Parallele Gleichungssystemlöser für Elastizitätsprobleme mit nichtlinearen oder zeitlich veränderlichen Materialparametern

Diplomarbeit

zur Erlangung des akademischen Grades eines
Magisters
an der Naturwissenschaftlichen Fakultät der
Karl-Franzens-Universität Graz

vorgelegt von

Patrick Ditz

betreut von

Univ.-Prof. Dipl.–Ing. Dr. **Gundolf Haase**

erstellt am 30.07.2012

Institut für Mathematik und wissenschaftliches Rechnen
Heinrichstraße 36, 8010 Graz

2012

Danksagungen

An dieser Stelle möchte ich mich bei Prof. Dipl.-Ing. Dr. Gundolf Haase bedanken, der mich während meiner Diplomarbeit betreut und umfangreich unterstützt hat. Außerdem möchte ich mich herzlich bei Mag. Dr.rer.nat. Manfred Liebmann und Dipl.-Ing. (FH) Aurel-Vasile Neic für die unerlässliche Hilfe bei der Implementation bedanken. Mein ganz besonderer Dank geht an meine Mutter für die finanzielle Unterstützung.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Bezeichnungen

Elastizitätstheorie

Ω	offene beschränkte Menge im $\mathbb{R}^3$
u	Verschiebung
ϕ	Deformation
C	Cauchy-Greenscher Tensor
E	Verzerrungstensor
ϵ	Verzerrung in der linearen Näherung
t	Cauchyscher Spannungsvektor
T	Cauchyscher Spannungstensor
σ	Spannung in der linearen Näherung
λ, μ	Lamè-Konstanten
$\mathtt{E}$	Elastizitätsmodul
ν	Poissonzahl
$\mathcal{C}$	Elastizitätstensor

Gleichungssystemlöser

ω	Relaxationsparameter
Ω	Indexmenge der Variablen
C	Grobgitterpunkte
F	Feingitterpunkte
A, A_h	Steifigkeits- oder Systemmatrix
P, I_{2h}^h	Prolongationsoperator
R, I_h^{2h}	Restriktionsoperator
A_C, A_{2h}	Grobgitteroperator
ν	Anzahl der Glättungen
$\mathtt{C}$	Matrix zur Vorkonditionierung

Einleitung

Zahlreiche Phänomene aus Natur und Technik lassen sich in einem mathematischen Modell darstellen. Ein derartiges Modell beschreibt die Beziehungen zwischen den für das Problem wichtigen Größen im Allgemeinen durch ein System von Differentialgleichungen. Die Lösung solcher Differentialgleichungen ist selten analytisch möglich, so dass auf numerische Verfahren zurückgegriffen werden muss.
Diese Arbeit beschäftigt sich mit dem Phänomen der elastischen Verformung von Materialien. Dabei wird das Material einer Kraft ausgesetzt und man ist an der dadurch resultierenden Verschiebung des Materials interessiert. Weiters studiert man die Verzerrungen und Spannungen, die durch diese Deformation hervorgerufen werden. Die 3 Grundbausteine der Elastizitätstheorie sind

- die Kinematik,
- die Gleichgewichtsbedingungen
- sowie die Materialgesetze,

welche in Kapitel 1 behandelt werden. Die mathematische Modellierung dieser Problemstellung liefert ein System von elliptischen partiellen Differentialgleichungen. Für die numerische Behandlung stellt die Finite Elemente Methode ein häufig verwendetes Hilfsmittel dar. Dazu wird das System in einen Integralausdruck übergeführt. Dies liefert die Variationsformulierung des Problems, welche durch eine Bilinearform dargestellt, und mittels Hilbertraum-Methoden behandelt wird. Eine endlichdimensionale Approximation liefert ein algebraisches (lineares oder nichtlineares) Gleichungssystem mit endlich vielen skalaren Unbekannten $u_1, \ldots, u_n$. Je nachdem wie die Materialien auf die Belastungen reagieren, wollen wir folgende Fälle untersuchen:

- das linear elastische Verhalten, welches durch das Hooksche Gesetz beschrieben wird. In diesem Fall liefert die Diskretisierung ein lineares Gleichungssystem

  $$Au = f$$

  mit $u, f \in \mathbb{R}^n$ und $A \in \mathbb{R}^{n \times n}$. Dieses Verhalten ist typisch für kleine Deformationen sowie für harte, spröde Materialien. Für zeitlich veränderliche Stoffparameter ist die Systemmatrix auch zeitabhängig und man bekommt

  $$A(t)\, u_t = f_t$$

  mit $u_t, f_t \in \mathbb{R}^n$ und $A(t) \in \mathbb{R}^{n \times n}$.

- das nicht-linear elastische Verhalten, bei der die Spannung nichtlinear von der Deformation abhängt. Das resultierende nichtlineare Gleichungssystem wird beschrieben durch

$$A\left(u_t\left(t\right)\right)u_t = f_t$$

mit $A\left(u_t\left(t\right)\right) \in \mathbb{R}^{n\times n}$. Davon betroffene Materialien, wie Gummi, sind fast inkompressible Materialien. In diesem Fall kann das Hookesche Gesetz nicht angewandt werden, bei dem die Verschiebung benachbarter Punkte proportional zum Abstand dieser Punkte ist. Diese Situation ist in der Natur die Regel. Wo möglich wird versucht dieses Problem in einer linearen Näherung zu behandeln. Insbesondere für kleine Deformationen ist diese Näherung häufig gerechtfertigt.

Die Lösung eines derartigen Gleichungssystems mittels einer direkten Methode, wie etwa dem Gaußalgorithmus, der *LU*- oder der Choleskyzerlegung, erfordern einen Rechenaufwand von $O\left(n^3\right)$ arithmetische Operationen und einen Speicheraufwand von $O\left(n^2\right)$ Werten. Die Diskretisierung der Differentialgleichungen liefert allerdings in der Regel dünnbesetzte Systemmatrizen, d.h. sie enthalten pro Matrixzeile unabhängig von der Problemgröße nur wenige Nichtnulleinträge, was den Speicheraufwand des Gleichungssystems auf $O\left(n\right)$ reduziert. Ziel ist es ein Verfahren zu entwickeln, welches $O\left(n\right)$ arithmetische Operationen beansprucht. Solche Verfahren werden als optimal bezeichnet.
Einen Ansatz in diese Richtung stellen iterative Löser, wie das Jakobi-, das Gauß-Seidel-Verfahren oder das Verfahren konjugierter Gradienten, dar. Die Konvergenzrate solcher Verfahren ist abhängig von der Konditionszahl der Systemmatrix, welche sich bei zunehmender Anzahl der Unbekannten verschlechtert. Wegen der steigenden Anzahl von zu berechnenden Variablen zum Erhalt einer besseren Approximationsgüte ist es wünschenswert, das die Berechnungszeit des Lösungsalgorithmus asymptotisch nur linear in der Anzahl der Unbekannten wächst. Dies erreicht man durch Verwendung von Mehrgitterverfahren, siehe [8]. Klassische geometrische Mehrgitterverfahren setzen Diskretisierungen auf unterschiedlichen Leveln voraus. Für komplizierte geometrische Strukturen erweist sich diese Vorgehensweise oftmals als schwierig. Deshalb wurden algebraische Mehrgitterverfahren entwickelt, welche ohne Kenntnis über die zugrundeliegende Diskretisierung, nur mit Hilfe der Einträge der Systemmatrix, eine "Gitter"-Hierarchie erzeugen, siehe [12]. Diese Thematik wird in Kapitel 2 behandelt. Darin wird für die Lösung des Ausgangsproblems das Algebraische Mehrgitterverfahren als Vorkonditionierer in einem konjugierten Gradientenverfahren präsentiert.
Da, speziell für 3D-Elastizitätsprobleme, die Anzahl der Unbekannten rasch in die Millionen gehen können, sind parallele Versionen der erwähnten Lösungsverfahren erforderlich. Diese werden in dem Softwarepaket [11] zur Verfügung gestellt, welches für effiziente parallele Umsetzungen auf modernen Hardwaresystemen designt wurde. Daher wählen wir diese Software-Applikation für die Umsetzung des Problems. Für eine ausführliche Anleitung hierzu sei auf [10] verwiesen. Kapitel 3 liefert einerseits einen kurzen Einblick in die Toolbox und das verwendete Speicherformat für die Systemmatrix und andererseits eine konkrete Anleitung für die Lösung eines diskretisierten Elastizitätproblems. Dabei wird eine Umsetzung sowohl auf den Haupt-

prozessoren (CPU's) als auch mit Zuhilfenahme der Graphikprozessoren (GPU's) des vorhandenen Computersystems präsentiert.
Abschließend werden numerische Ergebnisse eines 3D-Elastizitätproblems dargestellt. Hierbei zeigt sich einerseits, dass im Zusammenhang mit mehrdimensionalen Problemen skalare algebraische Mehrgitterverfahren wesentlich schlechtere Konvergenzeigenschaften als algebraische Mehrgitterverfahren für Systeme aufweisen, und andererseits, dass der Einsatz von Parallelisierungen mit GPU-Unterstützung sehr effiziente Realisierungen liefern.

Da die verwendete Version der Toolbox auf skalare Problemstellungen limitiert war, bestand meine Aufgabe darin diese für Systeme zu erweitern. Dafür war es von Nöten, einerseits die Ausgangsdaten für die numerischen Ergebnisse auf die gewünschte Blockstruktur zu übertragen, und andererseits die Quellcodes für das algebraische Mehrgitterverfahren, für die in Kapitel 2 präsentierte Theorie, zu adaptieren.

Kapitel 1

Elastizitätstheorie

In der Elastizitätstheorie betrachtet man den Zustand von Körpern unter der Einwirkung von Kräften. Insbesondere studiert man die Verzerrungen und Spannungen, die durch Deformationen erzeugt werden.
Für die wesentlichen Grundbegriffe der Elastizitätstheorie folgen wir den Darstellungen von Braess in [2]. Ausführlichere Beschreibungen entnehmen wir aus [9] und [1].

1.1 Kinematik - Beschreibung von Deformation

Man geht davon aus, dass für den Körper eine Referenzkonfiguration $\bar{\Omega}$ bekannt ist. Dabei sei $\bar{\Omega}$ der Abschluss einer beschränkten, offenen Menge $\Omega \subset \mathbb{R}^3$. Der aktuelle Zustand des Körpers wird durch eine Abbildung ϕ

$$\phi : \bar{\Omega} \longrightarrow \mathbb{R}^3$$

gegeben. Schreibt man die Funktion ϕ in der Form

$$\phi = Id + u \tag{1.1}$$

so erhält man mit $u : \bar{\Omega} \longrightarrow \mathbb{R}^3$ die Verschiebung, der jeder Punkt der Referenzkonfiguration unterworfen ist. Im Detail bedeutet dies, dass sich ein materieller Punkt x in $\bar{\Omega}$ durch Einfluss externer Kräfte in die neue Position $x + u(x)$ in der deformierten Konfiguration bewegt, was mittels der Abbildung 1.1 illustriert wird.
Die Abbildung ϕ wird im folgenden stets als genügend glatt vorausgesetzt. Es ist ϕ eine Deformation, wenn $det(\nabla\phi) > 0$ gilt. Dabei ist $\nabla\phi$ der Deformationsgradient. Wichtig sind die durch ϕ erzeugten Änderungen des Linienelements. Betrachtet man den Euklidischen Abstand

$$||\phi(x + dx) - \phi(x)||^2 = ||\nabla\phi \cdot dx||^2 + o\left(||dx||^2\right)$$

sieht man, dass für die lokale Änderung der Längen die Matrix

$$C := \nabla\phi^T \nabla\phi \tag{1.2}$$

ausschlaggebend ist. C heißt Cauchy-Greenscher Tensor.

Bemerkung 1.1 ***(Charakterisierung von Starrkörperbewegungen)*** *Es sei $\Omega \subset \mathbb{R}^3$ offen und zusammenhängend. Wenn für eine Bewegung $\phi \in C^1(\Omega)$ der zugehörige Verzerrungstensor die Relation*

$$C(x) = I \text{ für alle } x \in \Omega$$

erfüllt, stellt ϕ eine Starrkörperbewegung dar, d.h. es ist $\phi(x) = Qx + b$, wobei Q eine orthogonale Matrix ist.

Die durch

$$E := \frac{1}{2}(C - I) \tag{1.3}$$

definierte Abweichung von der Identität bezeichnet man als Verzerrung. Ausgangspunkt der Kontinuumsmechanik ist somit die mathematische Beschreibung von Verzerrungen. Verzerrungstensoren beschreiben das Verhältnis von Momentankonfiguration zur Ausgangskonfiguration bei der Deformation von kontinuierlichen Körpern und damit Veränderung der gegenseitigen Lagebeziehungen der Materieelemente.

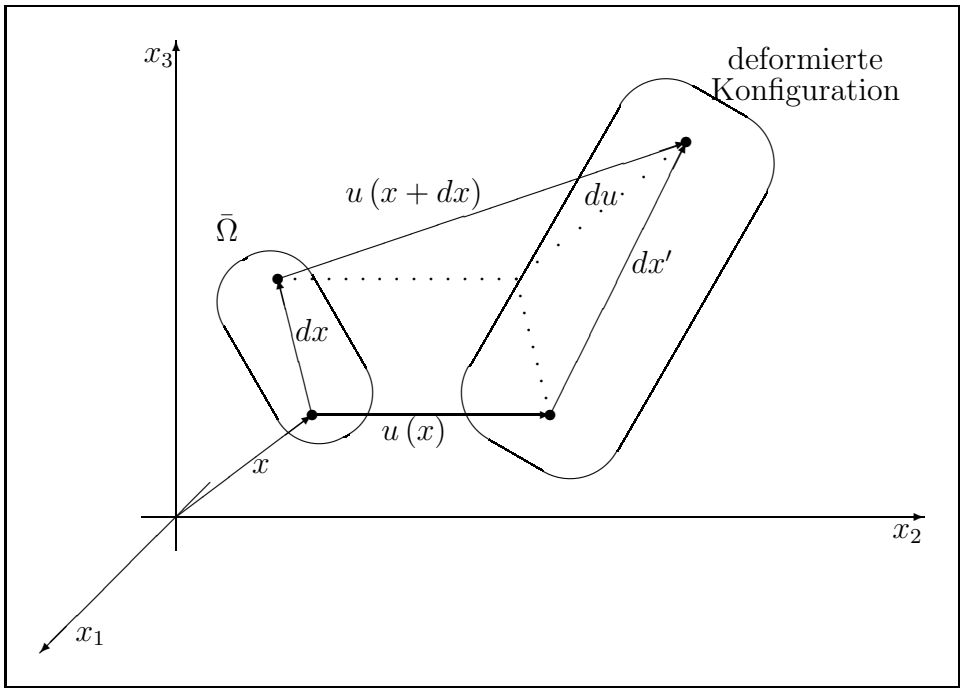


Abbildung 1.1: Der Abstand $||dx||$ des Punktes x zu einem benachbarten Punkt $x + dx$ ändert sich durch die Deformation zu $||dx'||$.

Durch Einsetzen von (1.1) und (1.2) in (1.3) ergibt sich

$$\begin{aligned} E &= \frac{1}{2}\left(\nabla\phi^T \nabla\phi - I\right) \\ &= \frac{1}{2}\left(\nabla(Id+u)^T \nabla(Id+u) - I\right) \\ &= \frac{1}{2}\left((I+\nabla u)^T (I+\nabla u) - I\right) \end{aligned}$$

$$= \frac{1}{2}\left(\nabla u + \nabla u^T + \nabla u^T \nabla u\right). \tag{1.4}$$

In der linearen Elastizitätstheorie werden die quadratischen Terme vernachlässigt, und wir erhalten für die Verzerrungen, die wir in der linearen Näherung mit ϵ bezeichnen, gerade die symmetrische Ableitung Du:

$$\epsilon := Du := \frac{1}{2}\left(\nabla u^T + \nabla u\right) \tag{1.5}$$

Diese Linearisierung liefert den sogenannten Cauchy-Verzerrungstensor. Seine Komponenten lauten

$$\epsilon_{ij} = \begin{pmatrix} \epsilon_{11} & \epsilon_{12} & \epsilon_{13} \\ \epsilon_{21} & \epsilon_{22} & \epsilon_{23} \\ \epsilon_{31} & \epsilon_{32} & \epsilon_{33} \end{pmatrix} = \begin{pmatrix} \frac{\partial u_1}{\partial x_1} & \frac{1}{2}\left(\frac{\partial u_1}{\partial x_2} + \frac{\partial u_2}{\partial x_1}\right) & \frac{1}{2}\left(\frac{\partial u_1}{\partial x_3} + \frac{\partial u_3}{\partial x_1}\right) \\ \frac{1}{2}\left(\frac{\partial u_2}{\partial x_1} + \frac{\partial u_1}{\partial x_2}\right) & \frac{\partial u_2}{\partial x_2} & \frac{1}{2}\left(\frac{\partial u_2}{\partial x_3} + \frac{\partial u_3}{\partial x_2}\right) \\ \frac{1}{2}\left(\frac{\partial u_3}{\partial x_1} + \frac{\partial u_1}{\partial x_3}\right) & \frac{1}{2}\left(\frac{\partial u_3}{\partial x_2} + \frac{\partial u_2}{\partial x_3}\right) & \frac{\partial u_3}{\partial x_3} \end{pmatrix}.$$

1.2 Gleichgewichtsbedingungen

Nachdem im vorangegangenen Abschnitt eine Einführung in die Geometrie des elastischen Körpers gegeben wurde, werden hier die Kräfte, welche auf den Körper wirken, behandelt. In der Mechanik wird das Wirken von Kräften axiomatisch eingeführt. Die Kraft-Wechselwirkungen können vollständig auf

- flächenhaft verteilte Kräfte (kurz Flächenkräfte),
- volumenhaft verteilte Kräfte (kurz Volumenkräfte)

zurückgeführt werden. Neben dieser Unterteilung nach dem Wirkungsort werden Kräfte nach ihrem Bezug zu dem Körper unterteilt in äußere Kräfte und innere Kräfte.
Wirkt eine externe Kraft auf einen Körper, so verformt sich dieser, entfällt die Belastung, so hat der Körper das Bestreben, die Formänderung rückgängig zu machen. Durch das Anbringen der äußeren Kräfte entstehen im Inneren des Körpers Kräfte, die sich den äußeren Kräften entgegensetzen und das Bestreben haben, die frühere Gestalt, den Referenzzustand, wieder herzustellen. Mit anderen Worten, äußere Kräfte wirken von außen auf ein mechanisches System, sie verursachen innere Kräfte.
Deutlich wird diese Unterteilung an den äußeren Kräften: Eine typische Volumenkraft ist die Schwerkraft, während die Belastung einer Brücke eine Flächenkraft darstellt.
Die Volumenkräfte $f : \Omega \longrightarrow \mathbb{R}^3$ liefern im Volumenelement dV die Kraft $f dV$. Die Flächenkräfte sind durch eine Funktion $t : \Omega \times S^2 \longrightarrow \mathbb{R}^3$ spezifiziert, wobei S^2 die Einheitssphäre im $\mathbb{R}^3$ bezeichnet: Sei V eine beliebige Teilmenge von Ω, und dA ein Oberflächenelement mit der Normalen n. Dann liefert das Flächenelement dA einen Beitrag $t(x, n)\, dA$ zur Kraft, der auch von der Richtung n abhängen kann. Der Vektor $t(x, n)$ heißt Cauchyscher Spannungsvektor.

In der Kontinuumsmechanik ist das folgende Axiom die Grundlage der Behandlung statischer Probleme. Es besagt, dass sich im Gleichgewichtszustand in jedem herausgeschnittenen Teilsystem alle Kräfte bzw. Momente zu Null addieren.

Axiom 1.1 ***([1, §1, S. 12] Axiom des statischen Gleichgewichts)*** *Der Körper $\overline{\Omega_1}$ befinde sich unter den Kräften f im Gleichgewicht. Dann existiert ein Vektorfeld $t : \overline{\Omega_1} \times S^2 \longrightarrow \mathbb{R}^3$ so dass in jeder Teilmenge V von $\overline{\Omega_1}$ gilt:*

- *(Impuls-Erhaltungssatz)*

$$\int_V f(x)\,dx + \int_{\partial V} t(x,n)\,dx = 0 \tag{1.6}$$

- *(Drehimpuls-Erhaltungssatz)*

$$\int_V x \times f(x)\,dx + \int_{\partial V} x \times t(x,n)\,dx = 0 \tag{1.7}$$

Hierbei bezeichnet $\times$ das Vektorprodukt im $\mathbb{R}^3$.

Der folgende Satz von Cauchy ist eine wichtige Konsequenz des Axioms des statischen Gleichgewichts. Im Wesentlichen sagt dieser aus, dass der Cauchysche Spannungsvektor linear im zweiten Argument ist und daher für alle Raumpunkte x ein Tensor $T(x) \in \mathbb{S}^3$, wobei $\mathbb{S}^3$ die Menge der symmetrischen $3\times 3-$Matrizen bezeichnet, existiert, mit dem man den Spannungsvektor darstellen kann. Des Weiteren steht dieser Tensor mit der Volumenkraftdichte f über eine partielle Differentialgleichung in Bezug.

Satz 1.1 ***([1, §1, S. 13] Satz von Cauchy)*** *Seien $t(\cdot,n) \in C^1\left(\overline{\Omega_1}\right)$, $t(x,\cdot) \in C(S^2)$ und $f \in C\left(\overline{\Omega_1}\right)$ im Gleichgewicht. Dann gibt es ein symmetrisches Tensorfeld $T \in C^1\left(\overline{\Omega_1}\right)$ mit folgenden Eigenschaften:*

$$\begin{aligned} t(x,n) &= T(x)\,n, && (1.8)\\ \mathrm{div}T(x) + f(x) &= 0, && (1.9)\\ T(x) &= T^T(x). && (1.10) \end{aligned}$$

Der Tensor T wird als Cauchyscher Spannungstensor bezeichnet.

Bemerkung 1.2 *Die Komponenten $T_{ij}, i,j = 1,2,3$ des Cauchyschen Spannungstensors im Punkt x können wie folgt interpretiert werden:*

- *der erste Index bezeichnet die Kraftrichtung,*
- *der zweite die (Schnitt-) Fläche.*

Die Diagonlaelemente $T_{ii}, i = 1,2,3$ werden Normalspannung und die Nichtdiagonalelemente $T_{ij}, i \neq j, i,j = 1,2,3$ Schubspannung genannt. Die Normalspannungen liegen in Richtung des Normalenvektors senkrecht zur Schnittfläche.

1.2.1 Piola-Transformation

Die Gleichgewichtsbedingungen haben wir in den Koordinaten des deformierten Körpers formuliert. Da diese Koordinaten erst berechnet werden sollen, ist es zweckmäßig, die Größen in den Referenzzustand zu transformieren.

Definition 1.1 ***(Zweite Piola-Kirchhoff-Transformierte)*** *Sei* $T : \overline{\Omega_1} \longrightarrow \mathbb{M}^3$ (:= *Menge der* 3×3 *Matrizen*) *ein Tensorfeld. Dann heißt das Tensorfeld* $\Sigma : \overline{\Omega_0} \longrightarrow \mathbb{M}^3_+$ (:= *Menge der Matrizen in* $\mathbb{M}^3$ *mit positiver Determinante*) *mit*

$$\Sigma(x) := det(\nabla\phi(x))\,\nabla\phi(x)^{-1}\,T(\xi)\,\nabla\phi(x)^{-T}$$

zweite Piola-Kirchhoff-Transformierte von $T(\xi)$.

Die Unterschiede der bisher besprochenen Spannungstensoren sind bei kleinen Deformationsgradienten zu vernachlässigen.

1.3 Materialgesetze

In der bisherigen Betrachtung wurde vernachlässigt, dass verschiedene Materialien unterschiedlich auf die gleiche (externe) Belastung reagieren. Die Deformation zu gegebenen Kräften hängt von den Materialeigenschaften ab. Die Spannung ist abhängig von der Verzerrung. Die physikalische Beziehung, die die beiden verbindet, ist ein Stoff- bzw. Materialgesetz, abhängig vom Werkstoff aus dem der Körper besteht.
Diese zusätzlichen Annahmen lösen auch das Problem der Unbestimmtheit der partiellen Differentialgleichungen. Die 3 skalaren Gleichgewichtsbedingungen in der Referenzkonfiguration haben neun Unbekannte, die sechs unabhängigen Komponenten des zweiten Piola-Kirchhoffschen Spannungstensors und die drei Komponenten der Deformation. Die fehlenden drei bzw. sechs Gleichungen erhält man, indem die Materialbeschaffenheit in die Betrachtung einfließt. Außerdem wird somit das zu optimierende Materialverhalten in das Modell mit aufgenommen.
Umgangssprachlich wird ein Material elastisch genannt, wenn es sich bei Belastung verformt und nach der Entlastung in den Ausgangszustand, den Referenzzustand, zurückkehrt. Man spricht in diesem Zusammenhang auch vom "Gedächtnis"des Materials, welches sich nur an einen Grundzustand erinnert. Innerhalb gewisser Grenzen ist die Verformung reversibel. Wird das Material über diese Grenzen hinaus belastet, so bleibt bei völliger Entlastung eine plastische Dehnung erhalten, in diesem Fall wird das Stoffverhalten plastisch genannt.
Hier heißt ein Material elastisch, wenn es eine Materialfunktion gibt, die die Deformation und den Spannungszustand in Bezug zueinander setzt. Die Materialfunktion beschreibt die Spannung des Materials als Antwort auf die Verschiebung/Verzerrung, daher auch die Bezeichnung als Antwortfunktion (engl.: response function). Die Materialfunktion wird wie die Gleichgewichtsbedingungen zunächst in Euler-Koordinaten angegeben.

Definition 1.2 *Ein Material heißt elastisch, wenn es eine Abbildung*

$$\widehat{T} : \mathbb{M}^3_+ \longrightarrow \mathbb{S}^3_+$$

gibt, so dass für jeden deformierten Zustand gilt

$$T(\xi) = \widehat{T}(\nabla\phi(x)), \text{ für alle } \xi \in \overline{\Omega_1}. \tag{1.11}$$

Die Abbildung $\widehat{T}$ heißt Antwortfunktion, und (1.11) bezeichnet man als konstitutives Gesetz.

Hinter dem konstitutiven Gesetz verbirgt sich insbesondere die Annahme, dass die Spannungen in lokaler Weise von den Verschiebungen abhängen, und auch nur von den ersten Ableitungen.
Die Antwortfunktion hängt in obiger Formulierung nicht explizit vom Ort x ab. Man unterscheidet je nachdem, ob die Materialfunktion vom Ort abhängt, homogenes und inhomogenes Material. Außerdem unterscheidet man, je nachdem, ob die Materialeigenschaften invariant gegenüber orthogonalen Drehungen sind oder nicht, isotropes und anisotropes Material.

Definition 1.3 *Ein Material heißt isotrop, wenn für alle $F \in \mathbb{M}^3_+$ gilt:*

$$\widehat{T}(F) = \widehat{T}(FQ), Q \in \mathbb{O}^3_+. \tag{1.12}$$

Dabei bezeichnet $\mathbb{O}^3_+$ die Menge der orthogonalen 3×3 Matrizen mit positiver Determinante.

Die Materialeigenschaft isotrop bedeutet, dass sich das Material in alle Richtungen gleich verhält, die physikalischen Eigenschaften des Materials also richtungsunabhängig sind. Der Spannungstensor ändert sich nicht, wenn man den Körper vor der Deformation dreht.
Obige Materialfunktionen sind für den Cauchyschen Spannungstensor und in Abhängigkeit vom Deformationsgradienten formuliert. Natürlich ist es auch möglich, Materialfunktionen für den zweiten Piola-Kirchhoffschen Spannungstensor zu formulieren. Des Weiteren ist es über die Zusammenhänge der Kinematik auch möglich, die Antwortfunktionen als Abbildungen des Cauchyschen Verzerrungstensors C oder des Verzerrungstensors E zu formulieren.
Hier wird nun ein Materialgesetz in Abhängigkeit von der Verzerrung formuliert, dieses beschreibt sogenanntes St.Venant-Kirchhoff-Material. Dies ist ein homogenes, isotropes Material.

Definition 1.4 ***(St. Venant-Kirchhoff Material)*** *Ein elastisches Material heißt St. Venant-Kirchhoff Material, falls gilt*

$$\tilde{\Sigma}(E) = \lambda spur(E) I + 2\mu E + o(||E||) \tag{1.13}$$

Hierbei werden die beiden Materialkonstanten λ und μ nach dem französischen Ingenieur Gabriel Lamè (1795-1870) Lamè-Konstanten genannt. Aus physikalischen Gründen gilt $\lambda > 0$ und $\mu > 0$.

Da der Verzerrungstensor eine nichtlineare Funktion der Verschiebung ist, hängt die Spannung selbst für eine lineare Materialfunktion nichtlinear von der Verschiebung u ab (die Abhängigkeit von x wird vernachlässigt):

$$\begin{aligned}\Sigma : u \to \tilde{\Sigma}\left(E\left(u\right)\right) &= \lambda spur\left(E\left(u\right)\right) I + 2\mu E\left(u\right) \\ &= \frac{\lambda}{2} spur\left(\nabla u + \nabla u^T + \nabla u^T \nabla u\right) + \mu\left(\nabla u + \nabla u^T + \nabla u^T \nabla u\right) \\ &= \lambda spur \nabla u + \mu\left(\nabla u + \nabla u^T\right) + \frac{\lambda}{2}\left(\nabla u^T \nabla u\right) + \mu \nabla u^T \nabla u\end{aligned}$$

Alternativ findet man häufig statt der Lamè-Konstanten den Elastizitätsmodul $\mathtt{E}$ und die Poissonzahl ν als Parameter. Zwischen diesen vier Konstanten besteht der folgende Zusammenhang:

$$\begin{array}{ll}\nu = \frac{\lambda}{2(\lambda+\mu)}, & \mathtt{E} = \frac{\mu(3\lambda+2\mu)}{\lambda+\mu}, \\ \lambda = \frac{\mathtt{E}\nu}{(1+\nu)(1-2\nu)}, & \mu = \frac{\mathtt{E}}{2(1+\nu)},\end{array}$$

wobei aus physikalischen Gegebenheiten $\mathtt{E} > 0$ und $0 < \nu < \frac{1}{2}$ ist.

Durch die Kinematik, die Gleichgewichtsbedingungen und die Stoffgesetze sind die Deformationen, Verzerrungen und Spannungen bestimmt. Wenn man von kleinen Deformationen ausgeht und die Verzerrung E durch die Linearisierung ϵ ersetzt, kommt man zur sogenannten geometrisch linearen Theorie.

1.4 Lineare Elastizitätstheorie

In der linearen Elastizitätstheorie werden in den Gleichungen nur Terme erster Ordnung in den Verschiebungen u berücksichtigt und Terme höherer Ordnung vernachlässigt. Das betrifft die Kinematik mit der Näherung (1.5) und das Materialgesetz mit (1.13). Es wird auch zwischen den verschiedenen Spannungstensoren nicht unterschieden. Man schreibt σ anstatt Σ und ϵ anstatt E.

1.4.1 Klassische gemischte Randwerttaufgabe

Hier werden die wesentlichen Gleichungen, die den linear-elastischen Körper unter den gemachten Annahmen beschreiben, zusammengefasst. Man erhält die klassische Darstellung eines elastischen Körpers $\Omega \subset \mathbb{R}^3$ unter dem Wirken einer externen Kraft t auf Γ_t im Gleichgewicht, der an einem Teil des Randes Γ_u eingespannt ist. Weiters bezeichne f die Volumenlast. Damit ergibt sich folgendes Modellproblem:

$$\begin{aligned}-\mathrm{div}\sigma\left(x\right) &= f\left(x\right), \quad x \in \Omega \text{ (Kräftegleichgewicht)} \\ \sigma\left(x\right) n &= t\left(x\right), \quad x \in \Gamma_t \text{ (Neumann-Randbedingung)} \\ u\left(x\right) &= 0, \quad x \in \Gamma_u \text{ (Dirichlet-Randbedingung)} \\ \epsilon\left(u\left(x\right)\right) &= \frac{1}{2}\left(\nabla u\left(x\right)^T + \nabla u\left(x\right)\right) \text{ (linearisierte Verzerrung)} \\ \sigma\left(x\right) &= \mathcal{C}\left(x\right)\epsilon\left(u\left(x\right)\right) \text{ (Hooksches Gesetz)}\end{aligned}$$

Der Elastizitätstensor $\mathcal{C} = \mathcal{C}(x)$ beschreibt hierbei das Verhalten des Materials im Punkt x, genauer die Steifigkeit: Je größer $\mathcal{C}$, desto steifer ist das Material im Punkt x, denn je größer $\mathcal{C}$ ist, um so kleiner ist die Verzerrung und damit die Deformation zu gegebener Spannung.

Im homogenen isotropen Fall sind die Elastizitätskoeffizienten unabhängig von x und können durch die Lamè-Koeffizienten beschrieben werden. Insbesondere ist der Elastizitätstensor von homogenem isotropen linear-elastischen Material elliptisch. Man betrachte das Materialgesetz für homogenes isotropes Material in Abhängigkeit von der linearisierten Verzerrung (λ, μ stückweise konstant):

$$\sigma = \lambda spur(\epsilon) I + 2\mu\epsilon \tag{1.14}$$

Alternativ können die Konstanten durch das Elastizitätsmodul und die Poissonzahl ausgedrückt werden:

$$\sigma = \frac{\mathtt{E}}{1+\nu}\left(\epsilon + \frac{\nu}{1-2\nu} spur(\epsilon) I\right) \tag{1.15}$$

Wegen $spur(I) = 3$ folgt aus (1.14) sofort $spur(\sigma) = \frac{\mathtt{E}}{1-2\nu} spur(\epsilon)$, und die Auflösung nach ϵ liefert

$$\epsilon = \frac{1+\nu}{\mathtt{E}}\sigma - \frac{\nu}{\mathtt{E}} spur(\sigma) I \tag{1.16}$$

Da die betrachteten Tensoren in $\mathbb{R}^3$ symmetrisch sind, und damit durch 6 Komponenten beschrieben werden, wählt man häufig eine komponentenweise Darstellung. Die Gleichung (1.15) komponentenweise geschrieben, ergibt

$$\begin{pmatrix}\sigma_{11}\\ \sigma_{22}\\ \sigma_{33}\\ \sigma_{12}\\ \sigma_{13}\\ \sigma_{23}\end{pmatrix} = \frac{\mathtt{E}}{(1+\nu)(1-2\nu)} \begin{pmatrix} 1-\nu & \nu & \nu & & & \\ \nu & 1-\nu & \nu & & & \\ \nu & \nu & 1-\nu & & & \\ & & & 1+\nu & & \\ & & & & 1+\nu & \\ & & & & & 1+\nu \end{pmatrix} \begin{pmatrix}\epsilon_{11}\\ \epsilon_{22}\\ \epsilon_{33}\\ \epsilon_{12}\\ \epsilon_{13}\\ \epsilon_{23}\end{pmatrix}$$

oder kurz

$$\sigma = \mathcal{C}\epsilon. \tag{1.17}$$

Dass die Matrix $\mathcal{C}$ für $0 \leq \nu < \frac{1}{2}$ positiv definit ist, erkennt man an der Inversen (ergibt sich aus (1.16)), welche dann ebenfalls positive definit sein muss, mit Hilfe des Satzes von Gerschgorin:

$$\mathcal{C}^{-1} = \frac{1}{\mathtt{E}} \begin{pmatrix} 1 & -\nu & -\nu & & & \\ -\nu & 1 & -\nu & & & \\ -\nu & -\nu & 1 & & & \\ & & & 1+\nu & & \\ & & & & 1+\nu & \\ & & & & & 1+\nu \end{pmatrix}$$

denn es liegen sämtliche Eigenwerte von $\mathcal{C}^{-1}$ in der Vereinigung der Kreise

$$R_i = \left\{ z \in \mathbb{C} : |z - c_{ii}| \leq \sum_{j=1, j\neq i}^{n} |c_{ij}| \right\}, \forall i = 1, \ldots, n \quad (c_{ij} \text{ Einträge von } \mathcal{C}^{-1})$$

und mithilfe von

$$|z - 1| \leq 2\nu \in [0, 1[\,, \text{ für } i = 1, 2, 3 \Rightarrow z \in]0, 2[$$

und

$$|z - (1 + \nu)| \leq 0 \Leftrightarrow z = 1 + \nu \in [1, 1.5[$$

müssen alle Eigenwerte von $\mathcal{C}^{-1}$ positive sein.

1.4.2 Variationsformulierung

Allgemein erhält man die schwache Formulierung, wenn man klassische Lösungen der PDEs betrachtet und die Gleichgewichtsbedingung mit Testfunktionen multipliziert und anschließend integriert. Die Grundidee dahinter besteht darin, das partielle Differentialgleichungssystem in einen Integralausdruck umzuformen, diesen als L^2-Skalarprodukt aufzufassen und das Problem mit Hilbertraum-Methoden zu lösen. Formal bedeutet dies, dass man die schwache Formulierung des elastischen Körpers im Gleichgewicht durch Multiplikation der Volumen-Gleichgewichtsbedingung mit Testfunktionen aus geeigneten Räumen und Dichteargumenten erhält. Was ein geeigneter Raum von Testfunktionen ist, hängt im Wesentlichen von den Randbedingungen ab. Unter der Annahme einer homogenen Dirichlet-Bedingung auf Γ_u bietet sich der folgender Raum für Testfunktionen, welche auf dem Dirichlet-Rand verschwinden, an, d. h.

$$\mathcal{H} := \left\{ u \in \mathbf{H}^1(\Omega) \,|\, u = 0 \text{ auf } \Gamma_u \right\}.$$

Hierbei bezeichne $\mathbf{H}^1(\Omega)$ den 3-dimensionalen Produktraum des Sobolevraums $H^1(\Omega)$. Dies ist der Raum der quadratisch integrierbaren Funktionen, deren erste schwache Ableitungen existieren und ebenfalls quadratisch integrierbar sind; siehe [1, §2.1.1].

Für die Herleitung der Variationsformulierung wird die gemischte Randwertaufgabe aus den vorherigen Kapitel in Komponentenschreibweise angegeben. Gesucht sind die Verschiebungskomponenten $u_i, i = 1, 2, 3$ mit

$$\begin{aligned}
-\sum_{j=1}^{3} \frac{\partial \sigma_{ij}(x)}{\partial x_j} &= f_i(x), \quad x \in \Omega \\
\sum_{j=1}^{3} \sigma_{ij}(x)\, n_j(x) &= t_i(x), \quad x \in \Gamma_t \\
u_i(x) &= 0, \quad x \in \Gamma_u \\
\epsilon_{ij}(u(x)) &= \frac{1}{2}\left(\frac{\partial u_i(x)}{\partial x_j} + \frac{\partial u_j(x)}{\partial x_i} \right) \\
\sigma_{ij}(x) &= \sum_{k,l=1}^{3} \mathcal{C}_{ijkl}(x)\, \epsilon_{kl}(u(x))
\end{aligned}$$

Multiplizieren der Gleichgewichtsbedingungen im Inneren mit einer Testfunktion $\varphi \in \mathcal{H}$ und anschließende Integration über Ω liefert (bei Aufsummieren über die Gleichgewichtsbedingungen):

$$\int_\Omega \sum_{i=1}^{3} \left(-\sum_{j=1}^{3} \frac{\partial \sigma_{ij}(x)}{\partial x_j} \right) \varphi_i(x)\, dx = \int_\Omega \sum_{i=1}^{3} f_i(x)\, \varphi_i(x)\,. \tag{1.18}$$

Mit partieller Integration erhält man für die linke Seite

$$-\int_\Omega \sum_{i,j=1}^{3} \frac{\partial \sigma_{ij}(x)}{\partial x_j} \varphi_i(x)\, dx = \int_\Omega \sum_{i,j=1}^{3} \sigma_{ij}(x)\, \frac{\partial \varphi_i(x)}{\partial x_j} dx - \int_\Gamma \sum_{i,j=1}^{3} \sigma_{ij}(x)\, n_j(x)\, \varphi_i(x)\, dx.$$

Durch Einsetzen der Randbedingungen ergibt dies

$$\int_\Omega \sum_{i,j=1}^{3} \sigma_{ij}(x)\, \frac{\partial \varphi_i(x)}{\partial x_j} dx = \int_\Omega \sum_{i=1}^{3} f_i(x)\, \varphi_i(x) + \int_{\Gamma_t} \sum_{i=1}^{3} t_i(x)\, \varphi_i(x)\, dx.$$

Mit dem Hookschen Gesetz (1.17) und der Symmetrie des Elastizitätstensors $\mathcal{C}$ folgt nunmehr für (1.18)

$$\int_\Omega \sum_{i,j,k,l=1}^{3} \mathcal{C}_{ijkl}(x)\, \frac{\partial u_k(x)}{\partial x_l} \frac{\partial \varphi_i(x)}{\partial x_j} dx = \int_\Omega \sum_{i=1}^{3} f_i(x)\, \varphi_i(x) + \int_{\Gamma_t} \sum_{i=1}^{3} t_i(x)\, \varphi_i(x)\, dx.$$

Man kann zeigen, dass die linke Seite der vorstehenden Gleichung eine bilineare beschränkte Abbildung

$$(u, v) \in \mathcal{H} \times \mathcal{H} \mapsto a(u, v) := \int_\Omega \sum_{i,j,k,l=1}^{3} \mathcal{C}_{ijkl}(x)\, \frac{\partial u_k(x)}{\partial x_l} \frac{\partial v_i(x)}{\partial x_j} dx$$

ist, wenn man annimmt, die Komponenten des Elastizitätstensors seien messbar und fast überall beschränkt. Andererseits definiert man

$$\langle F, v \rangle := \int_\Omega \sum_{i=1}^{3} f_i(x)\, v_i(x) + \int_{\Gamma_t} \sum_{i=1}^{3} t_i(x)\, v_i(x)\, dx.$$

Damit ergibt sich die schwache Variationsformulierung zu

$$a(u, v) = \langle F, v \rangle\,, \quad \forall u, v \in \mathcal{H}. \tag{1.19}$$

mit $f = (f_1, f_2, f_3)^T \in [L^2(\Omega)]^3, t = (t_1, t_2, t_3)^T \in [L^2(\Omega)]^3$ gegeben und $\mathcal{C}_{ijkl} \in L^\infty(\Omega)$ für $i, j, k, l = 1, 2, 3$.
Die Gleichung (1.19) wird im physikalischen Kontext auch Satz der virtuellen Arbeit genannt. Der Satz der virtuellen Arbeit basiert auf dem Prinzip der virtuellen Verschiebungen. Virtuelle Verschiebungen sind infinitesimale Verschiebungsfelder, die mit eventuellen Zwangsbedingungen verträglich sind. Die Arbeit, welche die Kräfte entlang der virtuellen (gedachten) Verschiebungen/Verzerrungen leisten, wird virtuelle Arbeit genannt. Das Prinzip der virtuellen Arbeit besagt nun, dass im Gleichgewicht die virtuelle Arbeit der inneren Kräfte gleich der virtuellen Arbeit der äußeren Kräfte ist.

Bemerkung 1.3 *Mithilfe der Kornschen Ungleichungen kann gezeigt werden, dass die Variationsformulierung elliptisch ist, und damit die Variationsaufgabe 1.19 der lineare Elastizitätstheorie genau eine Lösung hat. Für die Details sei auf [2] verwiesen.*

1.5 Nichtlineare Elastizitätstheorie

Im vorherigen Kapitel wurde die lineare Elastizitätstheorie behandelt. Hierfür wurde eine lineare Näherung der kinematischen Beschreibung 1.4 durch 1.5 durchgeführt und das lineare Materialgesetz für homogene isotrope Materialien 1.13 verwendet. Für nichtlineare Phänomene der Mechanik müssen demnach zwei Kategorien unterschieden werden:

- geometrisch nichtlinear: große Verformungen/Dehnungen, wodurch eine nichtlineare kinematische Beschreibung unumgänglich wird.
- physikalisch nichtlinear: Materialien, deren Stoffgesetz eine nichtlineare Funktion in Abhängigkeit von der Dehnung des Körpers ist, wie zum Beispiel bei Gummi oder Kunststoffen.
  Man unterscheidet außerdem kompressible und inkompressible Materialien. Körper aus inkompressiblen können zwar verformt werden, im Gegensatz zu solchen aus kompressiblen Materialien bleibt aber ihr Volumen stets erhalten; beispielsweise werden Herzmuskelfasern als inkompressibel angenommen.

Eine Einführung in die nichtlineare Elastizitätstheorie bietet beispielsweise [6].

Kapitel 2

Gleichungssystemlöser

Aus der Variationsformulierung (1.19) erhält man mittels Diskretisierung durch die Finite Elemente Methode (siehe beispielsweise [2, §II]) ein lineares Gleichungssystem

$$A_h u_h = f_h \qquad u_h \in \mathbb{R}^{N_h}, \tag{2.1}$$

wobei A_h eine symmetrische, positive definite, dünnbesetzte Matrix ist. Für die Lösung dieses Problems wird das Algebraische Mehrgitterverfahren als Vorkonditionierer in einem konjugierten Gradientenverfahren verwendet. Dieser Algorithmus wird in den folgenden Kapiteln näher erläutert.

2.1 Klassische Iterationsverfahren

Klassische Iterationsverfahren konvergieren sehr langsam und werden heute kaum noch als eigenständige Iterationsverfahren herangezogen. Trotzdem haben einige kaum an Bedeutung verloren, sie werden bei den modernen Iterationsverfahren als Hilfsprozeduren eingesetzt, und zwar beispielsweise bei den später beschriebenen Mehrgitterverfahren als Glätter. Die folgende Einführung wurde aus [2] und das Modellproblem aus [13] entnommen.

Viele Iterationsverfahren zur Lösung eines Gleichungssystems $Ax = b$ nehmen ihren Ausgangspunkt von einer Aufspaltung der Matrix

$$A = M - N.$$

Dabei ist M eine einfach invertierbare Matrix. Das gegebene System wird in die Form

$$Mx = Nx + b$$

gebracht. Dies führt zu der Iteration

$$Mx^{k+1} = Nx^k + b$$

bzw. $x^{k+1} = M^{-1}\left(Nx^k + b\right)$ oder

$$x^{k+1} = x^k + M^{-1}\left(b - Ax\right), \quad k = 0, 1, 2, \ldots.$$

Offensichtlich ist dies eine Iteration der Form

$$x^{k+1} = Gx^k + d \tag{2.2}$$

mit $G = M^{-1}N, d = M^{-1}b$.
Die Lösung x^* der Gleichung $Ax = b$ ist Fixpunkt des Prozesses (2.2), d.h. es gilt

$$x^* = Gx^* + d.$$

Die Subtraktion der letzten beiden Gleichungen liefert

$$x^{k+1} - x^* = G\left(x^k - x^*\right),$$

und durch Induktion folgt

$$x^k - x^* = G^k\left(x^0 - x^*\right). \tag{2.3}$$

Die Iteration (2.2) heißt konvergent, wenn für jedes beliebige x^0 die Konvergenz $\lim_{k\to\infty} x^k = x^*$ gewährleistet ist. Nach (2.3) ist das äquivalent zu

$$\lim_{k\to\infty} G^k = 0.$$

Satz 2.1 *([2, §IV.1, S. 173]) Sei G eine $n \times n$-Matrix. Dann sind folgende Aussagen äquivalent:*

i) *Die Iteration (2.2) konvergiert für jedes $x^0 \in \mathbb{R}^n$.*

ii) *Es ist $\lim_{k\to\infty} G^k = 0$.*

iii) *Es ist $\rho(G) < 1$, wobei $\rho(G)$ den Spektralradius der Matrix G bezeichnet.*

Einen ersten Ansatz für ein Iterationsverfahren erhält man aus der Aufspaltung

$$A = D - L - R.$$

Dabei sei D eine Diagonalmatrix, L eine subdiagonale und R eine superdiagonale Matrix

$$D_{ik} = \begin{cases} a_{ik} & \text{falls } i = k \\ 0 & \text{sonst,} \end{cases} \qquad L_{ik} = \begin{cases} -a_{ik} & \text{falls } i > k \\ 0 & \text{sonst,} \end{cases} \qquad R_{ik} = \begin{cases} -a_{ik} & \text{falls } i < k \\ 0 & \text{sonst.} \end{cases}$$

Als Gesamtschritt- oder Jacobi-Verfahren bezeichnet man die Iteration

$$Dx^{k+1} = (L + R)\, x^k + b.$$

Die zugehörige Iterationsmatrix ist $G = G_G = D^{-1}(L + R)$. Eine Alternative zum vorgestellten Verfahren ist das Einzelschrittverfahren oder Gauß-Seidel-Verfahren, dargestellt in [2].

Eine erste Konvergenzbeschleunigung brachte die sogenannte Overrelaxation. Das Jacobi-Relaxationsverfahren ist definiert durch

$$\frac{1}{\omega} D x^{k+1} = (L + R)\, x^k + b$$

oder

$$x^{k+1} = x^k + \omega D^{-1} \left(b - A x^k \right). \tag{2.4}$$

Das folgende Modellbeispiel 2.1 dient zur Konvergenzanalyse des Gesamtschrittverfahrens (2.4).

Beispiel 2.1 *Modellbeispiel: gesucht ist die Funktion $u(x)$ welche folgende Differentialgleichung löst:*

$$\begin{aligned} u''(x) &= f(x), \quad x \in (0,1) \\ u(0) &= u(1) = 0. \end{aligned}$$

Die Finite Elemente Diskretisierung der Variationsformulierung der betrachteten Differentialgleichung (siehe [13]) liefert

$$\frac{1}{h} \begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{pmatrix} \begin{pmatrix} u_1 \\ \vdots \\ u_{n-1} \end{pmatrix} = \begin{pmatrix} f_1 \\ \vdots \\ f_{n-1} \end{pmatrix},$$

oder

$$A_h u_h = f_h,$$

wobei $h = \frac{1}{n}$ und $N_h = n - 1$ gilt. Weiters sei $n \geq 4$ und n gerade vorausgesetzt.

Für die Untersuchung der Konvergenzeigenschaften der klassischen Iterationsverfahren wird die Fourieranalyse nach den Eigenfunktionen/Eigenvektoren von der Systemmatrix herangezogen. Die Eigenvektoren $\varphi_{h,k}$ des Eigenwertproblems $A_h \varphi_{h,k} = \lambda_{h,k} \varphi_{h,k}$ sind gegeben durch

$$\varphi_{h,k} = \sqrt{2} \begin{pmatrix} \sin k\pi \frac{1}{n} \\ \sin 2k\pi \frac{1}{n} \\ \vdots \\ \sin N_h k\pi \frac{1}{n} \end{pmatrix}, \qquad \text{für } k = 1, \ldots, N_h \tag{2.5}$$

wie folgende Rechnung zeigt. Die durch (2.5) definierten Eigenvektoren $\{\varphi_{h,1}, \ldots, \varphi_{h,N_h}\}$

bilden eine orthonormale Basis des $\mathbb{R}^{N_h}$.

$$\begin{aligned}
A_h\varphi_{h,k} &= \frac{\sqrt{2}}{h}\begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{pmatrix}\begin{pmatrix} \sin k\pi\frac{1}{n} \\ \sin 2k\pi\frac{1}{n} \\ \vdots \\ \sin N_h k\pi\frac{1}{n} \end{pmatrix} \\
&= \frac{\sqrt{2}}{h}\begin{pmatrix} 0 & + & 2\sin k\pi\frac{1}{n} & - & \sin 2k\pi\frac{1}{n} \\ -\sin k\pi\frac{1}{n} & + & 2\sin 2k\pi\frac{1}{n} & - & \sin 3k\pi\frac{1}{n} \\ & & \vdots & & \\ -\sin\left(N_h-2\right)k\pi\frac{1}{n} & + & 2\sin\left(N_h-1\right)k\pi\frac{1}{n} & - & \sin N_h k\pi\frac{1}{n} \\ -\sin\left(N_h-1\right)k\pi\frac{1}{n} & + & 2\sin N_h k\pi\frac{1}{n} & - & 0 \end{pmatrix} \\
&\overset{(*)}{=} \frac{\sqrt{2}}{h}\begin{pmatrix} 4\sin^2\frac{k\pi}{2n}\sin k\pi\frac{1}{n} \\ 4\sin^2\frac{k\pi}{2n}\sin 2k\pi\frac{1}{n} \\ \vdots \\ 4\sin^2\frac{k\pi}{2n}\sin\left(N_h-1\right)k\pi\frac{1}{n} \\ 4\sin^2\frac{k\pi}{2n}\sin N_h k\pi\frac{1}{n} \end{pmatrix} = \underbrace{\frac{4}{h}\sin^2\left(\frac{k\pi}{2n}\right)}_{=:\lambda_{h,k}}\varphi_{h,k}, \quad \forall k.
\end{aligned}$$

Dabei wurde im Schritt (*) verwendet, dass der Sinus eine ungerade Funktion ist, und dass aus den Additionstheoremen für Trigonometrische Funktionen gilt, dass $\forall a,b,c \in \mathbb{R} : 4\sin a\sin b\sin c = \sin\left(a+b-c\right)+\sin\left(b+c-a\right)+\sin\left(c+a-b\right)-\sin\left(a+b+c\right)$ *ist.*
Für die weitere Konvergenzanalyse benötigt man noch eine Abschätzung für die Eigenwerte:

$$8h \le \lambda_{h,1} < \lambda_{h,2} < \ldots < \lambda_{h,n-1} < \frac{4}{h} \quad \Rightarrow \quad \kappa\left(A_h\right) \overset{spd}{=} \frac{\lambda_{max}}{\lambda_{min}} = \frac{\lambda_{h,n-1}}{\lambda_{h,1}} \le \frac{1}{2}h^{-2}$$

Für die Lösung des linearen Gleichungssystems wird das Jacobi-Relaxationsverfahren (2.4) herangezogen:

$$u_h^{l+1} = \left(I - \omega\frac{h}{2}A_h\right)u_h^l + \omega\frac{h}{2}f, \quad l = 0,1,\ldots, \quad u_h^0 \in \mathbb{R}^{N_h}\, geg.$$

Nun betrachtet man den Fehler $z_h^l := u_h - u_h^l$, *und zerlegt den Anfangsfehler nach den Eigenvektoren (2.5):*

$$z_h^0 = u_h - u_h^0 = \sum_{k=1}^{n-1}\alpha_k\varphi_{h,k}, \quad \left\|z_h^0\right\|_h^2 = \left(z_h^0, z_h^0\right)_h = \sum_{k=1}^{n-1}\alpha_k^2.$$

Damit erhält man das folgende Fehlerschema

$$z_h^{l+1} = \underbrace{\left(I - \omega\frac{h}{2}A_h\right)}_{=:S(\omega)}z_h^l = \left(I - \omega\frac{h}{2}A_h\right)^{l+1}z_h^0 = \sum_{k=1}^{n-1}\alpha_k\left(1 - \omega\frac{h}{2}\lambda_{h,k}\right)^{l+1}\varphi_{h,k}.$$

Folglich gilt

$$\left\|z_h^{l+1}\right\|_h^2 = \sum_{k=1}^{n-1} \alpha_{h,k}^2 \left(1 - \omega\frac{h}{2}\lambda_{h,k}\right)^{2(l+1)} \leq \left[\max_k \left|1 - \omega\frac{h}{2}\lambda_{h,k}\right|\right]^{2(l+1)} \left\|z_h^0\right\|_h^2.$$

Damit liegt es nahe den Konvergenzfaktor als

$$\rho\left(h,\omega\right) = \rho\left(S\left(\omega\right)\right) = \max_{k=1,\ldots,n-1} \left|1 - \omega\frac{h}{2}\lambda_{h,k}\right|$$

zu definieren. Abbildung 2.1 zeigt den Konvergenzfaktor ρ in Abhängigkeit von ω für fixes h. Der optimale Konvergenzfaktor ρ_{opt} zur bestmöglichen Reduktion des Fehlers für alle Frequenzen wird für $\omega = 1$ erreicht, was folgende Rechnung zeigt:

$$\begin{aligned} 1 - \omega\frac{h}{2}\lambda_{h,1} &= -\left(1 - \omega\frac{h}{2}\lambda_{h,n-1}\right) \\ 1 - 2\omega\sin^2\frac{\pi}{2n} &= 2\omega\sin^2\left(\frac{n-1}{n}\frac{\pi}{2}\right) - 1 \\ 2 &= 2\omega\left(\sin^2\frac{\pi}{2n} + \sin^2\left(\frac{\pi}{2} - \frac{\pi}{2n}\right)\right) = 2\omega\left(\sin^2\frac{\pi}{2n} + \cos^2\frac{\pi}{2n}\right) = 2\omega. \end{aligned}$$

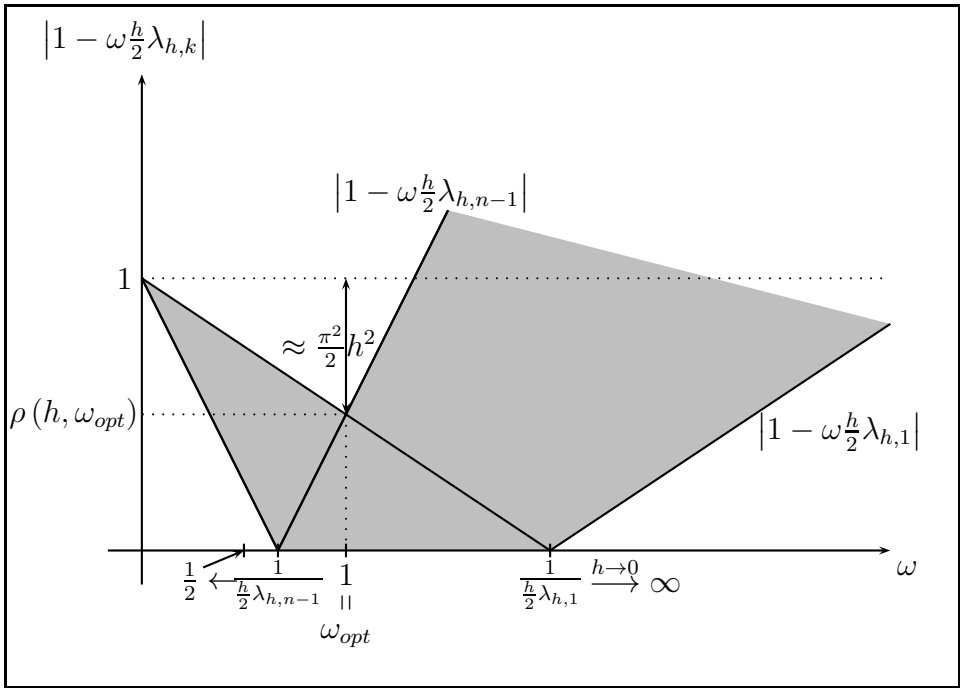


Abbildung 2.1: Fehlerreduktion des Jacobi-Relaxationsverfahren

Mit einer weiteren Hilfsrechnung

$$1 - \omega\frac{h}{2}\lambda_{h,k} = 1 - 2\omega\sin^2\frac{k\pi}{2n} = 1 - \omega\left(1 - \cos\frac{k\pi}{n}\right) = 1 - \omega\left(1 - \cos k\pi h\right)$$

kommt man zu folgendem Resultat:

$$\rho(h,\omega) \geq \rho(h,\omega_{opt}) = 1 - 2\sin^2\frac{\pi}{2n} = \cos\pi h \approx 1 - \frac{\pi^2}{2}h^2$$

$$\rho = \begin{cases} 1 - O(h^2) & ,0 < \omega \leq 1 \\ \geq 1 & \text{, für } \omega > 1 \text{ (fix) bei hinreichend kleinen } h \end{cases}$$

Weiters werden nun die Reduktionsfaktoren $1 - \omega\frac{h}{2}\lambda_{h,k}$ *für verschiedene* ω*-Werte in Abhängigkeit von den Frequenzen* $k = 1, \ldots, n-1$ *untersucht; siehe Abbildung 2.2. Dabei stehen die hochfrequenten Eigenfunktionen (*$k = \frac{n}{2}, \ldots, n-1$*) im Mittelpunkt der Betrachtungen. Dementsprechend definiert man 2 weitere Faktoren als Maß für den Glättungseffekt:*

$$\begin{aligned} \mu(h,\omega) = \mu(S) &:= \max_{k=\frac{n}{2},\ldots,n-1} \left|1 - \omega\frac{h}{2}\lambda_{h,k}\right| \quad \textit{(Glättungsfaktor)} \\ \mu^*(\omega) &:= \sup_{h\leq\frac{1}{4}} \mu(h,\omega) \quad \textit{(asymptotischer Glättungsfaktor)} \end{aligned}$$

Mithilfe des Auswertungen für $k = \frac{n}{2} : 1 - \omega(1 - \cos k\pi h) = 1 - \omega$ *und* $k = n-1 :$ $1 - \omega(1 - \cos k\pi h) \overset{cosx=-cos(\pi-x)}{=} 1 - \omega(1 + \cos\pi h)$ *ergibt sich*

$$\begin{aligned} \mu(h,\omega) &= \max\{|1-\omega|, |1-\omega(1+\cos\pi h)|\}, \\ \mu^*(\omega) &= \max\{|1-\omega|, |1-2\omega|\}, \end{aligned}$$

sodaß man für die optimalen (minimalen) Werte für μ *und* μ^* *erhält:*

$$\begin{aligned} \omega_h^* = \frac{2}{2+\cos\pi h} &: \mu(h,\omega_h^*) = \min_{0\leq\omega\leq1}\mu(h,\omega) = \frac{\cos\pi h}{2+\cos\pi h}, \\ \omega^* = \frac{2}{3} &: \mu^*(\omega^*) = \min_{0\leq\omega\leq1}\mu^*(\omega) = \frac{1}{3}, \end{aligned}$$

wobei sich ω_h^* *aus der Auflösung von* $1 - \omega = -(1 - \omega(1 + \cos\pi h))$ *ergibt.*

l	1	10	100
$[\mu^*(\omega^*)]^l$	$\frac{1}{3}$	$1.7 \cdot 10^{-5}$	$1.9 \cdot 10^{-48}$
$[\rho(h,\omega^*)]^l$	0.9999967	0.999967	0.99967

Tabelle 2.1: Konvergenzfaktor und asymptotischer Glättungsfaktor für steigende Iterationszahlen

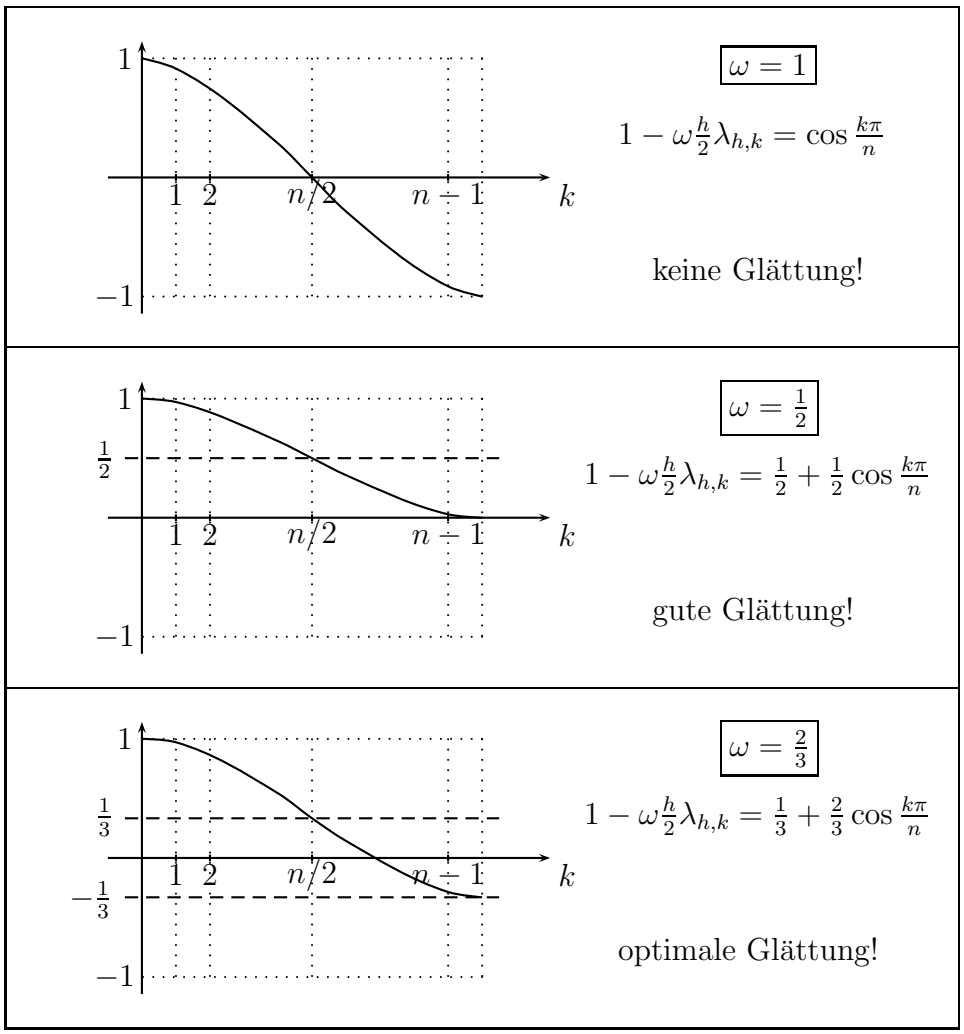


Abbildung 2.2: Glättungseigenschaften des Jacobi-Relaxationsverfahren

Abschließend wird das Konvergenzverhalten für wachsende Anzahl der Iterationen l untersucht. Dafür sei $h = 10^{-3}$ *und* $\omega^* = \frac{2}{3}$ *gewählt. Mit* $\mu^*(\omega^*) = \frac{1}{3}$ *und* $\rho(h,\omega^*) = 1 - \frac{4}{3}\sin^2\frac{\pi}{2n} \approx 1 - \frac{\pi^2}{3}h^2$ *wird der Konvergenzfaktor und der asymptotische Glättungsfaktor für steigende Anzahl der Iterationen in Tabelle 2.1 festgehalten.*

2.2 Mehrgitterverfahren

Die Mehrgitterverfahren zählen zu den schnellsten Gleichungslösern für große, dünnbesetzte lineare Gleichungssysteme, siehe [3]. Als Grundlage für die folgenden Ausführungen dienen [2] und [13].
Mehrgitterverfahren basieren auf der Beobachtung, dass die klassischen Iterationsverfahren glättend wirken, d.h. sie beseitigen sehr schnell die oszillierenden (kurzwelligen) Anteile der Fehlerfunktion; siehe Modellbeispiel 2.1. Die langwelligen Anteile werden dabei nur sehr langsam gedämpft, aber sie können auf gröberen Gittern gut erfaßt werden; siehe Abbildung 2.3.

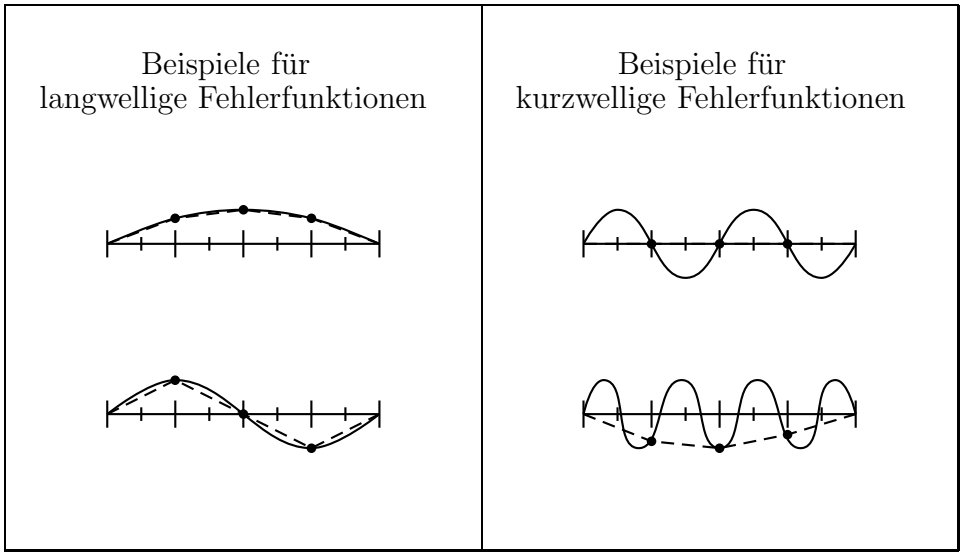


Abbildung 2.3: Approximation "glatter" Gitterfunktionen auf gröberem Gitter

Um diesen Sachverhalt zu verdeutlichen, betrachtet man die Eigenvektoren des Eigenwertproblems aus Modellbeispiel 2.1, welche bis auf einen konstanten Faktor den Fourier-Moden

$$\left(w_k^h\right)_j := \sin\left(\frac{jk\pi}{n}\right), \quad \underbrace{1 \le j \le n-1}_{\text{Komponenten}}, \quad \underbrace{1 \le k \le n-1}_{\text{Moden}} \tag{2.6}$$

entsprechen. Motiviert durch die geometrische Anschauung nennt man die Eigenvektoren mit $1 \le k \le \frac{n}{2}$ glatte Eigenvektoren und die Eigenvektoren mit $\frac{n}{2} < k \le n-1$ oszillierende Eigenvektoren. Für die glatten Eigenvektoren gilt, dass die Moden beim Übergang zum gröberen Gitter erhalten bleiben, denn es gilt

$$\left(w_k^h\right)_{2j} = \sin\left(\frac{2jk\pi}{n}\right) = \sin\left(\frac{jk\pi}{\frac{n}{2}}\right) = \left(w_k^{2h}\right)_j, \quad 1 \le k \le \frac{n}{2}.$$

Andererseits werden die oszillierende Eigenvektoren unsichtbar auf dem Grobgitter

(Aliasing-Effekt), denn

$$\begin{aligned}\left(w_k^h\right)_{2j} &= \sin\left(\frac{2jk\pi}{n}\right) = -\sin\left(\frac{2\pi j\,(n-k)}{n}\right)\\ &= -\sin\left(\frac{\pi j\,(n-k)}{\frac{n}{2}}\right) = -\left(w_{n-k}^{2h}\right)_j, \quad k > \frac{n}{2},\end{aligned}$$

d.h. für $k > \frac{n}{2}$ taucht die k-te Mode auf dem Feingitter als die (n-k)-te Mode am Grobgitter auf. Demnach sind glatte Eigenvektoren auf dem Feingitter für $\frac{n}{4} < k \leq \frac{n}{2}$ oszillierende Eigenvektoren auf dem Grobgitter und damit leicht zu glätten.

Mit diesen Vorüberlegungen kann folgende Mehrgitter-Idee festgehalten werden: Man führt einige Relaxationsschritte aus, um alle oszillierenden Bestandteile des Fehlers zu dämpfen. Dann geht man zu einem gröberen Gitter über und ermittelt dort näherungsweise den verbleibenden glatten Anteil. Das ist möglich, weil glatte Funktionen bereits auf gröberen Gittern gut approximiert werden können. Auf dem gröberen Gittern ist der Fehler (relativ) mehr oszillierend, wodurch weitere Relaxationsschritte effektiver als auf dem feineren Gitter sind. Die Glättungsschritte auf dem feinen Gitter und die Grob-Gitter-Korrekturen werden abwechselnd wiederholt. Man vergröbert das Gitter so lange bis ein kleines und damit einfach lösbares Gleichungssystem vorliegt. Die Operatoren, die die Daten zu einem feineren Gitter in die eines gröberen umwandelt, heißen Restriktions-, die in umgekehrter Richtung fungierenden heißen Prologations-Operatoren. Diese Idee zur Lösung von (2.1) wird nun im Mehrgitter-Algorithmus, Abbildung 2.4, festgehalten.

Im folgenden werden einige Bemerkungen zum Mehrgitter-Algorithmus aus Abbildung 2.4 angegeben:

- Alternativ kann nach Schritt 4 ein weitere Schritt 5 eingeführt werden, bei dem man die Korrektur $u_h^{k,\nu} + w_h$ ebenfalls glättet. In diesem Fall bezeichnet man Schritt 1 auch als Vorglättung und den neuen Schritt als Nachglättung.

- Für den Parameter γ, der den Aufwand bei der Grobgitter-Korrektur steuert, wird $\gamma = 1$ oder $\gamma = 2$ gewählt. Man spricht dann von V-Zyklus bzw. W-Zyklus. Die Bezeichnungen rühren daher, dass die in Abbildung 2.5 gezeigten Schemata den Buchstaben V und W ähneln.

- Das Gleichungssystem auf dem gröbsten Gitter löst man mit einem direkten Lösungsverfahren (z.B. LU-Faktorisierung).

- Ein Hilfsgitter kann durchaus so grob sein, dass man es in der Praxis nie als endgültiges Gitter heranziehen würde. Die Konvergenzrate wird dadurch nicht beeinträchtigt.

Übergang zwischen den Gittern:

Um Lösungen verschiedener Gitter vergleichen zu können, obwohl sie durch Vektoren unterschiedlicher Länge beschrieben werden, teilt man ein h-Gitter auf in

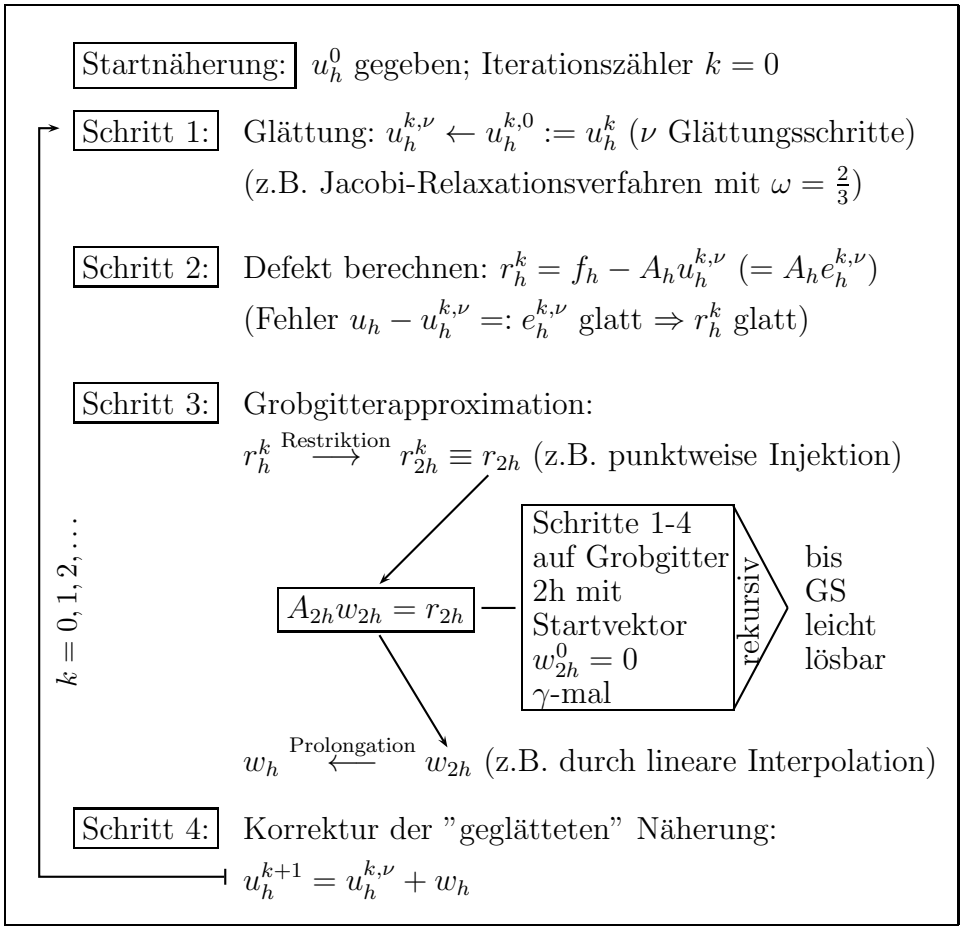


Abbildung 2.4: Idee für Mehrgitter-Algorithmus zur Lösung von $A_h u_h = f_h$

- Grobgitterpunkte, die auch im $2h$-Gitter vorkommen, und
- in die restlichen Feingitterpunkte.

Für die Abbildung vom groben zum feinen Gitter dient der Prolongationsoperator

$$I_{2h}^h : \mathbb{R}^{N_{2h}} \longrightarrow \mathbb{R}^{N_h},$$

der ein u_{2h} auf dem $2h$-Gitter auf ein u_h auf dem h-Gitter abbildet. Dabei gilt, daß u_h an den Grobgitterpunkten die Werte von u_{2h} übernimmt und an den Feingitterpunkten die Werte linear aus den benachbarten Grobgitterpunkten interpoliert. Abbildung 2.6 zeigt eine mögliche Variante einer eindimensionalen Prolongation.

Für den Übergang vom feinen zum groben Gitter definiert man den Restriktionsoperator

$$I_h^{2h} : \mathbb{R}^{N_h} \longrightarrow \mathbb{R}^{N_{2h}}.$$

Es gibt viele Möglichkeiten, wie man diesen Operator wählen kann. Abbildung 2.7 zeigt 2 dieser Möglichkeiten, nämlich durch Injektion und gewichtete Mittlungen. Betrachtet man die Interpolation aus Abbildung 2.6 und die gemittelte Restriktion

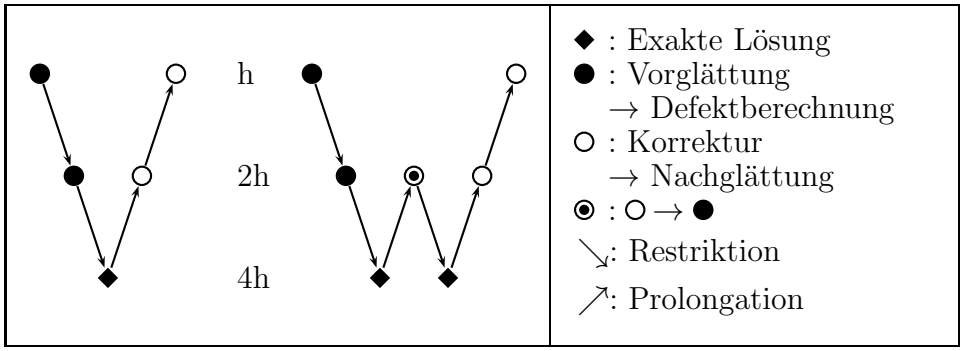


Abbildung 2.5: V-Zyklus und W-Zyklus auf 3 Gittern

aus Abbildung 2.7, offenbart sich ein Zusammenhang, denn man allgemein schreibt als

$$I_{2h}^{h} = c\left(I_{h}^{2h}\right)^{T}, \quad \text{für } c \in \mathbb{R}.$$

Diese Bedingung ist sehr hilfreich bezüglich der Analyse solcher Verfahren und wird dementsprechend auch häufig gefordert.

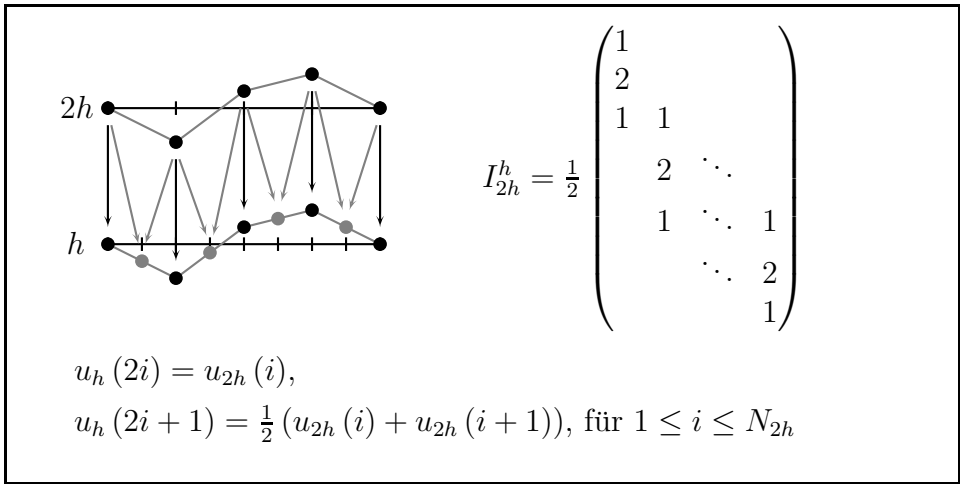


Abbildung 2.6: 1D Interpolation (Prolongation)

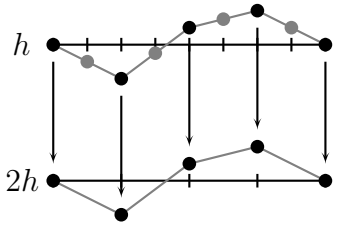
Damit benötigt man für das Verfahren nur mehr den Grobgitteroperator A_{2h}. Dieser kann aus der Grobgitterdiskretisierung gewonnen werden oder beispielsweise durch

$$A_{2h} = I_{h}^{2h} A_h I_{2h}^{h} \qquad \text{(Galerkinansatz)}. \tag{2.7}$$

Der Galerkinansatz für den Grobgitteroperator wird in Abbildung 2.8 für das Modellbeispiel 2.1 untersucht. Dabei wird der Interpolationsoperator aus Abbildung 2.6 und der gemittelte Restriktionsoperator aus Abbildung 2.7 verwendet. Man sieht, daß in diesem Fall der Galerkinansatz genau die Grobgitterdiskretisierung liefert.

Konvergenzanalyse:

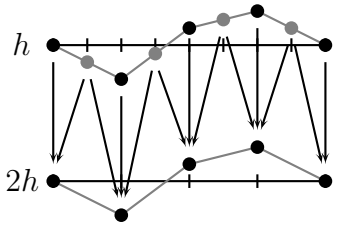
Injektion:



$$I_h^{2h} = \begin{pmatrix} 0 & 1 & 0 & & & & \\ & & 0 & 1 & 0 & & \\ & & & \ddots & \ddots & \ddots & \\ & & & & 0 & 1 & 0 \end{pmatrix}$$

$u_{2h}(i) = u_h(2i)$ für $1 \leq i \leq N_{2h}$

gewichtete Mittlungen:



$$I_h^{2h} = \tfrac{1}{4} \begin{pmatrix} 1 & 2 & 1 & & & & \\ & & 1 & 2 & 1 & & \\ & & & \ddots & \ddots & \ddots & \\ & & & & 1 & 2 & 1 \end{pmatrix}$$

$u_{2h}(i) = \frac{1}{4}\left(u_h(2i-1) + 2u_h(2i) + u_h(2i+1)\right)$ für $1 \leq i \leq N_{2h}$

Abbildung 2.7: 1D Restriktion

Für die Konvergenzanalyse folgen wir den Darstellungen von Hackbusch in [8]. Die Fehlerfortpflanzung des Zweigitterverfahrens mit Glättungsiteration

$$e_h^{l+1} = (I - B_h A_h)\, e_h^l$$

ist gegeben durch

$$e_h^{l+1} = \underbrace{(I - B_h A_h)^{\nu_2}}_{\text{Nachglätter}} \underbrace{\left(I - I_{2h}^h \left(A_{2h}\right)^{-1} I_h^{2h} A_h\right)}_{\text{Grobgitterkorrektur}} \underbrace{(I - B_h A_h)^{\nu_1}}_{\text{Vorglätter}} e_h^l,$$

denn für die Grobgitterkorrektur gilt

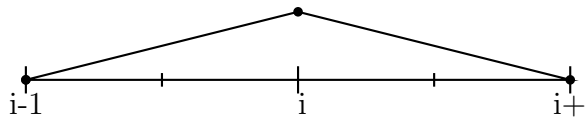
$$\begin{aligned} u_h^{l+1} &= u_h^l + I_{2h}^h \left(A_{2h}\right)^{-1} I_h^{2h} \left(f_h - A_h u_h^l\right) \\ e_h^{l+1} = u_h - u_h^{l+1} &= e_h^l - I_{2h}^h \left(A_{2h}\right)^{-1} I_h^{2h} A_h e_h^l. \end{aligned}$$

Für die Konvergenz von Mehrgitterverfahren muss die Qualität der Glättung (Glättungseigenschaft) und die Grobgitterkorrektur (Approximationseigenschaft) untersucht werden.

Definition 2.1 *(Glättungseigenschaft) Ein Iterationsverfahren*

$$u_h \leftarrow (I - B_h A_h)\, u_h$$

Berechne $A_{2h}e^i_{2h}$ mit $e^i_{2h} := (0,\ldots,0,1,0,\ldots,0)^T$



0		1		0	e^i_{2h}
0	$\frac{1}{2}$	1	$\frac{1}{2}$	0	$I^h_{2h}e^i_{2h}$
$-\frac{1}{2h}$	0	$\frac{1}{h}$	0	$-\frac{1}{2h}$	$A_hI^h_{2h}e^i_{2h}$
$-\frac{1}{4h}$		$\frac{1}{2h}$		$-\frac{1}{4h}$	$I^{2h}_hA_hI^h_{2h}e^i_{2h}$

$\Rightarrow$ ite Zeile von A_{2h} entspricht $\frac{1}{2h}\begin{bmatrix}-1 & 2 & -1\end{bmatrix}$, welche mit der Grobgitterdiskretisierung des Ausgangsproblems übereinstimmt

Abbildung 2.8: Grobgitteroperator mittels Galerkinansatz für das Modellbeispiel 2.1

erfüllt die Glättungseigenschaft, wenn eine von h unabhängige Funktion $\eta(\nu)$ mit $\lim_{\nu\to\infty}\eta(\nu)=0$ existiert, so dass $\forall\nu\in\mathbb{N}_0$ die Abschätzung

$$\|A_h\left(I-B_hA_h\right)^\nu\|_2 \le \eta(\nu)\,\|A_h\|_2 \tag{2.8}$$

gilt.

Definition 2.2 *(Approximationseigenschaft) Die Grobgitterkorrektur erfüllt die Approximationseigenschaft, wenn für ein c unabhängig von h die Abschätzung*

$$\left\|(A_h)^{-1}-I^h_{2h}\left(A_{2h}\right)^{-1}I^{2h}_h\right\|_2 \le \frac{c}{\|A_h\|_2} \tag{2.9}$$

gilt.

Zusammen ergibt sich die Konvergenz für das Zwei-Level-Verfahren:

Satz 2.2 *Es sei die Glättungseigenschaft und die Approximationseigenschaft erfüllt. Zu gegebenen $0<\zeta<1$ gibt es eine untere Schranke ν_0, so dass für alle $\nu>\nu_0$ gilt:*

$$\left\|\left(I-I^h_{2h}\left(A_{2h}\right)^{-1}I^{2h}_hA_h\right)\left(I-B_hA_h\right)^\nu\right\|_2 \le c\eta(\nu)\le\zeta. \tag{2.10}$$

Dabei sei betont, dass die Kontraktionsschranke $c\eta(\nu)$ unabhängig von h ist.

Mittels Rekursion lässt sich dieser Satz auch auf den W-Zyklus verallgemeinern, siehe [8]. Damit ist die Konvergenzrate des Mehrgitterverfahren bei geeigneter Wahl der MG-Operatoren unabhängig von h. Im Gegensatz zu den klassischen Iterationsverfahren wird dann die Konvergenz für $h\to 0$ nicht beliebig langsam.

2.3 Algebraische Mehrgitterverfahren

Im Gegensatz zum geometrischen Fall will man nun das lineare Gleichungssystem lösen, ohne Kenntnis über die zugrundeliegende Diskretisierung. Insbesondere gibt es keinerlei Information über die Grobgitterdiskretisierung. Dieser Ansatz führt zu Algebraischen Mehrgitterverfahren (AMG-Verfahren); siehe [3] und [12].
Diese Verfahren bestehen aus 2 Phasen:

1. Die Setupphase, in der die gröberen Gitter mitsamt ihren Gleichungssystemen und die zugehörigen Transferoperatoren erzeugt werden.

2. Die Lösungsphase, in der das Gleichungssystem mit Hilfe der zuvor erstellten Gitterhierachie gelöst wird.

Bemerkung 2.1 *Obwohl man hier von "Gittern" spricht, handelt es sich hierbei strikt genommen nur um Indexmengen. Der Begriff "Gitter" hat sich aber in der Literatur eingebürgert.*

Die Konstruktion der Mehrgitterhierachie erfordert auf jedem Level $l = 0, \ldots, L-1$ die Lösung der folgenden Aufgaben: (Dabei bezeichne 0 das feinste Level, L das gröbste.)

- Grob-Feingitter(C/F) Aufteilung; die Aufteilung des Gitters Ω_l in die Menge der Grobgitterpunkte $C_l = \Omega_{l+1}$ und die Menge der Feingitterpunkte F_l
- Die Konstruktion des Interpolationsoperators $I_{l+1}^l : \mathbb{R}^{N_{l+1}} \to \mathbb{R}^{N_l}$ und des Restriktionsoperators $I_l^{l+1} : \mathbb{R}^{N_l} \to \mathbb{R}^{N_{l+1}}$
- Die Aufstellung des Grobgitteroperators $A_{l+1} = I_l^{l+1} A_l I_{l+1}^l$ (Galerkinansatz).

Auf diese 3 Schritte wird im Folgenden näher eingegangen.

2.3.1 C/F-Aufteilung

Sei nun Ω die Indexmenge der Variablen auf einem beliebigen Level. Diese muss zur Erzeugung des nächstgröberen Gitters in die Menge der Grobgitterpunkte C und die Menge der Feingitterpunkte F aufgeteilt werden, mit

$$\Omega = C \cup F, \quad C \cap F = \emptyset.$$

Als ersten einfachen Aufteilungsansatz wird nur der Matrix-Graph der Systemmatrix A, definiert durch $G(A) := \{(i,j) \in \Omega \times \Omega | A_{ij} \neq 0\}$, für die Konstruktion eines Verfahrens miteinbezogen, siehe Algorithmus 1.

Algorithmus 1 startet mit der Wahl des ersten Punktes aus Ω und definiert diesen als Grobgitterpunkt. Alle Punkte, welche mit diesem verbunden sind, werden als Feingitterpunkte definiert; sprich alle Punkte aus der zugehörigen Matrixzeile, abgesehen vom Diagonalelement. Diese Vorgangsweise wird wiederholt, bis jeder Punkt aus Ω entweder ein Feingitter- oder ein Grobgitterpunkt ist.

Algorithmus 1 Einfache Aufteilung(Ω)

$C \leftarrow \emptyset, F \leftarrow \emptyset, T \leftarrow \Omega$
while $T \neq \emptyset$ **do**
  wähle $i \in T$
  $C \leftarrow C \cup \{i\}$
  $F \leftarrow F \cup \{j \in \Omega | j \notin C \cup F \wedge i \neq j \wedge A_{ij} \neq 0\}$
  $T \leftarrow T \backslash (C \cup F)$
end while
return C, F

Bemerkung 2.2 *Im Zusammenhang mit diesem ersten Aufteilungsansatz sei darauf hingewiesen, dass hieraus nicht klar hervorgeht, welche i für die Aufteilung aus der Indexmenge gewählt werden sollen oder müssen. Ein heuristischer Ansatz wäre die lexikographische Ordnung der Punkte für die Wahl zu nehmen. Ein andere Möglichkeit wäre die Knoten i mit den meisten Nachbarn zu wählen. In der Praxis zeigt sich, dass derartige einfache Vergröberungsverfahren beispielsweise bei anisotropen partiellen Differentialgleichungen zu großen Problemen führen. Verbesserte Heuristiken nützen die Größe der Matrixeinträge, worauf im folgenden eingegangen wird.*

Eine verbesserte Version liefert das Vergröberungsschema nach Rüge/Stüben [12]. Dabei werden die Größe der Matrixeinträge und nicht nur der Matrix-Graph berücksichtigt. Ein Punkt i ist mit einem Punkt $j \neq i$ stark gekoppelt, wenn

$$|A_{ij}| > \epsilon |A_{ii}| \tag{2.11}$$

mit $0 < \epsilon < 1$ gilt. Der neue Algorithmus 2 verläuft bis auf die Auswahl der Feingitterpunkte gleich wie der bereits besprochene. Für jeden Grobgitterpunkt werden nur diejenigen als Feingitterpunkte gewählt, welche mit dem Grobgitterpunkt stark gekoppelt sind. Für den Fall $\epsilon = 0$ erhält man Algorithmus 1.

Algorithmus 2 Verbesserte Aufteilung(Ω)

$C \leftarrow \emptyset, F \leftarrow \emptyset, T \leftarrow \Omega$
while $T \neq \emptyset$ **do**
  wähle $i \in T$
  $C \leftarrow C \cup \{i\}$
  $F \leftarrow F \cup \{j \in \Omega | j \notin C \cup F \wedge i \neq j \wedge |A_{ij}| > \epsilon |A_{ii}|\}$
  $T \leftarrow T \backslash (C \cup F)$
end while
return C, F

2.3.2 Gittertransferoperatoren

Bevor auf die Konstruktion der Transferoperatoren zwischen den Leveln genauer eingegangen wird, erweist sich ein genauerer Blick auf die Struktur des Gleichungs-

systems als nützlich. Ausgehend von einer C/F-Aufteilung, kann das Ausgangsproblem mittels geeigneter Permutation in der Form

$$\underbrace{\begin{pmatrix} A_{CC} & A_{CF} \\ A_{FC} & A_{FF} \end{pmatrix}}_{=:A} \begin{pmatrix} u_C \\ u_F \end{pmatrix} = \begin{pmatrix} f_C \\ f_F \end{pmatrix}$$

geschrieben werden. Definiert man weiters $T = \begin{pmatrix} I & 0 \\ A_{FF}^{-1}A_{FC} & I \end{pmatrix}$, so kann die neue Systemmatrix A in Blockdiagonalform dargestellt werden als

$$A = T^T \begin{pmatrix} S_C & 0 \\ 0 & A_{FF} \end{pmatrix} T,$$

wobei $S_C = A_{CC} - A_{CF}A_{FF}^{-1}A_{FC}$ das Schurkomplement bezüglich der F-Komponenten in A bezeichnet. Daraus motiviert, definiert man weiters

$$P := \begin{pmatrix} I \\ -A_{FF}^{-1}A_{FC} \end{pmatrix} \text{ und } R := \begin{pmatrix} I & -A_{CF}A_{FF}^{-1} \end{pmatrix}.$$

Lemma 2.1 *Bezüglich der Transferoperatoren P, R stellt das Schurkomplement den Galerkinoperator dar, d.h.*

$$S_C = RAP.$$

Dieser Sachverhalt soll anhand des Modellbeispieles 2.1 für $n = 8, N = 7, h = \frac{1}{8}$ in Abbildung 2.9 verdeutlicht werden.

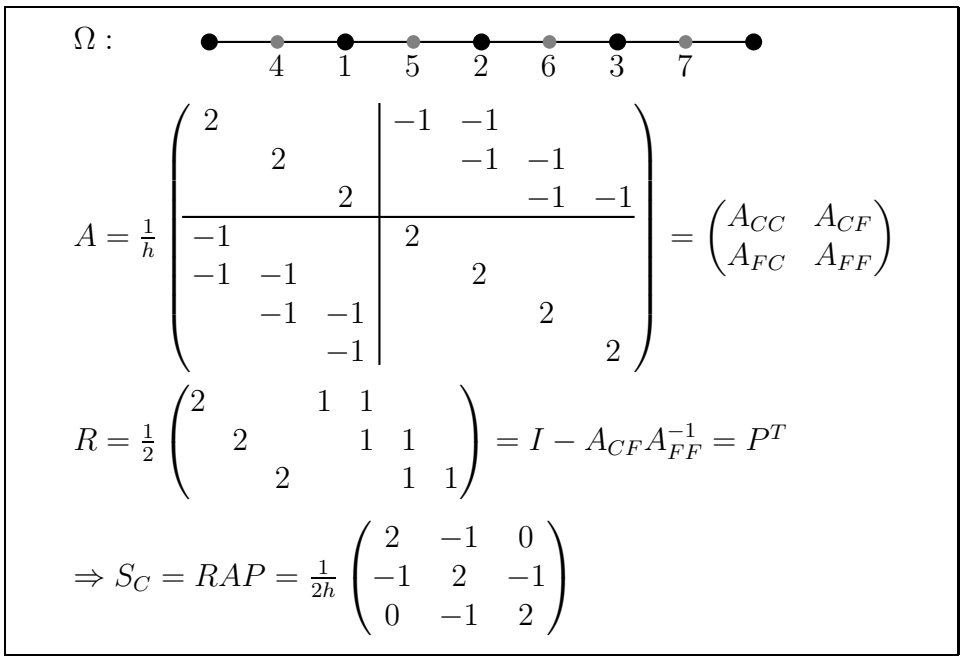


Abbildung 2.9: Schurkomplement für das Modellbeispiel 2.1 ($n = 8, N = 7$)

Definiert man $G := \begin{pmatrix} 0 & 0 \\ 0 & A_{FF}^{-1} \end{pmatrix}$, ergibt sich folgende Eigenschaft des Zweigitterverfahrens:

Lemma 2.2 *Die Zweigittermethode, bestehend aus der durch P, R, S_C definierten Grobgitterkorrektur und dem Nachglättungsschritt $I - GA$, ist exakt.*

Beweis: Sei e^l der Fehler, dann gilt für die Fehlerfortpflanzung $e^{l+1} = Me^l$, wobei

$$M = \underbrace{(I - GA)}_{\text{Nachglättung}} \underbrace{\left(I - PS_C^{-1}RA\right)}_{\text{Grobgitterkorrektur}}$$

den Mehrgitteroperator für die Zweigittermethode mit Nachglättung darstellt. Eine Umformung der Fehlerfortpflanzung liefert

$$\begin{aligned} e^{l+1} &= Me^l \\ u - u^{l+1} &= M\left(u - u^l\right) \\ u^{l+1} &= u - Mu + Mu^l \\ &= (I - M)\, A^{-1} f + Mu^l. \end{aligned}$$

Falls nun der Fehlerübergangsoperator M Null ist, ist die Zweigittermethode exakt. Dies verifiziert man durch Einsetzen der Operatoren. Für die detailierte Rechnung siehe [13, §3, S. 41]. □

Bemerkung 2.3 *Der Block A_{FF} ist für das Modellbeispiel 2.1 eine Diagonalmatrix und daher leicht zu invertieren. Im Allgemeinen ist dies aber nicht der Fall, siehe das 2D Beispiel aus [13]. Dies führt zur Idee, dass man $-A_{FF}^{-1}A_{FC}$ durch einen geeigneten Operator B_{FC} approximiert, siehe [13].*

Damit wählt man folgenden Ansatz für den Interpolationsoperator

$$P = \begin{pmatrix} I \\ P_{FC} \end{pmatrix}, \text{ so dass} \qquad (Pe)_i = \begin{cases} e_i & \text{, für } i \in C \\ \sum_{j \in C} w_{ij} e_j & \text{, für } i \in F \end{cases}$$

d.h. die Variablen aus C werden übernommen, wohingegen die Variablen in F aus den übrigen interpoliert werden. Dazu müssen noch die Gewichte $w_{ij} \in \mathbb{R}$ bestimmt werden. Dafür gibt es unterschiedliche Ansätze. Bezeichne N_i die Nachbarschaft eines Knoten $i \in \Omega$, d.h.

$$N_i := \{j \in \Omega \setminus \{i\} : (i, j) \in G(A)\}.$$

Da der Interpolationsprozess üblicherweise lokal ist, wählt man

$$(Pe)_i = \sum_{j \in P_i} w_{ij} e_j \quad , \forall i \in F,$$

wobei $P_i \subseteq (N_i \cap C)$ ist. Mithilfe der Charakterisierung des algebraisch glatten Fehlers ($Ae \simeq 0$), siehe [7], kann die Interpolation eines Feingitterpunktes i durch

$$(Pe)_i = \sum_{j \in P_i} w_{ij} e_j = -\frac{\sum_{j \in N_i} a_{ij}}{a_{ii} \sum_{j \in P_i} a_{ij}} \sum_{j \in P_i} a_{ij} e_j \tag{2.12}$$

angegeben werden, wobei diese Definition als die direkte Interpolation bezeichnet wird. Für alternative Interpolationsvorschriften sei auf Ruge/Stüben [12] verwiesen. Abschließend sei noch eine kostengünstige Variante aus [10] angegeben, welche definiert ist durch

$$(P_{FC})_{ij} = \begin{cases} \frac{1}{\sharp N_i} & , |A_{ij}| > \alpha \, |A_{ii}| \\ 0 & , \text{sonst.} \end{cases} \tag{2.13}$$

Den Restriktionsoperator definiert man durch den transponierten Interpolationsoperator, sprich

$$R = P^T,$$

denn dann gilt für symmetrisch, positiv definite Matrizen A, dass auch der Galerkinansatz $A_C := P^T AP$ symmetrisch und positiv definit ist.

2.3.3 Grobgitteroperator

Der Grobgitteroperator ergibt sich mittels des Galerkinansatzes zu

$$A_C := RAP = P^T AP.$$

2.3.4 AMG für Systeme

Die bisher präsentierten algebraischen Mehrgitterverfahren sind Methoden zur Lösung von linearen Systemen, welche von der Diskretisierung skalarer (elliptischer) partieller Differentialgleichungen herrührt. Betrachtet man Systeme von PDE's liefern derartige Verfahren keine guten Ergebnisse. Dies hängt damit zusammen, dass mittels skalarer Vergröberungsmethoden angewandt auf Systeme, physikalisch gekoppelte Größen getrennt werden, und damit die zugrundeliegende Struktur auf gröberen Gittern verloren geht. Daher muss im Zusammenhang mit 3D-Elastizitätsprobleme auf Methoden zurückgegriffen werden, welche für derartige Systeme geeignet sind, siehe [4]. Für eine Einführung in diese Thematik folgen wir den Darstellungen aus [7].

Ausgehend von der Vorausetzung, dass alle d physikalischen Unbekannten auf dem selben Gitter diskretisiert wurden, kann der Koeffizientenvektor u partitioniert werden in

$$u = (u_k)_{k=1}^n \,,$$

wobei $u_k \in \mathbb{R}^d$ (für 3D-Probleme gilt $d = 3$) den Vektor mit den Koeffizienten zum jeweiligen Gitterpunkt darstellt. Dementsprechend wird die Systemmatrix A in Blöcke aufgeteilt

$$A = \begin{pmatrix} A_{11} & \dots & A_{1n} \\ \vdots & \ddots & \vdots \\ A_{n1} & \dots & A_{nn} \end{pmatrix}$$

wobei $A_{ij} \in \mathbb{R}^{d \times d}$ ist.

Für die C/F-Aufteilung waren für skalare elliptische Probleme die Kopplungen zwischen den Unbekannten, welche in diesem Kontext Gitterpunkte repräsentierten, ausschlaggebend. Deshalb benötigt man nun eine Verallgemeinerung, welche weiters die Kopplungen zwischen den Gitterpunkten untersucht. Um die bereits eingeführten Algorithmen verwenden zu können, müssen starke Kopplungen zwischen den Blöcken der Systemmatrix, welche auf die jeweiligen Gitterpunkte referenzieren, betrachtet werden. Eine mögliche Vorgehensweise hierfür ist die Block-Matrix A in eine Skalar-Matrix $\tilde{A}$ überzuführen, wobei ein Block A_{ij} mittels einer geeigneten Matrixnorm $\|\cdot\|$ zu $\tilde{a}_{ij}$ wird, d.h.

$$\tilde{A} = (\tilde{a}_{ij}) := (\|A_{ij}\|) .$$

Dieser Ansatz wird in Abbildung 2.10 verdeutlicht. Nun ist noch offen, welche Norm für das Schema verwendet werden soll. Hinsichtlich der Wichtigkeit der Eigenwerte für die Vergröberung, wäre die L_2-Norm eine geeignete Wahl. Hinsichtlich der Berechnungskosten kommen auch billigere Alternativen in Frage, wie die Frobeniusnorm.

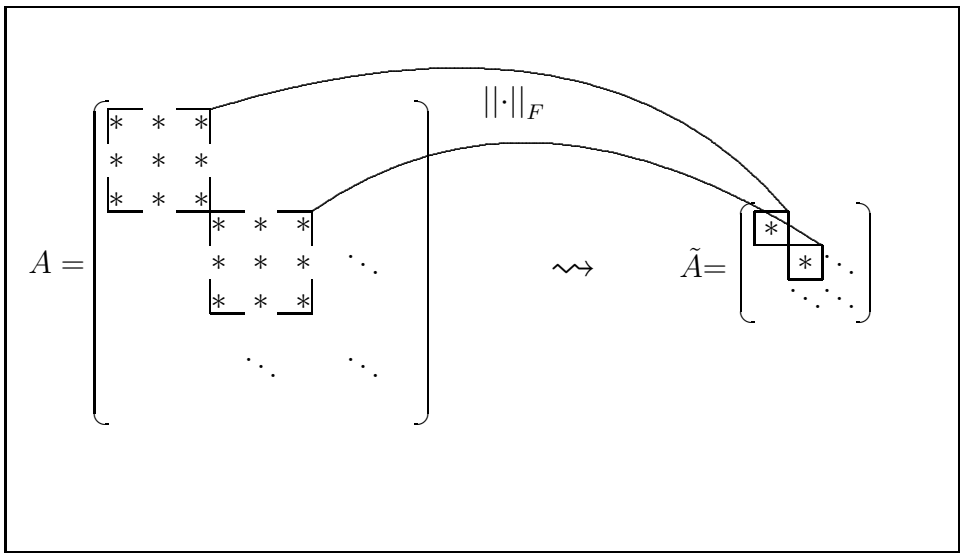


Abbildung 2.10: Transformation der Systemmatrix A in die reduzierte Matrix $\tilde{A}$

Auch für den Interpolationsoperator muss man sich von den skalaren Schemen trennen. Eine Verallgemeinerung könnte wie folgt ausschauen. Anstatt die Interpolation mit den skalaren Einträgen der Systemmatrix aufzusetzen, werden die Blöcke A_{ij} verwendet. Betrachtet man das direkte Interpolationsschema 2.12, folgt für das direkte Block-Interpolationsschema die Darstellung

$$(P_{FC}e)_i = -A_{ii}^{-1} \left(\sum_{k \in N_i} A_{ik} \right) \left(\sum_{k \in P_i} A_{ik} \right)^{-1} \left(\sum_{k \in P_i} A_{ik} e_k \right) . \tag{2.14}$$

Bemerkung 2.4 *Eine detailierte Betrachtung der Interpolationvorschrift 2.14 angewandt auf lineare Elastizitätsprobleme findet man in [7].*

2.4 Vorkonditioniertes konjugiertes Gradientenverfahren

Das Verfahren der konjugierten Gradienten ist ebenfalls eine effiziente Methode zur Lösung von symmetrischen, positive definiten Gleichungssystemen; siehe [2, §IV.3, S. 186]. Es gehört zur Klasse der Krylow Unterraumverfahren.
Ein Grundgedanke der Methode der konjugierten Gradienten besteht darin, dafür zu sorgen, dass aufeinander folgende Richtungen nicht fast parallel sind. Es werden orthogonale Richtungen angestrebt. Dabei wird dem Orthogonalitätsbegriff nicht die Euklidische Metrik zugrunde gelegt, sondern die Metrik vielmehr dem Problem angepasst.
Der vorkonditionierte konjugierte Gradientenalgorithmus (PCG) benötigt die Systemmatrix A, die rechte Seite f, eine Anfangsschätzung für die Lösung u, den Vorkonditionierer $\mathtt{C}^{-1}$, die maximale Anzahl von Iterationen $iter_{max}$ und eine vorgegebene Fehlertoleranz ϵ als Übergabewerte und liefert die Lösung u.

Algorithmus 3 PCG$(A, f, u, \mathtt{C}^{-1}, iter_{max}, \epsilon)$

$r \leftarrow f - Au$
$d \leftarrow \mathtt{C}^{-1}r$
$\sigma \leftarrow \langle d, r\rangle$
$\sigma_{start} \leftarrow \sigma_{alt} \leftarrow \sigma$
$iter \leftarrow 0$
while $iter < iter_{max} \wedge \sigma/\sigma_{start} > \epsilon^2$ **do**
  $iter \leftarrow iter + 1$
  $v \leftarrow Ad$
  $\alpha \leftarrow \sigma / \langle d, v\rangle$
  $u \leftarrow u + \alpha d$
  $r \leftarrow r - \alpha v$
  $h \leftarrow \mathtt{C}^{-1}r$
  $\sigma \leftarrow \langle h, r\rangle$
  $\beta \leftarrow \sigma/\sigma_{alt}$
  $d \leftarrow h + \beta d$
  $\sigma_{alt} \leftarrow \sigma$
end while
return u

Der Vorkonditionierungsschritt $\mathtt{C}^{-1}r$ wird über das AMG-Verfahren so realisiert, daß $\mathtt{C}^{-1}$ eine Approximation der Systemmatrix A^{-1} darstellt. Damit ist das gewünschte PCG-AMG Verfahren vollständig angegeben.

Bemerkung 2.5 *Der Vorkonditionierer* $\mathtt{C}$ *für das konjugierte Gradientenverfahren muss symmetrisch und positiv definit (spd) sein, um die Konvergenz des Verfahrens sicherzustellen. Um das AMG-Verfahren zur Vorkonditionierung verwenden zu können, müssen folgende Punkte erfüllt sein (siehe [13, §6.2, Satz 6.3, S. 141])*

- *Der A-priori-Vorkonditionierer* $\mathtt{C} \equiv \mathtt{C}_0$ *muss spd sein.*

- *Der Restriktionsoperator* I_l^{l+1} *ist transponiert zum Prolongationsoperator* I_{l+1}^l, *d.h.* $I_l^{l+1} = \left(I_{l+1}^l\right)^T$ *für* $l = 0, \ldots, L-1$. *Dadurch wurde aber gerade der Restriktionsoperator definiert.*

- *Die Hilfsoperatoren* $\mathtt{C}_l, l = 1, \ldots, L$ *auf den diversen Leveln müssen ebenfalls spd sein. Dies wird mithilfe der vorherigen Forderung und dem Galerkinansatz für die Operatoren sichergestellt.*

- *Der Nachglättungsfehlerübergangsoperator* $S_l^{(nach)}$ *ist adjungiert zum Vorglättungsfehlerübergangsoperator* $S_l^{(vor)}$ *im* $\mathtt{C}_l$*-energetischen Skalarprodukt* $(u_l, v_l)_{\mathtt{C}_l} := (\mathtt{C}_l u_l, v_l)_l$ *(*$(\cdot,\cdot)_l$ *bezeichnet das Euklidische Skalarprodukt), d.h. für* $l = 0, \ldots, L-1$ *gilt:*

$$\left(S_l^{(nach)} u_l, v_l\right)_{\mathtt{C}_l} = \left(u_l, S_l^{(vor)} v_l\right)_{\mathtt{C}_l} \quad \forall u_l, v_l \in \mathbb{R}^{N_l}$$

Nach [13, §6.2, Satz 6.6, S. 150] erfüllen die bisher besprochenen Glättungsiterationsverfahren diese Eigenschaft, sofern die Anzahl der Vorglättungsschritte gleich der Anzahl der Nachglättungsschritte entspricht.

2.5 Zeitlich veränderliche Materialparameter

Für zeitlich veränderliche Stoffparameter ist die Systemmatrix auch zeitabhängig und man erhält

$$A(t)\, u_t = f_t, \quad t \in \{0, 1, \ldots, T\} \tag{2.15}$$

mit $u_t, f_t \in \mathbb{R}^{N_h}$ und $A(t) \in \mathbb{R}^{N_h \times N_h}$. Für die Lösung dieses Problems kann das PCG-AMG Verfahren verwendet werden. Für die Umsetzung benötigt man in jedem Zeitschritt $t \in \{0, 1, \ldots, T\}$

1. die AMG-Setupphase für $A(t)$,
2. das PCG-Verfahren für $A(t)$ und darin die Vorkonditionierung $C(A(t))$.

Für den Fall, dass die zeitabhängigen Parameter nicht allzu stark variieren, kommen kostengünstigere Varianten in Frage, indem man die AMG-Setupphase nur für $t = 0$ oder zumindest nicht für alle $t \in \{0, 1, \ldots, T\}$ durchführt.

2.6 Nicht-linear elastisches Verhalten

Materialien, deren Stoffgesetz eine nichtlineare Funktion in Abhängigkeit von der Dehnung des Körpers ist, führen zu einem nichtlinearen Ausgangsproblem. Die Diskretisierung dessen liefert ein nichtlineares Gleichungssystem

$$A_h(u_h) = f_h \tag{2.16}$$

mit $u_h, f_h \in \mathbb{R}^{N_h}$ und $A_h(u_h) : \mathbb{R}^{N_h} \to \mathbb{R}^{N_h}$. Im quasilinearen Fall (oder mittels Linearisierung) lässt sich das Problem beschreiben durch

$$\hat{A}_h(u_h)\, u_h = f_h \tag{2.17}$$

mit $\hat{A}_h(u_h) \in \mathbb{R}^{N_h \times N_h}$.
Eine naheliegende Idee zur Lösung von (2.16) ist die Anwendung von Newton- bzw. newtonähnlichen Methoden. Damit muss statt (2.16) eine Folge von linearisierten Problemen, worauf sich die bereits beschriebenen Verfahren anwenden lassen, gelöst werden. Die Newtoniterationen für (2.16) sind gegeben durch

$$u_h^{k+1} = u_h^k + A_h'\left(u_h^k\right)^{-1}\left(f_h - A_h\left(u_h^k\right)\right), \quad k = 0, 1, 2, \ldots \tag{2.18}$$

wobei $A_h'\left(u_h^k\right)$ die Jacobi-Matrix von A_h bezeichnet, und u_h^0 ein gegebener Startvektor ist. Das Newtonverfahren (2.18) zur Linearisierung wird in Algorithmus 4 festgehalten.

Algorithmus 4 Newton-AMG$(A_h, A_h', f_h, u_h^0, \epsilon)$

$k = -1$
repeat
  $k = k + 1$
  $d_h^k = f_h - A_h\left(u_h^k\right)$
  $A_h'\left(u_h^k\right) w_h^k = d_h^k$ {← wird mittels Mehrgitterverfahren oder PCG-AMG gelöst}
  $u_h^{k+1} = u_h^k + w_h^k$
until $\left\|u_h^{k+1} - u_h^k\right\| < \epsilon$
return u_h^{k+1}

Bemerkung 2.6 *Die Konvergenzgeschwindigkeit des Newtonverfahrens ist quadratisch und die des Mehrgitterverfahrens linear, wodurch nur lineare Konvergenz für das Gesamtverfahren folgt. Zur Rettung der quadratischen Konvergenz muss die Anzahl der Mehrgitterzyklen sukzessive vergrößert, z.B. verdoppelt, werden; siehe [1, §3.5.1].*

Die Gleichung (2.17) kann auch mittels der Fixpunktiteration

$$u_h^{k+1} = u_h^k + \hat{A}_h\left(u_h^k\right)^{-1}\left(f_h - A_h\left(u_h^k\right)\right), \quad k = 0, 1, 2, \ldots$$

gelöst werden. Diese Vorgehensweise entspricht Algorithmus 4 sofern man $\hat{A}_h\left(u_h^k\right)$ als Systemmatrix für das lineare Verfahren verwendet. Diese Variante ist kostengünstiger, aber ein Nachteil ist die lineare Konvergenz.
Eine alternative Vorgehensweise ist die direkte Anwendung der Mehrgittermethode auf nichtlineare Probleme, wodurch beispielsweise keine Ableitungen benötigt werden; siehe [1, §3.5.2].

Kapitel 3

Implementation

Das vorherige Kapitel hat sich mit einem effizienten Verfahren zur Lösung linearer Gleichungssysteme beschäftigt, nämlich das Algebraische Mehrgitterverfahren als Vorkonditionierer in einem konjugierten Gradientenverfahren (PCG-AMG). Dieses Verfahren soll nun mittels einer benutzerfreundlichen C++ Toolbox [11] umgesetzt werden, welche parallelisierte Versionen des gewünschten Verfahrens bereitstellt. In diesem Abschnitt wird auf die Funktionsweise der Parallel Toolbox näher eingegangen. Eine ausführliche Beschreibung davon liefert [10].

3.1 Speicherformate

Da die Systemmatrix dünn besetzt und unstrukturiert ist, wird ein Speicherformat diskutiert, welches nicht alle Matrixeinträge speichert. Dazu betrachten wir das Compressed Row Storage(CRS) Format, mit welchem auch die Toolbox arbeiten kann. Eine Matrix $A \in \mathbb{R}^{n \times n}$, mit m Nichtnullelementen darin, kann wie folgt gespeichert werden:

count $\in \mathbb{N}^n$	Anzahl der Nicht-Null-Einträge pro Zeile
(dsp $\in \mathbb{N}^n$	Verschiebung)
col $\in \mathbb{N}^m$	Spalten-Indizes der Nicht-Null-Einträge zeilenweise
ele $\in \mathbb{R}^m$	Nicht-Null-Einträge für col

Diese Speichervariante wird im Folgenden an einer kleinen, dünn besetzten, Beispielmatrix illustriert:

$$\begin{pmatrix} 4 & 0 & -1 & 0 & -2 & 0 \\ 0 & 3 & 0 & 0 & -2 & -1 \\ 0 & 0 & 2 & 0 & -1 & 0 \\ 0 & -2 & 0 & 4 & 0 & 0 \\ 0 & 0 & -1 & -1 & 5 & 0 \\ -1 & 0 & 0 & 0 & -2 & 3 \end{pmatrix} \tag{3.1}$$

Abbildung 3.1 zeigt das CRS-Speicherformat für die Beispielmatrix 3.1. Dieses Format ist geeinget für Berechnungen auf der CPU, aber in Hinblick auf eine

3	3	2	2	3	3
0	3	6	8	10	13

1	3	5	2	5	6	3	5	2	4	3	4	5	1	5	6
4	-1	-2	3	-2	-1	2	-1	-2	4	-1	-1	5	-1	-2	3

Abbildung 3.1: Compressed Row Storage (CRS) - Datenformat für die Beispielmatrix 3.1 mit $count, dsp$ oben und col, ele darunter.

gewünschte GPU-Implementation liefert dieses Format keine effizienten Ergebnisse, siehe [10]. Zahlreiche Test für Finite Elemente Systemmatrizen zeigen, das das Interleaved Compressed Row Storage (ICRS) Format besser für derartige Berechnungen geeinget ist. Wie die nachfolgende ICRS-Speicherung Abbildung 3.2 für die obige Beispielmatrix illustriert, wird der *count*-Vektor gleichbehalten, jedoch *col* und *ele* reorganisiert. Dabei wird ein Block von L Zeilen von Spaltenindizes zusammengefasst. Wenn nicht alle Zeilen die gleich Anzahl von Nicht-Null-Einträgen besitzt, weißt dieses Datenformat Löcher auf. Obwohl dies die Speichereffizienz reduziert, gilt für Finite-Elemente-Simulationen, daß die Anzahl der Nichtnullelemente für jede Zeile annähernd gleich ist.

3	3	2	2	3	3
0	1	2	3	4	5

1	2	3	2	3	1	3	5	5	4	4	5	5	6			5	6
4	3	2	-2	-1	-1	-1	-2	-1	4	-1	-2	-2	-1			5	3

Abbildung 3.2: Interleaved Compressed Row Storage (ICRS) - Datenformat für die Beispielmatrix 3.1 mit $count, dsp$ oben und col, ele darunter.

3.2 Parallel Toolbox

Das Ziel der Parallel Toolbox ist es Parallelisierungen für partielle Differentialgleichungslöser basierend auf der Finiten Elemente Methode bereitzustellen. Der Finite Elemente Ansatz ist sehr flexibel bei der Erfassung komplizierter Geometrien des Berechnungsgebietes mit einfachen geometrischen Formen, den Elementen, und

stellt daher einen natürlichen Parallelisierungsansatz durch die Verteilung der Elemente auf die verschiedenen Recheneinheiten dar. Um die Kommunikation zwischen den Prozessoren so niedrig wie möglich zu halten, ist darauf zu achten, die Anzahl der Randknoten, welche von mehreren Prozessoren gemeinsam verwendet werden, zu minimieren. Optimale Aufteilungen können mittels Partitionierungsprogrammen, wie METIS, erzielt werden. Ausgehend von einer geeigneten Verteilung der Elemente auf die verschiedenen Prozessoren, stellt die Toolbox ein Kommunikatorobjekt zur Verfügung, welches die Informationen über die gemeinsamen Randknoten beinhaltet und für die Kommunikation unter den Prozessoren verantwortlich ist. Der Nachrichtenaustausch zwischen den Prozessoren wird dabei mittels dem Message Passing Interface (MPI) realisiert. Folgendes Unterkapitel zeigt einerseits wie man dieses Objekt verwendet und liefert andererseits einen Einblick auf den algebraischen Hintergrund der Parallelisierung. Daraufhin wird eine konkrete Umsetzung auf der CPU und auf der GPU illustriert.

3.2.1 Kommunikation - Parallelisierung

Die Idee des Kommunikator Objektes ist es, dem Benutzer der Toolbox ein Objekt bereitzustellen, welches die gesamte Kommunikation handhabt und verwaltet. Für die Realisierung wird eine C++ Templateklasse definiert. Die Templateargumente sind der Datentyp der Knotenindizes, üblicherweise *int*, und der Datentyp der Matrix und Vektor Einträge, typischerweise *double*. Folgender Code zeigt eine typische Instanziierung der Kommunikator Klasse, indem der Vektor der globalen Knotenpunkte *nod* übergeben wird.

```
communicator<int, double> com(nod);
```

Sämtliche Vektoren werden in der verwendeten Version der Toolbox mittels der C++ Standard Template Library (STL) *vector template class* repräsentiert. Dem Benutzer der Toolbox werden nun 3 Kommunikationsfunktionen zur Verfügung gestellt: *accumulate*, *distribute* und *collect*. Bevor auf diese Funktionen genauer eingegangen werden kann, müssen noch einige Konventionen und theoretische Überlegungen getroffen werden.
Das verwendete Parallelisierungskonzept ist eine Elementepartitionierung entlang der Kanten, sprich eine nicht überlappende Elementezerlegung. In diesem Zusammenhang muss zwischen 2 Arten von Vektoren unterschieden werden.

Definition 3.1 *(Akkumulierter Vektor) Sei $n \in \mathbb{N}$ die globale Problemgröße, $u \in \mathbb{R}^n$ ein Vektor, $P \in \mathbb{N}$ die Anzahl der verschiedenen Prozessoren, I_p die Menge der globalen Knotenpunkte auf dem Prozessor p und $n_p := \sharp I_p$ die Dimension von der Menge I_p mit $p \in \{1, \ldots, P\}$. Dann ist ein akkumulierter Vektor $\mathfrak{u}_\mathfrak{p} \in \mathbb{R}^{n_p}$ definiert als die Restriktion von u auf die Indexmenge I_p, d.h.*

$$\mathfrak{u}_\mathfrak{p} := u|_{I_p} \in \mathbb{R}^{n_p}$$

Definition 3.2 *(Verteilter Vektor) Sei $n \in \mathbb{N}$ die globale Problemgröße, I die globale Indexmenge mit $\sharp I = n$, $f \in \mathbb{R}^n$ ein Vektor, $P \in \mathbb{N}$ die Anzahl der verschiedenen*

Prozessoren, I_p die Menge der globalen Knotenpunkte auf dem Prozessor p und $n_p := \sharp I_p$ die Dimension von der Menge I_p mit $p \in \{1, \ldots, P\}$. Dann ist ein verteilter Vektor $f_p \in \mathbb{R}^{n_p}$ definiert als Teil der Zerlegung von f über der Indexmenge I_p, für welche die Relation

$$f = \sum_{p=1}^{P} f_p|^I \in \mathbb{R}^n$$

hält. Dabei bezeichne $f_p|^I \in \mathbb{R}^n$ die Nullerweiterung des lokalen Vektors $f_p \in \mathbb{R}^{n_p}$ auf einen globalen Vektor f mit der globalen Indexmenge I.

Für eine optisch leichtere Unterscheidung werden akkumulierte Vektoren rot (zusätzlich im fraktalen Schriftsatz) und verteilte Vektoren grün gekennzeichnet. Beide Vektortypen haben die gleiche Dimension auf den jeweiligen Prozessoren. Der Unterschied liegt in den numerischen Werten an den Randpunkten, welche von verschiedenen Prozessoren geteilt werden.
Die Prozedur *accumulate* nimmt einen verteilten Vektor als Übergabe und konvertiert diesen in einen akkumulierten Vektor. Dieser Prozess benötigt die Kommunikation zwischen den Prozessoren, und ist damit eine teure Operation. Der Quellcode für einen Akkumulationsprozess (b→𝔟) schaut damit unter Verwendung des bereits erstellten Kommunikators *com* wie folgt aus:

```
com.accumulate(b);
```

Die Prozedur *distribute* nimmt einen akkumulierten Vektor als Übergabe und konvertiert diesen in einen verteilten Vektor, sprich genau andersherum. Dieser Prozess ist allerdings im Gegensatz zur Akkumulation eine lokale Prozedur, wodurch keine Kommunikation erforderlich ist. Die Umsetzung des Verteilungsprozesses (𝔟→b) mit dem Kommunikator *com* könnte wie folgt durchgeführt werden:

```
com.distribute(b);
```

Die dritte Funktion der Kommunikator Klasse, *collect*, ermöglicht die Verteilung von numerischen Werten auf alle Prozessoren und berechnet die Summe all dieser Werte. Diese Prozedur wird typischerweise für die parallele Berechnung von Skalarprodukten benötigt. Sei beispielsweise *alpha* das Ergebnis eines lokalen Skalarproduktes eines Prozessors, dann wird mittels

```
com.collect(alpha);
```

in *alpha* das globale Resultat des Skalarproduktes gespeichert.

Dies sind also die Funktionen, welche die Toolbox dem Benutzer zur Verfügung stellt. Damit die gewünschten Lösungsverfahren auf den lokalen Prozessoren global korrekte Ergebnisse liefern, muss noch ein etwas genauerer Blick auf die parallelen algebraischen Operationen geworfen werden. Dazu benötigt man noch zu den Definitionen 3.1 und 3.2 für Vektoren die entsprechenden analogen Definitionen

	akkumuliert	verteilt
Vektoren	$\mathfrak{u}_\mathfrak{p}=M_p\mathfrak{u}$	$r=\sum_{i=1}^{P} M_i^T r_i$
Matrizen	$\mathfrak{A}_\mathfrak{p}=M_p\mathfrak{A}\ M_p^T$	$A=\sum_{i=1}^{P} M_i^T A_i M_i$

Tabelle 3.1: Datenrepräsentation für nichtüberlappende Gebietszerlegungen

für Matrizen. Dann kann die Datenrepräsentation über die Global-zu-lokal linearen Abbildungen

$$M_i := \begin{pmatrix} & 1 & \\ & \ddots & \\ & & 1 \end{pmatrix}, \quad \forall i \in \{1, \ldots, P\}$$

wobei die Eins-Einträge in den jeweiligen linearen Operatoren M_i aus der Indexmenge der lokalen Knotenpunkten bestimmt ist, in Tabelle 3.1 festgehalten werden.

Betrachtet man nun als erstes das parallele Skalarprodukt zwischen zwei Vektoren u und r, zeigt sich bereits, dass man bei den Vektortypen für die lokalen Berechnungen aufpassen muss, denn man erhält nur ein korrektes globales Resultat für das Skalarprodukt, wenn einer der beiden Vektoren verteilt und der andere akkumuliert ist. Dies wird durch folgende Rechnung verdeutlicht:

$$\langle \mathfrak{u}, r \rangle = \mathfrak{u}^T r = \mathfrak{u}^T \sum_{i=1}^{P} M_i^T r_i = \sum_{i=1}^{P} (M_i \mathfrak{u})^T r_i = \sum_{i=1}^{P} \langle \mathfrak{u}_\mathfrak{i}, r_i \rangle$$

Für eine weitere wichtige Operation, das Matrix-Vektor Produkt, ergibt sich

$$A\mathfrak{u} = \sum_{i=1}^{P} M_i^T A_i M_i \mathfrak{u} = \sum_{i=1}^{P} M_i^T A_i \mathfrak{u}_\mathfrak{i} = \sum_{i=1}^{P} M_i^T f_i = f.$$

Diese Rechnung wird zur Anschaulichkeit anhand eines pathologischen Beispiels in Abbildung 3.3 illustriert. Da für die Kosten des Verfahrens maßgeblich ist, ob für eine Operation eine Kommunikation erforderlich ist, werden im Folgenden die grundlegenden algebraischen Operationen diesbezüglich untersucht und eingeteilt.

- keine Kommunikation:
  - Matrix-Vektor Produkt (Beispiel: $v \leftarrow A\mathfrak{d}$)
  - Lineare Operationen bezüglich des selben Vektortyps (Beispiele: $r \leftarrow r - \alpha v$ und $\mathfrak{u} \leftarrow \mathfrak{u} + \alpha\mathfrak{d}$)
  - einen akkumulierten Vektor zu verteilen (Beispiel: $r \leftarrow R^{-1}\mathfrak{r}$)
    Der für diesen Prozess notwendige Operator R^{-1} ergibt sich aus

$$R = diag\,\{R_{ii}\}_{i=1}^{n} = \sum_{i=1}^{P} M_i^T \cdot M_i$$

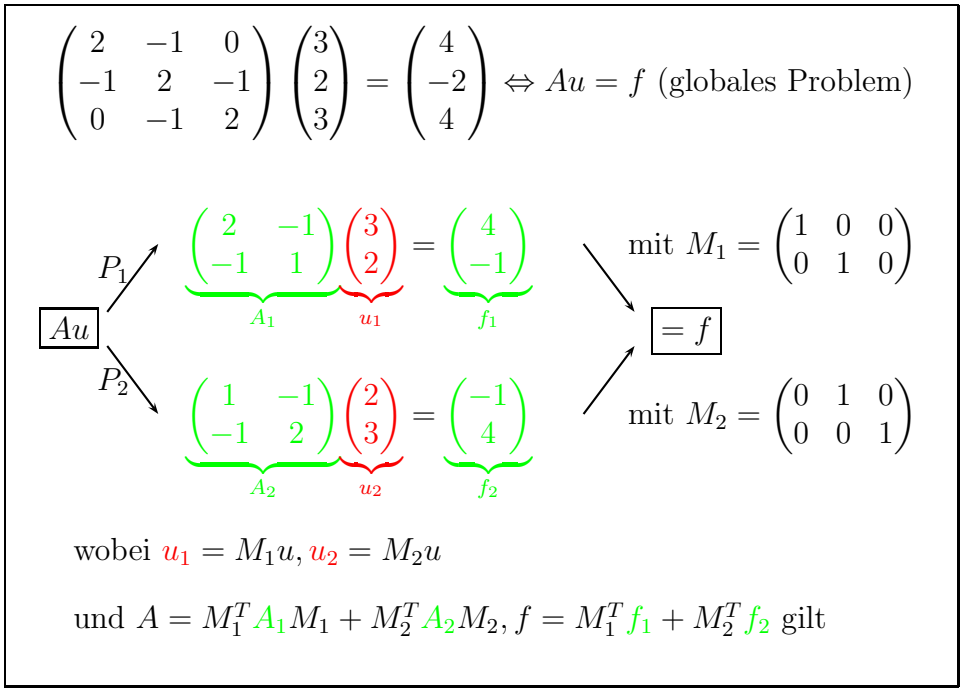


Abbildung 3.3: Beispiel für ein Matrix-Vektor Produkt auf 2 Prozessoren

- globale Kommunikation:
  - Skalarprodukt (Beispiel: $\alpha \Leftarrow \langle \mathfrak{u}, r \rangle = \sum_{i=1}^{P} \langle \mathfrak{u}_i, r_i \rangle$)
- Nachbar Kommunikation:
  - Akkumulation eines verteilten Vektors (Beispiel: $\mathfrak{u} \Leftarrow r = \sum_{i=1}^{P} M_i^T r_i$)
  - Akkumulation einer verteilten Matrix (Beispiel: $\mathfrak{A} \Leftarrow \sum_{i=1}^{P} M_i^T A_i M_i$)

Mit diesen Überlegungen lässt sich bereits eine parallele Variante des konjugierten Gradientenverfahrens in Algorithmus 5 festhalten, welche strukturell der sequentiellen Version (Algorithmus 3) entspricht. Neu ist die erforderliche Kommunikation aufgrund der Vorkonditionierung und des Skalarproduktes.

Bezüglich der klassischen Iterationsverfahren, welche als Glätter in den algebraischen Mehrgitterverfahren zum Einsatz kommen, erhält man beispielsweise für das Jakobiverfahren (2.4) die parallele Version zu

$$\mathfrak{u} = \mathfrak{u} + \omega \mathfrak{D}^{-1} \sum_{i=1}^{P} M_i^T \left(f_i - A_i \mathfrak{u}_i \right), \tag{3.2}$$

sodass ein lokaler Prozess $p \in \{1, \ldots, P\}$ nur die verteilte lokale Systemmatrix A_p benötigt und pro Iteration nur eine Nachbar-Kommunikation, nämlich die Akkumulation des verteilten Residuums r_p, erforderlich ist. Da die lokale akkumulierte Diagonalmatrix $\mathfrak{D}_\mathfrak{p}$ benötigt wird, sollte man diese aus $\mathfrak{D}$ einmalig ermitteln und

Algorithmus 5 parallel PCG($A, f, \mathfrak{u}, \mathtt{C}^{-1}, iter_{max}, \epsilon$)

$r \leftarrow f - A\mathfrak{u}$
$\mathfrak{d} \Leftarrow \mathtt{C}^{-1} r$
$\sigma \Leftarrow \langle \mathfrak{d}, r \rangle$
$\sigma_{start} \leftarrow \sigma_{alt} \leftarrow \sigma$
$iter \leftarrow 0$
while $iter < iter_{max} \wedge \sigma/\sigma_{start} > \epsilon^2$ **do**
  $iter \leftarrow iter + 1$
  $v \leftarrow A\mathfrak{d}$
  $\alpha \Leftarrow \sigma / \langle \mathfrak{d}, v \rangle$
  $\mathfrak{u} \leftarrow \mathfrak{u} + \alpha\mathfrak{d}$
  $r \leftarrow r - \alpha v$
  $\mathfrak{h} \Leftarrow \mathtt{C}^{-1} r$
  $\sigma \Leftarrow \langle \mathfrak{h}, r \rangle$
  $\beta \leftarrow \sigma/\sigma_{alt}$
  $\mathfrak{d} \leftarrow \mathfrak{h} + \beta\mathfrak{d}$
  $\sigma_{alt} \leftarrow \sigma$
end while
return $\mathfrak{u}$

abspeichern, da die Akkumulation der Diagonalen aus A_p für $\mathfrak{D}_\mathfrak{p}$ eine weitere Kommunikation erfordert.
Der parallele Mehrgitteralgorithmus 6 hat die gleiche Struktur wie die entsprechende sequentielle Variante. Darin benötigen die Matrix-Vektor Multiplikationen für die Prolongation und die Restriktion keine Kommunikation. Notwendig wird eine Kommunikation nur bei der Grobgitterlösung und, wie schon zuvor erwähnt, in den Glättungsschritten. Im Gegensatz zum parallelen Mehrgitterverfahren benötigt man für das parallele algebraische Mehrgitterverfahren noch zusätzlich eine parallele Setupphase, welche in [10] detailiert beschrieben wird.

Bemerkung 3.1 *Die Transformation des Vektortypes von verteilt zu akkumuliert und das Skalarprodukt inkludieren Kommunikationen und werden daher in den Algorithmen mit dem $\Leftarrow$ Pfeil hervorgehoben. Alle anderen Zuweisungen, welche keine Kommunikation erfordern, und damit nur lokal auf dem jeweiligen Prozessoren durchgeführt werden, sind mit $\leftarrow$ gekennzeichnet.*

3.2.2 Quellcode - Umsetzung

Um einen Überblick für die Umsetzung des PCG-AMG Verfahrens mithilfe der Parallel Toolbox [11] zu bekommen, wird im Folgenden die übliche Vorgehensweise anhand der strukturellen Quellcodeauflistung 3.1 skizziert:

Für die Benutzung dieser C++ Template Klassenbibliothek muss lediglich die Header-Datei `toolbox.h` (Zeile 1) inkludiert werden - ohne zusätzliche Kompilierungen oder Konfigurationen.

Algorithmus 6 parallel MG(l)

$f_0 \leftarrow f$
if $\ell < L$ **then**
  $u_\ell \leftarrow$ `presmooth`(K_ℓ, f_ℓ) {Vorglättung}
  $r_\ell \leftarrow f_\ell - K_\ell u_\ell$ {Residuumberechnung}
  $f_{\ell+1} \leftarrow$ `restrict`(r_ℓ) {Restriktion des Residuums}
  $u_{\ell+1} \leftarrow 0$
  `parallel MG`$(\ell + 1)$ {Mehrgitterrekursion}
  $s_\ell \leftarrow$ `prolongate`$(u_{\ell+1})$ {Prolongation der Änderung}
  $u_\ell \leftarrow u_\ell + s_\ell$ {Korrektur anwenden}
  $u_\ell \leftarrow$ `postsmooth`(K_ℓ, f_ℓ) {Nachglättung}
else
  $u_L \Leftarrow$ `direct-solve`(K_L, f_L) {Grobgitterlöser}
end if
$u \leftarrow u_0$
return u

Da die Parallelisierung mittels MPI realisiert wird, muss die MPI Bibliothek in Zeile 4 initialisiert werden, wobei die Übergabeargumente den Kommandozeilen-Parametern des Programms entsprechen. Nach Abschluss aller MPI-Operationen, meldet der Befehl in Zeile 61 den Prozess beim MPI-Laufzeitsystem wieder ab. In der Zeile 6 wird die Anzahl der Prozesse und in Zeile 7 die lokale eigene Nummer ermittelt. Diese und alle übrigen Befehle, im Zusammenhang mit der Kommunikation zwischen den Prozessen, werden in der Toolbox ähnlich den originalen MPI-Funktionen bezeichnet und es besteht ein äquivalenter Zusammenhang. Für eine detailierte Einführung in MPI siehe [14].
In den Zeilen 9 bis 17 werden einige wichtige Parameter initialisiert, auf welche kurz näher eingegangen wird.

- `epsilon`: Parameter für die Vergröberung in der AMG-Setupphase; siehe (2.11)
- `omega`: Relaxationsparameter des Glättungsverfahrens (Jacobiverfahren (2.4))
- `max_level`: maximale Anzahl von unterschiedlichen Diskretisierungsleveln; siehe Kapitel 2.3
- `min_nodes`: minimalste Anzahl von Knotenpunkten in der AMG Vergröberung (für 3D-Probleme beachte Kapitel 2.3.4)
- `gauss_seidel`: Jakobi oder Gauss-Seidel-Jakobi Glätter für das Mehrgitterverfahren; siehe [10] für Details
- `cascadic`: ermöglicht eine Alternative zum klassischen Mehrgitterverfahren zu verwenden: das Cascadische Mehrgitterverfahren; siehe [10]
- `error`: Relativer Fehler für das konjugierte Gradientenverfahren; damit ergibt sich das Abbruchskriterium

$$\frac{\sigma}{\sigma_{start}} < \text{error},$$

wobei σ_{start} die Norm des Anfangsresiduums und σ die Norm des Residuums nach den erforderlichen Iterationen bezeichnet; siehe Algorithmus 3

- `max_iterations`: maximale Anzahl von Iterationen für das konjugierte Gradientenverfahren
- `show`: das konjugierte Gradientenverfahren gibt in jeder Iteration das aktuelle und das relative Residuum aus

Listing 3.1: mainCPU.cpp

```
1  #include "../toolbox/toolbox.h"
2
3  int main(int argc, char *argv[]) {
4      network::init(argc, argv);  //Initialize MPI
5      int size, rank;
6      network::size(size);        //Number of processors
7      network::rank(rank);        //Rank of processor
8
9      double epsilon = 0.0;    //AMG parameter epsilon
10     double omega = 0.6;      //Omega for Jacobi smoother
11     int max_level = 16;      //Maximum number of levels in the multigrid hierarchy
12     int min_nodes = 3;       //Minimum number of nodes in AMG coarsening
13     bool gauss_seidel = false; //Enable Gauss-Seidel-Jacobi smoother
14     bool cascadic = false; //Enable cascadic multigrid algorithm
15     double error = 1e-6; //Relative error for CG algorithm
16     int max_iterations = 64; //Maximum iterations for CG algorithm
17     bool show = true; //Enable text output in CG algorithm
18
19     string root("../Data/"); //Location of the simulation data
20     vector<int> hdr;
21     vector<int> par;
22     vector<int> nod;
23     vector<double> ele;
24     read_header(root, hdr);                  //Read header information
25     read_partition(root, hdr, par);          //Read mesh partition
26     read_connection(root, hdr, par, nod);    //Read mesh connectivity
27     read_element(root, hdr, par, ele);       //Read element matrices
28     int gnumnode = hdr[4];  //global number of geometric nodes
29
30     vector<int> row;
31     vector<int> col;
32     element_accumulation(hdr, nod, row, col, ele); //Accumulate the system matrix
33     vector<int> cnt(nod.size(), 0);
34     for(int i = 0; i < (int)row.size(); i++) cnt[row[i]]++;//Initialize the count vector
35
36     communicator<int, double> com(nod); //Create the communicator object
37
38     vector<double> b(gnumnode, 1.0);
39     vector<double> x(gnumnode, 0.0);
40     binary_read(b.begin(), b.end(), root + "VectorB.bin", 0); //Read the right hand side
41     binary_read(x.begin(), x.end(), root + "VectorX.bin", 0); //Read intial solution vector
42     vector<double> _x(nod.size());
43     vector<double> _b(nod.size());
44     for(int i = 0; i < (int)nod.size(); i++) _x[i] = x[nod[i]];
45     for(int i = 0; i < (int)nod.size(); i++) _b[i] = b[nod[i]];
46     com.distribute(_b);
47
48     linear_operator<int, double> lin(cnt, col, ele); //Create the system matrix object
49     vector_product<int, double> sca(com); //Create the scalar product object
50
51     amg_solver<int, double> amg(nod, cnt, col, ele, //Create the AMG solver object
52     max_level, min_nodes, epsilon, omega, gauss_seidel, cascadic);
53
54     conjugate_gradient<int, double> cg(lin, amg, sca, //Create the CG solver object
55     error, max_iterations, show);
56
57     cg(_b, _x); //Solve the finite element system
58
59     binary_write(_x.begin(), _x.end(), root + "x.bin"); //store solution
60
61     network::finalize(); //Finalize MPI
62     return 0;
63 }
```

Nachdem in Zeile 19 der Ort der Binär-Dateien, welche für die Finite Elemente Berechnung benötigt werden, angegeben wird, dienen die Zeilen 24 bis 27 dem Einlesen der Informationen für die Systemmatrix. Auf das spezielle zugrundeliegende Datenformat wird in [10] genauer eingegangen. In Zeile 32 wird die lokale verteilte Systemmatrix für die jeweiligen Prozesse aufgestellt. Für das CRS-Datenformat

(siehe Kapitel 3.1) wird in Zeile 34 aus dem Vektor der Zeilenindizes `row` der speichereffizientere Vektor `cnt`, welcher die Anzahl der Nicht-Null-Einträge pro Zeile repräsentiert.
Das Kommunikator-Objekt, welches in Zeile 36 initialisiert wird, dient zur Parallelisierung und beinhaltet die notwendigen Informationen über die Knoten, welche sich unterschiedliche Prozesse teilen. Das vorherige Kapitel liefert eine detailierte Beschreibung davon. Für sequentielle Berechnungen wird der Kommunikator zwar erstellt hat aber keinerlei Funktion.
Die Zeilen 38 bis 46 dienen der Initialisierung der rechten Seite des Gleichungssystems und der Anfangsschätzung für die Lösung. Dabei kann die Zeile 41 auskommentiert werden, falls keine externe Datei mit einer Anfangsschätzung der Lösung vorhanden ist. Nach dem Aufstellen der lokalen Vektoren (Zeilen 44 und 45) ist darauf zu achten, dass die rechte Seite ein verteilter Vektor ist, was in Zeile 46 erreicht wird.
In den Zeilen von 48 bis 54 werden die benötigten Operatoren für den PCG-AMG Löser aufgebaut. Dabei ist Zeile 51 am rechenaufwendigsten, da das Aufstellen des AMG-Löser Objektes die notwendige Setupphase inkludiert.
In Zeile 57 wird der vorkonditionierte konjugierte Gradientenalgorithmus gestartet, wobei das AMG-Verfahren als Vorkonditionierung übergeben wird. Schließlich wird die erhaltene Lösung in Zeile 59 in die Datei `x.bin` geschrieben.

Parallelisierung auf GPUs

Grafikprozessoren (GPUs) sind Koprozessoren, welche den Hauptprozessor (CPU) bei seiner Arbeit unterstützen. Dabei handelt es sich hierbei um eine spezielle Art von Koprozessoren, nämlich Streamprozessoren, welche Datenströme verarbeiten können. Ursprünglich waren GPUs auf spezielle Funktionen im Zusammenhang mit graphischen Berechnungen, wie Texture Mapping, Antialiasing oder Rendering, limitiert, wobei das Hauptaugenmerk auf die Beschleunigung von 3D-Spielen gelegt wurde. Für Berechnungen über diesen Aufgabenbereich hinaus, führten GPU-Hersteller wie NVIDIA und AMD/ATI die sogenannten Allzweck-Berechnungen auf Grafikprozessoreinheiten (GPGPUs) ein. Dafür waren signifikante Änderungen an den Prozessorkernen von Nöten, um den wahlfreien/direkten Zugriff auf den Speicher (RAM) zu ermöglichen. Mit dieser Neuerung kann eine GPU als eine Shared Memory Maschine mit zahlreichen Prozessorkernen oder als Mehrkernprozessor interpretiert werden. Ein wichtiger Vorteil von GPU zu CPU Verwendung ist die hohe Rechenleistung, wobei die Geschwindigkeit hauptsächlich durch den hohen Grad an Parallelität der Rechenoperationen des Grafikprozessors erreicht wird. Für die Entwicklung GPGPU-fähiger Programme stellt NVIDIA CUDA zur Verfügung; siehe [5]. Das CUDA Toolkit beinhaltet C/C++ Kompiler und Laufzeitbibliotheken, wodurch Standard C/C++ Code, welcher auf einer GPU läuft, kompiliert werden kann. CUDA Programmierung basiert auf dem Konzept, dass man ein Programm auf dem Host laufen lässt während rechenintensive Funktionen auf den GPUs abgearbeitet werden. Damit kann ein C/C++ Code mithilfe des CUDA Toolkits Schritt für Schritt auf Berechnung-Kernels, welche auf der GPU laufen, ausgelagert werden.

Die Toolbox-Bibliotheken beinhalten bereits Routinen für die GPU Implementation mittels CUDA. Daher stehen bereits Funktionen zur Verfügung, welche für die Initialisierung und die Kopierung von Daten auf der GPU verantwortlich sind. Die Kernels für die benötigten Matrix- und Vektoroperationen sind ebenfalls bereits inkludiert. Im Folgenden wird in Listing 3.2 die entscheidenden Änderungen im Code skizziert und erläutert:

Listing 3.2: mainGPU.cpp

```
1  //...
2  device_vector<double> dev_x(_x); //GPU Memory Allocation and Copy Data to GPU
3  device_vector<double> dev_b(_b); //-''-
4
5  device_vector<int> dev_cnt;      //Declaration of the vectors used for ICRS
6  device_vector<int> dev_dsp;      //
7  device_vector<int> dev_col;      //
8  device_vector<double> dev_ele;   //
9
10 device_matrix_conversion<int, double> cvr; //Matrix format conversion CRS->ICRS
11 cvr(cnt, col, ele, dev_cnt, dev_dsp, dev_col, dev_ele); //-''-
12
13 device_communicator<int, double> dev_com(nod); //Create the communicator object
14
15 device_linear_operator<int, double> lin(dev_cnt, //Create the system matrix object
16 dev_dsp, dev_col, dev_ele);
17
18 device_vector_product<int, double> sca(dev_com); //Create the scalar product object
19
20 device_amg_solver<int, double> amg(nod, cnt, col, ele, //Create the AMG solver object
21 max_level, min_nodes, epsilon, omega, gauss_seidel, cascadic);
22
23 device_conjugate_gradient<int, double> cg(lin, amg, sca, //Create the CG solver object
24 error, max_iterations, show);
25
26 cg(dev_b, dev_x); //Solve the problem on the GPU
```

Zeile 1 soll daran erinnern, dass die Quellcodezeilenauflistung 3.2 nur die wesentlichen Änderungen für die gewünschte Verwendung der GPU illustriert und nicht die in dem Beispielcode 3.1 erforderlichen Vorgehensweisen ersetzt. Um das Verfahren mithilfe der GPU durchzuführen, müssen die Device-Vektoren (Speicherinitialisierung und/oder Speichertransfer) erstellt, und in weiterer Folge die Device-Funktionen verwendet werden. In den Zeilen 2 und 3 wird der benötigte Speicher auf der GPU allokiert um die entsprechenden Daten (hier die rechte Seite und die Anfangslösung) auf die GPU zu kopieren. Nach der Deklaration der benötigten Vektoren für das ICRS-Speicherformat in den Zeilen 5 bis 8, wird die lokale Systemmatrix vom CRS in das ICRS Format für GPU-Berechnungen in den Zeilen 10 und 11 konvertiert; siehe Kapitel 3.1. Die restliche Vorgangsweise verläuft analog zum CPU-Code, es müssen lediglich die `device_`-Varianten der Objekte erstellt werden, um dann das Problem auf der GPU zu lösen (Zeile 26).

3.3 Simulationen

Die Daten für das lineare Gleichungssystem stammen aus einer Finiten Elemente Disktretisierung eines 3D Elastizitätsproblems mit 9261 Knotenpunkten. Dabei handelt es sich um einen Würfel, der an einer Seite fixiert ist und auf dessen gegenüberliegende Seite eine Kraft einwirkt; siehe Abbildung 3.4. In Tabelle 3.2 werden die charakteristischen Eigenschaften der resultierenden Gleichungssysteme angegeben.

Für die ersten Analysen betrachten wir nur einen zufällig gewählten Teilschritt,

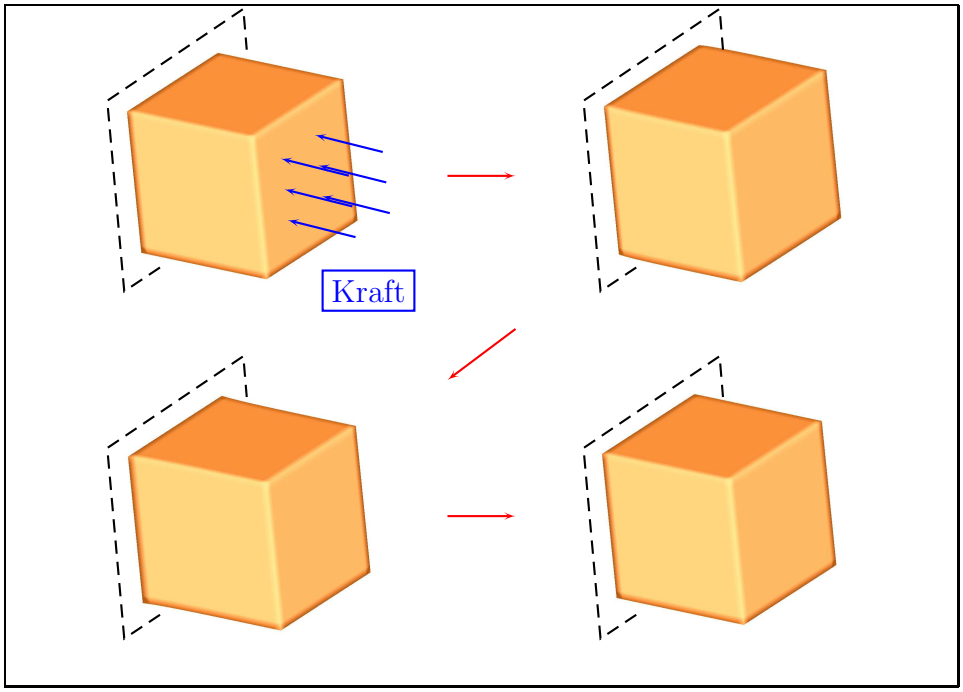


Abbildung 3.4: Testbeispiel: Würfel, der an einer Seite fixiert ist und auf dessen gegenüberliegende Seite eine Kraft einwirkt

Dimensionsdiskretisierung	21×21×21
Gesamtknotenanzahl	9261
Unbekannte (für 3D-Probleme 3× Knotenanzahl)	27783
Nicht-Null-Einträge in der Systemmatrix	≈ 1000000

Tabelle 3.2: Charakteristische Daten für das 3D-Elastizitätsproblem Würfel

der in Abbildung 3.4 illustrierten Deformation des Würfels, für welchen die Systemmatrix 895028 Nicht-Null-Einträge besitzt. Die Tests hierfür werden auf einem System mit nachstehenden Eigenschaften durchgeführt:

- CPU: Intel(R) Xeon 3.20 GHz (Irwindale)
  RAM: 2.0 GB

- GPU: NVIDIA GeForce GTX 550 Ti

Bevor auf das gewünschte PCG-AMG Verfahren näher eingegangen wird, wird fürs Erste nur das CG-Verfahren untersucht. Die Tabelle 3.3 zeigt Ergebnisse für unterschiedliche relative Fehler für das konjugierte Gradienten Verfahren ohne Vorkonditionierung, umgesetzt mittels Matlab (PCG-Funktion) und der Toolbox, wobei für die letztere Möglichkeit einerseits die CPU und andererseits die GPU Variante angeführt wird.

Lösungsvariante	Fehler	Iterationen	Zeit[s]	Beschleunigung
Matlab(PCG)	$1e-05$	177	4.3	
CPU Toolbox CG	$1e-05$	-	0.68	6.3
GPU Toolbox CG	$1e-05$	-	0.09	7.6
Matlab(PCG)	$1e-06$	202	4.8	
CPU Toolbox CG	$1e-06$	-	0.77	6.2
GPU Toolbox CG	$1e-06$	-	0.1	7.7
Matlab(PCG)	$1e-07$	219	5.3	
CPU Toolbox CG	$1e-07$	-	0.85	6.2
GPU Toolbox CG	$1e-07$	-	0.11	7.7
Matlab(PCG)	$1e-08$	238	5.7	
CPU Toolbox CG	$1e-08$	-	0.94	6.1
GPU Toolbox CG	$1e-08$	-	0.12	7.8

Tabelle 3.3: Würfelproblem gelöst mittels CG-Verfahren für unterschiedliche relative Fehler

Für die restlichen Auswertungen verwenden wir die Block-Systemmatrix mit 967851 Einträgen (alle Nicht-Null Blöcke). Diese beinhaltet auch Null-Einträge, da jeder Knotenpunkt mit 3×3 Blöcken in der Systemmatrix zusammenhängt, welche gegebenenfalls mit Nullen aufgefüllt werden; siehe Kapitel 2.3.4. Tabelle 3.4 zeigt die unterschiedlichen Verfahren ausgeführt auf der CPU. Dabei zeigt sich die Notwendigkeit der Blocküberlegungen bezüglich des AMG-Verfahrens für Systeme deutlich.

Verfahren (CPU)	Fehler	Iterationen	Setup[s]	Zeit[s]	Zeit/Iteration[s]
CG	$1e-05$	177	-	0.72	0.004
CG	$1e-06$	202	-	0.83	0.004
CG	$1e-07$	219	-	0.89	0.004
CG	$1e-08$	238	-	0.98	0.004
skalar PCG-AMG	$1e-05$	84	0.16	1.22	0.015
skalar PCG-AMG	$1e-06$	94	0.16	1.37	0.015
skalar PCG-AMG	$1e-07$	102	0.16	1.49	0.015
skalar PCG-AMG	$1e-08$	109	0.16	1.60	0.015
Block PCG-AMG	$1e-05$	16	1.11	0.35	0.022
Block PCG-AMG	$1e-06$	18	1.11	0.40	0.022
Block PCG-AMG	$1e-07$	20	1.11	0.45	0.022
Block PCG-AMG	$1e-08$	22	1.11	0.48	0.022

Tabelle 3.4: Würfelproblem (mit der Blocksystemmatrix) gelöst mittels unterschiedlicher Verfahren für unterschiedliche relative Fehler auf der CPU

Im Gegensatz zum Lösen eines einzelnen Systems (Tabellen 3.3 und 3.4) betrachten wir nun das Verhalten des Lösers im Rahmen zeitabhängiger Berechnungen im

Paket CARP[1] für dasselbe Beispiel.
Abschließend soll die Parallelisierung auf mehreren Prozessen analysiert werden. Dafür verwenden wir folgendes System (Mephisto GPU Cluster):

- 10 Intel Xeon X5650 CPUs, 60 Kerne mit 2.67 GHz
- 20 NVIDIA Tesla C2070 GPUs
- 480 GB CPU RAM, 120 GB GPU RAM

Die Tabellen 3.5 und 3.6 zeigen numerische Ergebnisse für unterschiedliche Anzahlen von Prozessen auf der CPU als auch mit GPU-Unterstützung. Der Speedup pro Iteration ist für 8 CPUs nicht ideal da die betrachtete Problemgröße relativ klein ist, wodurch sich die wachsende Kommunikation gegenüber den verringerten lokalen Berechnungen bereits bemerkbar macht. Dieses Verhältnis verschlechtert sich bei den GPUs noch weiter, da die Berechnungen 10 mal schneller ablaufen, jedoch die Kommunikation im besten Falle gleich schnell bleibt.

♯ Prozesse	abs. Fehler	Iterationen	Zeit[s]	Zeit/Iteration[s]	Speedup pro Iteration
1	$1e-06$	83	1.3402	0.01620	1
2	-	102	0.7886	0.00773	2.1
4	-	108	0.4442	0.00411	3.9
8	-	110	0.2926	0.00266	6.1

Tabelle 3.5: Würfelproblem für unterschiedliche Anzahl von Prozessen (CPU) für das Block PCG-AMG Verfahren (10^{-6} abs. Fehler $\approx 10^{-14}$ rel. Fehler; Speedup=Zeit (1 Prozess)/ Zeit (n Prozesse))

♯ Prozesse	abs. Fehler	Iterationen	Zeit[s]	Zeit/Iteration[s]	Speedup pro Iteration
1	$1e-06$	83	0.1259	0.00152	1
2	-	102	0.1279	0.00125	1.2
4	-	108	0.1340	0.00124	1.2

Tabelle 3.6: Würfelproblem für unterschiedliche Anzahl von Prozessen (GPU) für das Block PCG-AMG Verfahren (10^{-6} abs. Fehler $\approx 10^{-14}$ rel. Fehler; Speedup=Zeit (1 Prozess)/ Zeit (n Prozesse))

3.4 Zusammenfassung und Ausblicke

Die numerischen Auswertungen haben gezeigt, dass sich skalare AMG-Verfahren für 3D-Elastizitätsprobleme als ungeeignet herausstellen. Man muss sich von den klassischen variablen-basierenden Ansätzen trennen, und auf Erweiterungen zurückgreifen,

[1] http://carp.meduni-graz.at/

welche physikalische Unbekannte koppeln. Weiters erweist sich der Einsatz von Parallelisierungskonzepten in Verbindung mit GPU-Unterstützung als sehr hilfreich für eine effektive Umsetzung des Verfahrens.
Für zukünftige Analysen wäre es interessant das Verfahren auf alternative (nichtlineare) Problemstellungen mit variierten Problemgrößen und komplizierteren Geometrien anzuwenden, und die resultierenden Konvergenzverhalten zu vergleichen. Im Zusammenhang mit AMG Überlegungen könnte man einerseits in der AMG-Setupphase alternative Varianten bezüglich verwendeter Interpolationsschemata und der damit verbundenen Parameter und Normen testen, als auch andererseits unterschiedliche Block-Glättungsverfahren einsetzen.

Literaturverzeichnis

[1] Katharina Becker-Steinberger. Freie Materialoptimierung - Modellierung und Optimierung elastischer Materialien. Master's thesis, Universität Hamburg, 2007.

[2] D. Braess. *Finite Elemente: Theorie, schnelle Löser und Anwendungen in der Elastizitätstheorie*. Springer-Verlag Berlin Heidelberg, 3. edition, 2003.

[3] W. L. Briggs, V. E. Henson, and S. F. McCormick. *A Multigrid Tutorial*. SIAM Books, Philadelphia, 2000. Second edition.

[4] Tanja Cless. *AMG Strategies for PDE Systems with Applications in Industrial Semiconductor Simulation*. PhD thesis, Universität zu Köln, 2005.

[5] CUDA. Nvidia cuda. Website, 2012. Available online at `http://www.nvidia.com/cuda/`.

[6] Axel Feldmann. Nichtlineare Elastizität inkompressibler Materialien. Master's thesis, Westfälische Wilhelms-Universität Münster, Institut für Numerische und instrumentelle Mathematik, 2000.

[7] Michael Griebel, Daniel Oeltz, and Marc Alexander Schweitzer. An algebraic multigrid method for linear elasticity. *SIAM J. Sci. Comp*, 25:407, 2003.

[8] Wolfgang Hackbusch. *Multi-Grid Methods and Applications*. Springer-Verlag Berlin Heidelberg, 1985.

[9] Ulrich Langer. Numerische Festkörpermechanik (computational mechanics). Technical report, Johannes Kepler Universität Linz, 1997.

[10] Manfred Liebmann. *Efficient PDE Solvers on Modern Hardware with Applications in Medical and Technical Sciences*. PhD thesis, Karl-Franzens-Universität Graz, 2009.

[11] Manfred Liebmann. Parallel toolbox home page. Website, 2009. Available online at `http://paralleltoolbox.sourceforge.net/`.

[12] J. W. Ruge and K. Stüben. Algebraic multigrid (AMG). In S.F. McCormick, editor, *Multigrid Methods*, volume 3 of Frontiers in Applied Mathematics, pages 73–130. SIAM, Philadelphia, 1987.

[13] Gundolf Haase und Ulrich Langer. MULTIGRID - METHODEN. Technical report, Johannes Kepler Universität Linz, 2004.

[14] E. Lusk W. Gropp and A. Skjellum. *Using MPI: Portable Parallel Programming with the Message Passing Interface*. The MIT Press, Cambridge, 1999.