_{l-1}$ und die des feinen Gitters mit Level l mit $j = 1, \ldots, n_l$ durchnummeriert. Lege zunächst den Stern wie eine Schablone zentral über den Grobgitterpunkt i. Der Wert jedes vom Stern überdeckten Feingitterpunkts j wird nun in die Zeile j der Spalte i eingetragen. Geht man so für jeden Grobgitterpunkt i vor, erhält man die Prolongationsmatrix. Alle freien Stellen werden dabei mit Nullen aufgefüllt.

4.2.4. Beispiel Diffusion

Im Folgenden werden wir ein Beispiel für ein erweitertes Gleichungssystem betrachten. Als Problem wählen wir die Diffusion im Zweidimensionalen, die wir auf einem uniformen, quadratischen Gitter diskretisieren. Der Übersichtlichkeit halber wählen wir Level 2 als das feinste Level und damit ein 3×3 Gitter. In Abschnitt 3.3.1 haben wir die Steifigkeitsmatrix für diese Problem bereits aufgestellt

$$A^{B_2} = \frac{1}{3}\begin{pmatrix} 8 & -1 & & -1 & -1 & & & & \\ -1 & 8 & -1 & -1 & -1 & -1 & & & \\ & -1 & 8 & & -1 & -1 & & & \\ -1 & -1 & & 8 & -1 & & -1 & -1 & \\ -1 & -1 & -1 & -1 & 8 & -1 & -1 & -1 & -1 \\ & -1 & -1 & & -1 & 8 & & -1 & -1 \\ & & & -1 & -1 & & 8 & -1 & \\ & & & -1 & -1 & -1 & -1 & 8 & -1 \\ & & & & -1 & -1 & & -1 & 8 \end{pmatrix}.$$

Um die erweiterte Steifigkeitsmatrix A^{E_2} aufstellen zu können, benötigen wir die Transformation

$$S^{E_2B_2} = \begin{bmatrix} P_1^2 & I_2 \end{bmatrix}.$$

Die Prolongationsmatrix P_1^2 erhalten wir, wie oben beschrieben, aus dem Prolongationsstern (4.4)

$$P_1^2 = \begin{pmatrix} \frac{1}{4} & \frac{1}{2} & \frac{1}{4} & \frac{1}{2} & 1 & \frac{1}{2} & \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \end{pmatrix}^T,$$

somit folgt

$$S^{E_2B_2} = \begin{pmatrix} \frac{1}{4} & 1 & & & & & & & \\ \frac{1}{2} & & 1 & & & & & & \\ \frac{1}{4} & & & 1 & & & & & \\ \frac{1}{2} & & & & 1 & & & & \\ 1 & & & & & 1 & & & \\ \frac{1}{2} & & & & & & 1 & & \\ \frac{1}{4} & & & & & & & 1 & \\ \frac{1}{2} & & & & & & & & 1 \\ \frac{1}{4} & & & & & & & & \end{pmatrix}.$$

Mit $S^{E_2B_2}$ können wir die Steifigkeitsmatrix A^{B_2} in die erweiterte Steifigkeitsmatrix

$$A^{E_2} = {S^{E_2B_2}}^T \; A^{B_2} \; S^{E_2B_2}$$

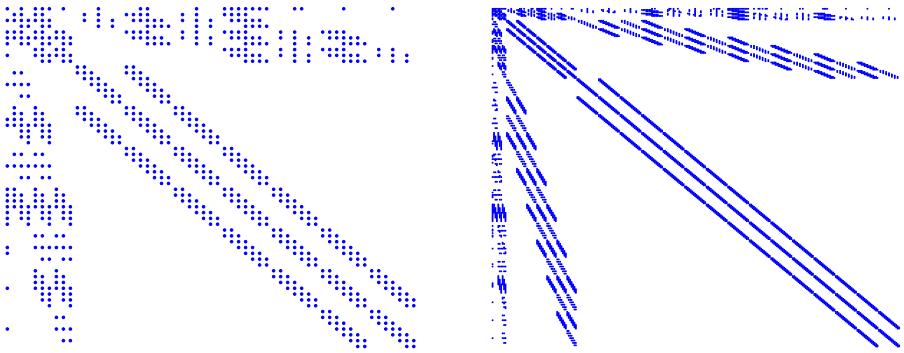
transformieren. Für unser Beispiel ergibt sich

$$A^{E_2} = \frac{1}{3} \left(\begin{array}{c|ccccccccc} 8 & 1.5 & & 1.5 & 5 & 1.5 & & 1.5 & & \\ \hline & 8 & -1 & & -1 & -1 & & & & \\ 1.5 & -1 & 8 & -1 & -1 & -1 & -1 & & & \\ & & -1 & 8 & & -1 & -1 & & & \\ 1.5 & -1 & -1 & & 8 & -1 & & -1 & -1 & \\ 5 & -1 & -1 & -1 & -1 & 8 & -1 & -1 & -1 & -1 \\ 1.5 & & -1 & -1 & & -1 & 8 & & -1 & -1 \\ & & & & -1 & -1 & & 8 & -1 & \\ 1.5 & & & & -1 & -1 & -1 & -1 & 8 & -1 \\ & & & & & -1 & -1 & & -1 & 8 \end{array}\right).$$

In diesem Beispiel lässt sich die Struktur der Multilevelmatrizen schon erahnen, doch um diese wirklich sehen zu können, müssen mehr Level hinzugenommen werden. Betrachtet man das schematische Beispiel einer erweiterten Steifigkeitsmatrix des Levels 3 oder 4 (siehe Abbildung 4.9), so kann man die Fingerstruktur, die diese Matrizen aufweisen, deutlich erkennen.

4.2.5. Konvergenz und Stabilität

Wir untersuchen nun das Konvergenzverhalten von klassischen Iterationsverfahren auf einem Erzeugendensystem. Wir rufen uns in Erinnerung, dass der Diskretisierungsfehler, der bei den Finiten Elementen entsteht, kleiner wird, wenn die Maschenweite feiner gewählt wird. Wie am Anfang dieses Kapitels beschrieben, hat die Wahl einer feinen Maschenweite allerdings auch zur Folge, dass ein Iterationsverfahren zwar den Fehler nach wenigen Schritten schnell glättet, aber viele Iterationen braucht, um diesen im Ganzen genügend zu reduzieren (siehe Abbildung 4.1).



(a) Level 3 (b) Level 4

Abb. 4.9.: Schematische Darstellung der Struktur von Multilevelmatrizen

Im Mehrgitterverfahren wird das feine Level daher vor allem zum Glätten des Fehlers verwendet, auf den gröberen Gittern wird der Fehler dann approximiert und als Korrekturterm verwendet.

Im Multilevelverfahren wird diese Idee weitergeführt. Die verschieden feinen Gitter werden in einem Erzeugendensystem vereinigt und gemeinsam in einem Gleichungssystem abgebildet. Daraus, dass nun nicht nur das feine Gitter sondern auch die gröberen Gitter einbezogen werden, resultiert, dass die Iterationsverfahren deutlich bessere Konvergenzraten in diesem erweiterten Gleichungssystem erreichen (siehe Abbildung 4.10).

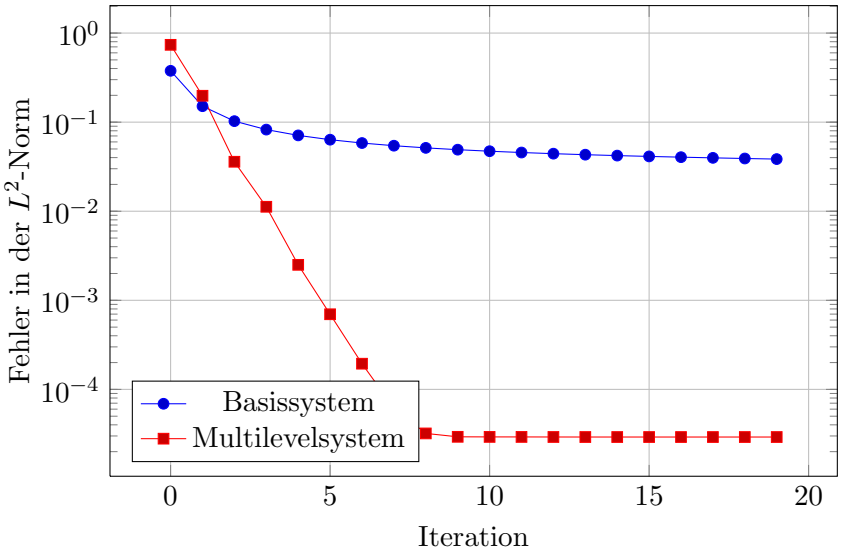


Abb. 4.10.: Vergleich des Konvergenzverhaltens des Gauß-Seidel-Verfahrens anhand des Poisson-Problems mit der Maschenweite $h = 1/64$ auf dem feinsten Gitter und 6 Leveln im Multilevelsystem

Wir betrachten in dieser Abbildung als Beispiel die Diskretisierung der Diffusion im Zweidimensionalen auf einem uniformen quadratischen Gitter im Einheitsquadrat. Zur Berechnung der rechte Seite und zur Berechnung des Fehlers wählen wir entsprechend eine Modellfunktion als bekannte Lösung. Für die genaue Beschreibung der Aufstellung des Modellproblems sei auf Abschnitt 6.1 des Ergebniskapitels verwiesen.

Wir diskretisieren das Gebiet mit einer Maschenweite von $h = 1/64$ und erhalten somit 3969

Gitterpunkte. Über das aus der Diskretisierung des feinen Gitters resultierende Gleichungssystem (im Folgenden als Basissystem bezeichnet) lassen wir das Gauß-Seidel-Verfahren iterieren und messen nach jeder Iteration den Fehler in der L^2-Norm. Dabei beobachten wir eine Konvergenzrate ρ, die bei $0,9$ liegt, das Iterationsverfahren konvergiert folglich nur sehr langsam gegen die Lösung. Wenn wir das Gleichungssystem nun in ein erweitertes Gleichungssystem transformieren und das gleiche Iterationsverfahren über das entstandene Multilevelsystem mit sechs Leveln laufen lassen, erreichen wir eine deutlich bessere Rate von $\rho < 0,29$, wie Abbildung 4.10 zeigt.

Wir haben mit dem Multilevelverfahren eine deutliche Verbesserung der Konvergenzrate erreicht, aber wie verhält sich die Konvergenzrate, wenn wir weiter verfeinern und die Levelzahl erhöhen? Betrachtet man das Konvergenzverhalten vom Gauß-Seidel-Verfahren über Basissystemen (siehe Abbildung 4.11), erkennt man, dass die Konvergenzrate mit jeder Verfeinerung schlechter wird. Im Multilevelsystem dagegen bleibt die Konvergenzrate auch auf viel feineren Leveln nahezu identisch (siehe Abbildung 4.12). Diese Beobachtung zeigt, dass die Konvergenzrate unabhängig von der Maschenweite der Diskretisierung ist.

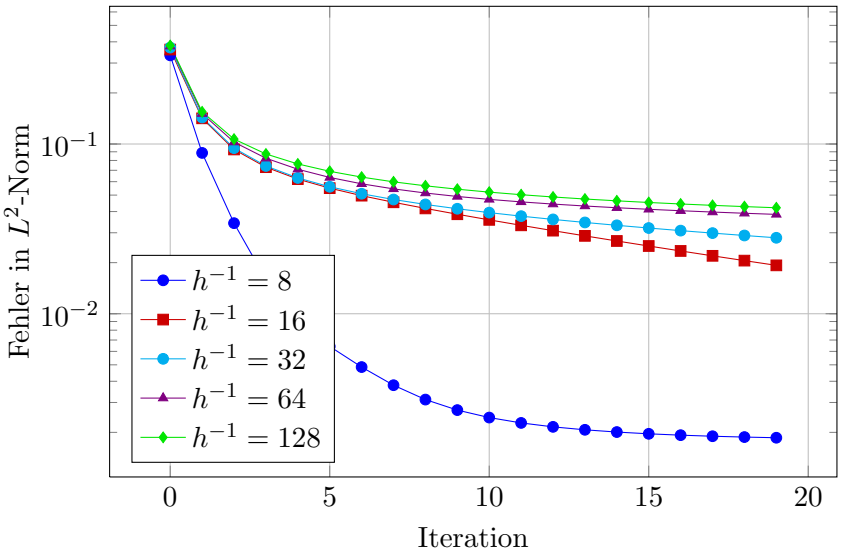


Abb. 4.11.: Konvergenzverhalten des Gauß-Seidel-Verfahrens in Basissystemen mit verschiedenen Maschenweiten h

Genaue Untersuchungen zum Konvergenzverhalten von Iterationsverfahren über Erzeugendensystemen finden sich in [Gri94] und [Osw94]. Darin wird für additive Methoden wie Jacobi, Richardson oder Gradientenverfahren und für die multiplikativen Methoden wie Gauß-Seidel oder SOR gezeigt, dass diese unabhängig von der Anzahl der Unbekannten bzw. der Anzahl der Level konvergieren.

Damit erreicht das Multilevelverfahren mit den klassischen Iterationsverfahren die Unabhängigkeit der Konvergenzrate von der Maschenweite der Diskretisierung. Das Verfahren ist *stabil*.

Wichtig ist für uns auch, dass für die multiplikativen Verfahren gilt, dass diese unabhängig von ihrer Durchlaufreihenfolge eine gemeinsame obere Schranke der Konvergenzraten haben.

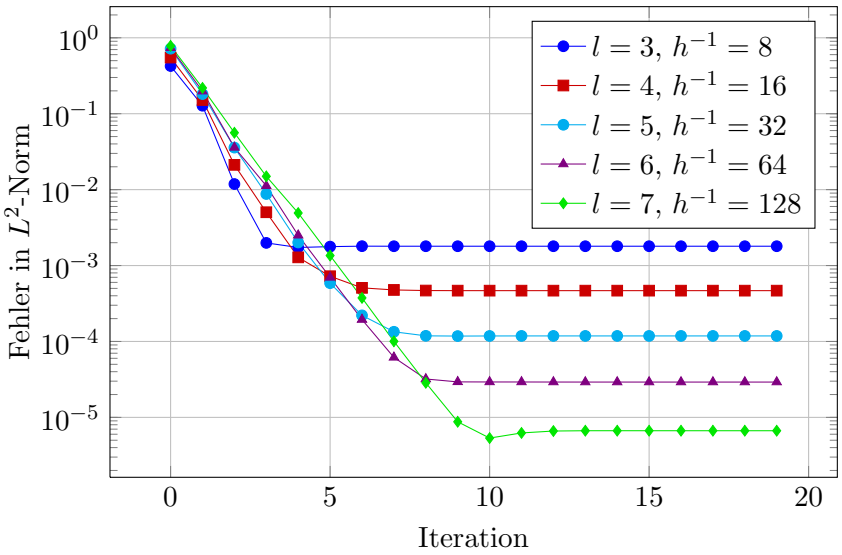


Abb. 4.12.: Konvergenzverhalten des Gauß-Seidel-Verfahrens in Multilevelsystemen mit verschiedenen Maschenweiten h und entsprechenden Leveln l

4.3. Startwertberechnung mit Nested-Iteration

In diesem Abschnitt werden wir die Möglichkeit betrachten, mit dem Mehrgitterverfahren gute Startwerte für die Lösung eines Gleichungssystems zu berechnen. Anschließend übertragen wir dieses Verfahren auf Multilevelsysteme.

Dieses als *Nested-Iteration* bezeichnete Verfahren gründet auf zwei Überlegungen zur Diskretisierung und zum Mehrgitterverfahren.

Die erste Idee ist es, für ein Iterationsverfahren über einem Gleichungssystem des Levels l die Lösung des Gleichungssystems des Levels $l-1$ als Startvektor zu verwenden. Zu diesem Zweck ließe sich die Lösung des gröberen Gitters auf das feinere prolongieren. Dabei muss beachtet werden, dass dazu keine ausiterierte Lösung nötig ist, schon eine Näherung bildet einen vernünftigen Startwert. Der bei der Prolongation entstehende Interpolationsfehler steht der Idee folglich nicht im Wege.

Die zweite Überlegung ist die, dass eine Lösung immer mit dem Diskretisierungsfehler behaftet ist. Es ist also nicht sinnvoll, die Näherungslösung weiter als bis zur Genauigkeit der Diskretisierung zu berechnen. Ist diese erreicht, bringen weitere Schritte keine Verbesserung des Gesamtfehlers mehr.

Wir skizzieren das Verfahren in Algorithmus 4. Man beachte, dass wir in diesem Algorithmus nur einen Schritt der Mehrgitteriteration MGM_l ausführen. Das ist tatsächlich ausreichend, wenn die Konvergenzrate des Verfahrens $\rho < \frac{1}{4}$ ist [Bra97]. Wenn ein Iterationsverfahren eingesetzt wird, das mehr als einen Schritt benötigt, um eine deratige Konvergenzrate zu erreichen, werden pro Iteration q Schritte durchgeführt, wobei q gemäß $\rho^q < \frac{1}{4}$ gewählt wird.

Die Genauigkeit des so gewonnenen Startwerts wird in [Bra97] wie folgt abgeschätzt, wobei u_l die diskrete Lösung auf dem Level l und der Diskretisierungsfehler von der Ordnung $\mathcal{O}(h^2)$ sei,

$$||v^l - u_l|| \quad \leq \quad \frac{5\rho}{1-4\rho}\, c \cdot h_l^2.$$

Geht man davon aus, dass eine Konvergenzrate von $\rho < \frac{1}{6}$ realistisch ist, dann liefert der

Algorithmus 4: Nested-Iteration $\mathbf{NI}_l$

Input : Matrix A_l, rechte Seite b_l, Prolongation P_{l-1}^l
Output : Startvektor v^l

if $l = 0$ **then**
  `// Löse das gröbste Level direkt`
  $v^0 = u_0 = A_0^{-1} b_0$
  return v^0
else
  `// Berechne Näherung` v^{l-1} `von` $A_{l-1}u_{l-1} = b_{l-1}$ `rekursiv mit` $\mathbf{NI}_{l-1}$
  $v^{l-1} = \mathbf{NI}_{l-1}$
  `// Prolongiere` v^{l-1}
  $v^{l,0} = P_{l-1}^l v^{l-1}$
  `// Führe einen Schritt der Mehrgitteriteration aus`
  $v^{l,1} = \mathrm{MGM}_l\left(v^{l,0}\right)$
  `// Wähle Näherung als Startwert für das Level` l
  $v^l = v^{l,1}$
  return v^l
end

Algorithmus mit einem Fehler von $\frac{5}{2}ch_l^2$ eine gute Näherung für Startwerte.

4.3.1. Nested-Iteration auf Multilevelsystemen

Das Nested-Iteration-Verfahren lässt sich von der Mehrgitteriteration analog auf das Multilevelverfahren übertragen, wie wir im Folgenden zeigen.

Will man einen guten Startwert zum Level l berechnen, stellt man das erweiterte Gleichungssystem wie gewohnt auf. Danach führt man das Iterationsverfahren aus, doch lässt man dieses nicht wie gewohnt auf dem gesamten Gleichungssystem operieren, sondern modifiziert das Verfahren wie folgt. Wir legen ein Fenster über das erweiterte System und lassen das Verfahren nur über den Bereich iterieren, der innerhalb des Fensters liegt. Dabei enthält das Fenster immer die Zeilen der Level, die kleiner oder gleich dem aktuellen Level sind.

Beginnend mit dem ersten Level lässt man das Iterationsverfahren einen Iterationsschritt auf den aktuell sichtbaren Komponenten des Iteriertenvektors laufen. Wenn das Iterationsverfahren auf einem Level ausgeführt wurde, muss danach die Iterierte prolongiert werden, um als Startwert für das folgende Level dienen zu können. Dazu wird diese zunächst in das Basissystem transformiert, um dann in das Basissystem des nächstfeineren Levels prolongiert zu werden. Der Startvektor der nächsten Iteration des Verfahrens wird dann erstellt, indem der so prolongierte Vektor derart in die Multileveliterierte eingebettet wird, dass er den seinem Level entsprechenden Bereich belegt. Der Rest des neuen Startvektors wird mit Nullen aufgefüllt. Die Matrix selber muss nicht erneut aufgestellt werden, es ist lediglich nötig, das Fenster, auf das das Iterationsverfahren im nächsten Schritt zugreifen kann, um das folgende Level zu erweitern. Nun kann über das Multilevelsystem mit vergrößertem Fenster iteriert werden. So fahren wir fort, bis wir das feinste Level erreicht haben.

Als Beispiel für den in Algorithmus 4 benötigten Iterationsschritt liefert Algorithmus 6 ein

Algorithmus 5: Nested-Iteration im erweiterten Gleichungssystem

Input : Matrix A^{E_k}, rechte Seite b^{E_k}, Startvektor v^1
Output : Startvektor v^k

$l = 1$
while $l \leq k$ **do**
  `// Lege das Fenster fest, auf dem das Iterationsverfahren operiert`
  `// Sei` m_l `die Anzahl der Freiheitsgrade auf Level` l `mit` $l = 1, \ldots, k$
  $F_l = \left[1, \ldots, \sum_{i=1}^{l} m_i\right]$
  `// Führe das Iterationsverfahrens aus`
  $v^{l,1} = \text{Iterationsschritt}\left(A^{E_k}, b^{E_k}, v^l, F_l\right)$
  `// Transformiere Vektor in das Basissystem`
  $v_{B_l} = S^{E_l B_l} v^l$
  `// Prolongiere diesen in das nächstfeinere Basissystem`
  $v_{B_{l+1}} = P_l^{l+1} v_{B_l}$
  `// und bette diesen in den neuen Startvektor ein`
  $v^{l+1} = \Lambda_{l+1}^{k} v_{B_{l+1}}$
  $l = l + 1$
end
return v^k

Iterationsverfahren, das wir derart modifiziert haben, dass es nur innerhalb eines gegebenen Fensters iteriert. Der Einfachheit halber wählen wir das Jacobi-Verfahren als Beispiel.

Auf diese Weise können wir die erweiterten Gleichungssysteme des Multilevelverfahrens zur Startwertberechnung nutzen und müssen nur kleine Änderungen an den Iterationsverfahren vornehmen.

4.4. Lösung von Multilevelsystemen

Nun betrachten wir einige Methoden, die sich zur Lösung des erweiterten Gleichungssystems (4.2) verwenden lassen.

Ein konventionelles direktes Verfahren, wie das der Gauß-Elimination, können wir nicht einsetzen, da die erweiterte Steifigkeitsmatrix A^{E_k} nicht invertierbar ist und damit eine Voraussetzung für dieses Verfahren nicht erfüllt ist (siehe Abschnitt 2.4.2.1). Außerdem sind direkte Löser für große Gleichungssysteme nicht praktikabel.

Stattdessen betrachten wir die klassischen Iterationsverfahren. Wendet man den Gauß-Seidel-Algorithmus geschickt auf das erweiterte Gleichungssystem an, kommt dies dem Mehrgitterverfahren gleich. Ein Schritt des symmetrischen Gauß-Seidel-Algorithmus, beginnend mit der letzten Zeile des Systems, entspricht dann exakt einem V-Zyklus im Mehrgitterverfahren. Die beim Mehrgitter nötigen Prolongations- und Restriktionsoperatoren müssen hier jedoch nicht ausgeführt werden, da diese durch die Struktur des Multilevelsystems bereits gegeben sind. So wird eine Operation auf dem groben Gitter stets entsprechend auf das feine Gitter prolongiert. Analog gilt dies für Operationen auf dem feinen Gitter und die Restriktion.

Für weitere Lösungsverfahren und ihr Verhalten in einem erweiterten Gleichungssystem sei auf

Algorithmus 6: Iterationsschritt des modifizierten Jacobi-Verfahrens

Input : $n \times n$ Matrix A, rechte Seite b, Startvektor v, Fenster F
Output : Startvektor w

for $j \in F$ **do**
  $w_j := 0$
  `// Relaxiere nur mit den Komponenten innerhalb des Fensters` F
(I) **for** $i \in F$ **do**
    if $j \neq i$ **then**
      $w_i = w_i + a_{i,j}\, v_j$
    end
  end
  $w_j = \frac{1}{a_{i,i}}(b_i - w_j)$
end
return w

[Gri94] verwiesen. So lässt sich beispielsweise der BPX-Vorkonditionierer als Jacobi-Verfahren auf der diagonalskalierten erweiterten Steifigkeitsmatrix interpretieren. Jacobi konvergiert auf dieser singulären Matrix zwar nicht unbedingt, lässt sich für die Vorkonditionierung jedoch einsetzen.

In dieser Arbeit möchten wir uns im Folgenden mit einem speziellen Löser beschäftigen, der bestimmte Probleme der anderen wohlbekannten Verfahren vermeidet. Das sind zum einen Probleme, die daraus resultieren, dass das Lösungsverfahren häufig die Vorteile des Multilevel- bzw. Mehrgitterkonzepts annulliert, indem es bei Operationen auf groben Gittern den Fehler auf feinere Gitter verschmiert. Dies ist nicht zu vermeiden, wenn Grobgitterkorrekturschritte gemacht werden müssen. Da jedoch die meisten Verfahren die Level statisch ablaufen, wird immer wieder über die gröberen Level iteriert. Auch wenn dem Verfahren bestimmte geschickte Durchlaufreihenfolgen (V-, W-, F-Zyklus) vorgegeben werden, kann dieses zu keinem Zeitpunkt wissen, ob ein Schritt auf einem gröberen Level nötig ist und ob insbesondere dieser den zuvor im feineren Gitter reduzierten Fehler wieder verstärkt.

Zum anderen sind die meisten bekannten Verfahren problemabhängig konstruiert oder müssen für spezielle Anwendungen vorkonditioniert werden. Dies erfordert stets, dass man das Problem erst auf seine Eigenschaften hin untersuchen und geeignete Verfahren finden muss, bevor man mit der Lösung des Systems beginnen kann.

Wir werden im Folgenden mit dem Gauß-Southwell ein adaptives Iterationsverfahren untersuchen. Dieses soll sich durch die adaptiv gewählte Durchlaufreihenfolge an das gegebene Problem anpassen können, so dass es beispielsweise einem richtungsorientierten Operator folgen kann. Weiterhin soll das Verfahren die Multilevelstruktur des Gleichungssystems so gut wie möglich ausnutzen und stets nur dann auf einem Level operieren, wenn dies der Lösung des Systems direkt dient und keine unnötigen oder ungeschickten Schritte durchgeführt werden.

Es ist zu erwarten, dass dadurch der Gauß-Southwell auch weniger abhängig von den Parametern der Differentialgleichung ist als ein klassisches Iterationsverfahren. Das bedeutet, dass mit einer adaptiven Durchlaufreihenfolge die Robustheit eines Verfahrens erhöht werden kann.

5. Gauß-Southwell-Verfahren

In diesem Kapitel werden wir den Algorithmus von Gauß-Southwell behandeln. Dieses iterative Verfahren zur Lösung eines linearen Gleichungssystems wurde 1823 von Gauß [Gau03] erstmals vorgeschlagen. Später entwickelte Southwell [Sou40] das Verfahren auf der Basis physikalischer Prozesse und wandte es auf verschiedene Probleme an [Sou46]. In [Fox48] wird Southwells Verfahren von den physikalischen Grundlagen gelöst und als Iterationsverfahren für Gleichungssysteme unabhängig von der Problemstellung verwendet. Erstmals als *Gauß-Southwell-Relaxation* wird es in [FW60] bezeichnet. In [LY08] wird die Vorgehensweise des Gauß-Southwell-Verfahrens auf Minimierungsprobleme angewandt und als *Gauß-Southwell-Method* benannt.

Dieses Verfahren unterscheidet sich von den klassischen Iterationsverfahren dadurch, dass nicht ein ganzer Iterationsschritt durchgeführt wird, in welchem die komplette Iterierte relaxiert wird, sondern dass in einzelnen Schritten die Iterierte komponentenweise relaxiert wird. Der Algorithmus geht hierbei so vor, dass er stets die Komponente auswählt, die im Residuenvektor den betragsmäßig größten Wert hat. Dieser Wert des Residuenvektors ist danach Null. So passt der Algorithmus sich adaptiv an das gegebene Gleichungssystem an und erreicht eine geschickte Durchlaufreihenfolge, die unnötige Schritte vermeidet. Berücksichtigt man, dass die Strategie des Verfahrens ist, immer die Komponente zu relaxieren, die zum aktuellen Zeitpunkt der Iteration den größten Gewinn verspricht, kann man den Algorithmus in die Klasse der Greedy-Algorithmen einordnen.

Warum ist es sinnvoll, sich am Residuum des Gleichungssystems zu orientieren? Alle Elemente des Gleichungssystems beeinflussen das Residuum, denn es ist wie folgt definiert

$$r := Ax - b.$$

Das Residuum beinhaltet also alle Informationen des Gleichungssystems bzw. der zugrunde liegenden diskretisierten partiellen Differentialgleichung. Sowohl die Parameter des Differentialoperators als auch die aktuelle Iterierte und die rechte Seite haben ihren Anteil daran. Dadurch spiegelt das Residuum sowohl Daten- als auch Lösungsänderungen wider, und der Gauß-Southwell-Algorithmus nutzt dies aus, indem er durch seine Wahl des betragsmäßig größten Residuumwertes dies alles mit berücksichtigt.

Da das Verfahren somit Informationen über die Parameter gewinnt, ist zu erwarten, dass es sich an diese anpassen und so die Abhängigkeit der Konvergenzgeschwindigkeit von diesen reduzieren kann. Würde die Konvergenzrate des Verfahrens vollständig von den Parametern unabhängig werden, hätte man Robustheit erreicht.

Um diese Vorgehensweise zu ermöglichen, muss der Algorithmus stets auf das betragsmäßig größte Element des Residuenvektors zugreifen können. Aus dieser Bedingung folgen Kosten, die wir im weiteren Verlauf dieses Kapitels minimieren werden.

5.1. Das Verfahren

Betrachten wir nun das Verfahren im Detail, wobei wir der Darstellung in [Bol03] folgen. Wir wollen ein lineares Gleichungssystem

$$Ax = b$$

lösen (siehe Abschnitt 2.4.1). Mit $x^{(k)} \in \mathbb{R}^n$ bezeichnen wir die Iterierte und mit $r^{(k)} \in \mathbb{R}^n$ den Residuenvektor im Schritt k. Der Gauß-Southwell-Algorithmus führt in jedem Einzelschritt die Relaxation einer Komponente aus, so dass das Residuum dieser Komponente Null wird. Sei

$$i_k := \text{argmax}_{j\in\{1,2,..,n\}} \left(|r_j^{(k)}|\right)$$

der Index der Komponente, die im k-ten Schritt relaxiert wird.

Seien außerdem ϕ_{i_k} und ψ_{i_k} die Abbildungen, welche die Änderungen an x bzw. r beschreiben, das heißt

$$\begin{aligned} \phi_{i_k} &: \mathbb{R}^n \to \mathbb{R}^n \quad \text{mit} \quad x^{(k)} \mapsto \phi_{i_k}(x^{(k)}) = x^{(k+1)} \text{ und} \\ \psi_{i_k} &: \mathbb{R}^n \to \mathbb{R}^n \quad \text{mit} \quad r^{(k)} \mapsto \psi_{i_k}(r^{(k)}) = r^{(k+1)}. \end{aligned}$$

Damit können wir das Verfahren von Gauß-Southwell formal definieren.

Definition 5.1 (Verfahren von Gauß-Southwell)**.** *Sei $A \in \mathbb{R}^{n\times n}$, $b \in \mathbb{R}^n$ gegeben, $x^{(0)} \in \mathbb{R}^n$ beliebig, ϕ_{i_k} und ψ_{i_k} wie oben definiert. Dann ist die Iterationsvorschrift für das Verfahren von Gauß-Southwell gegeben durch*

$$\begin{aligned} r^{(0)} &= b - A\,x^{(0)}, \\ i_k &= \text{argmax}_{i\in\{1,2,\ldots,n\}} \left(|r_i^{(k)}|\right), \\ x^{(k+1)} &= \phi_{i_k}\left(x^{(k)}\right), \\ r^{(k+1)} &= \psi_{i_k}\left(r^{(k)}\right). \end{aligned}$$

Um die Abbildungen definieren zu können, müssen wir zunächst einige Beziehungen herstellen. Es gilt

$$x_i^{(k)} := \begin{cases} \frac{1}{a_{i_k,i_k}} \left(b_{i_k} - \sum\limits_{\substack{j=1 \\ j\neq i_k}}^{n} a_{i_k,j} x_j^{(k-1)} \right) & , \text{ falls } i = i_k \text{ und} \\ x_i^{k-1} & , \text{ sonst.} \end{cases} \tag{5.1}$$

Das Residuum nach jedem Iterationsschritt neu auszurechnen, ist sehr teuer, da es eine Matrix-Vektor-Multiplikation erfordern würde. Wir zeigen im Folgenden, dass es eine Methode gibt, das Residuum schneller auszurechnen, indem man die aktuelle Änderung entsprechend einbezieht.

Betrachten wir die Abbildung ψ_{i_k}, die das Residuum behandelt.

Lemma 5.1. *Für die Einträge des Residuenvektors im k-ten Schritt gilt nach der Relaxation der i_k-ten Komponente*

$$r_i^{(k)} = \begin{cases} 0 & , \text{ falls } i = i_k \text{ und} \\ r_i^{(k-1)} - \frac{a_{i,i_k}}{a_{i_k,i_k}} r_{i_k}^{(k-1)} & , \text{ sonst.} \end{cases} \tag{5.2}$$

Beweis. Für den Fall $i = i_k$ gilt

$$\begin{aligned} r_{i_k}^{(k)} &= (b - Ax^{(k)})_{i_k} \\ &= b_{i_k} - \sum_{j=1}^{n} a_{i_k,j}\, x_j^{(k)} \\ &\overset{(5.1)}{=} b_{i_k} - \sum_{\substack{j=1 \\ j\neq k}}^{n} a_{i_k,j}\, x_j^{(k-1)} - a_{i_k,i_k} \frac{1}{a_{i_k,i_k}} \left(b_{i_k} - \sum_{\substack{j=1 \\ j\neq k}}^{n} a_{i_k,j}\, x_j^{(k-1)} \right) \\ &= 0. \end{aligned} \tag{5.3}$$

Wir überlegen zunächst, dass

$$x_{i_k}^{(k)} = x_{i_k}^{(k-1)} + \frac{r_{i_k}^{(k-1)}}{a_{i_k,i_k}} \tag{5.4}$$

gilt, denn

$$\begin{aligned} x_{i_k}^{(k-1)} + \frac{r_{i_k}^{(k-1)}}{a_{i_k,i_k}} &= \frac{1}{a_{i_k,i_k}} \left(a_{i_k,i_k}\, x_{i_k}^{(k-1)} + \underbrace{r_{i_k}^{(k-1)}}_{\left(b - Ax^{(k-1)}\right)_{i_k}} \right) \\ &= \frac{1}{a_{i_k,i_k}} \left(a_{i_k,i_k}\, x_{i_k}^{(k-1)} + b_{i_k} - \sum_{j=1}^{n} a_{i_k,j}\, x_j^{(k-1)} \right) \\ &= \frac{1}{a_{i_k,i_k}} \left(b_{i_k} - \sum_{\substack{j=1 \\ j\neq k}}^{n} a_{i_k,j}\, x_j^{(k-1)} \right) \\ &\overset{(5.1)}{=} x_{i_k}^{(k)}. \end{aligned}$$

Für alle übrigen Komponenten von $r^{(k)}$ mit $i \neq i_k$ gilt

$$\begin{aligned}
r_i^{(k)} &= (b - Ax^{(k)})_i \\
&= b_i - \sum_{j=1}^{n} a_{i,j}\, x_j^{(k)} \\
&= b_i - \sum_{\substack{j=1 \\ j\neq i_k}}^{n} a_{i,j}\, x_j^{(k)} - a_{i,i_k} x_{i_k}^{(k)} \\
&\overset{(5.1)}{=} b_i - \sum_{\substack{j=1 \\ j\neq i_k}}^{n} a_{i,j}\, x_j^{(k-1)} - a_{i,i_k} x_{i_k}^{(k)} - a_{i,i_k} x_{i_k}^{(k-1)} + a_{i,i_k} x_{i_k}^{(k-1)} \\
&= b_i - \sum_{j=1}^{n} a_{i,j}\, x_j^{(k-1)} - a_{i,i_k}\, x_{i_k}^{(k)} + a_{i,i_k}\, x_{i_k}^{(k-1)} \\
&= r_i^{(k-1)} + a_{i,i_k} \left(x_{i_k}^{(k-1)} - x_{i_k}^{(k)} \right) \\
&\overset{(5.4)}{=} r_i^{(k-1)} + a_{i,i_k} \left(x_{i_k}^{(k-1)} - \left(x_{i_k}^{(k-1)} + \frac{r_{i_k}^{(k-1)}}{a_{i_k,i_k}} \right) \right) \\
&= r_i^{(k-1)} - \frac{a_{i,i_k}}{a_{i_k,i_k}} r_{i_k}^{(k-1)}.
\end{aligned} \tag{5.5}$$

□

Damit haben wir gezeigt, dass wir das Residuum stets nach einem Iterationsschritt günstig aktualisieren können.

Ist die Matrix A zusätzlich dünn besiedelt, kann dieses Wissen hier verwendet werden. Indem nur die Elemente i des Residuenvektors r aktualisiert werden, für die der entsprechende Matrixeintrag nicht Null ist, lassen sich unnötige Kosten einsparen, da gilt

$$r_i^{(k)} = r_i^{(k-1)}, \text{ falls } a_{i,i_k} = 0.$$

5.1.1. Konvergenz

Aus obigen Beziehungen erkennt man, dass Vektoren, deren i_k-te Komponente Null ist, Fixpunkte der Residuumsabbildung ψ_{i_k} darstellen. Somit wird jeder Vektor auf einen Fixpunkt abgebildet. Daraus folgt insbesondere, dass eine mehrfache Hintereinanderausführung einer Abbildung ein Ergebnis nicht mehr ändert.

Mit Hilfe dieser Überlegung lässt sich eine erste Konvergenzaussage für das Gauß-Southwell-Verfahren formulieren. Im Folgenden sei stets $\|\cdot\|_1$ die Summennorm und $\|\cdot\|_\infty$ die Maximumsnorm.

Lemma 5.2. *Sei $A \in \mathbb{R}^{n\times n}$, A^T strikt diagonaldominant und ψ_{i_k} mit $i_k \in \{1, 2, \ldots, n\}$ beliebig die Abbildung des Residuums $r^{(k)}$. Dann gilt mit $\kappa := \max_{l\in\{1,2,\ldots,n\}} \sum\limits_{\substack{j=1 \\ j\neq l}}^{n} |\frac{a_{l,j}}{a_{l,l}}| < 1$*

$$||\psi_{i_k}(r^{(k)})||_1 \quad \leq \quad \sum_{\substack{j=1 \\ j\neq i_k}}^{n} |r_j^{(k)}| + \kappa\, |r_{i_k}^{(k)}| \quad < \quad ||r^{(k)}||_1.$$

Beweis.

$$
\begin{aligned}
||\psi_{i_k}(r^{(k)})||_1 &= \sum_{j=1}^{n} |\psi_{i_k}(r^{(k)})_j| \\
&\overset{(5.3)}{=} \sum_{\substack{j=1 \\ j\neq i_k}}^{n} |\psi_{i_k}(r^{(k)})_j| \\
&\overset{(5.5)}{=} \sum_{\substack{j=1 \\ j\neq i_k}}^{n} |r_j^{(k)} - \frac{a_{j,i_k}}{a_{i_k,i_k}} r_{i_k}^{(k)}| \\
&\leq \sum_{\substack{j=1 \\ j\neq i_k}}^{n} |r_j^{(k)}| + \underbrace{\sum_{\substack{j=1 \\ j\neq i_k}}^{n} |\frac{a_{j,i_k}}{a_{i_k,i_k}}|}_{\leq \kappa < 1} |r_{i_k}^{(k)}| \\
&\leq \sum_{\substack{j=1 \\ j\neq i_k}}^{n} |r_j^{(k)}| + \kappa |r_{i_k}^{(k)}| \\
&< ||r^{(k)}||_1.
\end{aligned}
$$

□

Da $\kappa < 1$ gilt, ist die Folge $\{||r^{(k)}||_1\}_{k=0}^{\infty}$ streng monoton fallend.

In Lemma 5.2 haben wir gezeigt, dass das Verfahren von Gauß-Southwell konvergent ist, da $||\cdot||_1 \geq 0$ und somit eine untere Schranke existiert. Damit jedoch der Algorithmus gegen die Lösung des Gleichungssystems konvergiert, muss das Residuum gegen Null gehen. Das besagt der folgende Satz.

Satz 5.1 (Konvergenz des Verfahrens von Gauß-Southwell)**.** *Seien $A \in \mathbb{R}^{n\times n}$ mit A^T strikt diagonaldominant, $b \in \mathbb{R}^n$ und $x^{(k)} \in \mathbb{R}^n$ gegeben. Dann gilt bei der Anwendung des Verfahrens von Gauß-Southwell*

$$||r^{(k+1)}||_1 \leq \delta ||r^{(k)}||_1 \quad \forall\, k \in \mathbb{N}$$

mit $\delta := \frac{n-1+\kappa}{n}$ und $\kappa := \max_{l\in\{1,2,\ldots,n\}} \sum_{\substack{j=1 \\ j\neq l}}^{n} |\frac{a_{l,j}}{a_{l,l}}|$. Da δ ein fester Wert und unabhängig von k ist, konvergiert das Residuum gegen die untere Schranke gegen Null.

Beweis. Da $\kappa < 1$ gilt, folgt $\delta \in [0, 1)$. Weiterhin sei $i_k := \text{argmax}_{i \in \{1,2,\ldots,n\}} \left(|r_i^{(k)}| \right)$. Damit gilt

$$\begin{aligned}
||r^{(k+1)}||_1 &= ||\psi_{i_k}(r^{(k)})||_1 \\
&\overset{\text{Lemma 5.2}}{\leq} \sum_{\substack{j=1 \\ j \neq i_k}}^{n} |r_j^{(k)}| + \kappa |r_{i_k}^{(k)}| \\
&= \sum_{\substack{j=1 \\ j \neq i_k}}^{n} |r_j^{(k)}| + \kappa ||r^{(k)}||_\infty \\
&= ||r^{(k)}||_1 - ||r^{(k)}||_\infty + \kappa ||r^{(k)}||_\infty \\
&= ||r^{(k)}||_1 - (1-\kappa) ||r^{(k)}||_\infty \\
&= ||r^{(k)}||_1 - \frac{(1-\kappa)}{n} n ||r^{(k)}||_\infty \\
&\leq ||r^{(k)}||_1 - \frac{(1-\kappa)}{n} ||r^{(k)}||_1 \\
&= \frac{n-1+\kappa}{n} ||r^{(k)}||_1 \\
&= \delta ||r^{(k)}||_1.
\end{aligned}$$

□

Weiterführende Untersuchungen der theoretischen Konvergenz des Gauß-Southwell-Verfahrens, insbesondere in Bezug auf eine für uns relevante Erweiterung auf positiv (semi-)definite Systeme, finden sich in [Fox48], [WK02] sowie [Rü93] und den Referenzen darin.

Wichtig ist, dass die Frage nach theoretischer Konvergenz für uns nicht so bedeutsam ist wie die Frage nach praktischer Konvergenz. Selbst ein konvergentes Verfahren kann nicht praktikabel sein, wenn die Konvergenzgeschwindigkeit nicht konkurrenzfähig ist. In Kapitel 6 untersuchen wir die Konvergenzraten einiger Problemstellungen.

5.2. Der Algorithmus

Im folgenden Abschnitt beschäftigen wir uns mit der Komplexität und den Kosten des Verfahrens von Gauß-Southwell. Rufen wir uns nochmal in Erinnerung, was dieses Verfahren von den klassischen Iterationsverfahren unterscheidet. Die klassischen Iterationsverfahren (siehe Abschnitt 2.4.3) wie Jacobi oder Gauß-Seidel und deren Varianten, die mit einem Relaxationsparameter arbeiten, haben gemeinsam, dass sie das Gleichungssystem stets in einer festen Reihenfolge durchlaufen.

Das Gauß-Southwell-Verfahren dagegen legt die Durchlaufreihenfolge dynamisch fest. Anhand des Residuums wird in jedem Einzelschritt entschieden, welche Komponente der Iterierten als nächstes relaxiert wird. Dadurch erreicht das Verfahren eine geschickte Durchlaufreihenfolge und vermeidet unnötige Schritte. Das Gauß-Southwell-Verfahren passt sich adaptiv an das Gleichungssystem an. Ist das Gleichungssystem besonders strukturiert, wie es beim erweiterten Gleichungssystem der Multilevelmethode (siehe Kapitel 4) der Fall ist, ermöglicht diese Vorgehensweise dem Verfahren, sich selbstständig an die Struktur anzupassen.

Wir definieren das Verfahren zunächst als Algorithmus 7.

Algorithmus 7: Gauß-Southwell-Verfahren

Input : $n \times n$ Matrix A, rechte Seite b, Startvektor x^0
Output : Näherungslösung x^c

`// Berechne Residuum`
(I) $r^0 = b - Ax^0$
for j = *1,...,n* **do**
  $r_j^0 = \left|r_j^0\right|$
end
`// Führe` c `Relaxierungen durch`
(II) **for** i = *1,...,c* **do**
  $x^{(i)} = x^{(i-1)}$
  `// finde größten Eintrag` i_k `im Residuenvektor` $r^{(i)}$
(III) $i_k = \text{argmax}_{j \in \{1,2,..,n\}}(|r_j^{(i)}|)$
  $x_{i_k}^{(i)} = 0$
(IV) **for** $j \atop {j \neq i_k}$ = *1,...,n* **do**
    $x_{i_k}^{(i)} = x_{i_k}^{(i)} + a_{i_k,j} \cdot x_j^{(i-1)}$
    `// Residualkomponente` j `neu berechnen`
(V) $r_j^{(i)} = \left|r_j^{(i-1)} - \frac{a_{i,i_k}}{a_{i_k,i_k}} r_{i_k}^{(i-1)}\right|$
  end
  $x_{i_k}^{(i)} = \frac{1}{a_{i_k,i_k}}(b_{i_k} - x_{i_k}^{(i)})$
  `// Residualkomponente` i_k `setzen`
(VI) $r_{i_k}^{(i)} = 0$
end

5.2.1. Komplexität

Wir untersuchen nun die Komplexität des Algorithmus von Gauß-Southwell und die Möglichkeiten, die Kosten des Verfahrens zu reduzieren. Im Folgenden ist N immer die Anzahl der Unbekannten des Gleichungssystems. Zunächst analysieren wir die Kosten in Abhängigkeit von N. Entscheidend sind hierbei diejenigen, die innerhalb der ersten **for**-Schleife (II) auftreten. Speichern wir das Residuum in einem Vektor, setzen sich die Kosten wie folgt zusammen.

- Die lineare Suche nach dem größten Eintrag im Residuenvektor (III) kostet $\Theta(N)$.
- Wenn wir einen Vektor für das Residuum benutzen, können wir die Kosten für das Speichern von Elementen in (V) und (VI) mit jeweils $\mathcal{O}(1)$ vernachlässigen.
- Die Berechnung des neuen Eintrags im Vektor der Iterierten x kostet $\mathcal{O}(N)$, dominiert durch die **for**-Schleife in (IV).

Um die Kosten des Verfahrens zu senken, ist der erste Ansatz, die Suche nach dem größten Eintrag im Residuum zu optimieren. Wenn man den Vektor nach der Größe der Einträge sortiert,

kann man stets schnell auf das größte Element zugreifen. Dann tritt jedoch das Problem auf, dass man in (V) und (VI) über den Index des Elements auf Elemente unbekannten Wertes zugreifen muss. Dieses Problem lässt sich mittels einer Datenstruktur, die einen Index für den Schlüssel und einen Index für den Wert des Vektors bereit hält, lösen.

Wenn wir die Kosten für die doppelte Indizierung vernachlässigen, besteht weiterhin das Problem, dass der Vektor sortiert werden muss. Die Kosten für gängige Sortierverfahren, wie einem *Heapsort* oder einem *Quicksort*, liegen im durchschnittlichen Fall bei $\mathcal{O}(N \log N)$, im schlechtesten Fall benötigt der *Quicksort* sogar $\mathcal{O}(N^2)$ [Sch01]. Wählt man den *Bucketsort* als Sortieralgorithmus, lässt sich eine Komplexität von $\mathcal{O}(N)$ erreichen, wenn man davon ausgeht, dass die Schlüsselmenge unabhängig und gleichverteilt in einem gegeben Intervall ist. Ist das nicht der Fall, kann der schlechteste Fall eintreten und alle Schlüssel landen im selben Bucket. Dies hätte zur Folge, dass dieses Sortierverfahren eine Komplexität von $\mathcal{O}(N^2)$ erreichen würde. Selbst wenn man garantieren könnte, dass diese Bedingung zu erfüllen sei, läge man mit den Kosten in der gleichen Größenordnung wie der, den die *lineare Suche* im unsortieren Residuenvektor kostet, und man hätte dieser gegenüber nichts gewonnen.

Somit lässt sich der Schluss ziehen, dass wir über das reine Sortieren des Residuenvektors die Kosten des Algorithmus nicht reduzieren können. Im weiteren Verlauf dieses Kapitels werden wir allerdings feststellen, dass wir nicht ständig alle Komponenten des Residuenvektors ändern werden und es somit unnötig ist, den Vektor komplett neu zu sortieren.

Betrachten wir nun einige Lösungsansätze, die auf Datenstrukturen beruhen, die es ermöglichen, Elemente geordnet abzulegen.

5.3. Datenstrukturen

Um aus der Vielzahl existierender Datenstrukturen wählen zu können, überlegen wir zunächst die Anforderungen, die das Gauß-Southwell-Verfahren an die Datenstruktur des Residuenvektors stellt. Die Elemente des Vektors sind Paare des Zeilenindex und des Werts. Der Zeilenindex ist ganzzahlig und eindeutig, wogegen der Wert eine Fließkommazahl ist.

Der Algorithmus greift auf die folgenden vier Weisen auf die Datenstruktur zu.

1. Einfügen von Elementen (I).
2. Zugriff auf das Element mit dem größten Wert (III).
3. Zugriff auf das Element anhand des Index (V).
4. Änderung des Wertes eines Elements anhand des Index (V)+(VI).

Hierbei muss man beachten, dass das Element mit dem größten Wert, auf das in (III) zugegriffen wird, stets in (VI) geändert wird, es somit auch aus der Datenstruktur entfernt und wieder eingefügt werden sollte.

Wenn man außerdem davon ausgeht, dass die Matrix des Gleichungssystems dünn besiedelt ist, wie es bei Matrizen, die beim Multilevelverfahren zum Einsatz kommen, der Fall ist, werden meist nur sehr wenige Komponenten des Residuenvektors geändert. Folglich wird es sinnvoll sein, die Datenstruktur so sortiert wie möglich zu halten, um unnötige Sortieroperationen zu vermeiden.

Im Folgenden werden wir Priority-Queues untersuchen, die diesen Anforderungen gerecht werden können.

5.3.1. Priority-Queues und Heaps

Ein Heap ist eine Datenstruktur aus der Familie der Priority-Queues. Die zentrale Eigenschaft dieser Datenstrukturen ist die, dass stets das Element mit der höchsten Priorität entnommen werden kann. Der Zugriff auf die Elemente mit niedrigerer Priorität ist nachrangig, wenn nicht ganz unmöglich. Eine Priority-Queue stellt in der Regel die Operationen *insert*, *remove*, *extractMax/Min* und *in-/decreaseKey* bereit. Wir fordern im Weiteren, dass ebenfalls die Operation *changeKey* zur Verfügung gestellt wird. Diese soll es ermöglichen, nicht nur die Priorität eines Elements zu erhöhen, wie es *in-/decreaseKey* können soll, sondern auch die Priorität senken zu können. Die meisten existierenden Priority-Queues stellen diese Operation nicht zur Verfügung, doch ist es in allen Priority-Queues möglich, diese durch die Nacheinanderausführung von *remove* und *insert* zu realisieren, wenn sich keine andere Möglichkeit bietet.

Die Operationen sind wie folgt definiert.

- *insert* fügt ein neues Element in den Heap ein.
- *remove* entfernt ein Element aus dem Heap.
- *extractMax/Min* gibt das Element mit der höchsten Priorität zurück und entfernt es aus der Datenstrutur. Die Bezeichnung rührt daher, dass Heaps meistens ab- oder aufsteigend nach der Schlüsselgröße sortiert sind.
- *in-/decreaseKey* erhöht die Priorität des Schlüssels eines Elements. Je nachdem, ob der Heap auf- oder absteigend sortiert ist, wird diese Funktion anders bezeichnet.
- *changeKey* ändert den Schlüssel eines Elements, kann die Priorität also beliebig ändern.

Bei den Operationen *remove* und *changeKey* muss beachtet werden, dass alle Heaps davon ausgehen, dass man die Position des Elementes im Heap bereits kennt. Um die Position jedes Elements zu kennen, kann es nötig sein, die Datenstruktur des Heaps stark erweitern zu müssen, was zur Folge hat, dass die Kosten der Operationen beeinflusst werden können und zusätzlich Informationen über die Elemente gespeichert werden müssen. Es wird vorausgesetzt, dass für jede dieser Operationen gilt, dass nach ihrer Ausführung die Heap-Struktur erhalten geblieben ist.

Betrachten wir nun einige spezielle Priority-Queues, die sich uns anbieten würden.

Der **Binär-Heap** [Sch01] ist eine Priority-Queue, die alle für uns notwendigen Operationen in $\mathcal{O}(\log N)$ zur Verfügung stellt, wenn die Position aller Elemente im Heap bekannt ist. Die Struktur dieses Heaps basiert auf einem balancierten Binärbaum. Bemerkenswert ist, dass sich der Binär-Heap sehr einfach in einem Array realisieren lässt und damit wenig Aufwand für die Struktur der Daten nötig ist. Im nächsten Kapitel werden wir diesen Ansatz genauer betrachten.

Der **Binominal-Heap** [Vui78] hat gegenüber dem Binär-Heap den Vorteil, dass er die *merge* Operation günstiger zur Verfügung stellt. Bei dieser Operation geht es darum, zwei Priority-Queues miteinander zu vereinigen. Der Binominal-Heap profitiert davon, dass er aus mehreren Binominal-Bäumen (siehe Abbildung 5.1) besteht, die jeweils einen Heap darstellen. Bei der Vereinigung von Binominal-Heaps müssen dann lediglich die Binominal-Bäume der Heaps vereinigt werden, was sich sehr günstig umsetzen lässt. Da wir die *merge* Operationen jedoch nicht benötigen und alle anderen Operationen zu den gleichen Kosten wie beim Binär-Heap ($\mathcal{O}(\log N)$) durchgeführt werden können, würden wir nicht besonders vom Binominal-Heap profitieren können.

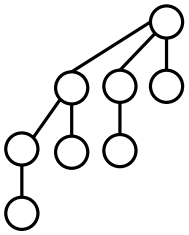


Abb. 5.1.: Struktur eines Binominal-Baums mit 8 Elementen

Vorteile gegenüber dem Binominal-Heap bietet der **Fibonacci-Heap** [FT87]. Die Struktur des Fibonacci-Heaps erlaubt es, dass sich die Kosten einer Umstrukturierung in den darauf folgenden Operationen amortisieren. In diesem Fall sind für *insert* und *merge* Operationen Kosten von lediglich $\mathcal{O}(1)$ nötig. Damit das möglich ist, wird vorausgesetzt, dass die Anzahl der *delete* Operationen im Vergleich zu allen anderen Operationen signifikant kleiner ist. Da aber zum einen unsere Anforderung, einen *changeKey* ausführen zu können, jeweils *insert* und *delete* Operationen benötigt und zum anderen *delete* Operationen im worst case $\mathcal{O}(N)$ kosten können, bietet der Fibonacci-Heap für uns keine Vorteile gegenüber den Binominal-Heaps. Durch unsere Anforderungen können wir auch nicht davon profitieren, dass der Fibonacci-Heap amortisierte Kosten für *insert* und *merge* Operationen ermöglicht. Weiterhin lässt die Struktur der Fibonacci-Heaps (Abbildung 5.2) erahnen, dass diese Datenstruktur für den Fall, dass fortlaufend Elemente geändert werden, nicht praktikabel ist.

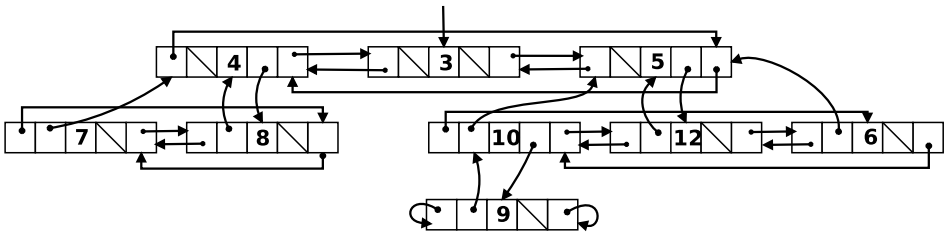


Abb. 5.2.: Beispiel eines Fibonacci-Heaps mit neun Elementen

Der **Relaxed-Heap** [DGST88] ist eine Alternative zum Fibonacci-Heap, der eine weniger komplexe Struktur aufweist. Auch hier stellt unsere Forderung, die Operation *changeKey* durchführen zu können, das größte Problem dar, da diese Operation wie beim Binär-Heap $\mathcal{O}(\log N)$ kostet und uns somit keine Vorteile gegenüber diesem bietet.

Nach dieser Betrachtung von verschiedenen Priority-Queues müssen wir feststellen, dass der Binär-Heap alle unsere Anforderungen erfüllt. Aus den Vorteilen der anderen Lösungen können wir keinen Nutzen ziehen. Da wir die Komponenten des Residuen-Vektors häufig ändern und damit die Struktur der Priority-Queue häufig umbauen müssen, ist es in unserem Interesse, eine Datenstruktur zu verwenden, die möglichst einfach gehalten ist, aber die von uns benötigten Operationen effizient durchführen kann. Je komplexer die Datenstruktur ist, desto mehr Aufwand muss in eine Umstrukturierung gesteckt werden. Das anfangs erwähnte Problem, dass wir die Position der Elemente in der Datenstruktur kennen müssen, wenn wir effizient *remove* oder *changeKey* Operationen durchführen wollen, ist für komplexere Strukturen deutlich schwieriger zu lösen als für eine einfach gehaltene.

Im nächsten Abschnitt stellen wir den Binär-Heap ausführlich vor und untersuchen diesen darauf, wie er mit unserer Problemstellung umgehen kann. Danach erweitern wir den Binär-Heap

um die Möglichkeit, auf ein Element anhand seines Werts zuzugreifen und dessen Schlüssel zu ändern.

5.3.2. Binär-Heap

Ein Binär-Heap stellt sämtliche Heap-Operationen mit einer Komplexität von $\mathcal{O}(\log N)$ zur Verfügung. Wir betrachten nun die Datenstruktur und die Bedingungen, die das ermöglichen.

Die zugrunde liegende Datenstruktur des Binär-Heaps ist ein balancierter Binärbaum. Bei diesem müssen alle Ebenen bis auf die unterste vollständig besetzt sein. Die letzte Schicht wird dabei stets von links aufgefüllt. Um eine Heap-Struktur zu erreichen, werden die Elemente im Baum derart abgelegt, dass jeder Elternknoten eine höhere Priorität hat als seine Kinder. Daraus folgt, dass der Baum die Heap-Bedingung erfüllt und somit an der Wurzel der Knoten mit der höchsten Priorität steht.

Eine wichtige Eigenschaft eines balancierten Binärbaums ist, dass dieser mit N Elementen eine maximale Höhe von $\mathcal{O}(\log N)$ besitzt.

Jede der oben genannten Operationen kann zur Folge haben, dass ein Knoten die Heap-Sortierung des Binärbaums verletzt. Um diese wiederherzustellen, wird eine Funktion benutzt, die den Knoten so lange mit seinen Eltern- (*upheap*) bzw. seinen Kindknoten (*downheap*) vertauscht, bis die Heap-Bedingung wieder für jeden Knoten im Baum erfüllt ist.

Diese *heapify* Operation muss den Knoten maximal von der Wurzel bis in die unterste Ebene bzw. von der untersten Ebene bis zur Wurzel transportieren und hat somit höchstens die Kosten in der Höhe des Baumes $\mathcal{O}(\log N)$. Die Eigenschaft, dass ein Element leicht sowohl nach oben, als auch nach unten bezüglich der Sortierfolge bewegt werden kann, ist eine Eigenschaft des Binär-Heaps, die es uns ermöglicht, die geforderte Operation *changeKey* zu implementieren, ohne dass das Element erst aus dem Baum entfernt und dann neu eingefügt werden muss.

Die zur Realisierung eines Binärbaums nötige Datenstruktur lässt sich sehr einfach mittels eines Arrays implementieren. Dabei werden die Knoten beginnend mit der Wurzel Ebene für Ebene durchnummeriert, wie es in Abbildung 5.3 abgebildet ist. Entsprechend dieser Nummerierung werden die Knoten ohne zusätzliche Informationen in einem Array abgelegt. Diese Möglichkeit, die Daten zu speichern, hat zur Folge, dass die Zugriffe und Änderungen an der Struktur deutlich günstiger sind als bei den anderen vorgestellen Priority-Queues.

Die Wurzel liegt an Position 0 im Array, und für einen beliebigen Knoten an der Position i des Arrays gilt, dass der Elternknoten stets an der Stelle $\lfloor \frac{i-1}{2} \rfloor$, das linke Kind an der Stelle $2i+1$ und das rechte Kind an der Stelle $2i+2$ zu finden ist.

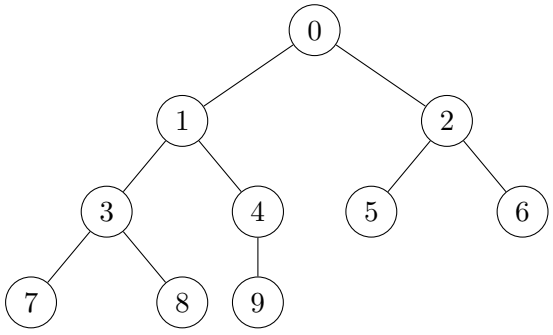


Abb. 5.3.: Ein balancierter Binärbaum

Im Folgenden untersuchen wir, wie sich der Binär-Heap erweitern lässt, um sich in die Gegebenheiten des Gauß-Southwell-Algorithmus einpassen und dessen Anforderungen gerecht werden zu können.

5.3.3. Binär-Heap mit extra indizierten Schlüssel-Wert-Paaren

Der Gauß-Southwell-Algorithmus hat an die Heap-Datenstruktur, die das Residuum speichert, die Anforderung, dass der Wert jeden Elements des Residuums den Schlüssel darstellt, der auf den Index des Elements zeigt. Wir betrachten ein Beispiel. Speichert man den Residuenvektor

$$r = (0, 1.0, 4.2, 2.7, 8.5, 4.1, 0.5, 0, 1.0)^T$$

in einem Binär-Heap ab, ergibt sich die Struktur, die in Abbildung 5.4 abgebildet ist. Man beachte, dass die Residualeinträge, die bereits Null sind und somit nicht mehr relaxiert werden müssen, nicht in die Datenstruktur mit aufgenommen worden sind. Dieser Trick erspart unnötige Umstrukturierungen des Binär-Heaps. Wir können aus dem Binär-Heap problemlos den jeweils

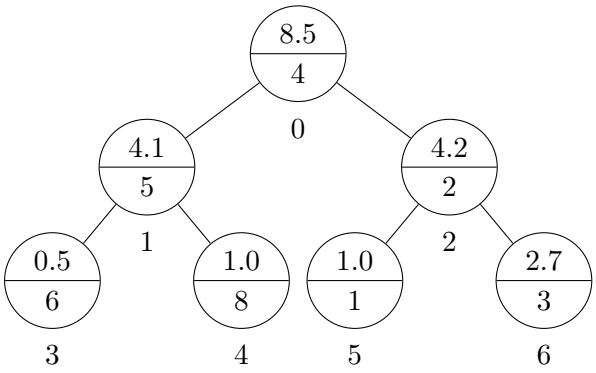


Abb. 5.4.: Residuum im Binärbaum

größten Eintrag des Residuums entnehmen (III) und neue Einträge hinzufügen (I). Wenn wir jedoch einen existierenden Eintrag ändern wollen (V), stellt sich das Problem, dass unbekannt ist, ob und wo sich der Eintrag in der Struktur befindet.

Wenn wir die Einträge des Residualvektors aktualisieren, ist uns zwar der im Baum verwendete Schlüssel bekannt, doch ist dieser nicht eindeutig. Da wir jedoch eigentlich einen bestimmten Eintrag i des Residualvektors ändern wollen, müssten wir den kompletten Baum nach dem entsprechenden Knoten durchsuchen, was Kosten von $\mathcal{O}(N)$ zur Folge hätte.

Zur Lösung dieses Problems haben wir uns das folgende Vorgehen überlegt, das geschickt die Anforderungen unseres Algorithmus ausnutzt. Bemerkenswert bei dieser Idee ist die Einfachheit, mit der sich die Kosten für das Finden bestimmter Elemente reduzieren lassen.

Zunächst benötigen wir ein weiteres Array, das in seiner Größe der Dimension des Residuums entspricht. Dieses werden wir als Index verwenden. Jeder Eintrag i entspricht der Zeile i im Residuum und deutet auf die jeweilige Speicherposition im Binärbaum. Wenn der entsprechende Eintrag des Vektors nicht im Baum gespeichert wurde, weil er Null ist, wird -1 in den Index eingetragen.

Für das betrachtete Beispiel ist der Index in Tabelle 5.1 dargestellt.

Index i	0	1	2	3	4	5	6	7	8
Position im Baum	-1	5	2	6	0	1	3	-1	4

Tabelle 5.1.: Index des Beispiel-Heaps

Bei jeder Änderung der Positionen im Baum, die in der *heapify* Operation vorgenommen wird, wird in diesem Index eingetragen, gelöschte Werte auf -1 gesetzt und neue Einträge entsprechend ergänzt. Eine Änderung im Array und der Zugriff auf einen Eintrag geschieht in $\mathcal{O}(1)$.

Nun erweitern wir den Heap, um die Anforderung des Gauß-Southwell-Algorithmus erfüllen zu können, einen Wert im Residuenvektor zu ändern, wobei über den Index dieses Wertes zugegriffen werden soll. Wir führen die neue Operation *changeKeyByValue* für den Binär-Heap ein, die erst im Index überprüft, ob der entsprechende Eintrag schon im Baum existiert und diesen, wenn dies nicht der Fall ist, hinzufügt und dessen Position im Index speichert. Ist der Eintrag bereits im Baum vorhanden, wird sein Schlüssel geändert und *heapify* für die entsprechende Position aufgerufen, die die Heap-Eigenschaft des Baums wiederherstellt und den Indexeintrag entsprechend aktualisiert.

Mit dieser Erweiterung des Binär-Heaps sind alle für den Gauß-Southwell nötigen Operationen des Binär-Heaps in $\mathcal{O}(\log N)$ durchführbar.

5.4. Anwendung auf Multilevelsysteme

Wir werden im Folgenden überlegen, wie der Gauß-Southwell-Algorithmus sich in einem Multilevelsystem verhält. Dabei werden wir zuerst untersuchen, ob der Algorithmus von der Struktur des erweiterten Gleichungssystems profitieren kann. Im weiteren Verlauf betrachten wir die vom Gauß-Southwell für das Gleichungssystem gewählte Durchlaufreihenfolge und beobachten diese aus der Sicht der geschachtelten Gitter.

Als Beispiel betrachten wir im Folgenden den Laplace-Operator im Zweidimensionalen auf dem Einheitsquadrat, das wir auf einem uniformen Gitter mit einer quadratischen Triangulierung diskretisiert haben (siehe Abschnitt 3.3).

Die mit dem Galerkin-Verfahren erzeugten Matrizen haben die Eigenschaft, dünn besiedelt zu sein. Diese Eigenschaft überträgt sich auf die Matrix im erweiterten Gleichungssystem des Multilevelverfahrens (siehe Abbildung 4.9). Folglich können wir uns diese Eigenschaft stets zu Nutze machen. Betrachten wir nun, was die Dünnbesiedeltheit in der Praxis bedeutet. In der Tabelle 5.2 ist für verschiedene Multilevelmatrizen die Dimension der Matrix und die Belegung in Prozent eingetragen.

Level	Dimension	Belegung
2	10×10	60%
3	59×59	21%
4	284×284	6,3%
5	1245×1245	1,8%
6	5214×5214	0,5%
7	21343×21343	0,1%

Tabelle 5.2.: Belegung der Steifigkeitsmatrizen des erweiterten Gleichungssystems

Um das Verhalten des Gauß-Southwell genauer in Augenschein zu nehmen, müssen wir die Struktur der Multilevelmatrix detaillierter untersuchen. Die verschiedenen in der Matrix ge-

schachtelten Level nehmen unterschiedlich große Blöcke innerhalb der erweiterten Matrix ein.

Block zum Level	Anteil an der Matrix	Belegung des Blocks	Durchschnittl. Anzahl Elemente pro Zeile
1	$\ll 1\%$	13 %	2938
2	$\ll 1\%$	5 %	1143
3	$\ll 1\%$	2 %	437
4	1%	0,8 %	192
5	4%	0,3 %	85
6	18%	0,2 %	39
7	76%	0,09 %	19

Tabelle 5.3.: Besiedeltheit der Blöcke einer Multilevelmatrix des Levels 7

Man sieht deutlich, dass der größte Teil der Matrix vom feinsten Level beansprucht wird, welches auch die geringste Besiedelung hat. Daraus kann man folgern, dass für Algorithmen, die die Dünnbesiedeltheit der Matrix in einem Gleichungssystem ausnutzen können, das Abarbeiten der Zeilen mit jedem gröberen Level immer teurer wird. Für den Gauß-Southwell-Algorithmus gilt im Besonderen, dass die Relaxation eines feinen Levels viel weniger Änderungen im Residuenvektor nach sich ziehen als eine in einem groben Level.

Damit ist die vom Gauß-Southwell gewählte Durchlaufreihenfolge für uns von großem Interesse. Wir untersuchen diese im erweiterten Gleichungssystem.

Der Übersichtlichkeit halber betrachten wir nur ein Erzeugendensystem mit 5 Leveln. Wie oben nehmen wir dazu wir den Laplace-Operator im Zweidimensionalen, den wir diskretisiert und zur Multilevelmatrix erweitert haben.

Die Besiedeltheit dieser Matrix setzt sich wie in Tabelle 5.4 aufgezeigt zusammen. Die Verteilung der Matrixeinträge über die einzelnen Blöcke der Level ist sehr ähnlich zum Beispiel mit 7 Leveln. Auch hier nimmt das dünn besiedelte feinste Level den größten Teil der Matrix ein. Die verhältnismäßig nicht so dünn besiedelten Level haben dagegen einen nur sehr geringen Anteil.

Block zum Level	Anteil an der Matrix	Belegung des Blocks	Durchschnittl. Anzahl Elemente pro Zeile
1	$\ll 1\%$	17 %	222
2	$< 1\%$	12 %	149
3	4%	6 %	71
4	18%	2 %	34
5	77%	1 %	23

Tabelle 5.4.: Besiedeltheit der Blöcke einer Multilevelmatrix des Levels 5

Um die rechte Seite aufzustellen, wählen wir eine glatte, zweimal differenzierbare Funktion. Auf die kontinuierliche Funktion wenden wir den Laplace Operator an und werten sie dann an den Gitterpunkten des feinsten Levels aus. Da wir die rechte Seite für das erweiterte Gleichungssystem benötigen, wenden wir den Transformationsoperator an (siehe Abschnitt 4.2.2). Damit erhalten wir eine rechte Seite in der Form wie wir sie hier benötigen.

Man beachte, dass wir einen Zyklus, der im Aufwand einem Schritt des Gauß-Seidel- oder Jacobi-Verfahrens entsprechen soll, auf die Länge von N Einzelschritten gesetzt haben, wobei N die Anzahl der Unbekannten ist.

Wir starten mit einem randomisierten Startvektor $x^{(0)}$. Um die zufälligen Effekte, die durch diese Wahl auftreten können, zu minimieren, ignorieren wir bei dieser Analyse der Durchlaufreihenfolge den ersten Zyklus. Die darauf folgenden sechs Zyklen reichen aus, um einen Eindruck von der Verteilung der Relaxationen, die der Gauß-Southwell-Algorithmus vornimmt, zu gewinnen.

Block zum Level	Zeilen im Block	Schritte im Block	Anteil an Schritten gesamt
1	1	3	$\ll 1\%$
2	9	45	$<1\%$
3	49	243	3%
4	225	1013	14%
5	961	6165	82%

Tabelle 5.5.: Schrittverteilung

In der Tabelle 5.5 und Abbildung 5.5 lässt sich erkennen, dass der Algorithmus die meisten Schritte auf dem feinsten Level vornimmt. Dabei muss man beachten, dass die feineren Level häufiger besucht werden, als das beim Gauß-Seidel-Algorithmus der Fall wäre. Besucht der Gauß-Seidel-Algorithmus dagegen in sechs Zyklen das gröbste Level jedesmal, greift der Gauß-Southwell-Algorithmus hier nur halb so oft darauf zu.

Ebenfalls ist von Interesse, wie der Gauß-Southwell-Algorithmus durch die Level läuft. Betrachtet man Abbildung 5.6, so lässt sich erkennen, dass der Algorithmus die meisten Relaxationen auf dem feinsten Level vornimmt. In größeren Abständen werden gröbere Level relaxiert. In diesem Beispiel kann man auch eine Art „W-Zyklus“ erkennen, in welchem der Algorithmus bis auf das gröbste Level herabsteigt.

Welche Schlussfolgerungen kann man aus diesem Verhalten ziehen? Einige Probleme, die in Abschnitt 4.4 beschrieben werden, kann der Gauß-Southwell-Algorithmus besser meiden als die Verfahren, die statisch durch das Gleichungssystem laufen. Wie man im vorangegangenen Beispiel erkennen kann, arbeitet der Gauß-Southwell automatisch seltener auf den gröberen Leveln, weil der Gewinn durch die Relaxationen dort nicht groß genug ist. Nur wenn durch die Schritte auf feinen Gittern das Residuum auf den groben Gittern groß genug geworden ist, werden dort Relaxationen durchgeführt. Demnach können wir daraus schließen, dass das Verfahren sich adaptiv an das erweiterte Gleichungssystem anpassen und dessen Struktur ausnutzen kann.

Theoretische Aussagen darüber, wie dieses Verhalten die Konvergenz des Verfahrens beeinflusst, lassen sich nur schwer machen. Wie wir aber schon festgestellt haben, sind die Überlegungen zur theoretischen Konvergenz für uns von keinem so großen Interesse. Entscheidend für uns ist es, wie praktikabel ein Verfahren ist. In Kapitel 6 untersuchen wir die Konvergenz des Gauß-Southwell-Verfahrens in der Praxis und vergleichen sie mit dem klassischen Gauß-Seidel-Iterationsverfahren.

5.5. Vergleich mit dem Gauß-Seidel-Verfahren

Wir wollen nun die Kosten des Gauß-Southwell-Verfahrens in Bezug zu den Kosten des Gauß-Seidel-Verfahrens setzen, welches wir im Kapitel 6 als Referenz verwenden. Um die Kosten der beiden Algorithmen vergleichen zu können, ist es zunächst nötig zu überlegen, inwiefern die Algorithmen sich ähnlich sind. Im Folgenden bezeichnen wir mit N immer die Größe des Iteriertenvektors, also die Anzahl der Unbekannten. Mit M bezeichnen wir die mittlere Anzahl

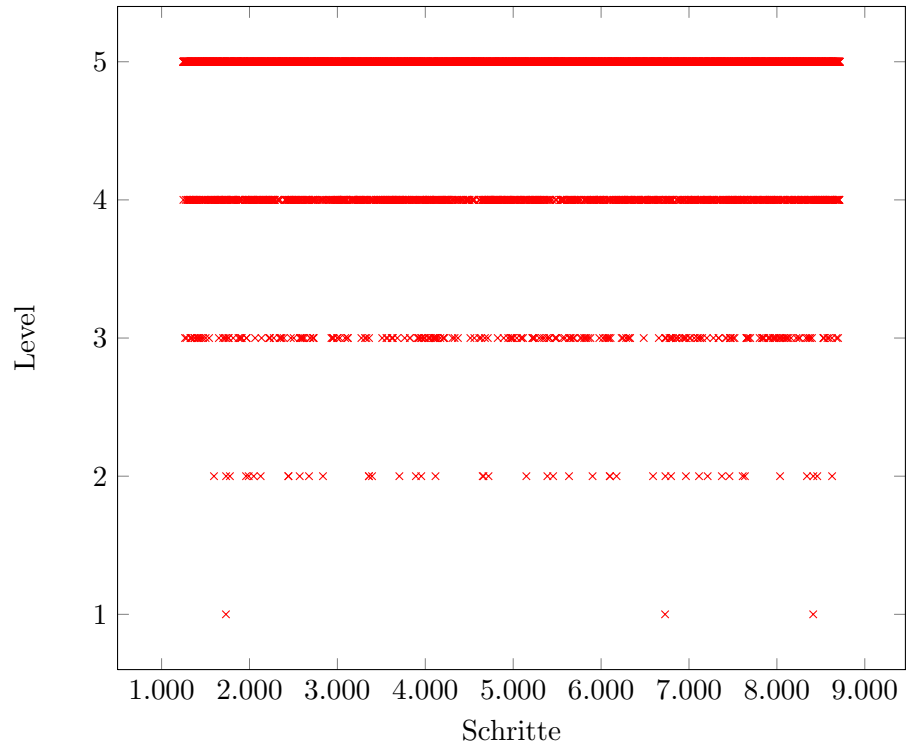


Abb. 5.5.: Verteilung der einzelnen Relaxationsschritte auf die Level der erweitertern Matrix

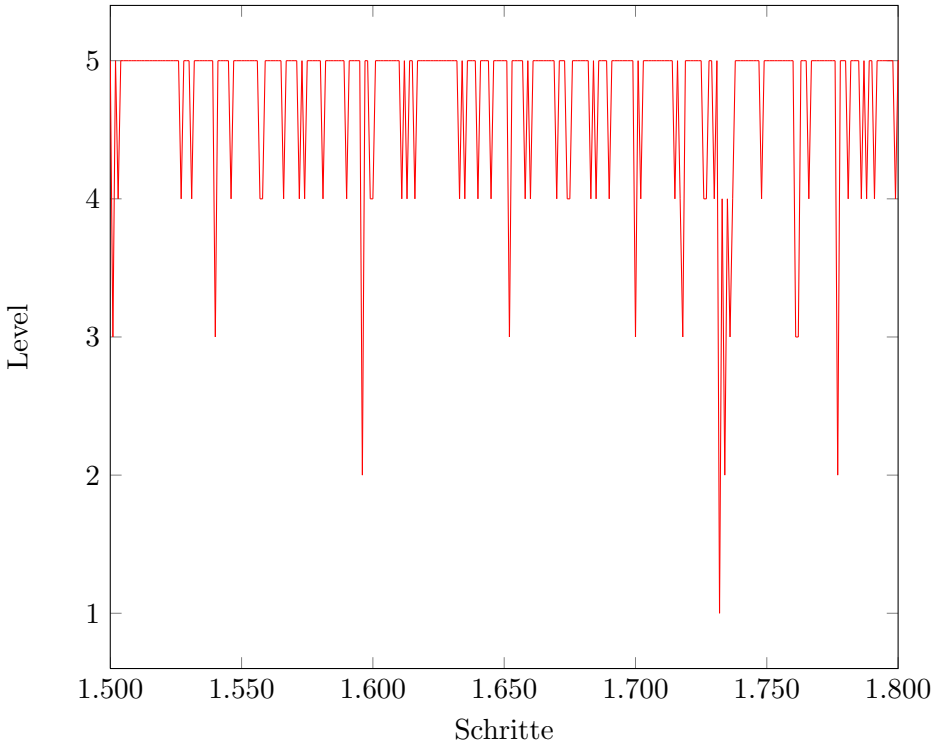


Abb. 5.6.: Durchlaufreihenfolge – „V-Zyklen“ des Gauß-Southwell-Algorithmus

der Nicht-Null-Einträge in einer Zeile der Matrix A des Gleichungssystems. Da wir in dieser Arbeit erweiterte Gleichungssysteme lösen möchten, ist die Matrix A des Gleichungssystems stets dünn besiedelt. Daraus folgt, dass M sehr viel kleiner ist als N, wie wir im vorhergehenden Abschnitt bei der Untersuchung der Gestalt der Matrix sehen konnten.

Der Gauß-Seidel-Algorithmus (siehe Algorithmus 3 in Abschnitt 2.4.3.2) relaxiert in jedem Iterationsschritt jede Komponente der Iterierten x, wogegen der Gauß-Southwell in jedem Schritt nur eine einzelne Komponente x_{i_k} relaxiert. Wir interessieren uns für die Kosten der Relaxation einer einzelnen Komponente und vergleichen diese für beide Algorithmen miteinander. Für den Gauß-Seidel-Algorithmus betragen diese Kosten $\mathcal{O}(N)$, da in jedem Einzelschritt jeweils das Produkt von den Elementen x_j der Iterierten und der Komponenten der entsprechenden Matrixzeile aufsummiert werden müssen. Wenn der Algorithmus die Dünnbesiedeltheit der Matrix ausnutzen kann, reduzieren sich die Kosten auf $\mathcal{O}(M)$ für einen einzelnen Relaxationsschritt.

Verfahren	Kosten			Gesamt
Gauß-Seidel	$\mathcal{O}(N)$			$\mathcal{O}(N)$
Gauß-Seidel (dünn)	$\mathcal{O}(M)$			$\mathcal{O}(M)$
	(III)	+ [(IV)	· (V)]	
Gauß-Southwell mit lin. Suche	$\Theta(N)$	$\mathcal{O}(N)$	$\mathcal{O}(1)$	$\Theta(N)$
Gauß-Southwell mit lin. Suche (dünn)	$\Theta(N)$	$\mathcal{O}(M)$	$\mathcal{O}(1)$	$\Theta(N)$
Gauß-Southwell mit Heap	$\mathcal{O}(\log N)$	$\mathcal{O}(N)$	$\mathcal{O}(\log N)$	$\mathcal{O}(N \log N)$
Gauß-Southwell mit Heap (dünn)	$\mathcal{O}(\log N)$	$\mathcal{O}(M)$	$\mathcal{O}(\log N)$	$\mathcal{O}(M \log N)$

Tabelle 5.6.: Vergleich der Kosten, die von Gauß-Seidel und Gauß-Southwell für einen einzelnen Relaxationsschritt aufgewandt werden müssen. Mit N wird die Zahl der Unbekannten, mit M die Zahl der Nicht-Null-Elemente in der jeweiligen Zeile der dünn besiedelten Matrix bezeichnet. Es gilt $M \ll N$.

Der Gauß-Southwell (siehe Algorithmus 7) muss dagegen in jedem Schritt deutlich mehr Aufwand betreiben. Zunächst muss in (III) der größte Eintrag im Residuenvektor bestimmt werden. Durch den Einsatz des modifizierten Binär-Heaps ließen sich diese Kosten von $\Theta(N)$ für die lineare Suche auf $\mathcal{O}(\log N)$ reduzieren. Doch auch hier muss über das Produkt der Elemente der Iterierten x und der entsprechenden Einträge der Matrixzeile summiert werden. Innerhalb der for-Schleife (IV) wird diese und eine weitere Operation ausgeführt. Diese weitere Operation (V) ist das Aktualisieren der entsprechenden Einträge des Residuums. Die Kosten hierfür betragen beim Einsatz unseres Heaps $\mathcal{O}(\log N)$. Demzufolge ergeben sich Kosten von $\mathcal{O}(N \log N)$ für die for-Schleife (IV), und wenn die Dünnbesiedeltheit ausgenutzt werden kann $\mathcal{O}(M \log N)$. Diese Kosten sind entscheidend für die Analyse, da sie die Kosten für das Finden des größten Eintrags im Residuenvektor dominieren.

Die Kosten für die verschiedenen Fälle und Verfahren sind in Tabelle 5.6 eingetragen. Der Unterschied zwischen dem Verfahren von Gauß-Seidel und dem von Gauß-Southwell liegt darin, dass im Gauß-Seidel-Verfahren eine statische Durchlaufreihenfolge beibehalten wird, wogegen Gauß-Southwell sich adaptiv an das Gleichungssystem anpasst und mit Hilfe des Residuums die Durchlaufreihenfolge dynamisch festlegt. Der Mehraufwand, den das Verfahren dafür betreiben muss, können wir nach diesen Überlegungen beziffern, er geht mit dem Faktor $\mathcal{O}(\log N)$ in die Kosten des Verfahrens ein.

Im Folgenden gilt es zu untersuchen, ob sich diese höheren Kosten im Vergleich zum Gauß-Seidel lohnen.

Wie gut der Gauß-Southwell von der Adaptivität profitieren kann, analysieren wir im anschließenden Kapitel, in dem wir das Lösungsverhalten beider Verfahren für verschiedene Modellprobleme und die daraus resultierenden Konvergenzraten untersuchen werden.

6. Numerische Ergebnisse

In den vorherigen Kapiteln haben wir Verfahren zur Lösung von partiellen Differentialgleichungen vorgestellt und uns theoretisch mit ihnen auseinander gesetzt. Die Diskretisierung der partiellen Differentialgleichungen führt zu Gleichungssystemen, die zu lösen sind. Dazu setzen wir Iterationsverfahren ein, mit denen wir uns in diesem Kapitel in der Praxis befassen. Wir werden für verschiedene Problemstellungen den Gauß-Southwell-Algorithmus auf seine Konvergenzrate untersuchen, um diese dann mit den Raten zu vergleichen, die das Gauß-Seidel-Verfahren erreichen kann. Im weiteren Verlauf werden wir analysieren, wie sich die adaptive Durchlaufreihenfolge auf die Robustheit des Gauß-Southwell auswirkt und wie sich dessen Abhängigkeit von den Parametern der Differentialgleichung im Vergleich zum Gauß-Seidel verhält.

Zunächst behandeln wir die genaue Aufgabenstellung, die es zu lösen gilt, und betrachten die Möglichkeiten, den Iterationsfehler zu messen und die Konvergenzrate zu untersuchen.

6.1. Aufgabenstellung

Die Aufgabe, die wir uns stellen, ist die, eine partielle Differentialgleichung zu lösen. Derartige Aufgabenstellungen resultieren beispielsweise aus der mathematischen Modellierung physikalischer Prozesse. Gegeben sind dabei Differentialoperatoren und eine rechten Seite. Abhängig von diesen wird ein Gleichungssystem

$$Ax = f$$

aufgestellt, für das wir mit einem iterativen Lösungsverfahren eine Näherungslösung berechnen wollen. Die Matrix A resultiert aus der Diskretisierung der Differentialoperatoren der Aufgabenstellung, f stellt die rechte Seite dar. Wir diskretisieren die Differentialoperatoren mittels des Galerkin-Verfahrens und der Finiten Elemente (siehe Kapitel 3). Dabei betrachten wir auch in diesem Kapitel das Einheitsquadrat als zweidimensionales Gebiet.

Wir wollen die Konvergenzrate der Iterationsverfahren untersuchen, welche wir über den Iterationsfehler definiert haben (siehe Abschnitt 2.4.4). Die beste Möglichkeit, den Fehler zu messen, haben wir, wenn wir die exakte Lösung des Problems kennen. Dies ist natürlich im Allgemeinen nicht der Fall, da sich das Lösen der Differentialgleichung erübrigt, wenn man die Lösung auch analytisch berechnen kann. Da unser Ziel hier jedoch nicht die Lösung eines realen Problems ist, können wir eine Problemstellung wählen, deren Lösung uns bekannt ist. Zu diesem Zweck benötigen wir eine geeignete kontinuierliche Modellfunktion.

Bevor wir die rechte Seite aufstellen, ist eine Vorüberlegung nötig. Wir diskretisieren stets mit dem Galerkin-Verfahren, welches die Differentialgleichung mit einer Testfunktion v multipliziert und mit Integration in die schwache Form überführt. Für einen allgemeinen Differentialoperator lautet die schwache Form

$$a(u, v) = l(v),$$

dabei ist

$$l(v) = \int_\Omega f v \, dV \quad \forall v \in V$$

(siehe Abschnitt 3.1). Betrachten wir nun die rechte Seite. Analog zum Vorgehen in Kapitel 3 diskretisieren wir diese, indem wir f zur Basis $\{\varphi_i\}_{i=1,\dots,N}$ darstellen, und überführen sie somit in den endlichdimensionalen Raum V_N

$$\int_\Omega \sum_i f_i \varphi_i \, v \, dV \qquad \forall v \in V_N.$$

Wie in Kapitel 3 wählen wir als Testfunktion die Basis, also $v = \varphi_j \ \forall j = 1, \dots, N$. Damit gilt

$$\int_\Omega \sum_i f_i \varphi_i \varphi_j \, dV = \sum_i f_i \underbrace{\int_\Omega \varphi_i \varphi_j \, dV}_{M(i,j)} \qquad \forall \, j = 1, \dots, N.$$

Das Integral $M(i,j)$ lässt als Matrix darstellen und wird dann als Massenmatrix M bezeichnet (siehe Abschnitt 3.3.1).

Nun können wir die rechte Seite unseres Modellproblems diskret aufstellen. Man wendet den Differentialoperator dafür auf die kontinuierliche Modellfunktion an, wertet diese dann an den Gitterpunkten aus und multipliziert den daraus resultierenden Vektor mit der Massenmatrix M.

Auf diese Weise können wir für beliebige Differentialoperatoren Gleichungssysteme aufstellen, deren Lösung wir kennen, und so stets den Iterationsfehler berechnen.

6.1.1. Variable Parameter

Wenden wir uns nun der Möglichkeit zu, das Gleichungssystem mit einem Parameter Epsilon zu versehen, um eine singuläre Störung erzeugen zu können. In [Rü93] wird die Problemstellung derart konstruiert, dass der Operator nicht von einem Parameter beeinflusst werden kann, statt dessen besteht die Möglichkeit, die rechte Seite zu stören

$$Ax_\epsilon = f_\epsilon,$$

wodurch auch die Lösung von Änderungen des Parameters betroffen ist.

Es werden also nur reine Operatoren eingesetzt, so dass die Konsistenz des Gleichungssystems unbeeinflusst bleibt und die Konstante der Fehlerabschätzung [Hac96] unabhängig von Epsilon ist. Außerdem ist die Multilevelmatrix für verschiedene Epsilon identisch und somit auch ihre Kondition, da der Operator sich nicht ändert. Allerdings wird die Regularität der Lösung beeinflusst.

Wir konstruieren unsere Problemstellung hingegen so, dass der Operator selbst gestört wird. Diese Form der Aufgabenstellung ist besonders praxisrelevant, da physikalische Modelle häufig mit flexiblen Parametern ausgestattet sind. Die Lösung lassen wir jedoch glatt

$$A_\epsilon x = f_\epsilon.$$

Das hat zur Folge, dass eine Änderung des Parameters Epsilon eine Änderung der Konstante $c(\epsilon)$ der Fehlerabschätzung, der Konsistenz und Stabilität des Verfahrens und der Kondition der Multilevelmatrix bewirkt. So können wir das Gauß-Southwell-Verfahren auf Operatorrobustheit untersuchen.

6.1.2. Modellfunktion

Der Einfachheit halber fordern wir von der Modellfunktion, die die Lösung der partiellen Differentialgleichung sei, dass sie Null auf dem Rand des Gebiets und genügend glatt ist. Inbesondere sollte die Funktion zweimal stetig differenzierbar sein, da wir Differentialoperatoren zweiten Grades einsetzen wollen. Außerdem fordern wir, dass die zweite Ableitung der Modellfunktion nicht verschwindet, da sonst die Nullfunktion als triviale Lösung existiert.

Man beachte auch, dass wir die Funktion nicht nur nutzen, um die diskrete Lösung der Differentialgleichung zu berechnen, sondern auch, um die rechte Seite des Gleichungssystems aufzustellen.

Wir wählen die Funktionen

$$p(x,y) := x(1-x)\cdot y(1-y) \quad \text{und} \quad q(x,y) := \sin(2\pi x)\sin(2\pi y)$$

als Modellfunktionen. Mit dieser Wahl haben wir mit Polynomen und den trigonometrischen Funktionen zwei große Familien von Funktionen abgedeckt. Diese sind zweimal differenzierbar und erfüllen für unser Gebiet alle geforderten Bedingungen. In Abbildung 6.1 sind die Funktionen dargestellt.

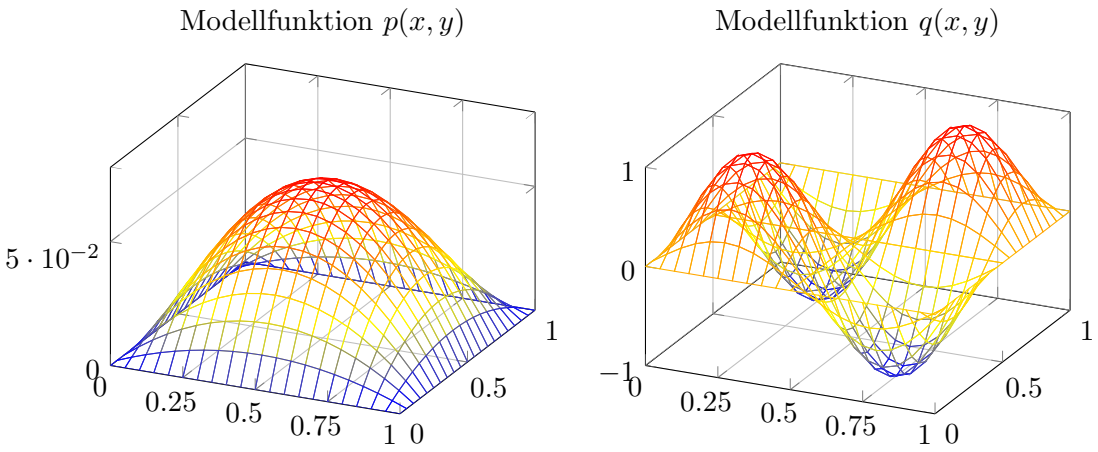


Abb. 6.1.: Zweidimensionale Modellfunktionen $p(x,y) = x(1-x)\cdot y(1-y)$ und $q(x,y) = \sin(\pi x)\sin(\pi y)$ auf dem Einheitsquadrat

6.1.3. Iterationsfehler

Um den Iterationsfehler einer Näherungslösung zu bestimmen, benötigen wir eine Referenzlösung. Diese erhalten wir bei unseren Modellproblemen dadurch, dass wir die Modellfunktion an den Gitterpunkten der Diskretisierung auswerten. Um den absoluten Iterationsfehler $e^{(m)}$ im Schritt m zu bestimmen, bilden wir die Differenz von der Näherungslösung $x^{(m)}$ und der ausgewerteten Referenzlösung u mit der Maschenweite h

$$e^{(m)} = \left|x^{(m)} - I_h u\right|.$$

Der dabei entstehende Vektor $x^{(m)} - I_h u$ enthält für jeden Gitterpunkt den genauen Iterationsfehler im Schritt m. Um die Größe des gesamten Fehlers messen zu können, benötigen wir noch eine Norm $|\cdot|$, auf die wir im nächsten Abschnitt genauer eingehen.

Um zu vermeiden, dass Superkonvergenzeffekte auftreten, messen wir den Fehler nicht auf dem aktuellen Gitter, sondern stets auf einem feineren Gitter. Dazu prolongieren wir die Iterierte $x_l^{(m)}$, die auf Level l errechnet worden ist, auf das feinere Level k mit $k > l$, werten die Referenzlösung u entsprechend auf diesem Level aus und bilden die Differenz zwischen der Lösung und der prolongierten Iterierten

$$e_l^{(m)} = \left| P_l^k x_l^{(m)} - I_k u \right|.$$

Wenn wir in jedem Schritt eines Iterationsverfahrens den Iterationsfehler messen, können wir damit die Konvergenzrate des Verfahrens für die aktuelle Problemstellung bestimmen (siehe Abschnitt 2.4.4).

6.1.4. Normen

Wenn wir den Fehler, wie im vorangegangenen Abschnitt beschrieben, berechnet haben, erhalten wir einen Vektor, der den Fehler in jedem Punkt des Gitters enthält. Um die Größe dieses Vektors und damit den Iterationsfehler zu messen, benötigen wir eine Norm, also eine reellwertige Funktion, die einem Vektor $x \in \mathbb{R}^n$ eine reelle Zahl $||x||$ zuordnet und eine Reihe weiterer Eigenschaften erfüllt [Sto94].

Wir verwenden in dieser Arbeit zwei Normen. Das ist zum einen die L^2-Norm, die wir unter der Verwendung der Massenmatrix M mit

$$||x||_{L^2}^2 := x^T M x$$

definieren, da für ein kontinuierliches $x \in V$

$$||x||_{L^2}^2 := \int_\Omega |x|^2 dV$$

gilt und sich somit diskret für $x \in V_h$

$$\begin{aligned} ||x||_{L^2}^2 &= \int_\Omega \langle x, x \rangle \, dV \\ &= \int_\Omega \sum_{i=1}^N x_i \varphi_i \cdot \sum_{j=1}^N x_j \varphi_j \, dV \\ &= \sum_{i,j=1}^N x_i x_j \underbrace{\int_\Omega \varphi_i \varphi_j \, dV}_{M(i,j)} \\ &= x^T M x \end{aligned} \tag{6.1}$$

übertragen lässt.

Zum anderen verwenden wir die Energienorm $|| \cdot ||_A$. Die Energienorm ist im Gegensatz zur L^2-Norm abhängig von der Problemstellung. Für den diskreten Laplace-Fall lässt sich die Ener-

gienorm analog zu (6.1) schreiben

$$\begin{aligned} ||x||_A^2 &= \int_\Omega \langle \nabla x, \nabla x \rangle^2 \, dV \\ &= \int_\Omega \sum_{i=1}^N x_i \nabla \varphi_i \cdot \sum_{j=1}^N x_j \nabla \varphi_j \; dV \\ &= \sum_{i,j=1}^N x_i x_j \underbrace{\int_\Omega \nabla \varphi_i \nabla \varphi_j \, dV}_{= a(\varphi_i, \varphi_j) = A(i,j)} \\ &= x^T A x. \end{aligned} \tag{6.2}$$

Nach diesem Prinzip wird die Energienorm allgemein für einen symmetrischen, positiv definiten Operator als die von der Bilinearform $a(\cdot, \cdot)$ induzierte Norm

$$||x||_A^2 := a(x, x) = x^T A x$$

definiert. Für Finite Elemente-Diskretisierungen ist die Energienorm eine natürliche Wahl, da das Finite Elemente-Verfahren bezüglich dieser Norm optimal ist. Genauer misst die Energienorm des Fehlers die minimale Differenz zwischen exakter Lösung und Finite-Elemente-Lösung. Es gilt die Bestapproximation [Hac96]

$$||u - u_h||_A \leq ||u - v_h||_A \quad \forall v_h \in V_h.$$

In der Energienorm liefert die Finite Elemente-Methode also die bestmögliche Näherung an die exakte Lösung. Der Name der Norm ergibt sich physikalisch daraus, dass die Energienorm sich auch als innere Energie der Fehlerfunktion deuten lässt.

Wir wollen nun die Fehlerabschätzungen für beide Normen betrachten und prüfen, wie sich eine Verfeinerung der Maschenweite bei der Diskretisierung auf die Norm auswirkt.

Es gelten die Fehlerabschätzungen aus [Bra97] und [Gri03], das heißt für die L^2-Norm

$$||u - u_h||_{L^2} \leq c \cdot h^2 ||f||_{L^2}$$

eine Konvergenzordnung von $\mathcal{O}(h^2)$ und für die Energienorm

$$||u - u_h||_A \leq c \cdot h ||u||_{H^2}$$

entsprechend die Konvergenzordnung $\mathcal{O}(h)$.

Da wir bei einer Verfeinerung immer die Maschenweite des feinsten Gitters halbieren, interessieren wir uns dafür, wie sich das auf den Fehler auswirkt. Untersuchen wir das zunächst für die L^2-Norm

$$\begin{aligned} ||u - u_{\frac{h}{2}}||_{L^2} &\leq c \cdot \left(\frac{h}{2}\right)^2 ||f||_{L^2} \\ &= \frac{1}{4} \, c \cdot h^2 ||f||_{L^2} \\ &= \frac{1}{4} \, ||u - u_h||_{L^2}. \end{aligned}$$

Demzufolge erreicht man mit der Halbierung der Maschenweite eine Verbesserung des Fehlers in der L^2-Norm um den Faktor 4. Für die Energienorm gilt analog

$$\begin{aligned} ||u - u_{\frac{h}{2}}||_A &\leq c \cdot \frac{h}{2}||u||_{H^2} \\ &= \frac{1}{2}\, c \cdot h||f||_{H^2} \\ &= \frac{1}{2}\, ||u - u_h||_A. \end{aligned}$$

Hier ergibt sich, dass sich der Fehler in der Energienorm bei der Halbierung der Maschenweite um den Faktor 2 verbessert.

Diese Abschätzungen gelten für den Fehler des gesamten Verfahrens, der sich aus dem Diskretisierungs-, dem Iterations- und dem Interpolationsfehler zusammensetzt.

6.2. Diffusion, Reaktion und anisotrope Diffusion im Multilevelsystem

Im Folgenden untersuchen wir das Lösungsverhalten des Gauß-Southwell-Algorithmus für verschiedene Modell-Differentialgleichungen. Wir vergleichen dessen Konvergenzverhalten mit dem des Gauß-Seidel-Verfahrens, mit welchem er eng verwandt ist. Der Gauß-Seidel-Algorithmus folgt einer festen Durchlaufreihenfolge, während unser Algorithmus sich der Struktur des Gleichungssystems anpasst und den nächsten Schritt immer so wählt, dass er den größten Gewinn für die Lösung des Gleichungssystems bringt. Durch diesen Vergleich können wir direkt feststellen, wie wir von der Adaptivität profitieren können und ob sich die daraus resultierenden Kosten rechtfertigen lassen.

Da wir ein Multilevelsystem lösen wollen, setzen wir nicht den klassischen Gauß-Seidel-Algorithmus ein, sondern lassen ihn symmetrisch durch das Gleichungssystem iterieren. Das klassische Verfahren würde immer von den groben zu den feinen Gittern laufen und profitiert somit nicht so gut von den Eigenschaften des Multilevelsystems. Der symmetrische Gauß-Seidel-Algorithmus, der vom feinsten Level zum gröbsten Level und zurück läuft, entspricht einem V-Zyklus im Mehrgitterverfahren (siehe Abschnitt 4.4) und verhält sich deutlich stabiler, als es der klassische Gauß-Seidel tun würde.

Um sicherzustellen , dass wir beide Iterationsverfahren gerecht miteinander vergleichen, zählen wir den V-Zyklus des Gauß-Seidel-Algorithmus und $2N - 1$ Einzelschritte des Gauß-Southwell-Algorithmus als jeweils einen vollen Iterationsschritt dieser Verfahren. So führen beide in einem Iterationsschritt gleich viele Relaxierungsoperationen auf dem Gleichungssystem aus.

6.2.1. Diffusion

Wir beginnen die Untersuchung des Gauß-Southwell-Algorithmus und den Vergleich mit dem symmetrischen Gauß-Seidel-Verfahren mit dem Problem der Diffusion. Die zu lösende Differentialgleichung ist als Poisson-Gleichung bekannt und lautet

$$-\Delta u = f$$

mit zugehörigen Randbedingungen. Die Poisson-Gleichung dient für viele Verfahren als Modellproblem. Wir diskretisieren die Gleichung mittels der Finiten Elemente (siehe Abschnitt 3.3.1).

Die rechte Seite stellen wir, wie in Abschnitt 6.1 beschrieben, mit Hilfe der Modellfunktion $p(x, y)$ auf, die uns hier damit auch als Lösung dient, wobei wir auf dem Rand des Gebiets $u|_{\partial\Omega} = 0$ setzen. Als Startwert nehmen wir einen Vektor mit randomisierten Werten aus dem Intervall $[-1, 1]$.

Wir betrachten den Verlauf des Fehlers für verschiedene Level in der L^2-Norm (Abbildung 6.2) und in der Energienorm (Abbildung 6.3).

Die erste Beobachtung, die wir machen, ist, dass der Iterationsfehler direkt vom Level und damit von der Maschenweite des feinsten Gitters abhängt. Beide Verfahren konvergieren gegen die gleiche Schranke, welche nicht vom Iterationsverfahren, sondern von dem Problem, das dem Gleichungssystem zugrunde liegt, und dessen Diskretisierung herrührt. Wir bezeichnen diese Schranke im Folgenden als Diskretisierungsfehler, wobei jedoch in diesen nicht nur der Fehler der Diskretisierung sondern auch der Interpolationsfehler der Fehlermessung in geringem Maße einfließt.

Mit jeder Verfeinerung der Maschenweite des feinsten Gitters wird der Diskretisierungsfehler geringer. Dabei gilt, dass die L^2-Norm des Diskretisierungsfehlers, der die untere Schranke des Iterationsfehlers darstellt, mit jeder Halbierung der Maschenweite um den Faktor 4, in der Energienorm jeweils um den Faktor 2, kleiner wird. Diese von uns beobachteten Faktoren für die Verbesserung des Fehlers entsprechen den Abschätzungen aus Abschnitt 6.1.4. Dort wurde der Gesamtfehler abgeschätzt, also fließt neben dem Diskretisierungs- und dem Interpolationsfehler auch der Iterationsfehler ein. Da wir unsere Beobachtungen jedoch für den Fall gemacht haben, dass die Verfahren ausiteriert sind, geht dieser gegen Null, folglich entsprechen unsere Beobachtungen aus der Praxis genau dem, was wir in der Theorie ausgerechnet haben.

Wenn wir die Konvergenzrate des Gauß-Seidel-Algorithmus betrachten, erkennen wir, dass sich diese stabil, d.h. unabhängig vom Level des erweiterten Gleichungssystems verhält. In diesem Fall ist die Konvergenzrate $\rho = 0,29$ in der L^2-Norm. Die Entwicklung im ersten Schritt können wir außer acht lassen, da wir stets mit einem randomisierten Startvektor $x^{(0)}$ begonnen haben. Ebenso ist der letzte Schritt, bevor der Algorithmus den Diskretisierungsfehler erreicht, von geringer Bedeutung, da sich der Fehler bereits in der Asymptotik befindet. Der Gauß-Southwell verhält sich auf allen Leveln ähnlich, zeigt jedoch einen unverkennbar steileren Abstieg des Fehlers. Das schlägt sich in der Konvergenzrate mit $\rho = 0,03$ nieder. Analog verhält sich auch die Anzahl der Iterationsschritte.

Wenn wir die Gleichung derart modifizieren, dass wir den Laplace-Operator mit einem $\epsilon > 0$ multiplizieren, lässt sich beobachten, dass sich das Konvergenzverhalten beider Algorithmen nicht ändert. Damit sind die Beobachtungen für dieses Modellproblem beendet, und wir wenden uns einem neuen Problem zu.

6.2.2. Diffusion-Reaktion

Die nächste Differentialgleichung, an der wir das Lösungsverhalten des Gauß-Southwell-Algorithmus untersuchen wollen, ist die Diffusions-Reaktionsgleichung

$$-\Delta u + u = f$$

mit der Randbedingung $u|_{\partial\Omega} = 0$. Wie oben diskretisieren wir die Gleichung mit Hilfe der Finiten Elemente und stellen die Gleichungssysteme wie beschrieben für verschieden feine Gitter mitsamt der rechten Seite auf und transformieren sie in Multilevelsysteme. Analog zur reinen Diffusion starten wir in jedem Level mit einem randomisierten Startvektor.

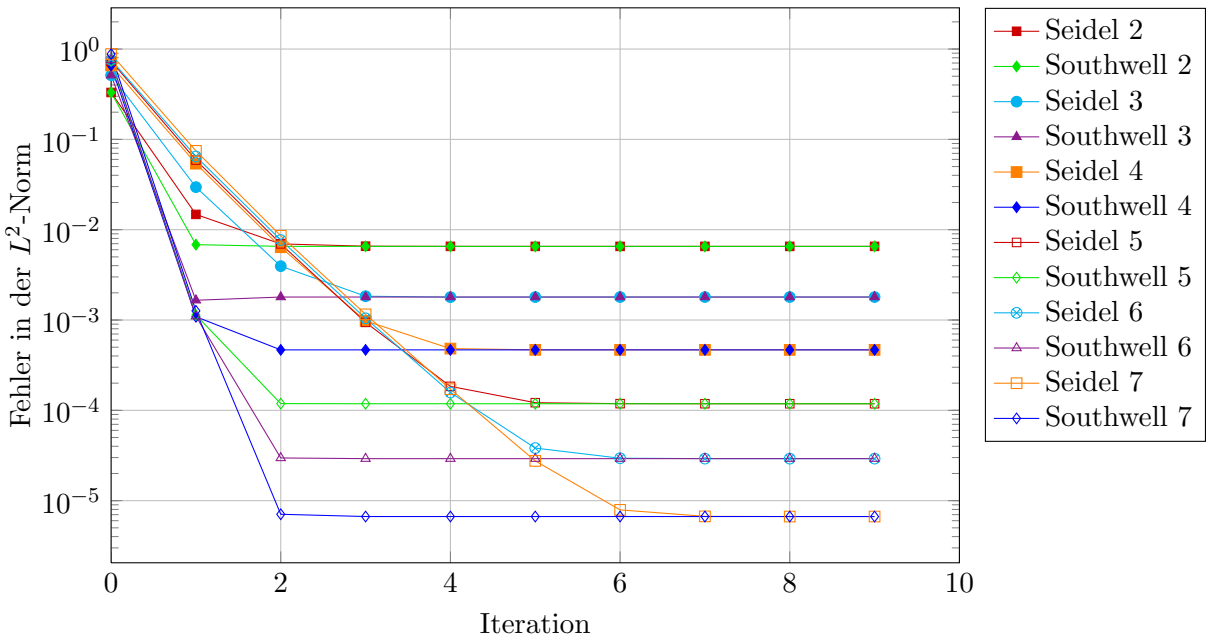


Abb. 6.2.: Vergleich von Gauß-Seidel und Gauß-Southwell für verschiedene Level. Diffusion mit der Modellfunktion $p(x, y)$ als Lösung, gemessen in der L^2-Norm

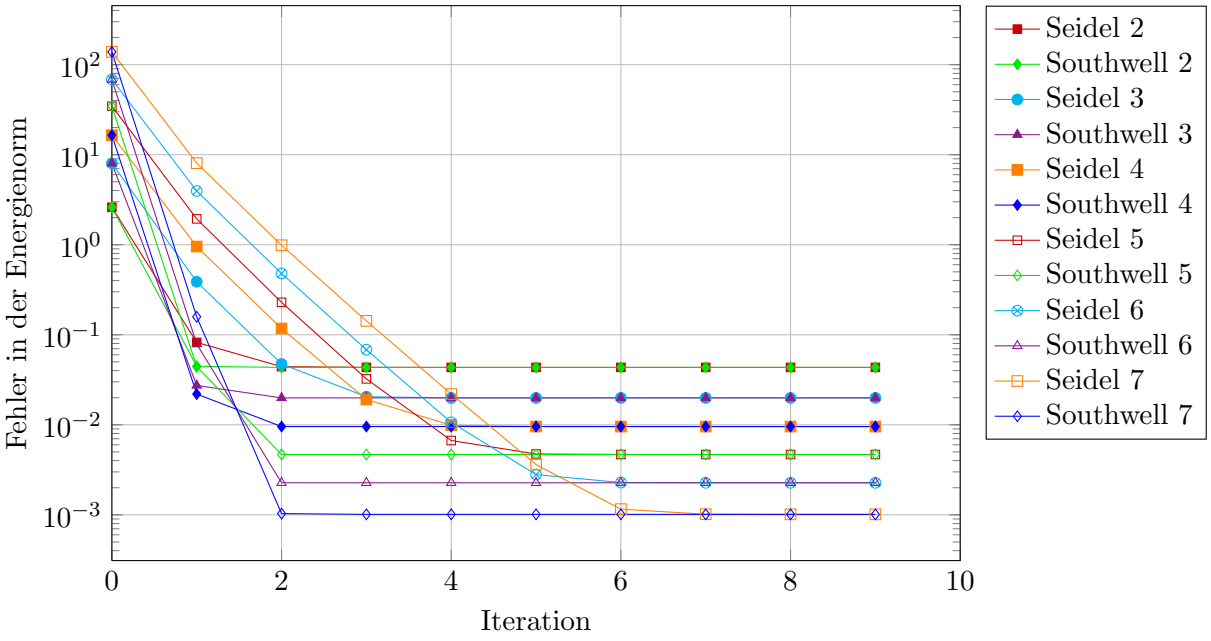


Abb. 6.3.: Vergleich von Gauß-Seidel und Gauß-Southwell für verschiedene Level. Diffusion mit der Modellfunktion $p(x, y)$ als Lösung, gemessen in der Energienorm

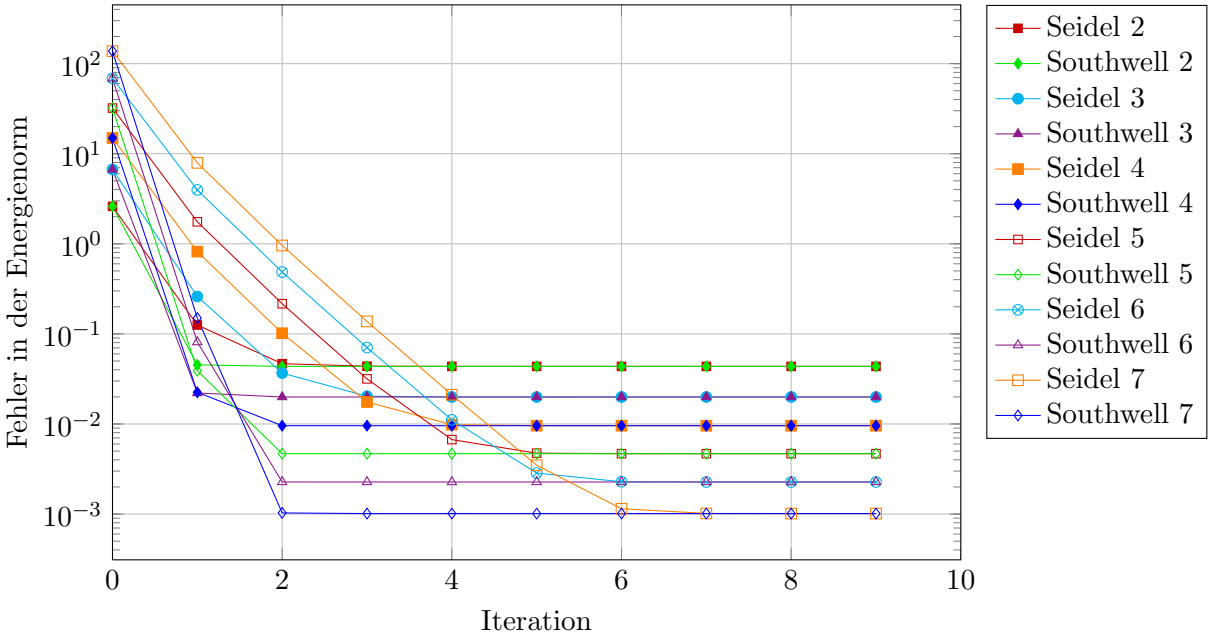


Abb. 6.4.: Vergleich von Gauß-Seidel und Gauß-Southwell für verschiedene Level. Diffusions-Reaktionsgleichung mit der Modellfunktion $p(x, y)$ als Lösung, gemessen in der Energienorm

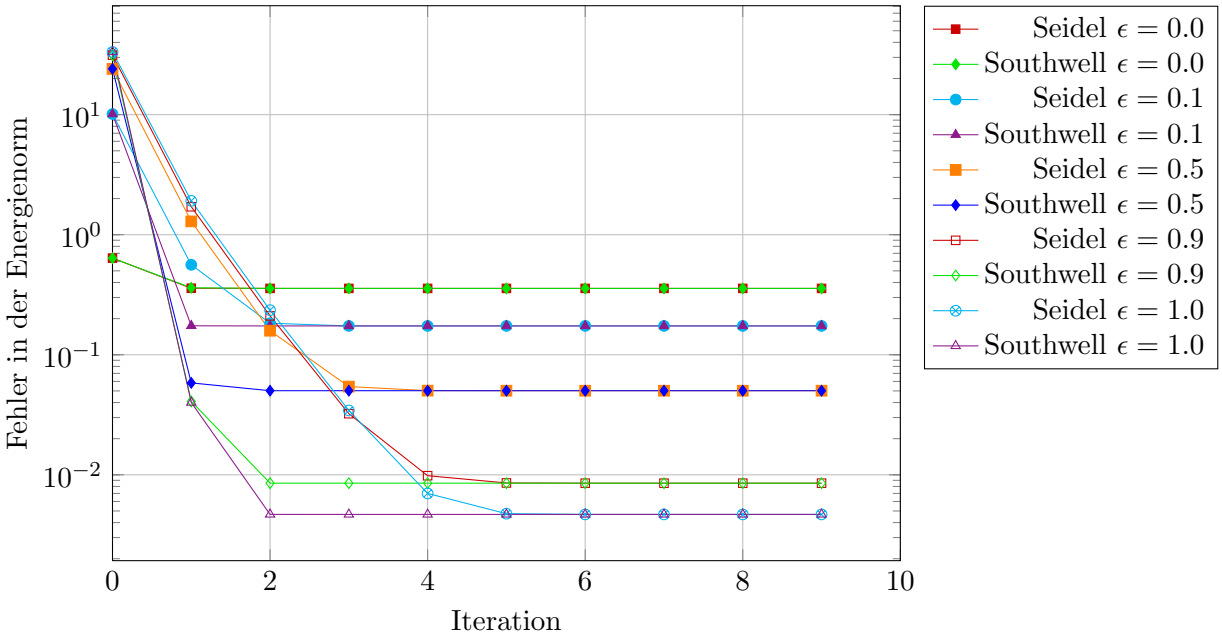


Abb. 6.5.: Vergleich von Gauß-Seidel und Gauß-Southwell im Level 5. Diffusions-Reaktionsgleichung mit ϵ-abhängig gewichteter Diffusion und der Modellfunktion $p(x, y)$ als Lösung, gemessen in der Energienorm

Wir betrachten den Verlauf des Fehlers für verschiedene Level und beobachten, dass der Reaktionsterm keinen nennenswerten Einfluss auf das Konvergenzverhalten der beiden Iterationsverfahren hat und sich somit kaum von dem der reinen Diffusion unterscheidet. Der Verlauf des Fehlers in der Energienorm ist in Abbildung 6.4 abgebildet.

Wie bei der reinen Diffusion erkennt man, dass mit jedem Level der Diskretisierungsfehler kleiner wird, in der abgebildeten Energienorm um den Faktor 2, in der L^2-Norm analog um den Faktor 4. Beide Löser brauchen nur einige Schritte, um den Diskretisierungsfehler zu erreichen, doch auch hier ist der Gauß-Southwell-Algorithmus nach deutlich weniger Schritten am Ziel als der Gauß-Seidel.

Wir untersuchen nun das Verhalten der Iterationsverfahren für einen unterschiedlich starken Einfluss der Diffusion. Zu diesem Zweck modifizieren wir die Diffusions-Reaktionsgleichung soweit, dass ein $\epsilon \geq 0$ vor dem Diffusionsterm steht. Die Gleichung lautet somit

$$-\epsilon \Delta u + u = f.$$

Wir betrachten das Lösungsverhalten der Iterationsverfahren für ein festes Level 5 mit verschiedenen Epsilon in Abbildung 6.5. Als erstes stellt sich heraus, dass der Diskretisierungsfehler bei gleichem Level vom Epsilon der Gleichung abhängt. Da der Reaktionsterm von der Maschenweite abhängig ist, ist der Diskretisierungsfehler umso größer, je stärker der Reaktionsterm Einfluss nimmt.

Für die Iterationsverfahren beobachtet man, dass für kleine Epsilon beide Löser sehr schnell den Diskretisierungsfehler erreichen. Insbesondere fällt auf, dass der Gauß-Seidel-Algorithmus mit jedem kleineren Epsilon immer besser wird, bis er genauso schnell die Lösung erreicht wie der Gauß-Southwell-Algorithmus. Der Grund dafür liegt darin, dass mit kleinerem Epsilon zum einen der Diskretisierungsfehler größer wird, und damit schneller erreicht werden kann, und zum anderen, dass die Gleichung immer einfacher zu lösen wird, wenn der Reaktionsterm stärker ins Gewicht fällt. Für den Grenzfall $\epsilon = 0$, also der reinen Reaktion, entarten beide Verfahren, wie zu erwarten, zu direkten Lösern.

Für den Fall $\epsilon > 1$ ist kein großer Unterschied im Verlauf der Kurven zu sehen. Für immer größere Epsilon wird der Einfluss des Reaktionsterms immer geringer, so dass sich das Konvergenzverhalten der Iterationsverfahren dem der reinen Diffusion angleicht.

Die Diffusions-Reaktionsgleichung stellt also keinen der beiden Löser vor größere Probleme, und wir beobachten die zu erwartende Annäherung des Gauß-Seidel-Verfahrens an den Gauß-Southwell für einen dominanten Reaktionsterm.

Nun wenden wir uns der anisotropen Diffusion zu, die beide Lösungsverfahren vor größere Herausforderungen stellt.

6.2.3. Anisotrope Diffusion

Die Problemstellung, der wir uns nun widmen, ist die der anisotropen Diffusion. O.b.d.A. lassen wir die Diffusion für die x-Richtung fest und variieren die Diffusion in y-Richtung mittels eines Epsilon. Für $\epsilon = 1$ entspricht die Gleichung der reinen Diffusion (siehe Abschnitt 6.2.1).

Die Gleichung dieser Problemstellung lautet

$$-(\partial_{xx} u + \epsilon \cdot \partial_{yy} u) = f,$$

erneut mit der Randbedingung $u|_{\partial\Omega} = 0$.

Wie zuvor diskretisieren wir die Gleichung mit Hilfe der Finiten Elemente und stellen die Gleichungssysteme für verschieden feine Gitter mitsamt der rechten Seite auf und transformieren sie in Multilevelsysteme. Analog zu den vorherigen Beispielproblemen starten wir in jedem Level mit einem randomisierten Startvektor.

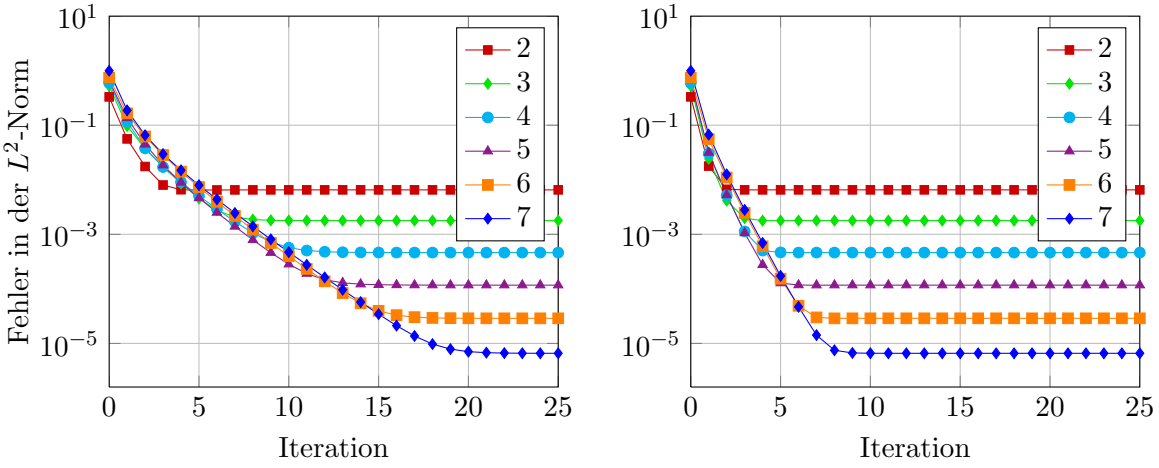
Bevor wir das Verhalten der iterativen Löser für verschiedene Epsilon untersuchen, stellen wir einige Überlegungen zur anisotropen Diffusion im Multilevelsystem an. Bei der Aufstellung dieses Systems wird wie üblich das feine Ausgangsgitter mit gröberen Gittern zusammengefasst, wobei wir mittels Prolongation und Restriktion zwischen jeweiligen Leveln interpolieren. Dabei sind sowohl die Prolongation als auch die Restriktion isotrop konstruiert, ebenso wie das Gitter, das uniform gewählt ist. Dieses Prinzip behalten wir um der Allgemeinheit willen bei, so dass zu erwarten ist, dass für den nun folgenden Fall, dass ein anisotroper Differentialoperator diskretisiert wird, das Problem deutlich schwieriger zu lösen ist.

Natürlich könnte man ein auf das Problem angepasstes Verfahren mit anisotropen Gittern und anisotropen Interpolationen konstruieren, jedoch wollen wir hier das Verhalten der Iterationsverfahren für den Fall untersuchen, dass weder die Diskretisierung noch das Multilevelverfahren a priori an das Problem angepasst wurden. Folglich können wir erwarten, dass das Verfahren langsamer konvergiert als bei der isotropen Diffusion und die Konvergenzrate mit stärker werdender Anisotropie schlechter wird. Sehen wir uns nun die konkreten Ergebnisse an.

Der Fall $\epsilon = 1$ entspricht der reinen Diffusion, die wir in Abschnitt 6.2.1 behandelt haben. Wir analysieren den Fall $\epsilon = 10$ in der L^2-Norm. Betrachtet man den Verlauf der Fehler in Abbildung 6.6, erkennt man, dass der Diskretisierungsfehler wieder vom Level des erweiterten Gleichungssystems und damit von der Maschenweite des feinsten Gitters abhängt. Die nächste Beobachtung, die wir machen, ist, dass beide Algorithmen in allen Leveln jeweils eine sehr ähnliche Konvergenzrate haben. Der Fehler fällt jedoch beim Gauß-Southwell-Algorithmus deutlich steiler ab als beim Gauß-Seidel-Verfahren, so dass der Diskretisierungsfehler auf jedem Level nach viel weniger Schritten erreicht ist. Dieser Unterschied wird deutlicher, je größer das Level ist. Im Level 7 des erweiterten Gleichungssystems ist für den Gauß-Seidel der Diskretisierungsfehler erst nach über 20 Iterationsschritten erreicht, während der Gauß-Southwell schon nach 10 Schritten am Ziel ist. Hier zeigt sich, dass die adaptive Durchlaufreihenfolge von großem Vorteil gegenüber der statischen Durchlaufreihenfolge des symmetrischen Gauß-Seidel ist.

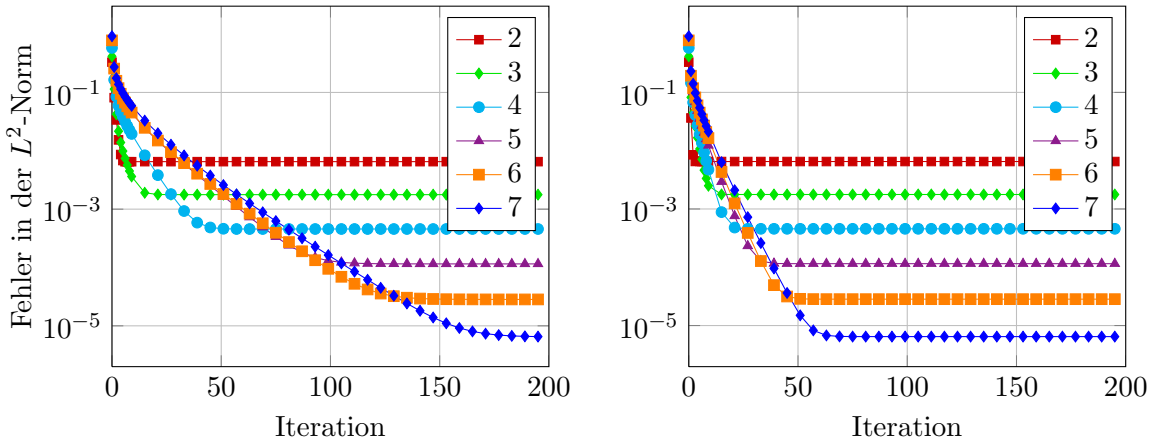
Die anisotrope Diffusion stellt die Iterationsverfahren vor wesentlich größere Herausforderungen als die vorhergegangenen Beispiele. Mit der Wahl des Epsilon können wir den Grad der Anisotropie beeinflussen und somit den Schwierigkeitsgrad erhöhen. Wir betrachten den Verlauf des Fehlers in der L^2-Norm für $\epsilon = 100$ in Abbildung 6.7 und für $\epsilon = 1000$ in Abbildung 6.8. Obwohl die Anzahl der Iterationsschritte für beide Verfahren stark ansteigt, erkennt man, dass der Gauß-Southwell-Algorithmus immer deutlich schneller den Diskretisierungsfehler erreicht. Für $\epsilon = 100$ erreicht der Gauß-Southwell im Level 7-System die Asymptote nach knapp 70 Schritten, wogegen der Gauß-Seidel über 200 Schritte benötigt. Im Fall $\epsilon = 1000$ auf dem Level 7 wird der Schwierigkeitsgrad der anisotropen Diffusion noch deutlicher, aber auch hier benötigt das adaptive Verfahren mit 500 Schritten viel weniger Iterationen als das entsprechende Verfahren im V-Zyklus mit 1600 Schritten.

Misst man dieselben Fälle in der Energienorm, stellt man ein ähnliches Verhalten fest, vergleiche Abbildungen 6.9-6.11. Hier lässt sich allerdings noch eine weitere interessante Beobachtung in Bezug auf den Diskretisierungsfehler machen. In der Energienorm ist der Diskretisierungsfehler für ein festes Level und variierendes Epsilon abhängig von Epsilon, also vom Grad der Anisotropie. Das ist auch zu erwarten, da die Energienorm ja über die Matrix A definiert ist.



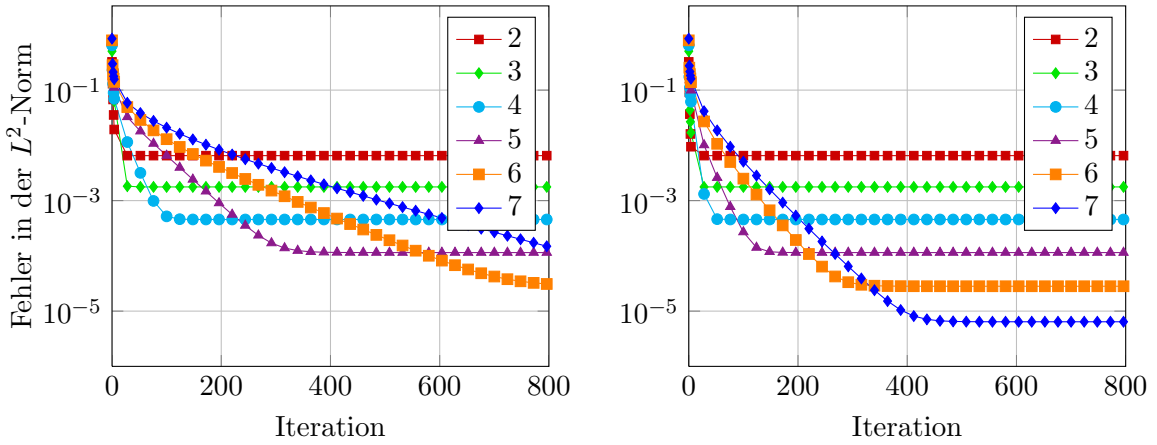
(a) Gauß-Seidel (b) Gauß-Southwell

Abb. 6.6.: Vergleich von Gauß-Seidel und Gauß-Southwell für verschiedene Level. Anisotrope Diffusion mit $\epsilon = 10$ und der Modellfunktion $p(x, y)$ als Lösung, gemessen in der L^2-Norm



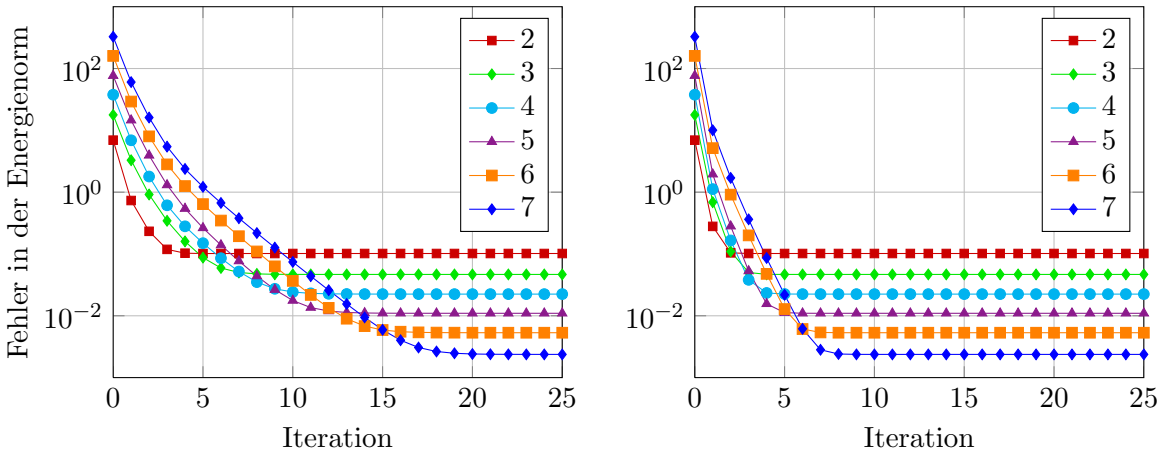
(a) Gauß-Seidel (b) Gauß-Southwell

Abb. 6.7.: wie Abbildung 6.6 mit $\epsilon = 100$



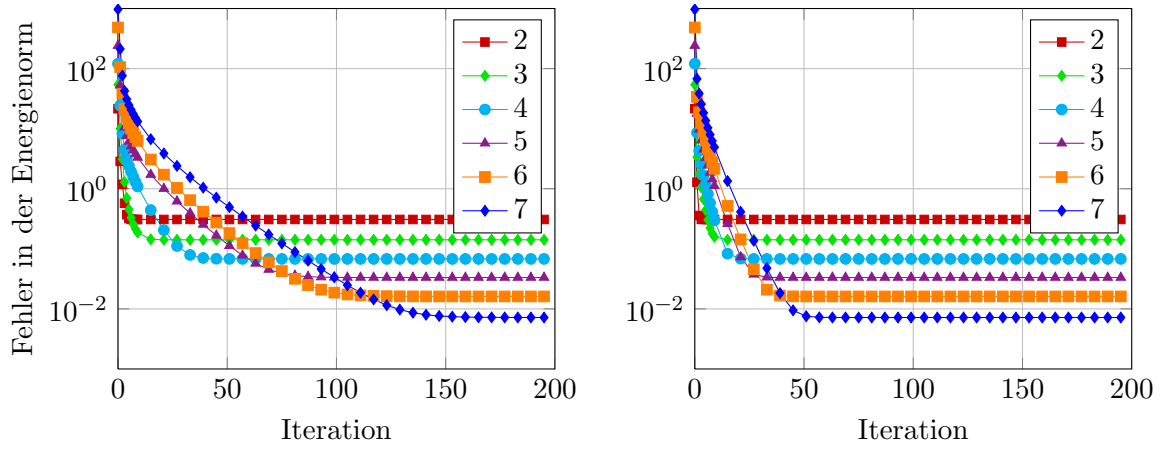
(a) Gauß-Seidel (b) Gauß-Southwell

Abb. 6.8.: wie Abbildung 6.6 mit $\epsilon = 1000$



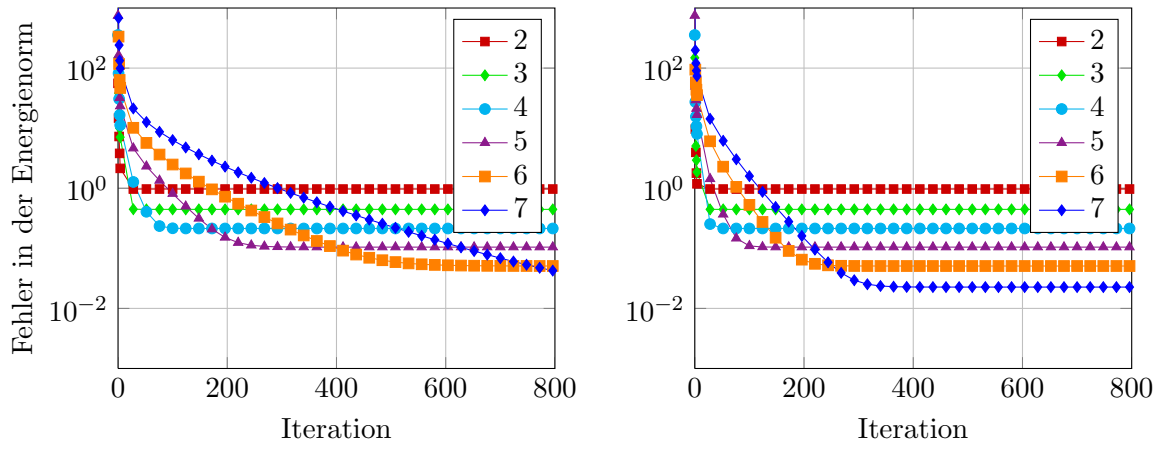
(a) Gauß-Seidel (b) Gauß-Southwell

Abb. 6.9.: Vergleich von Gauß-Seidel und Gauß-Southwell für verschiedene Level. Anisotrope Diffusion mit $\epsilon = 10$ und der Modellfunktion $p(x, y)$ als Lösung, gemessen in der Energienorm



(a) Gauß-Seidel (b) Gauß-Southwell

Abb. 6.10.: wie Abbildung 6.9 mit $\epsilon = 100$

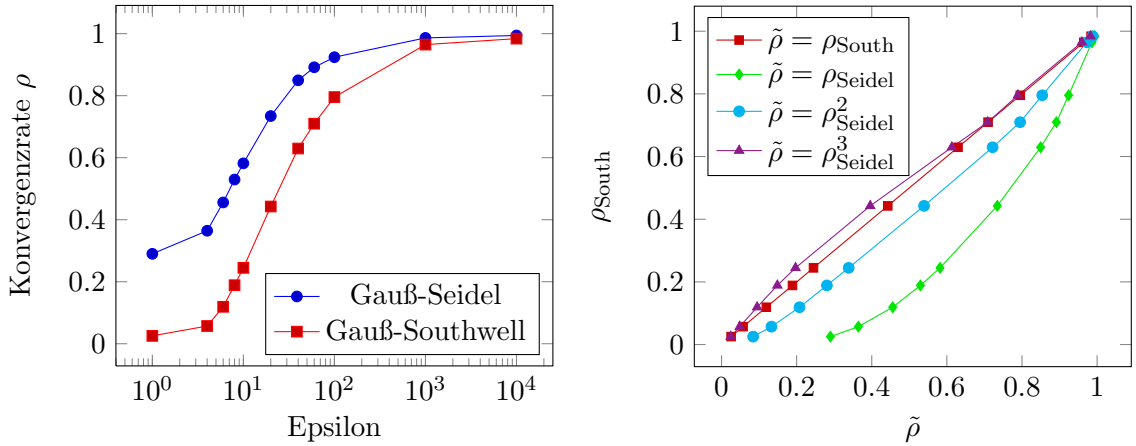


(a) Gauß-Seidel (b) Gauß-Southwell

Abb. 6.11.: wie Abbildung 6.9 mit $\epsilon = 1000$

In der L^2-Norm stellen wir allerdings fest, dass der Diskretisierungsfehler für ein beliebig festes Level unabhängig von Epsilon ist. Wir können also stets dieselbe Genauigkeit unserer Lösung erreichen, egal wie anisotrop das Problem gestellt ist. Das überrascht auf den ersten Blick, lässt sich jedoch anhand des Zusammenhangs zwischen Stabilität, Konsistenz und Konvergenz des Verfahrens zeigen. Da sich dies unseres Wissens nach nicht in der Literatur finden lässt, fügen wir in Anhang A eine Beweisskizze zu dieser Unabhängigkeit bei.

Um beurteilen zu können, ob sich die höheren Kosten des Gauß-Southwell-Verfahrens gegenüber dem Gauß-Seidel-Verfahren rechtfertigen lassen, benötigen wir ein Maß dafür, wieviel schneller das adaptive Verfahren ist. Die Konvergenzrate ist die naheliegende Größe, um beide Verfahren für verschiedene Epsilon vergleichen zu können und die Kosten in Relation setzen zu können.



(a) Konvergenzrate ρ abhängig von Epsilon (b) Konvergenzraten in Relation zueinander

Abb. 6.12.: Vergleich der Konvergenzraten ρ des Gauß-Southwell- und des Gauß-Seidel-Verfahrens für die anisotrope Diffusion in einem Level 6-Multilevelsystem

Um die Konvergenzrate messen zu können, wählen wir ein festes Level, auf welchem wir diese betrachten wollen. Für beide Verfahren suchen wir den Iterationsschritt, in dem der Diskretisierungsfehler bis auf eine Genauigkeit von 10^{-8} erreicht wird, wobei der Startfehler für beide Verfahren identisch ist. Mit Hilfe der gemessenen Fehler und der Anzahl der Iterationsschritte errechnen wir die Konvergenzrate (siehe Abschnitt 2.4.4) der Verfahren zwischen Start der Rechnung und Erreichen des Ziels. Wir betrachten hier den Fehler gemessen in der L^2-Norm, wobei sich die Konvergenzrate für die Energienorm analog verhält.

In Tabelle 6.1 sind die Konvergenzraten der beiden Verfahren für verschiedene Epsilon abgetragen, in Abbildung 6.12(a) grafisch dargestellt. Deutlich erkennt man, dass das Gauß-Southwell-Verfahren eine bessere Konvergenzrate liefert als das Gauß-Seidel-Verfahren, wobei sich für große Epsilon die Konvergenzgeschwindigkeit beider Verfahren stark reduziert.

Direkt miteinander in Relation gesetzt (siehe Abbildung 6.12(b)) erkennt man, dass der Verlauf der Kurve der Konvergenzraten des Gauß-Seidel-Verfahrens ρ_{Seidel} deutlich unter der Kurve des Gauß-Southwell-Verfahrens ρ_{South} liegt, sich jedoch in der 3. Potenz in etwa angenähert hat. Daraus ergibt sich die Schlussfolgerung, dass im Multilevelsystem mit dem symmetrischen Gauß-Seidel-Verfahren die dreifache Anzahl von Iterationen nötig ist, um die gleichen Konvergenzraten zu erreichen, die der Gauß-Southwell benötigt. Dem Gewinn, dass das Gauß-Southwell-Verfahren nur ein Drittel der Iterationsschritte des Gauß-Seidel-Verfahrens benötigt, um eine Lösung mit

Epsilon	ρ_{Seidel}	ρ_{South}
1	0,29	0,02
4	0,36	0,05
6	0,45	0,12
8	0,53	0,19
10	0,58	0,24
20	0,73	0,44
40	0,85	0,63
60	0,89	0,71
100	0,92	0,79
1000	0,98	0,96
10000	0,99	0,98

Tabelle 6.1.: Konvergenzraten ρ des Gauß-Southwell- und des Gauß-Seidel-Verfahren für die anisotrope Diffusion in einem Level 6-Multilevelsystem

dem gleichen Fehler zu erreichen, stehen die Kosten, die die adaptive Durchlaufreihenfolge (siehe Abschnitt 5.5) mit sich bringt, gegenüber. Diese betragen in jedem Schritt den Faktor $\mathcal{O}(\log N)$ gegenüber dem Gauß-Seidel-Verfahren. Demzufolge rechtfertigen die Kosten, die die geschickte Wahl der Relaxationen im *worst case* verursacht, den Gewinn gegenüber der statischen Durchlaufreihenfolge nicht.

Level	N	$\log_2(N)$	Laufzeit pro Iteration [s]		$\frac{\text{Southwell}}{\text{Seidel}}$
			G,-Seidel	G,-Southwell	
2	10	3	0,30	0,32	1,07
3	59	6	0,24	0,34	1,39
4	284	8	0,26	0,37	1,45
5	1245	10	0,30	0,50	1,64
6	5214	12	1,31	2,15	1,63
7	21343	14	17,02	25,28	1,48

Tabelle 6.2.: Vergleich der Laufzeiten jeweils eines Iterationsschritts mit je $2N - 1$ Relaxationen für das Gauß-Seidel- und das Gauß-Southwell-Verfahren in Multilevelsystemen verschiedenen Levels. N ist die Anzahl der Unbekannten in jedem Level.

Wenn wir die Laufzeit eines einzelnen Iterationsschritts messen, können wir so abschätzen, wieviel aufwändiger das Gauß-Southwell-Verfahren in der Praxis wirklich ist. Dies lässt sich nicht wie eine *average case*-Abschätzung verwenden, ist aber richtungsweisend, um ein Gefühl für den Aufwand des adaptiven Verfahrens zu entwickeln. Man beachte, dass die hier gemessenen Zeiten nur zum Vergleich der beiden Verfahren untereinander dienen. Zum Vergleich mit anderen Verfahren sind die Ergebnisse nicht verwertbar, da beispielsweise die Berechnung des Fehlers, die aufwändig ist und nach jedem Iterationsschritt durchgeführt werden muss, mitgemessen wurde.

Es zeigt sich, dass das Gauß-Southwell-Verfahren deutlich weniger Kosten benötigt, als die *worst case*-Abschätzung vermuten lässt (siehe Tabelle 6.2). Demzufolge rentiert sich der Einsatz des Gauß-Southwell-Verfahrens gegenüber dem Gauß-Seidel-Verfahren für die Lösung von Multilevelsystemen, denen das Problem anisotroper Diffusion zugrunde liegt, in der Praxis sehr wohl. Jedoch bleibt das Problem, dass die Konvergenzgeschwindigkeit (siehe Tabelle 6.1) auch für den Gauß-Southwell bei starker Anisotropie sehr langsam ist, lediglich in den ersten Schritten konnten wir für alle Epsilon eine deutlich bessere Rate beobachten. Können wir aus dieser

Beobachtung einen Nutzen ziehen?

Wir betrachten nun ein Verfahren, dem die Idee zugrunde liegt, die Ergebnisse eines jeden Levels als Startwert für das jeweils nächste Level zu verwenden. Dazu kommt die Anforderung, auf jedem Level nur wenige Iterationsschritte auszuführen. Wir werden sehen, dass dieses Verfahren von der adaptiven Durchlaufreihenfolge des Gauß-Southwell-Algorithmus profitieren kann und sich der dadurch entstehende Aufwand deutlich bezahlt macht.

6.3. Anisotrope Diffusion mit Nested-Iteration im Multilevelsystem

In den Abbildungen 6.6-6.11 lässt sich für beide Verfahren beobachten, dass der Verlauf des Iterationsfehlers in den ersten Schritten deutlich steiler ist als in den später folgenden Schritten. Insbesondere bei größeren Epsilon fällt auf, dass sich dieses Verhalten auch in diesen Fällen zeigt, obwohl die Konvergenzgeschwindigkeit insgesamt stark abgenommen hat.

Diese Beobachtung führt uns zu der Idee, den Gauß-Southwell-Algorithmus in Verbindung mit der Nested-Iteration anzuwenden. Dieses Verfahren haben wir in Abschnitt 4.3 eingeführt und in Abschnitt 4.3.1 auf Multilevelsysteme erweitert. Wir untersuchen nun das Konvergenzverhalten des Gauß-Southwell-Algorithmus in diesem Verfahren und vergleichen es wieder mit dem symmetrischen Gauß-Seidel-Algorithmus. Dabei zählen wir die Iterationsschritte wie in Abschnitt 6.2, damit wir beide Algorithmen korrekt gegenüberstellen können. Die Ergebnisse, die wir so erhalten, lassen zu, dass wir direkt abschätzen können, wie gut wir von der Adaptivität des Gauß-Southwell-Algorithmus profitieren können.

Die Problemstellung, für die wir das Konvergenzverhalten der beiden Iterationsverfahren in der Nested-Iteration untersuchen wollen, ist die der anisotropen Diffusion, wie wir sie in Abschnitt 6.2.3 beschrieben haben. Wir erwarten hierbei, mit der Nested-Iteration bessere Konvergenzraten zu erzielen, als wir in o.g. Abschnitt erreichen konnten.

Für die verschiedenen Fälle werden wir überprüfen, wie viele Iterationsschritte von den beiden Verfahren notwendig sind, um dieses als ausitertiert zu betrachten. Wir definieren diesen Zeitpunkt so, dass sich der Iterationsfehler mit jedem Schritt nur noch im Bereich von 10^{-8} verbessert. Ist dies der Fall, so hat sich für uns im Folgenden der Iterationsfehler „genügend nah“ dem Diskretisierungsfehler genähert.

Wir betrachten das Konvergenzverhalten beider Verfahren für die Modellfunktionen $p(x, y)$ und $q(x, y)$ jeweils in der L^2-Norm und in der Energienorm. Wir lösen das Level 1 direkt, prolongieren das Ergebnis auf das zweite Level und nutzen dieses als Startvektor für das Nested-Iteration-Verfahren. Die Vorgehensweise ist dann die, dass wir auf jedem Level so lange iterieren, bis wir den Diskretisierungsfehler genügend genau erreicht haben, und dann das Ergebnis als Startvektor für das folgende Level verwenden. Damit unterscheiden wir uns vom ursprünglichen Nested-Iteration-Verfahren, welches auf jedem Level eine fest vorgegebene Anzahl von Iterationsschritten durchführt. Wir wollen hier jedoch das Konvergenzverhalten der Iterationsverfahren für sehr gute Startwerte auf den verschiedenen Leveln untersuchen und lassen aus diesem Grund die Verfahren ausiterieren. Man beachte, dass, um die Lesbarkeit der folgenden Abbildungen zu erhöhen, nicht immer der komplette Verlauf des Iterationsfehlers gezeigt wird.

Als Epsilon wählen wir $\epsilon = 1$, was der isotropen Diffusion entspricht, sowie $\epsilon = 100$ und $\epsilon = 10000$. Bevor wir uns mit den verschiedenen Fällen beschäftigen, betrachten wir exemplarisch den Diskretisierungsfehler der beiden Funktionen in Abbildung 6.13 und 6.19 auf den verschiedenen Leveln. Mit der Verbesserung des Fehlers von Level zu Level und den verschiedenen Faktoren für die Normen haben wir uns in Abschnitt 6.2.3 beschäftigt, hier vergleichen wir nun den

Unterschied zwischen den Modellfunktionen. Dabei fällt auf, dass der Diskretisierungsfehler für den Fall, dass die trigonometrische Funktion $q(x, y)$ die Lösung darstellt, größer ist als für die polynomielle Funktion $p(x, y)$. Das hat zur Folge, dass bei gleicher Konvergenzrate ein Verfahren für die Funktion $q(x, y)$ den Diskretisierungsfehler schneller erreicht als für die Funktion $p(x, y)$.

Für den ersten Fall mit $\epsilon = 1$ (siehe Abbildungen 6.13, 6.16, 6.19, 6.22) beobachten wir, dass sich das Gauß-Seidel-Verfahren auf dem feinsten Level nach vier Schritten dem Diskretisierungsfehler genügend genähert hat. Das gilt für beide Funktionen und für die Messung in beiden Normen. Für den Gauß-Southwell stellt sich heraus, dass er für alle Level nur einen Schritt benötigt, um das gleiche Ziel zu erreichen.

Für $\epsilon = 100$ sieht man, dass sich die beiden Verfahren deutlich voneinander entfernen (siehe Abbildungen 6.14, 6.17, 6.20, 6.23). Sowohl in der Energienorm als auch in der L^2-Norm gemessen benötigt das Gauß-Southwell-Verfahren bei beiden Funktionen in jedem Level weniger als 10 Iterationsschritte, wogegen das Gauß-Seidel-Verfahren bis zu 90 Schritte benötigt.

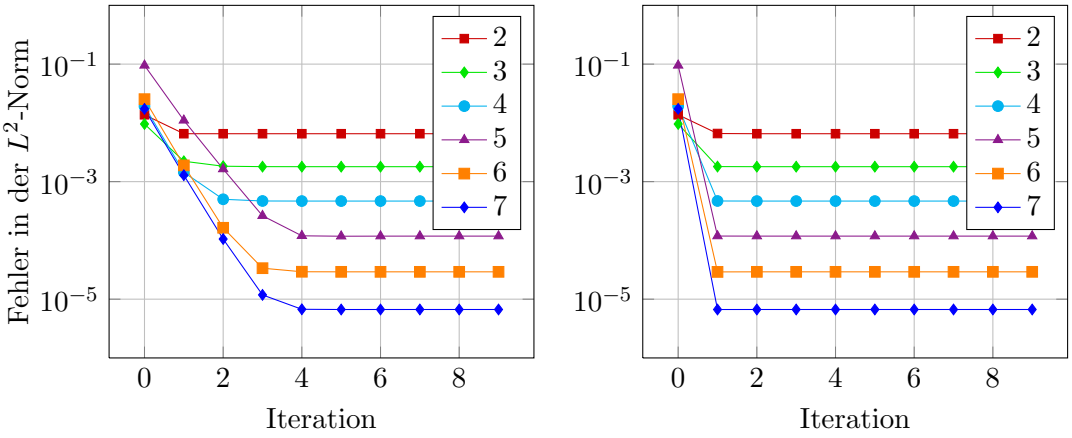
Wir betrachten einen Fall im Detail. Für die polynomielle Modellfunktion $p(x, y)$ mit dem Fehler gemessen in der Energienorm (siehe Abbildung 6.17) benötigt der Gauß-Southwell 4 Iterationsschritte auf dem Level 7, das Gauß-Seidel-Verfahren dagegen mehr als 70. Die Unbekannte auf Level 7 hat $N = 21343$ Elemente. Damit kostet ein Iterationsschritt des Gauß-Southwell-Verfahrens für dieses Level im *worst case* $\log_2(N) \approx 14$-mal soviel wie ein Schritt des Gauß-Seidel-Verfahrens. Konkret bedeutet das für den betrachteten Fall, dass der Gauß-Southwell mit $4 \cdot 14$-fachen Kosten besser ist als der Gauß-Seidel mit seinen $70 \cdot 1$-fachen Kosten. Das Lösen dieses Gleichungssystems mit dem Gauß-Southwell-Verfahren kostet hier also weniger Aufwand als mit dem Gauß-Seidel-Verfahren.

Wenn wir nun die Anisotropie noch mehr erhöhen, wird das Gleichungssystem noch schwieriger zu lösen. Wir betrachten das Verhalten des Fehlers der Iterationsverfahren für die Aufgabenstellung mit $\epsilon = 10000$ (siehe Abbildungen 6.15, 6.18, 6.21, 6.24). Hier sehen wir, dass sich die beiden Verfahren noch stärker voneinander unterscheiden.

Für die trigonometrische Funktion $q(x, y)$ tritt das nicht ganz so deutlich hervor, da der Diskretisierungsfehler in diesem Fall viel größer und damit schneller erreicht ist. Sehen wir uns zunächst den Verlauf für den Fall an, dass $q(x, y)$ die exakte Lösung ist und der Fehler in der Energienorm gemessen wird (siehe Abbildung 6.24). Für das Level 7 benötigt das Gauß-Southwell-Verfahren 24 Iterationsschritte, bis es genügend nah an den Diskretisierungsfehler heran gekommen ist. Das Gauß-Seidel-Verfahren benötigt dagegen 372 Schritte. Wie oben gilt, dass ein Iterationsschritt des adaptiven Verfahrens im *worst case* um den Faktor $\log_2(N) \approx 14$ teurer ist. Demnach entsprechen die Kosten der 24 Gauß-Southwell Schritte denen von 336 Gauß-Seidel Schritten, was weniger ist als die 372 Schritte, die der Gauß-Seidel tatsächlich benötigt, um den Fehler in gleichem Maße zu verringern.

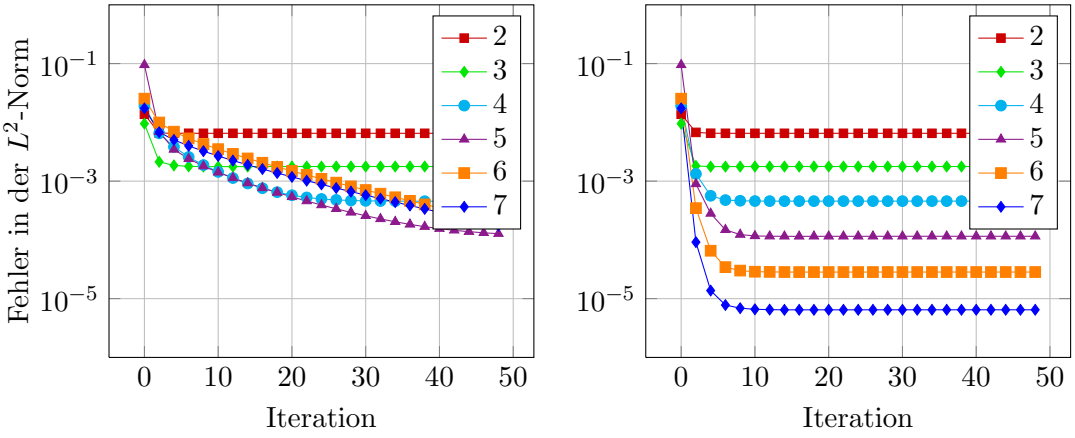
Funktion	Level	$\log_2(N) \approx$	Iterationsschritte		
			G.-Seidel	G.-Southwell	G.-Southwell $\cdot \log_2(N)$
$q(x, y)$	6	12	74	13	182
$q(x, y)$	7	14	372	14	336
$p(x, y)$	6	12	400	31	372
$p(x, y)$	7	14	600	40	560

Tabelle 6.3.: Vergleich der zur genügend guten Näherung an den Diskretisierungsfehler benötigten Iterationsschritte der Algorithmen von Gauß-Seidel und Gauß-Southwell im Nested-Iteration-Verfahren für die anisotrope Diffusion mit $\epsilon = 10000$



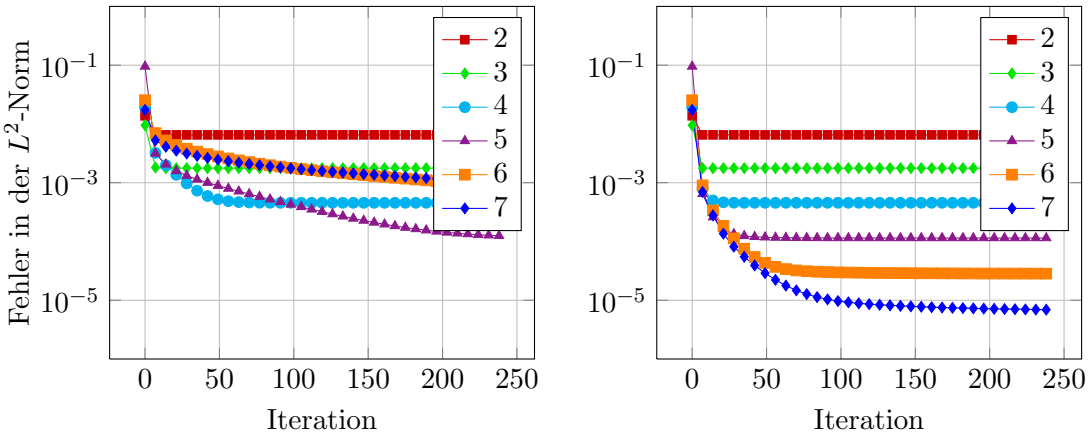
(a) Gauß-Seidel (b) Gauß-Southwell

Abb. 6.13.: Vergleich von Gauß-Seidel und Gauß-Southwell im Nested-Iteration-Verfahren. Anisotrope Diffusion mit $\epsilon = 1$ und der Modellfunktion $p(x, y)$ als Lösung, gemessen in der L^2-Norm



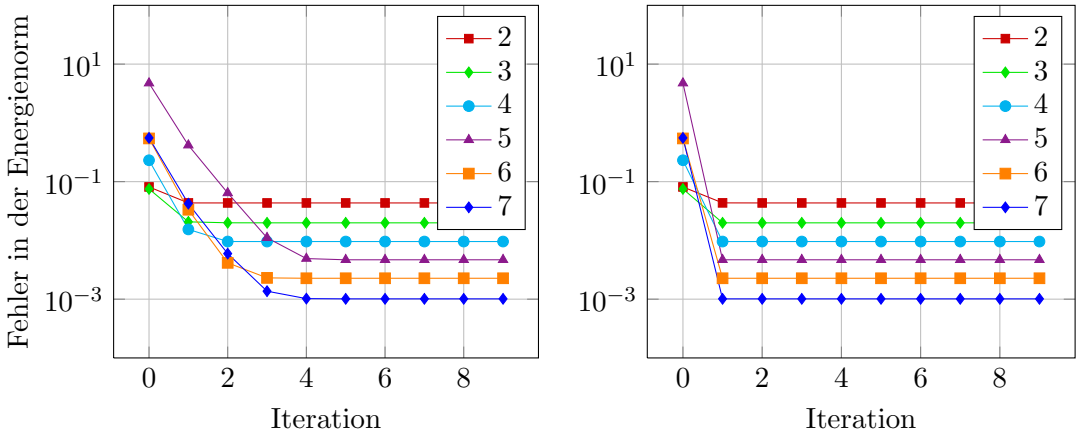
(a) Gauß-Seidel (b) Gauß-Southwell

Abb. 6.14.: Wie Abbildung 6.13 mit $\epsilon = 100$



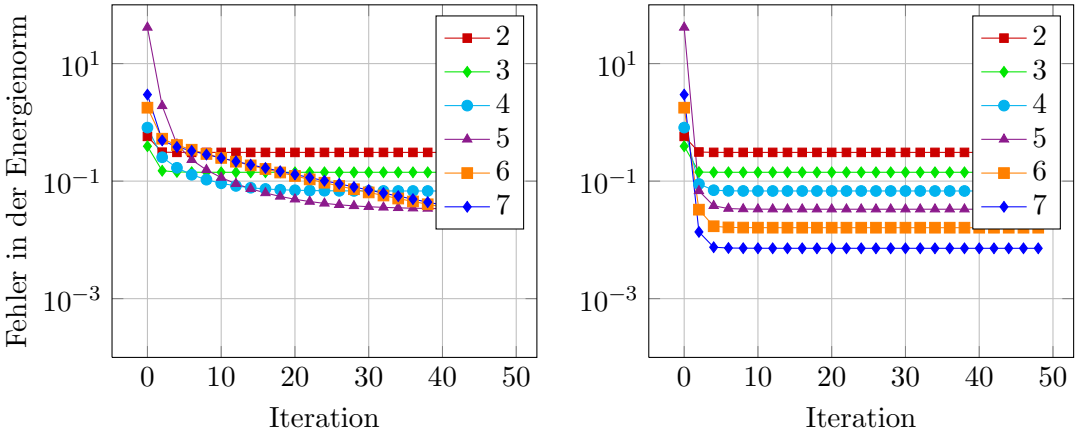
(a) Gauß-Seidel (b) Gauß-Southwell

Abb. 6.15.: Wie Abbildung 6.13 mit $\epsilon = 10000$



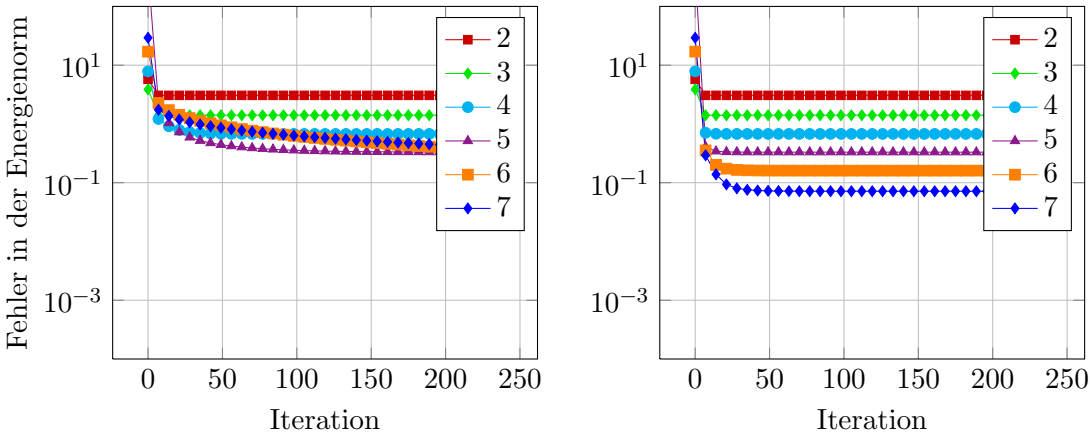
(a) Gauß-Seidel (b) Gauß-Southwell

Abb. 6.16.: Vergleich von Gauß-Seidel und Gauß-Southwell im Nested-Iteration-Verfahren. Anisotrope Diffusion mit $\epsilon = 1$ und der Modellfunktion $p(x, y)$ als Lösung, gemessen in der Energienorm



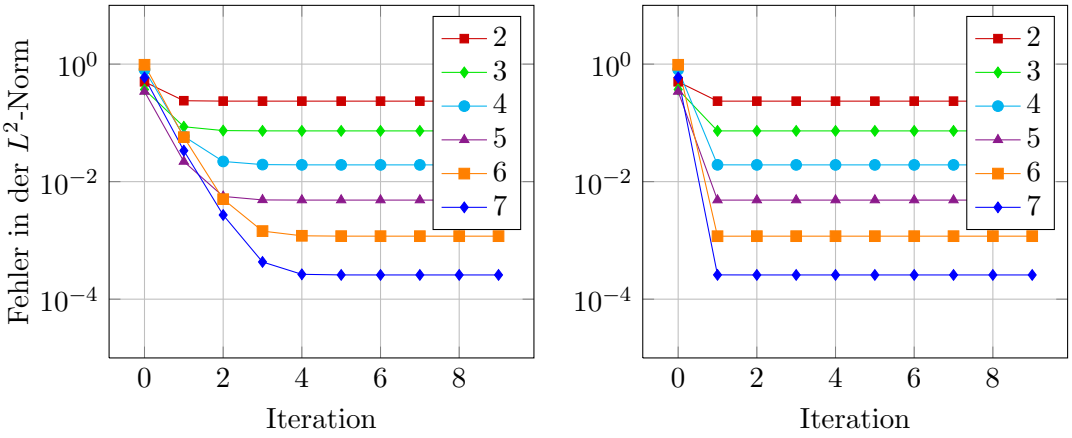
(a) Gauß-Seidel (b) Gauß-Southwell

Abb. 6.17.: Wie Abbildung 6.16 mit $\epsilon = 100$



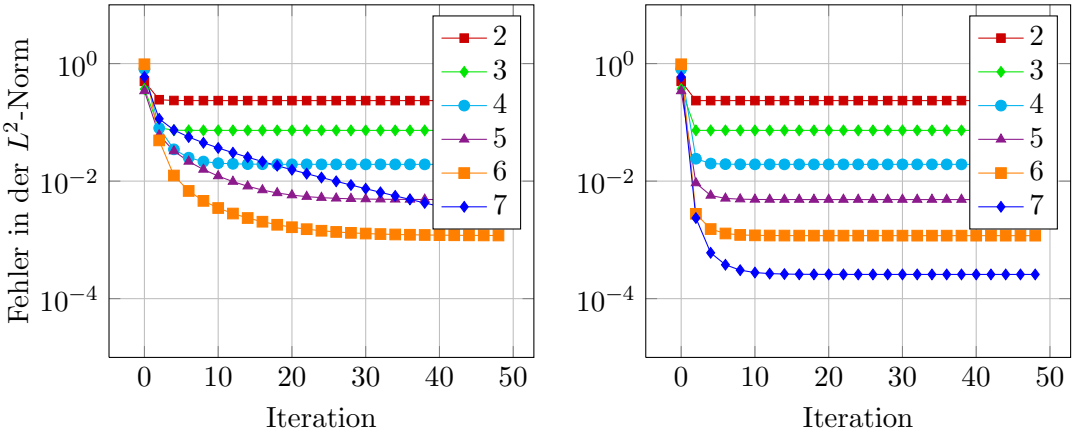
(a) Gauß-Seidel (b) Gauß-Southwell

Abb. 6.18.: Wie Abbildung 6.16 mit $\epsilon = 10000$



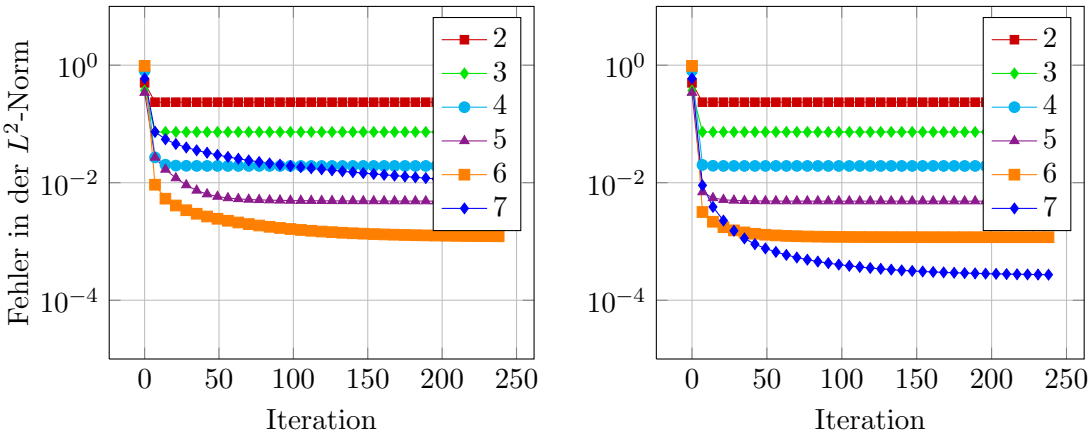
(a) Gauß-Seidel (b) Gauß-Southwell

Abb. 6.19.: Vergleich von Gauß-Seidel und Gauß-Southwell im Nested-Iteration-Verfahren. Anisotrope Diffusion mit $\epsilon = 1$ und der Modellfunktion $q(x, y)$ als Lösung, gemessen in der L^2-Norm



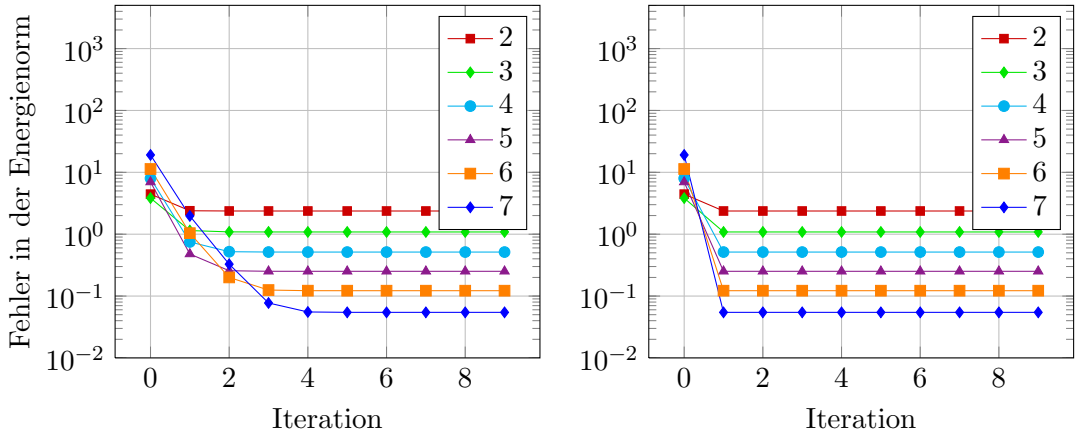
(a) Gauß-Seidel (b) Gauß-Southwell

Abb. 6.20.: Wie Abbildung 6.19 mit $\epsilon = 100$



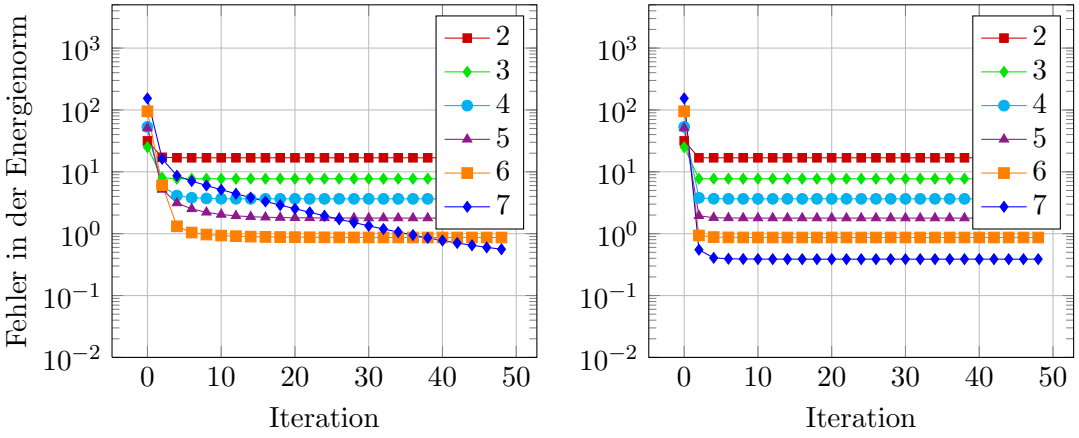
(a) Gauß-Seidel (b) Gauß-Southwell

Abb. 6.21.: Wie Abbildung 6.19 mit $\epsilon = 10000$



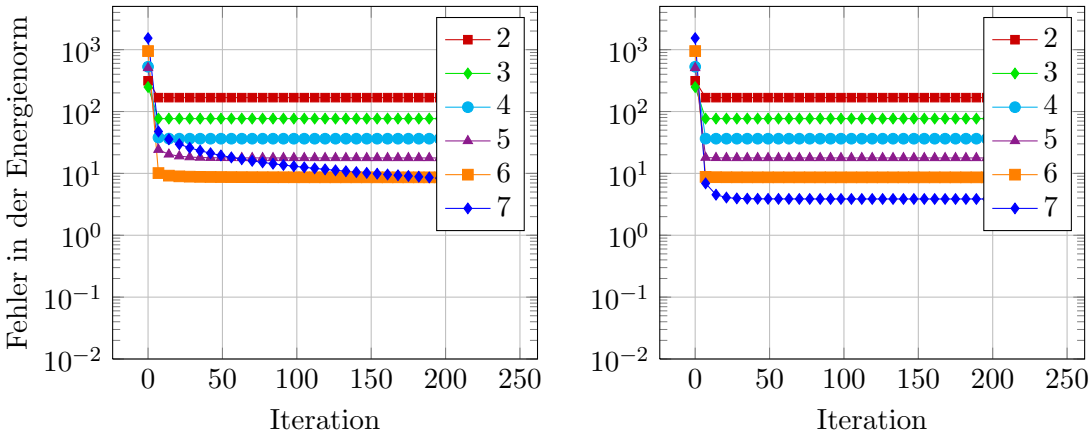
(a) Gauß-Seidel (b) Gauß-Southwell

Abb. 6.22.: Vergleich von Gauß-Seidel und Gauß-Southwell im Nested-Iteration-Verfahren. Anisotrope Diffusion mit $\epsilon = 1$ und der Modellfunktion $q(x, y)$ als Lösung, gemessen in der Energienorm



(a) Gauß-Seidel (b) Gauß-Southwell

Abb. 6.23.: Wie Abbildung 6.22 mit $\epsilon = 100$



(a) Gauß-Seidel (b) Gauß-Southwell

Abb. 6.24.: Wie Abbildung 6.22 mit $\epsilon = 10000$

Untersuchen wir die polynomielle Modellfunktion $p(x, y)$, deren Diskretisierungsfehler kleiner ist, so ist zu erwarten, dass ein Iterationsverfahren bei gleicher Konvergenzrate mehr Schritte ausführen muss, um diesen genügend genau zu erreichen. Wie man in Abbildung 6.18 sieht, benötigt das Gauß-Southwell-Verfahren, um den Diskretisierungsfehler zu erreichen, für jedes Level weniger als 40 Schritte. Das Gauß-Seidel-Verfahren muss dagegen deutlich mehr Iterationsschritte ausführen. Auf Level 6 sind zum Erreichen des Diskretisierungsfehlers 400 Schritte nötig, auf Level 7 über 600 Schritte. Rechnet man die Kosten der Schritte des Gauß-Southwell-Verfahrens mit dem jeweiligen Faktor um (siehe Tabelle 6.3), ergeben sich für Gauß-Southwell Kosten von 372 Gauß-Seidel-Schritten für das Level 6 und 560 Gauß-Seidel-Schritten für das Level 7.

Es zeigt sich also, dass das Gauß-Southwell-Verfahren für das Lösen dieser Gleichungssysteme ab einem bestimmten Level weniger Kosten verursacht als die Lösung mit dem Gauß-Seidel-Verfahren, insbesondere für immer größere Systeme. Der Aufwand, den die adaptive Durchlaufreihenfolge verursacht, ist damit im *worst case* gerechtfertigt und somit natürlich auch für die durchschnittlichen und besten Fälle.

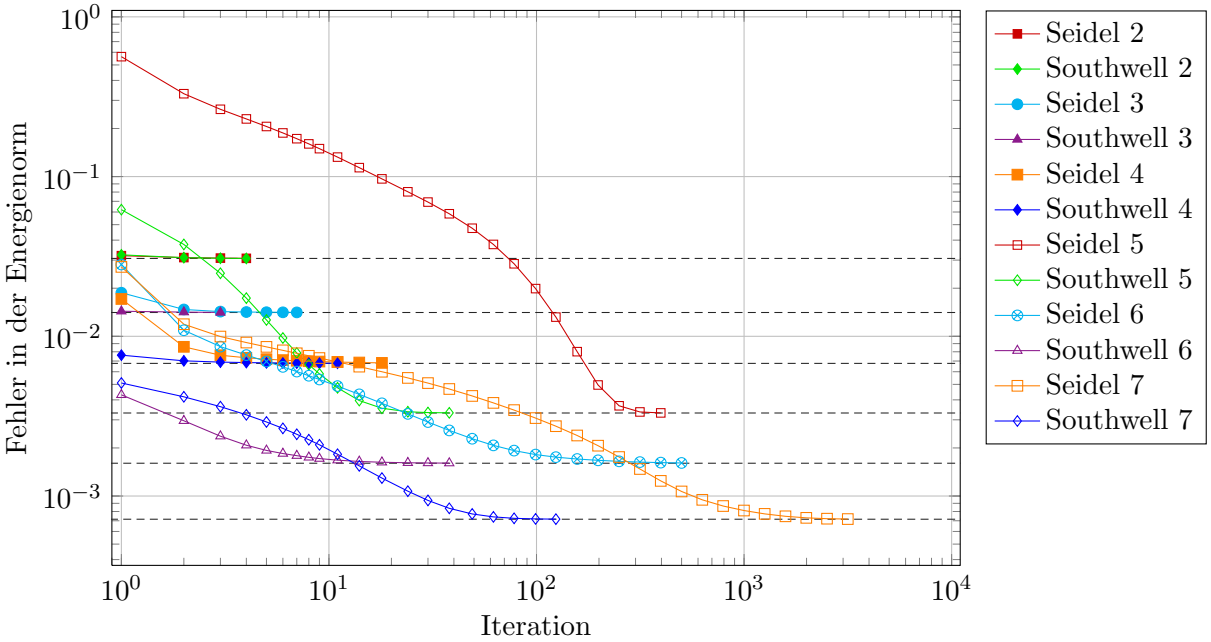


Abb. 6.25.: Vergleich von Gauß-Seidel und Gauß-Southwell im Nested-Iteration-Verfahren. Modellfunktion $p(x, y)$, anisotrope Diffusion mit $\epsilon = 0$. Die Iteration wurde jeweils abgebrochen, wenn eine Delta-Umgebung um den Diskretierungsfehler erreicht wurde

In Abbildung 6.25 wird noch einmal das Gauß-Southwell-Verfahren dem Gauß-Seidel-Verfahren exemplarisch für den Extremfall $\epsilon = 0$ gegenübergestellt. Man beachte, dass hier auch die x-Achse logarithmisch skaliert ist und diesmal die Iteration jeweils abgebrochen wurde, sobald ein Verfahren eine Delta-Umgebung um den Diskretisierungsfehler erreicht hat.

An diesem Beispiel sehen wir ganz deutlich, wie der Gauß-Southwell von der adaptiven Durchlaufreihenfolge profitiert. Auf den ersten Leveln kann das Verfahren gegenüber dem Gauß-Seidel-Algorithmus keinen bemerkenswerten Vorteil erreichen, wogegen mit steigendem Level die Unterschiede immer deutlicher werden. So ist die Anzahl der nötigen Iterationen des Gauß-Southwell ein bis eineinhalb Größenordnungen geringer als die, die der Gauß-Seidel benötigt, um den Fehler in gleichen Maße zu verringern. Je mehr Level im erweiterten Gleichungssystem vereinigt sind,

desto größer wird die Anzahl der Unbekannten. Der Mehraufwand, den das Gauß-Southwell-Verfahren investieren muss, um die adaptive Durchlaufreihenfolge zu finden, steigt jedoch logarithmisch in der Anzahl der Unbekannten und wächst folglich sehr viel langsamer als das Gleichungssystem.

6.3.1. Robustheitsuntersuchung

Bevor wir uns der Analyse der Verfahren in der Praxis widmen, wollen wir untersuchen, ob wir die Abhängigkeit von den Parametern der Differentialgleichung reduzieren konnten. Um für die anisotrope Diffusion Unabhängigkeit vom Grad der Anisotropie zu erreichen, müsste die Konvergenzgeschwindigkeit des Gauß-Southwell-Verfahrens unabhängig vom Parameter Epsilon sein. Wir betrachten den Verlauf des Fehlers im Nested-Iteration-Verfahren im 7-Level Multilevelsystem für verschiedene Epsilon.

In Abbildung 6.26 ist der Verlauf des Fehlers gemessen in der Energienorm dargestellt. Hier machen wir zunächst die Beobachtung, dass der Fehler von Epsilon abhängt und mit diesem Parameter zusammen anwächst. Wie zu erwarten, erreicht das Gauß-Southwell-Verfahren dabei stets schneller den Diskretisierungsfehler als das Gauß-Seidel-Verfahren. Es zeigt sich jedoch anhand des Zeitpunkts, an dem die Verfahren diese Grenze erreichen, dass der Gauß-Southwell weniger vom Grad der Anisotropie beeinflusst wird als der Gauß-Seidel. Anhand einer Geraden ist skizziert, wann diese Grenze jeweils erreicht wird. Beachtet man, dass die Anzahl der Iterationen hier logarithmisch aufgetragen ist, wird klar, dass diese im Gauß-Southwell-Verfahren sehr viel steiler verläuft als im Gauß-Seidel-Verfahren. Wir erreichen hiermit also einen großen Gewinn bezüglich der Robustheit.

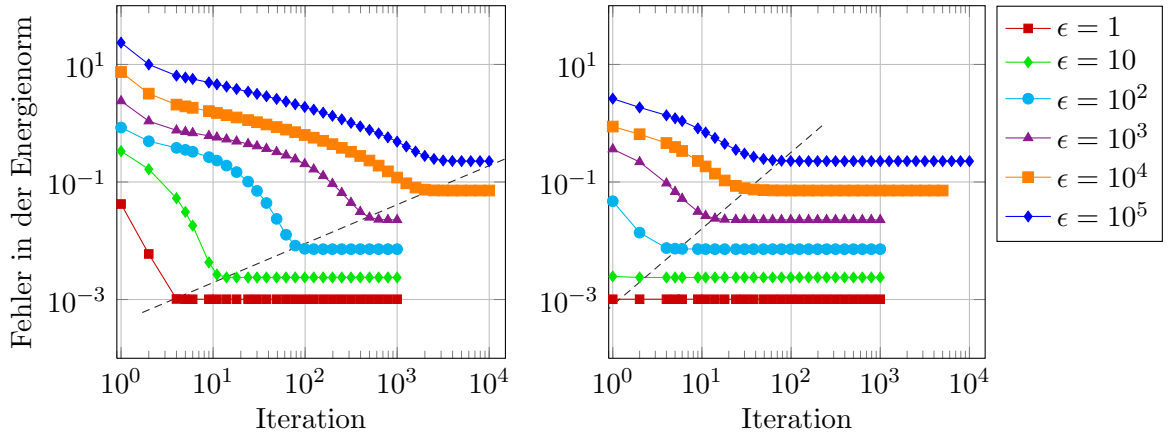
Der Vollständigkeit halber stellt Abbildung 6.27 denselben Sachverhalt in der L^2-Norm dar. Hier lässt sich noch einmal deutlich erkennen, dass der Diskretisierungsfehler in der L^2-Norm unabhängig von Epsilon ist, vergleiche dazu Anhang A. Der Gewinn in der Robustheit zeigt sich dabei in der starken Stauchung der Fehlerkurven des Gauß-Southwell entlang der x-Richtung im Vergleich zu denen des Gauß-Seidel.

Zwar erreichen wir mit dem Gauß-Southwell keine absolute Robustheit, allerdings wird deutlich, dass die Konvergenzgeschwindigkeit des Gauß-Southwell-Verfahrens für die anisotrope Diffusion sehr viel weniger abhängig vom Grad der Anisotropie ist, als dies für das Gauß-Seidel-Verfahren der Fall ist.

Im Folgenden werden wir untersuchen, wie sich dieser Gewinn in der Praxis auswirkt. Dabei können wir erwarten, dass der Gauß-Southwell-Algorithmus zur Lösung der gleichen Aufgabenstellung nur einen Bruchteil der Zeit benötigen wird, die der Gauß-Seidel-Algorithmus aufwenden muss.

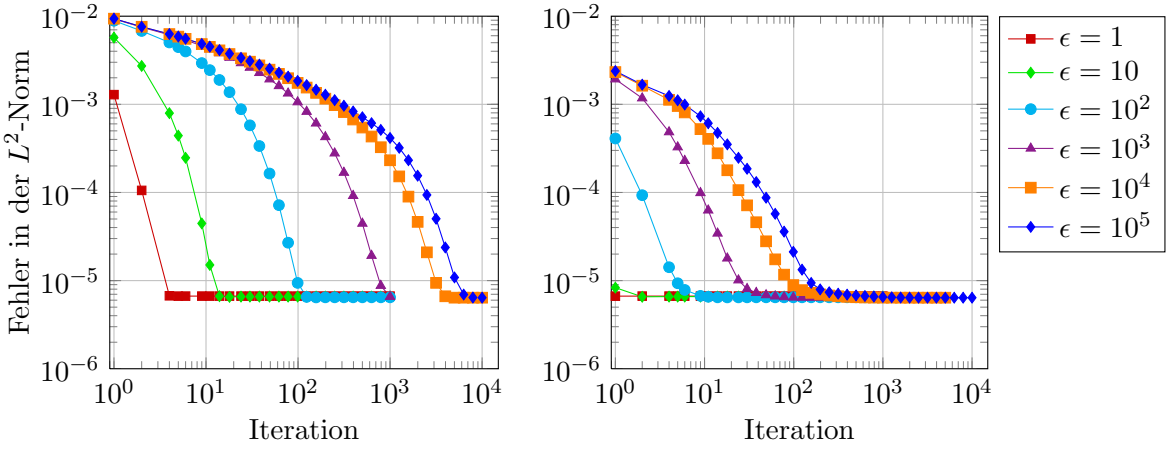
6.3.2. Laufzeitanalyse

Aus den vorangegangenen Überlegungen und Beobachtungen wissen wir, dass sich der Mehraufwand rentiert, den das Gauß-Southwell-Verfahren gegenüber dem Gauß-Seidel-Algorithmus investieren muss. Dabei hatten wir die Kosten der einzelnen Schritte für den *worst case* abgeschätzt, d.h. selbst in diesem schlechtesten Fall lohnt sich der Aufwand des Gauß-Southwell. Uns interessieren aber auch die durchschnittlich auftretenden Kosten, die das adaptive Verhalten des Algorithmus verursacht. Diese lassen sich theoretisch nicht berechnen, da wir keine genauen Annahmen über die Struktur des Residuums und damit der Priority-Queue treffen können. Um den



(a) Gauß-Seidel (b) Gauß-Southwell

Abb. 6.26.: Vergleich der Nested-Iteration auf Level 7, anisotrope Diffusion mit verschiedenen Epsilon und mit der Modellfunktion $p(x, y)$ als Lösung, gemessen in der Energienorm



(a) Gauß-Seidel (b) Gauß-Southwell

Abb. 6.27.: Vergleich der Nested-Iteration auf Level 7, anisotrope Diffusion mit verschiedenen Epsilon und mit der Modellfunktion $p(x, y)$ als Lösung, gemessen in der L^2-Norm

wirklich entstehenden Aufwand zu beurteilen, können wir nur die Zeit messen, die die Verfahren zur Lösung des Gleichungssystems tatsächlich benötigen.

Da wir die Algorithmen im Nested-Iteration-Verfahren einsetzen, können wir die Verfahren am besten vergleichen, wenn wir Zeiten für die Lösung des jeweiligen Levels messen. Damit erhalten wir auch die Gesamtzeit, die notwendig ist, um das Verfahren komplett durchzuführen. Wir setzen als Ziel, dass bis auf einen Faktor von $\delta = 1,01$ das Problem jeweils bis zur Diskretisierungsgenauigkeit gelöst werden muss. Wir messen hier nur die Zeit, die die iterativen Löser tatsächlich benötigen, da wir diese miteinander vergleichen wollen. Die zusätzlichen Operationen, die für die Nested-Iteration durchgeführt werden müssen, wie das direkte Lösen auf dem gröbsten Level oder das Interpolieren zwischen den Leveln, sind bei beiden identisch und fließen nicht in die Messung ein. Die gemessenen Zeiten dienen nur zum Vergleich der beiden Verfahren untereinander und sind zum Vergleich mit anderen Verfahren untauglich, da beispielsweise die Berechnung des Fehlers mitgemessen wurde, die sehr aufwändig ist und nach jedem Iterationsschritt durchgeführt werden muss.

Wir prüfen das Verhältnis der Zeit, die die Algorithmen benötigen, um die Fehlerschranke zu erreichen, indem wir für jedes Level den Quotient (Laufzeit Gauß-Southwell/Laufzeit Gauß-Seidel) der verschiedenen gemessenen Zeiten berechnen und betrachten. Um die beiden Löser über das ganze Nested-Iteration-Verfahren miteinander vergleichen zu können, bilden wir die Summe über die Zeiten aller Level für beide Löser. In Tabelle 6.4 sind die Zeiten für verschiedene Epsilon eingetragen, dabei dient die Modellfunktion $p(x, y)$ als Lösung. Für die trigonometrische Modellfunktion $q(x, y)$ liefert die Zeitmessung analoge Ergebnisse.

Level	Laufzeit [s]		$\frac{\text{Southwell}}{\text{Seidel}}$
	G.-Seidel	G.-Southwell	
2	0.30	0.32	1.07
3	0.63	0.34	0.54
4	0.64	0.37	0.58
5	1.20	0.50	0.42
6	5.00	2.15	0.43
7	67.68	25.28	0.37
$\sum$	75.45	28.96	0.38

(a) $\epsilon = 1$

Level	Laufzeit [s]		$\frac{\text{Southwell}}{\text{Seidel}}$
	G.-Seidel	G.-Southwell	
2	0.42	0.67	1.60
3	0.91	0.68	0.75
4	4.19	1.34	0.32
5	12.91	3.64	0.28
6	128.53	20.95	0.16
7	2145.60	307.51	0.14
$\sum$	2292.56	334.79	0.15

(b) $\epsilon = 100$

Level	Laufzeit [s]		$\frac{\text{Southwell}}{\text{Seidel}}$
	G.-Seidel	G.-Southwell	
2	0.41	0.64	1.56
3	1.28	0.80	0.63
4	9.16	3.56	0.39
5	65.16	18.97	0.29
6	1735.00	337.04	0.19
7	63543.39	5871.25	0.09
$\sum$	65354.40	6232.26	0.10

(c) $\epsilon = 10000$

Level	Laufzeit [s]		$\frac{\text{Southwell}}{\text{Seidel}}$
	G.-Seidel	G.-Southwell	
2	0.63	0.65	1.03
3	1.55	0.69	0.45
4	5.72	3.43	0.60
5	95.03	25.25	0.27
6	1415.74	243.30	0.17
7	90825.87	6894.42	0.08
$\sum$	92344.54	7167.74	0.08

(d) $\epsilon = 0$

Tabelle 6.4.: Vergleich der gemessen Laufzeit von Gauß-Southwell und Gauß-Seidel im Nested-Iteration-Verfahren. Problemstellung war die anisotrope Diffusion mit verschiedenen Epsilon und der polynomiellen Modellfunktion $p(x, y)$ als Lösung

Für das Level 2 lässt sich in allen Fällen beobachten, dass das Gauß-Southwell-Verfahren mehr Zeit benötigt, um ein genügend gutes Ergebnis zu berechnen. Für diese Fälle brauchen beide Algorithmen nur wenige Schritte, um das entsprechende Ziel zu erreichen, und die Kosten, die durch die adaptive Durchlaufreihenfolge des Gauß-Southwell-Algorithmus entstehen, fallen dadurch mehr ins Gewicht. Auf den höheren Leveln benötigt der Gauß-Southwell-Algorithmus allerdings im Verhältnis immer weniger Iterationsschritte verglichen mit dem Gauß-Seidel-Verfahren. Das schlägt sich auch in der Laufzeit des Algorithmus nieder. Für alle verschiedenen Epsilon, von der isotropen Diffusion mit $\epsilon = 1$ bis zur extremen Anisotropie mit $\epsilon = 0$, benötigt das Gauß-Southwell-Verfahren auf jedem Level $l > 2$ weniger Zeit, bis die Fehlerschranke überschritten ist, als das Gauß-Seidel-Verfahren.

Betrachtet man die Gesamtlaufzeiten der Nested-Iteration für die verschiedenen Epsilon, erkennt man, wie der Grad der Anisotropie die Schwierigkeit der Lösung beeinflusst. Je mehr der Unterschied der Diffusion in den verschiedenen Richtungen wächst, desto länger benötigen die Algorithmen zur Lösung. Die Problemstellung mit $\epsilon = 1$ lösen die Verfahren am schnellsten, mit wachsender Anisotropie benötigen die Verfahren immer mehr Zeit.

Je schwieriger die Problemstellung wird, desto mehr profitiert das adaptive Verfahren von der geschickten Durchlaufreihenfolge gegenüber der statischen Durchlaufreihenfolge des symmetrischen Gauß-Seidel-Verfahrens. Der Quotient der Gesamtlaufzeiten beider Verfahren wird immer kleiner, was bedeutet, dass das Gauß-Southwell-Verfahren im Verhältnis mit wachsendem Schwierigkeitsgrad immer schneller gegenüber dem Gauß-Seidel-Verfahren wird. Für $\epsilon = 1$ ist Gauß-Southwell mehr als doppelt so schnell wie der Gauß-Seidel, für $\epsilon = 100$ sechsmal, für $\epsilon = 10000$ zehnmal und bei der anisotropen Diffusion mit Diffusion nur in einer Richtung ($\epsilon = 0$) mehr als zwölfmal so schnell. Konkret bedeutet das einen Unterschied von 25 Stunden für den Gauß-Seidel und zwei Stunden für den Gauß-Southwell (siehe Tabelle 6.4(d)).

Mit diesen Ergebnissen können wir zum einen die bei den *worst case*-Abschätzungen gewonnenen Erkenntnisse bestätigen, denn das Gauß-Southwell-Verfahren kann den Mehraufwand der adaptiven Durchlaufreihenfolge ohne Schwierigkeiten rechtfertigen, da der Gewinn gegenüber der statischen Durchlaufreihenfolge entsprechend groß ist. Zum anderen zeigen wir hiermit, dass das Gauß-Southwell-Verfahren in der praktischen Anwendung einen noch viel deutlicheren Gewinn bringt und damit insbesondere für große Systeme um ein Vielfaches weniger Rechenzeit benötigt als das Gauß-Seidel-Verfahren.

Daraus ergibt sich die Schlussfolgerung, dass sich die Kosten der Adaptivität des Gauß-Southwell-Verfahrens durch den damit erzielten Gewinn eindeutig rechtfertigen lassen.

7. Zusammenfassung und Ausblick

Zusammenfassung

In dieser Arbeit beschäftigten wir uns mit der effizienten adaptiven Lösung von Multilevelsystemen. Die diesen Gleichungssystemen zugrunde liegende Problematik ist die der numerischen Lösung von partiellen Differentialgleichungen. Partielle Differentialgleichungen werden zum Beispiel zur mathematischen Modellierung physikalischer Vorgänge benötigt und sind allgemein formuliert Gleichungen, in denen partielle Ableitungen auftreten, von denen die gesuchte Funktion abhängt. Um diese Gleichungen numerisch zu lösen, ist es notwendig, von der kontinuierlichen Aufgabenstellung zu einer diskreten zu gelangen, dessen Lösung mittels endlich vieler Daten beschrieben werden kann.

Zur Diskretisierung setzten wir das Galerkin-Verfahren [Bra97] ein. Mit diesem Verfahren lässt sich unter Einsatz der Finiten Elemente eine partielle Differentialgleichung derart diskretisieren, dass ein dünn besiedeltes Gleichungssystem entsteht, das numerisch gelöst werden muss.

Mittels klassischer Iterationsverfahren kann für diese Gleichungssysteme eine Näherungslösung berechnet werden, doch ist die Konvergenzrate dieser Verfahren von der Diskretisierung abhängig. Um eine möglichst gute Näherung an die Lösung der Differentialgleichung zu erhalten, ist es notwendig, die Maschenweite der Diskretisierung sehr fein zu wählen. Die Konvergenzgeschwindigkeit der Iterationsverfahren hängt allerdings von der Maschenweite der Diskretisierung ab. Je feiner die Maschenweite gewählt wird, desto schlechter konvergieren die Lösungsverfahren.

Die Multilevelmethoden [Gri94] liefern ein Verfahren, das stabil ist, d.h. die Konvergenzrate der iterativen Löser ist mit dieser Methode unabhängig von der Maschenweite der Diskretisierung. Mit dem Multilevelverfahren wird ein erweitertes Gleichungssystem aufgestellt, dessen effiziente Lösung der Hauptteil dieser Arbeit war.

Wir stellten das Gauß-Southwell-Verfahren [Sou40] vor, das gegenüber den klassischen Iterationsverfahren die Durchlaufreihenfolge im Gleichungssystem adaptiv wählt und sich so an die Struktur des Systems anpasst. Das Verfahren folgt dem Prinzip der Greedy-Algorithmen und wählt mittels des Residuums jeweils die Komponente aus, deren Relaxation den größten Gewinn verspricht. Dazu ist es notwendig, dass das Verfahren immer den betragsmäßig größten Eintrag des Residuenvektors kennt. Diese Voraussetzung zu erfüllen, ist sehr aufwändig, da zum einen immer das Residuum neu berechnet und zum anderen entweder durchsucht oder sortiert werden muss.

Wir untersuchten das Verfahren darauf, wie sich die entstehenden Kosten reduzieren lassen. In [Rü93] wurde die Idee verfolgt, die Kosten zu reduzieren, indem immer nur ein Teil der Unbekannten betrachtet wurde, doch widersprach dies unserem Ziel, das Gauß-Southwell-Verfahren nicht grundlegend zu ändern bzw. abzuschwächen. Wir analysierten das in [Fox48] allgemein definierte Verfahren und fanden Möglichkeiten, die entstehenden Kosten zu reduzieren. Wir konnten zeigen, dass es nicht notwendig ist, in jedem Schritt das komplette Residuum neu zu berechnen, sondern dass es ausreicht, jeweils nur die betroffenen Komponenten zu aktualisieren.

Die nächste Herausforderung war die, stets effizient auf die jeweils größte Komponente des Residuums zugreifen zu können. Um dieses Problem zu lösen, prüften wir verschiedene Datenstrukturen auf die Möglichkeit, das Residuum so zu speichern, dass es sich schnell ändern ließ und schnell das größte Element bereitstellen konnte. Mit dem Binär-Heap fanden wir eine Datenstruktur, die zu unseren Anforderungen passte, und konnten diese derart erweitern, dass sich alle vom Gauß-Southwell benötigten Operationen in $\mathcal{O}(\log N)$ durchführen lassen, wobei N die Anzahl der Elemente des Residuenvektors ist.

Um den Gauß-Southwell-Algorithmus weiter untersuchen zu können, benötigten wir ein Verfahren, das uns zum Vergleich dienen konnte. Der Gauß-Seidel-Algorithmus war als Standardverfahren die natürliche Wahl dafür, da er dem Gauß-Southwell eng verwandt ist und die gleichen Voraussetzungen hat. Zunächst untersuchten wir die Kosten der beiden Verfahren und stellten fest, dass das Gauß-Southwell-Verfahren für die adaptive Relaxationsreihenfolge einen zusätzlichen Faktor von $\mathcal{O}(\log N)$ gegenüber den Kosten des Gauß-Seidel aufwenden muss.

Als nächsten Schritt war es nötig zu beurteilen, ob sich diese höheren Kosten gegenüber denen einer statischen Durchlaufreihenfolge rechtfertigen lassen. Dazu untersuchten wir für verschiedene Modellprobleme wie Diffusion, Reaktion und anisotrope Diffusion das Konvergenzverhalten beider Algorithmen. Angewandt wurden diese stets auf Multilevelsysteme, um Stabilität zu gewährleisten.

Mit der anisotropen Diffusion konnten wir beide Verfahren schließlich vor große Herausforderungen stellen, denn je größer wir die Unterschiede in den Diffusionsrichtungen wählten, desto aufwändiger war die Lösung der Multilevelsysteme. Es zeigte sich, dass das Gauß-Southwell-Verfahren stets bessere Konvergenzraten als Gauß-Seidel erreichte, für extreme Beispiele aber ebenfalls stark an Konvergenzgeschwindigkeit verlor. Der direkte Vergleich der Konvergenzraten zeigte, dass das adaptive Verfahren für beliebig starke Anisotropie nur ein Drittel der Iterationsschritte des Vergleichsverfahrens benötigte. Dieser Gewinn war nicht ausreichend, um die im *worst case* auftretenden Kosten zu rechtfertigen. Eine Messung der Iterationen in der Praxis zeigte allerdings, dass das Verfahren deutlich weniger Kosten produzierte, als die *worst case*-Abschätzung vermuten ließ. Der Mehraufwand des adaptiven Verfahrens ließ sich daher mit den tatsächlich gemessenen Kosten rechtfertigen, doch zeigte es sich im weiteren Verlauf der Arbeit, dass wir noch einen wesentlich größeren Gewinn gegenüber dem Gauß-Seidel-Verfahren erreichen können.

Es ließ sich nämlich beobachten, dass auch bei extremer Anisotropie das Gauß-Southwell-Verfahren in den ersten Schritten eine sehr viel höhere Konvergenzgeschwindigkeit erreichte als im weiteren Verlauf der Iteration. Dies führte uns zu der Idee, den Algorithmus im Nested-Iteration-Verfahren einzusetzen. Dabei wird, beginnend mit einem kleinen Level, die Näherungslösung jeweils als Startwert für das nächstgrößere Level eingesetzt. Auf die Art können beide Iterationsverfahren im Multilevelsystem immer mit einem sehr guten Startwert beginnen.

Wie erwartet profitierten beide Iterationsverfahren von dieser Vorgehensweise, und wir analysierten die Anzahl der Iterationsschritte, die jeweils nötig waren, um eine genügend gute Lösung zu erreichen. Der Gauß-Southwell erreichte diese stets schneller, und es zeigte sich, dass der Abstand zwischen der Anzahl der im Gauß-Southwell-Verfahren nötigen Schritte zu denen, die das Gauß-Seidel-Verfahren durchführen musste, immer größer wurde, wenn die Anisotropie zunahm oder das Level groß genug wurde. Um den Aufwand der beiden Verfahren vergleichen zu können, gewichteten wir die Anzahl der Iterationsschritte des Gauß-Southwell-Verfahrens mit der *worst case*-Abschätzung des Mehraufwands und verglichen das Ergebnis mit den im Gauß-Seidel-Verfahren notwendigen Schritten. Damit konnten wir für unsere Probleme zeigen, dass das

Gauß-Southwell-Verfahren selbst in dem Fall, dass die Datenstruktur immer die größtmöglichen Kosten produzieren würde, schneller zum Ergebnis gelangt als das Gauß-Seidel-Verfahren.

Da der Gauß-Southwell schon bei der *worst case*-Abschätzung schneller das Ergebnis liefern konnte, ließen sich für die Anwendung in der Praxis noch sehr viel größere Gewinne gegenüber dem Gauß-Seidel-Verfahren erwarten. Folglich war die nächste Untersuchung, der wir die beiden Verfahren unterzogen, die Laufzeitmessung in der Praxis. Zu diesem Zweck legten wir für jedes Level eine Fehlerschranke fest, für die eine Delta-Umgebung um den Diskretisierungsfehler gelegt wurde. Jedes Verfahren musste solange iterieren, bis diese erreicht war. Wir prüften die so gemessenen Laufzeiten im Nested-Iteration-Verfahren für jedes einzelne Level und für das gesamte Verfahren. Dabei zeigte sich klar, dass das Gauß-Southwell-Verfahren um ein Vielfaches schneller zu einer Lösung gelangte als das Gauß-Seidel-Verfahren. Wir beobachteten, dass mit der Größe des Gleichungssystems der Vorsprung des adaptiven Verfahrens immer deutlicher wurde.

Wir konnten mit dem Gauß-Southwell zwar keine absolute Robustheit erreichen, allerdings zeigte sich eindrucksvoll, dass die Konvergenzgeschwindigkeit des Gauß-Southwell-Verfahrens für die anisotrope Diffusion sehr viel weniger abhängig vom Grad der Anisotropie ist, als dies für das Gauß-Seidel-Verfahren der Fall ist. Besonders deutlich ließ sich das bei den Laufzeitvergleichen veranschaulichen. Für den Extremfall, bei dem die anisotrope Diffusion nur eine Diffusionsrichtung hat, stellte sich für das Nested-Iteration-Verfahren mit sieben Leveln heraus, dass der Gauß-Southwell mit einer Gesamtlaufzeit von zwei Stunden über zwölfmal schneller war als der Gauß-Seidel, der unter gleichen Bedingungen 25 Stunden benötigte.

Damit konnten wir zeigen, dass sich die Kosten des Gauß-Southwell-Verfahrens, die wir mit einer geschickten Aktualisierung des Residuums und einer für dieses Problem angepassten Datenstruktur senken konnten, nicht nur problemlos rechtfertigen lassen, sondern wir konnten zeigen, dass die adaptive Durchlaufreihenfolge dem Algorithmus die Möglichkeit gibt, wesentlich effizienter zu arbeitet. Dadurch tritt der benötigte Aufwand weit hinter dem erzielten Gewinn zurück. Das Gauß-Southwell-Verfahren ist also insbesondere bei großen dünn besiedelten Gleichungssystemen, wie sie üblicherweise bei Multilevelverfahren entstehen, dem Gauß-Seidel-Verfahren bei weitem vorzuziehen.

Ausblick

Wir konnten bei der Untersuchung der numerischen Ergebnisse zeigen, dass sich der Gauß-Southwell-Algorithmus effizient einsetzen lässt. Auch stellten wir fest, dass das Verfahren in der Praxis deutlich weniger Kosten produziert, als die *worst case*-Abschätzung das vermuten lässt. Die Untersuchung der Einsatzmöglichkeiten des Verfahrens sind allerdings noch nicht abgeschlossen. So haben wir in dieser Arbeit nur partielle Differentialgleichungen mit homogenen Dirichlet-Randbedingungen betrachtet, die Untersuchung des Verhaltens des Gauß-Southwell für Probleme mit inhomogenen oder Neumann-Randbedingungen stehen aus.

Weiterhin sollte das Verhalten des Gauß-Southwell für die Konvektions-Diffusionsgleichung untersucht werden. Hier besteht die Hoffnung, dass sich das Verfahren insbesondere für konvektionsdominante Probleme besonders gut eignet, da es sich an die Konvektionsrichtung adaptiv anpassen und dieser folgen kann. So wäre es möglich, einen noch größeren Vorsprung gegenüber Verfahren mit einer festen Durchlaufreihenfolge herauszuarbeiten.

Wir haben den Gauß-Southwell in dieser Arbeit für Multilevelsysteme eingesetzt, dabei konnten wir beobachten, dass das Verfahren die Struktur dieser Systeme ausnutzt, ohne speziell dafür modifiziert worden zu sein. Für andere Gleichungssysteme, die eine Struktur besitzen, gilt

es zu untersuchen, ob der Gauß-Southwell sich dieser ebenfalls anpassen und damit wieder einen Vorteil gegenüber anderen Iterationsverfahren erreichen kann.

Für die anisotrope Diffusion haben wir das Verfahren eingehend untersucht und dabei festgestellt, dass dieses insbesondere für extreme Anisotropie deutlich schneller konvergierte als klassische Iterationsverfahren. Statt wie bisher Modellfunktionen als Lösung zu verwenden, wäre es nun interessant, konkrete Anwendungen zu betrachten, in deren Problemstellung starke Anisotropie auftritt, wie etwa bei der Kantendetektion mit anisotroper Diffusion aus [PM90]. Bei solchen Aufgabenstellungen sollte der Gauß-Southwell deutlich von seiner Adaptivität profitieren und sie sehr gut lösen können.

Eine weitere Idee bezieht sich auf den Algorithmus selbst. Dieser entscheidet nur anhand des Residuums über die Durchlaufreihenfolge, dabei wird stets die Komponente mit dem größten Residuumswert relaxiert. Überlegungen wie die aus [Fox48], sich nicht alleine an $\operatorname{argmax}_i(|r_i|)$, also an dem jeweils größten Eintrag des Residuums, zu orientieren, sondern die Einträge zu gewichten, beispielsweise mit den Diagonaleinträgen der Steifigkeitsmatrix und dann $\operatorname{argmax}_i(|r_i/a_{ii}|)$ zu suchen, stehen aus. Für diese Modifikation müsste analysiert werden, inwiefern das Verfahren dadurch beschleunigt, oder für welche Problemstellungen dies sinnvoll eingesetzt werden kann.

Wir haben in dieser Arbeit den Gauß-Southwell nur insofern modifiziert, dass wir die im Verfahren nötigen Kosten reduziert haben. Man könnte den Algorithmus aber auch grundlegend erweitern, so dass man nicht versucht, das aktuelle Residuum auf Null zu reduzieren, sondern in Kauf nimmt, dass es größer Null bleibt, dafür aber die Residuenwerte der aktuellen und der benachbarten Komponenten soweit verkleinert werden, dass in der Summe eine größere Verbesserung des Residuums des Gleichungssystems erreicht werden kann. Mittels dieser Überrelaxation würde man das Verfahren zu einer Art SOR-Gauß-Southwell erweitern.

Desweiteren lohnt es sich, Verfahren, die ähnlich zum Gauß-Southwell einzelne Komponenten aus einem sortieren Suchraum auswählen, darauf zu untersuchen, ob sich die hier verwendete Datenstruktur analog zur Kostenreduktion einsetzen lässt. Beispielsweise setzen die Optimierungsverfahren in [TB91] und die Data Mining-Algorithmen in [RMW02] eine mit dem Gauß-Southwell verwandte Methode ein. Dabei wird in jedem Schritt jeweils über die Variable minimiert, die den größten Gewinn verspricht. Für diese zur Klasse der *Greedy*-Algorithmen gehörenden Verfahren ließen sich die Kostenreduktionen, die wir im Gauß-Southwell-Verfahren erreichen konnten, indem wir das Residuum in einer entsprechenden Datenstruktur speicherten, analog verwenden, um die dort auftretenden Kosten zu reduzieren.

So verbleiben interessante Fragestellungen, die in künftigen Arbeiten untersucht werden können.

A. Anhang

Bei der Untersuchung der anisotropen Diffusion in Abschnitt 6.2.3 beobachteten wir, dass der Diskretisierungsfehler in der L^2-Norm pro Level jeweils unabhängig vom Grad der Anisotropie ist.

Wir konnten in der Literatur keine Bemerkung oder einen Beweis zu dieser Beobachtung finden, daher wollen wir hier kurz unsere Überlegungen dazu skizzieren. Wir beschränken uns dabei auf die Finite Differenzen-Diskretisierung mit dem 5-Punkte-Stern für den Laplace und die L^∞-Norm, um mit zentralen Sätzen aus [Hac96] argumentieren zu können. Für die Finiten Elemente mit den 9-Punkte-Stern und der L^2-Norm sollte sich diese Aussage analog übertragen lassen.

Satz A.1. *Der Finite Differenzen-Diskretisierungsfehler ist für den anisotropen Laplace-Operator mit dem 5-Punkte-Stern und in der L^∞-Norm unabhängig vom Grad der Anisotropie.*

Beweisskizze. Um zu zeigen, dass der Diskretisierungsfehler für die anisotrope Diffusion in der L^∞-Norm von Epsilon unabhängig ist, bedienen wir uns der Konvergenzaussage [Hac96, Satz 4.5.3]. Man zeigt die Konvergenz eines Diskretisierungsverfahrens üblicherweise damit, dass man beweist, dass das Verfahren konsistent und stabil ist.

Sei u die exakte Lösung, u_h die Näherungslösung, Δ der Laplace-Operator, Δ_h der diskrete Laplace-Operator, Δ^ϵ der anisotrope Laplace, Δ_h^ϵ der diskretisierte anisotrope Laplace, R_h die Restriktion und L_h die zum Differentialoperator gehörende Matrix.

Wir erhalten laut [Hac96, Satz 4.5.3] für den Diskretisierungsfehler des Laplace-Operators die folgende Darstellung

$$u_h - R_h u \quad = \quad L_h^{-1}\left(\Delta_h R_h u - R_h \Delta u\right)$$

und entsprechend in der L^∞-Norm die Abschätzung

$$||u_h - R_h u||_\infty \quad \leq \quad \left|\left|L_h^{-1}\right|\right|_\infty \cdot \; ||\Delta_h R_h u - R_h \Delta u||_\infty$$

Falls ein Diskretisierungsverfahren stabil und konsistent ist, müssen weiterhin folgende Ungleichungen

$$\begin{aligned} & \left|\left|L_h^{-1}\right|\right|_\infty \leq c && \text{(Stabilität)} \\ \text{und} \qquad & ||\Delta_h R_h u - R_h \Delta u||_\infty \leq K \cdot h^k ||u||_{C^{k+2}} && \text{(Konsistenz)} \end{aligned}$$

erfüllt sein. Nun gilt es, den Stabilitäts- und Konsistenzterm konkret abzuschätzen und zu überprüfen, inwiefern diese im anisotropen Fall abhängig von Epsilon sind.

Wenden wir uns zunächst dem Konsistenzterm zu. Für diesen gilt mittels der Taylor-Entwicklung die Abschätzung

$$||\Delta_h R_h u - R_h \Delta u||_\infty \leq h^2 \frac{1}{6} ||u||_{C^4},$$

vergleiche dazu [Hac96, Bemerkung 4.5.2]. Wir möchten hier allerdings den anisotropen Laplace betrachten. In diesem Fall gilt analog

$$||\Delta_h^\epsilon R_h u - R_h \Delta^\epsilon u||_\infty \;\leq\; (1+\epsilon)\, h^2 \frac{1}{12}\, ||u||_{C^4}\,.$$

Folglich ist die Konsistenz abhängig von Epsilon.

Wenden wir uns nun der Stabilität zu und schätzen diese ab. Da L_h positiv definit ist, gilt

$$\left|\left|L_h^{-1}\right|\right|_2 \;=\; \frac{1}{\lambda_{\min}}, \tag{A.1}$$

und über die Eigenwerte der zum 5-Punkte-Stern gehörenden Matrix L_h wissen wir, dass

$$\lambda_{\nu\mu} \;=\; 4h^{-2}\left(\sin^2\left(\nu h\frac{\pi}{2}\right) + \sin^2\left(\mu h\frac{\pi}{2}\right)\right)$$

mit $1 \leq \nu, \mu \leq n-1$, vergleiche dazu [Hac96, Lemma 4.4.2]. Für den anisotropen Fall ist also entsprechend

$$\lambda_{\nu\mu}^\epsilon \;=\; 4h^{-2}\left(\sin^2\left(\nu h\frac{\pi}{2}\right) + \epsilon\cdot\sin^2\left(\mu h\frac{\pi}{2}\right)\right). \tag{A.2}$$

Um den größtmöglichen Wert für (A.1) zu finden, gilt es, den kleinsten Eigenwert abzuschätzen. Folglich müssen wir das Minimum von (A.2) finden

$$\begin{aligned}
\lambda_{\min} \;=\; \min_{1\leq\nu,\mu\leq n-1}\lambda_{\nu\mu}^\epsilon \;&=\; \min_{\nu,\mu}\left(4h^{-2}\left[\sin^2\left(\nu h\frac{\pi}{2}\right) + \epsilon\cdot\sin^2\left(\mu h\frac{\pi}{2}\right)\right]\right)\\
&=\; 4h^{-2}\left(\min_{\nu,\mu}\left[\underbrace{\sin^2\left(\nu h\frac{\pi}{2}\right)}_{\in\,[0,1]} + \epsilon\cdot\underbrace{\sin^2\left(\mu h\frac{\pi}{2}\right)}_{\in\,[0,1]}\right]\right)\\
&\overset{(*)}{=}\; 4h^{-2}\left[\sin^2\left(\min_\nu\left(\nu h\frac{\pi}{2}\right)\right) + \epsilon\cdot\sin^2\left(\min_\mu\left(\mu h\frac{\pi}{2}\right)\right)\right]\\
&\overset{(**)}{=}\; 4h^{-2}\left[\sin^2\left(h\frac{\pi}{2}\right) + \epsilon\cdot\sin^2\left(h\frac{\pi}{2}\right)\right]\\
&=\; 4h^{-2}(1+\epsilon)\sin^2\left(h\frac{\pi}{2}\right).
\end{aligned}$$

$(*)$: Der Fall $\sin^2(\nu h\frac{\pi}{2}) = 0$ kann nicht angenommen werden, da $1 \leq \nu \leq n-1$ und $h > 0$. Außerdem gilt $h = \frac{1}{n}$, also $\nu h < 1$ und somit $\left(\nu h\frac{\pi}{2}\right) \in \left(0, \frac{\pi}{2}\right)$, analog für μ. In diesem Intervall ist $\sin^2$ streng monoton steigend, also folgt der nächste Schritt.
$(**)$: Das Minimum wird für $\mu = \nu = 1$ angenommen.

Es gilt also

$$\left|\left|L_h^{-1}\right|\right|_2 \;=\; \frac{1}{\lambda_{\min}} \;=\; \frac{h^2}{4}\,(1+\epsilon)^{-1}\sin^{-2}\left(h\frac{\pi}{2}\right).$$

Da $L^\infty \subseteq L^2$ gilt, lässt sich dies auch auf die L^∞-Norm übertragen

$$\left|\left|L_h^{-1}\right|\right|_\infty \;\leq\; \frac{h^2}{4}(1+\epsilon)^{-1}\sin^{-2}\left(h\frac{\pi}{2}\right).$$

Folglich ist die Stabilität ebenfalls von Epsilon abhängig.

Somit gilt also für die Konvergenz des Diskretisierungsverfahrens für den anisotropen Laplace

$$\begin{aligned}
||u_h - R_h u||_\infty &\leq \left|\left|L_h^{-1}\right|\right|_\infty \cdot \; ||\Delta_h^\epsilon R_h u - R_h \Delta^\epsilon u||_\infty \\
&\leq \frac{h^2}{4}\,(1+\epsilon)^{-1} \sin^{-2}\left(h\frac{\pi}{2}\right) \cdot (1+\epsilon)\, h^2 \frac{1}{12}\, ||u||_{C^4} \\
&= \frac{h^4}{48} \sin^{-2}\left(h\frac{\pi}{2}\right) ||u||_{C^4} \\
&= \frac{h^2}{48}\, ||u||_{C^4} \cdot \underbrace{h^2 \sin^{-2}\left(h\frac{\pi}{2}\right)}_{\leq 1 \;\; (***)} \\
&\leq \frac{h^2}{48}\, ||u||_{C^4}\,.
\end{aligned}$$

$(***)$: da $h^2 \leq \sin^{-2}\left(h\frac{\pi}{2}\right)$ für $h \in (0,1]$.

Damit konnten wir zeigen, dass die Konvergenz der Diskretisierung des anisotropen Laplace mit den Finiten Differenzen in der L^∞-Norm unabhängig von Epsilon ist. Folglich ist auch der Iterationsfehler im ausiterierten Zustand unabhängig vom Grad der Anisotropie. □

Literaturverzeichnis

[Ape99] Apel, T.: *Anisotropic finite elements: Local estimates and applications.* Teubner Stuttgart, 1999.

[Bol03] Bolten, M.: *Adaptive iterative Gleichungssystemlöser in der Bildverarbeitung.* Universität zu Lübeck, Institut für Mathematik, 2003. Studienarbeit.

[boo] *BOOST C++ Libraries.* `http://www.boost.org`.

[Bra97] Braess, D.: *Finite Elemente.* Springer-Verlag, Berlin, 1997.

[DGST88] Driscoll, J.R., H.N. Gabow, R. Shrairman und R.E. Tarjan: *Relaxed heaps: an alternative to Fibonacci heaps with applications to parallel computation.* Commun. ACM, 31(11):1343–1354, 1988.

[Fox48] Fox, L.: *A short account of relaxation methods.* The Quarterly Journal of Mechanics and Applied Mathematics, 1(1):253, 1948.

[FT87] Fredman, M.L. und R.E. Tarjan: *Fibonacci heaps and their uses in improved network optimization algorithms.* J. ACM, 34(3):596–615, 1987.

[FW60] Forsythe, G.E. und W.R. Wasow: *Finite-difference methods for partial differential equations.* Wiley New York, 1960.

[Gau03] Gauss, C.F.: *Werke Bd. 9*, Kapitel Brief von Gauß an Gerling, Seite 280ff. Teubner, Leipzig, 1903.

[Gri90] Griebel, M.: *Zur Lösung von Finite-Differenzen- und Finite-Element-Gleichungen mittels der Hierarchischen Transformations-Mehrgitter-Methode.* Technischer Bericht, SFB Bericht 342/4/90 A, Institut für Informatik, TU München, 1990.

[Gri94] Griebel, M.: *Multilevelmethoden als Iterationsverfahren über Erzeugendensystemen.* Teubner, Stuttgart, 1994.

[Gri03] Griebel, M.: *Wissenschaftliches Rechnen 1 und 2.* Rheinische Friedrich-Wilhelms-Universität Bonn, Institut für Numerische Simulation, 2003. Vorlesungsskript.

[Hac91] Hackbusch, W.: *Iterative Lösung großer schwachbesetzter Gleichungssysteme.* Teubner-Verlag, Stuttgart, 1991.

[Hac96] Hackbusch, W.: *Theorie und Numerik elliptischer Differentialgleichungen.* Teubner-Verlag, Stuttgart, 1996.

[LY08] Luenberger, D.G. und Y. Ye: *Linear and nonlinear programming*. Springer Verlag, 2008.

[Mei08] Meister, A.: *Numerik linearer Gleichungssysteme*. Vieweg u. Sohn Verlag, Wiesbaden, 2008.

[Osw94] Oswald, P.: *Multilevel finite element approximation*. Teubner Stuttgart, 1994.

[PM90] Perona, P. und J. Malik: *Scale-space and edge detection using anisotropic diffusion*. IEEE Transactions on pattern analysis and machine intelligence, 12(7):629–639, 1990.

[RMW02] Rätsch, G., S. Mika und M.K. Warmuth: *On the convergence of leveraging*. Advances in Neural Information Processing Systems, 1:487–494, 2002.

[Rü93] Rüde, U.: *Mathematical and computational techniques for multilevel adaptive methods*. Society for Industrial and Applied Mathematics, 1993.

[Sch01] Schöning, U.: *Algorithmik*. Spektrum, Akad. Verl., 2001.

[Smi85] Smith, G.D.: *Numerical solution of partial differential equations: finite difference methods*. Oxford University Press, USA, 1985.

[Sou40] Southwell, R.V.: *Relaxation methods in engineering science: a treatise on approximate computation*. Oxford University Press Oxford, 1940.

[Sou46] Southwell, R.V.: *Relaxation methods in theoretical physics: a continuation of the treatise Relaxation methods in engineering science*. Clarendon Press, Oxford, 1946.

[Sto94] Stoer, J.: *Numerische Mathematik 1 und 2*. Springer-Verlag, Berlin, 1994.

[Str95] Strauss, W.A.: *Partielle Differentialgleichungen*. Vieweg u. Sohn Verlag, Wiesbaden, 1995.

[TB91] Tseng, P. und D.P. Bertsekas: *Relaxation methods for problems with strictly convex costs and linear constraints*. Mathematics of operations research, 16:462–481, 1991.

[Vui78] Vuillemin, J.: *A data structure for manipulating priority queues*. Commun. ACM, 21(4):309–315, 1978.

[WK02] Wollny, G. und F. Kruggel: *Computational cost of nonrigid registration algorithms based on fluid dynamics*. IEEE transactions on medical imaging, 21(8):946–952, 2002.

Index