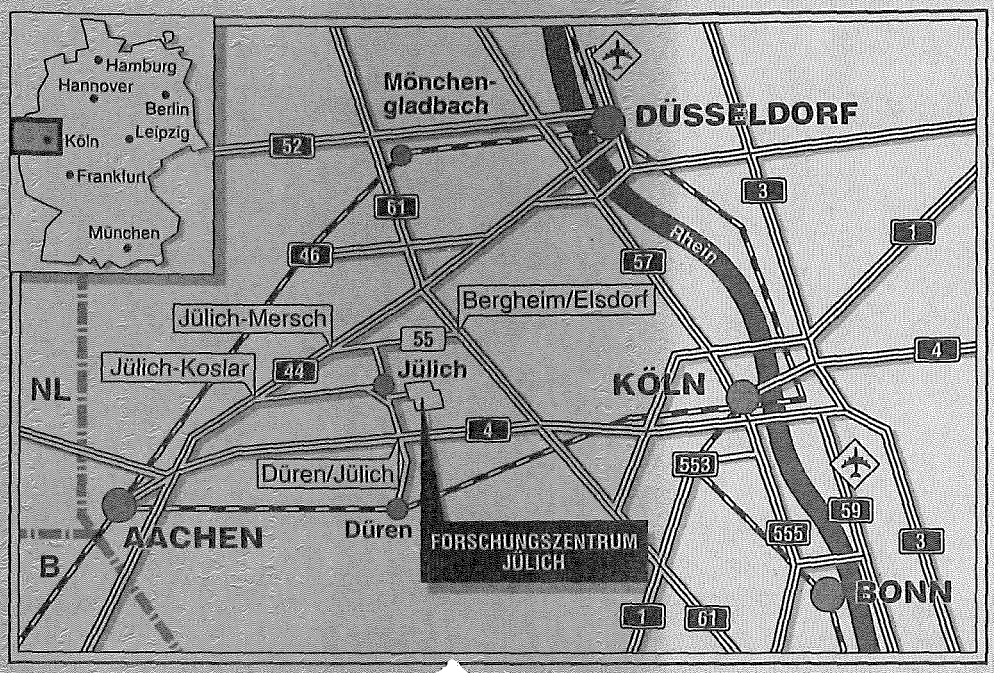
FORSCHUNGSZENTRUM JÜLICH GmbH

Zentralinstitut für Angewandte Mathematik

Vergleich von parallelen Verfahren zur Vorkonditionierung für die Methode der konjugierten Gradienten

Christof Schelthoff



Berichte des Forschungszentrums Jülich ; 2913
ISSN 0944-2952
Zentralinstitut für Angewandte Mathematik Jül-2913

Zu beziehen durch: Forschungszentrum Jülich GmbH · Zentralbibliothek
D-52425 Jülich · Bundesrepublik Deutschland
Telefon: 02461/61-6102 · Telefax: 02461/61-6103 · Telex: 833556-70 kfa d

Vergleich von parallelen Verfahren zur Vorkonditionierung für die Methode der konjugierten Gradienten

Christof Schelthoff

Kurzfassung

Im Rahmen dieser Arbeit werden Methoden der Vorkonditionierung für die Methode der konjugierten Gradienten zur Lösung großer dünn besetzter linearer Gleichungssysteme mit symmetrischer und positiv definiter Koeffizientenmatrix vorgestellt. Mit dem Einsatz massiv-paralleler Systeme stellt sich die Frage nach gut parallelisierbaren Vorkonditionierungen. In diesem Zuammenhang erweist sich die polynomiale Vorkonditionierung als gut geeignet. Das Verfahren erhöht die Anzahl der Matrix-Vektor-Produkte, während die Gesamtzahl der Iterationsschritte und somit die Anzahl der Skalarprodukte reduziert wird. Die parallele Berechnung der Matrix-Vektor-Produkte führt gewöhnlich zu Kommunikation mit wenigen Prozessoren; Skalarprodukte erfordern hingegen globale Synchronisation. Weiterhin verringern sich durch die Reduktion der Anzahl der Iterationen die im Verfahren der konjugierten Gradienten durchzuführenden Orthogonalisierungen; dieses hat einen positiven Einfluß auf die Stabilität des Verfahrens. Im praktischen Teil dieser Arbeit wird die polynomiale Vorkonditonierung anhand einer Implementierung mit Tschebyscheff-Polynomen auf zwei massiv-parallelen Systemen der KFA Jülich untersucht.

Abstract

This report presents preconditioning techniques for the conjugate gradient method (CG), an iterative method for the solution of large sparse symmetric positive definite systems. The availability of massively parallel processors draws attention to parallel preconditioners. In this context, polynomial preconditioning is shown to be suitable. The preconditioner increases the number of matrix-vector products, whereas the total number of iterations and therefore the number of dot products decreases. The parallel computation of matrix-vector products usually results in communication with a small number of processors. Dot products, however, require global synchronisation. This reduction also results in better stability of the CG-method because fewer orthogonalizations are executed. The implementation is based on Chebyshev polynomials; performance tests were carried out on two massively parallel computer systems of the Research Centre Jülich (KFA).

Inhalt

Abbildungsverzeichnis

Kapitel 1

Einleitung

Die mathematische Modellierung unterschiedlichster physikalischer Vorgänge wie der Schadstoffausbreitung im Boden oder der Verformung von Materialien führt auf das Problem der Lösung partieller Differentialgleichungen. Da diese in der Regel nicht geschlossen gelöst werden können, kann ein Diskretisierungsverfahren, beispielsweise die Methode der finiten Elemente, verwendet werden, um eine Näherungslösung zu erhalten.
Hierzu sind Gleichungssysteme zu lösen, welche häufig eine große symmetrische und positiv definite Koeffizientenmatrix mit wenigen von Null verschiedenen Einträgen besitzen. In vielen Fällen werden iterative Verfahren verwendet, da diese einen geringen Speicherbedarf haben. Ein übliches iteratives Verfahren ist die Methode der konjugierten Gradienten (CG-Verfahren). Der wesentliche Aufwand dieses Verfahrens besteht in dem in jedem Iterationsschritt zu berechnenden Matrix-Vektor-Produkt mit der Koeffizientenmatrix. Die Konvergenzgeschwindigkeit dieses Verfahrens hängt von der Kondition der Matrix ab. Wird das Gleichungssystem in ein äquivalentes System mit besserer Kondition transformiert, so ist ein besseres Konvergenzverhalten zu erwarten. Weiterhin bewirkt eine Verbesserung der Kondition auch eine verminderte Anfälligkeit des Verfahrens gegenüber Rundungsfehlern und somit eine erhöhte Stabilität des Verfahrens. Der Aufwand dieser Transformation – der Vorkonditionierung – darf hierbei jedoch nicht zu hoch sein.

Der Wunsch nach mehr Arbeitsspeicher zur Modellierung feinerer Gitter bei der Diskretisierung, erhöhter Genauigkeit und schnelleren Rechenzeiten führt auf den Einsatz massiv-paralleler Rechnersysteme. Bei Systemen mit verteiltem Speicher besteht die Schwierigkeit der Parallelisierung in der Implementierung eines effizienten Matrix-Vektor-Produkts. Hieraus ergibt sich die Fragestellung nach gut parallelisierenden Vorkonditionierungsmethoden. Eine Übertragung sequentiell gut geeigneter Methoden, wie die unvollständige Cholesky-Zerlegung, erweist sich als schwierig und häufig ineffizient, da diese Verfahren in hohem Maße sequentiellen Charakter haben.

Neben der einfach zu implementierenden Skalierung unter Verwendung der Diagonalen (Diagonal Scaling) erweist sich eine andere Klasse von Vorkonditionie-

rungsmethoden als gut parallelisierbar – die polynomiale Vorkonditionierung. Diese Verfahren beruhen im wesentlichen darauf, statt einer Matrix-Vektor-Multiplikation eine Folge von Matrix-Vektor-Produkten mit der Koeffizientenmatrix zu berechnen. Durch die Bereitstellung des Matrix-Vektor-Produkts im zugrundeliegenden CG-Verfahren ist somit weder zusätzlicher Speicherbedarf noch anderer zusätzlicher Aufwand zur Parallelisierung erforderlich.

Die hier vorgestellten Algorithmen sind unabhängig vom verwendeten CG-Verfahren einfach zu implementieren. Ziel dieser Arbeit ist die Entwicklung und Implementierung der polynomialen Vorkonditionierung mit Tschebyscheff-Polynomen und ihre Kombination mit dem Diagonal Scaling als Erweiterung eines parallelen CG-Verfahrens im Forschungszentrum Jülich. Es sollen die wesentlichen bei der Parallelisierung entstehenden Effekte herausgearbeitet werden.
Entscheidend für den Einsatz der polynomialen Vorkonditionierung ist die Frage, ob ein CG-Verfahren mit Vorkonditionierung besser, gleich oder schlechter skaliert als das Verfahren ohne Vorkonditionierung. Weiterhin ist zu untersuchen, ob der Effekt der erhöhten Stabilität des CG-Verfahrens mit polynomialer Vorkonditionierung in praktischen Anwendungen von Bedeutung ist.

Im Rahmen dieser Arbeit wird in Kapitel 2 eine Einleitung zu iterativen Verfahren, insbesondere zum Verfahren der konjugierten Gradienten, gegeben und der Begriff der Kondition vorgestellt. Weiterhin wird der Einfluß der Kondition auf die Konvergenz des Verfahrens der konjugierten Gradienten erläutert. In Kapitel 3 wird die Theorie der Vorkonditionierung beschrieben und auf die polynomiale Vorkonditionierung mit Tschebyscheff-Polynomen ausführlich eingegangen. Kapitel 4 beschreibt die verwendeten Rechnerarchitekturen, die Datenverteilung auf die Prozessoren, das zur Berechnung des Matrix-Vektor-Produkts benötigte Kommunikationsschema und die benutzte Speichertechnik. Dies bildet die Basis für die Implementierung des Diagonal Scaling und der polynomialen Vorkonditionierung mit Tschebyscheff-Polynomen auf zwei massiv-parallelen Systemen.
In der hier vorgestellten Implementierung lassen sich beide Methoden kombinieren.

Kapitel 5 beschreibt die verwendeten Testbeispiele, numerische Ergebnisse und Performance werden in Kapitel 6 behandelt. Schließlich werden im letzten Kapitel die Resultate zusammengefaßt und Ansatzpunkte für Erweiterungen gegeben.

Kapitel 2

Grundlagen

In diesem Kapitel wird zu Beginn eine kurze Übersicht über einige Iterationsverfahren für lineare Gleichungssysteme gegeben. Literatur hierzu findet sich in [10],[13],[28] und [33]. Ausführlich wird dann in Kapitel 2.3 das Verfahren konjugierter Gradienten behandelt, welches sich in [10],[11],[24] und [33] wiederfindet. Außerdem wird der Begriff der Kondition eingeführt und die Bedeutung der Kondition für die Stabilität eines Iterationsverfahrens, sowie für die Konvergenz des Verfahrens konjugierter Gradienten analysiert. Dieses ist entnommen aus [10],[11],[23], [24] und [28].

2.1 Lineare Gleichungssysteme

Im folgenden sind mit großen lateinischen Buchstaben $n \times n$ Matrizen, mit kleinen Vektoren und mit griechischen Buchstaben reelle Zahlen und gelegentlich auch Abbildungen bezeichnet.

Zur Lösung eines linearen Gleichungssystems

$$Ax = v \tag{2.1}$$

mit einer regulären Koeffizientenmatrix A gibt es zwei Klassen von Verfahren. Die **direkten Verfahren** (z.B. Gauß–Algorithmus) liefern nach einer endlichen Anzahl von Schritten die exakte Lösung x^*. Diese Berechnungsschritte lassen sich als Umformungen der Matrix A und des Vektors v darstellen.

Ist die Matrix A dünn besetzt, so führen auf dem Gauß–Verfahren basierende Algorithmen in der Regel zu fill–in, d.h. Nullelemente von A werden zu Nicht-Nullelementen.

Da hier große, unregelmäßig dünnbesetzte Matrizen betrachtet werden, erfordern direkte Verfahren einen sehr hohen Speicherbedarf für die Koeffizientenmatrix bzw. deren Umformungen.

Iterative Verfahren hingegen verwenden nur Matrixelemente ungleich Null und verändern A nicht. Sie haben daher minimalen Speicherbedarf.

Aus diesen Gründen sind für Matrizen der hier betrachteten Besetzungsstrukturen aus Diskretisierungsverfahren für partielle Differentialgleichungen, insbesondere aus Finite-Elemente-Verfahren, die iterativen Verfahren den direkten vorzuziehen. Sie liefern zu einem Startvektor x_0 durch eine Iterationsvorschrift eine Folge von *Iterierten* $x_0 \to x_1 \to x_2 \to \ldots$, die unter geeigneten Voraussetzungen gegen die gesuchte Lösung konvergiert. Iterative Verfahren liefern bei den hier betrachteten Matrizen in der Regel mit weniger Speicherbedarf und weniger Rechenoperationen als direkte Verfahren bereits eine gute Approximation der Lösung.

Für das weitere Vorgehen wird noch folgende Definition benötigt:

Def 2.1 *Eine $n \times n$ -Matrix A heißt* **symmetrisch**, *falls $A^T = A$, und* **positiv definit**, *falls $x^T Ax > 0 \quad \forall\, x \in \mathbb{R}^n \backslash \{0\}$.*

2.1.1 Iterationsverfahren der Form $Bx_{k+1} = Qx_k + v$

In diesem Abschnitt sollen einige iterative Lösungsverfahren kurz vorgestellt werden, deren Kenntnis für das weitere Vorgehen wichtig ist.

Zur Lösung von (2.1) betrachte man das äquivalente System

$$Bx + (A - B)x = v \,. \tag{2.2}$$

Eine mögliche Iterationsvorschrift lautet dann

$$Bx_{k+1} = (B - A)x_k + v \,. \tag{2.3}$$

Hierbei ist B eine beliebige $n \times n$ Matrix.

Ist B invertierbar, so ergibt sich mit $H := I - B^{-1}A$

$$x_{k+1} = Hx_k + B^{-1}v \,. \tag{2.4}$$

Die Klasse der hier betrachteten Verfahren unterscheidet sich in der speziellen Wahl von B.

Ist B leicht zu invertieren, so ist das Verfahren gemäß (2.4) aufzustellen. Ist die Inverse der Matrix B nur mit sehr hohem Aufwand zu bestimmen, so empfiehlt sich Darstellung (2.3), da dort in jedem Iterationsschritt ein Gleichungssystem der Form $Bx_{k+1} = f(x_k)$ zu lösen ist.

2.1.2 Konvergenzeigenschaften

Die folgenden Betrachtungen gelten für beliebige Normen. Es ist jedoch darauf zu achten, daß, falls im gleichen Zusammenhang von Vektor- und Matrixnorm gesprochen wird, diese Normen miteinander verträglich gewählt werden.

Für die exakte Lösung $x^* = A^{-1}v$ gilt

$$x^* = Hx^* + B^{-1}v \; . \tag{2.5}$$

Durch Subtraktion von (2.4) erhält man

$$\begin{aligned} \| x_{k+1} - x^* \| &= \| H(x_k - x^*) \| \\ &\leq \| H \| \cdot \| x_k - x^* \| \\ &\leq \; \ldots \leq \| H \|^{k+1} \cdot \| x_0 - x^* \| \; . \end{aligned} \tag{2.6}$$

Somit ist $\| H \| < 1$ eine hinreichende Bedingung für die Konvergenz des Verfahrens. Insbesondere ist $\| H \|$ als Kontraktionsfaktor ein Maß für die Konvergenzgeschwindigkeit des Verfahrens.

Die Matrix B muß also so gewählt werden, daß $\| H \|$ möglichst klein wird. Insbesondere hängt die Konvergenzgeschwindigkeit von der betrachteten Norm ab.

Wäre $B = A$, so ergäbe dies

$$\| H \| = \| I - B^{-1}A \| = 0$$

und somit wg. (2.6) die Konvergenz in einer Iteration. Dieses Ergebnis ist jedoch nur von theoretischem Interesse, da der Arbeitsaufwand zur Berechnung von B^{-1} dem der Berechnung der exakten Lösung $x^* = A^{-1}v$ entspricht.

Dieser Aspekt ist aber nützlich bei der Wahl von B, denn B wird umso brauchbarer sein, je mehr es folgenden zwei Bedingungen genügt:

a) B ist eine "gute Approximation" von A

b) B ist leicht zu invertieren bzw. x_{k+1} in (2.3) ist leicht zu bestimmen

Als a-priori Abschätzung über den Fehler der Iterierten ergibt sich [30]:

$$\| x_k - x^* \| \leq \frac{\| H \|^k}{1 - \| H \|} \| x_1 - x_0 \| \; . \tag{2.7}$$

und als a-posteriori Abschätzung:

$$\| x_k - x^* \| \leq \frac{\| H \|}{1 - \| H \|} \| x_k - x_{k-1} \| \ . \tag{2.8}$$

Bei Kenntnis von $\| H \|$ läßt sich mit (2.7) nach der ersten Iteration eine Abschätzung angeben für die maximale Anzahl der Iterationen, die man benötigt, um den exakten Wert bis auf einen Fehler ε anzunähern.

$$k_{max} = \left[\frac{\log(\varepsilon) - \log(\| x_1 - x_0 \|) + \log(1 - \| H \|)}{\log(\| H \|)} \right]^* , \tag{2.9}$$

wobei $[\ \cdot\ ]^*$ die obere Gaußklammer, also die nächstgrößere ganze Zahl bezeichnet.

Für die folgenden speziellen Verfahren betrachte man die Zerlegung $A = D - L - R$ mit

$$D = \begin{pmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & \ddots & & \vdots \\ \vdots & & & 0 \\ 0 & \cdots & 0 & a_{nn} \end{pmatrix} ,$$

$$L = -\begin{pmatrix} 0 & \cdots & \cdots & 0 \\ a_{21} & \ddots & & \vdots \\ \vdots & \ddots & & \vdots \\ a_{n1} & \cdots & a_{n,n-1} & 0 \end{pmatrix} ,$$

$$R = -\begin{pmatrix} 0 & a_{12} & \cdots & a_{1n} \\ \vdots & \ddots & \ddots & \vdots \\ \vdots & & & a_{n-1,n} \\ 0 & \cdots & 0 & 0 \end{pmatrix} .$$

2.1.3 Das Gesamtschrittverfahren (GSV)

In (2.4) sei $B := D$.

Somit ergibt sich

$$\begin{aligned} x_{k+1} &= (I - D^{-1}A)x_k + D^{-1}v \\ &= D^{-1}(L + R)x_k + D^{-1}v \end{aligned} \tag{2.10}$$

als Iterationsverfahren, welches als **Gesamtschritt–** oder **Jacobi–Verfahren** bekannt ist. Für die Existenz von D^{-1} ist es notwendig und hinreichend, daß alle Diagonalelemente von A ungleich Null sind.

2.1.4 Das Einzelschrittverfahren (ESV)

Nun sei $B := D - L$. Die Inverse ist in diesem Fall nicht so leicht zu berechnen wie im Gesamtschrittverfahren.

Mit (2.3) ergibt sich

$$\begin{aligned} (D-L)x_{k+1} - Rx_k &= v \\ Dx_{k+1} &= Lx_{k+1} + Rx_k + v \\ x_{k+1} &= D^{-1}Lx_{k+1} + D^{-1}Rx_k + D^{-1}v \,, \end{aligned} \tag{2.11}$$

wobei diese Iteration wohldefiniert ist, da in Lx_{k+1} wegen der Dreiecksgestalt von L nur bereits berechnete Komponenten von x_{k+1} benutzt werden. Dieses Verfahren ist als **Einzelschritt**– oder **Gauß–Seidel–Verfahren** bekannt. Die Invertierung von $D - L$ ist wieder genau dann möglich, falls A nur Diagonalelemente ungleich Null hat.

2.1.5 Relaxationsverfahren

Die Idee der Relaxationsverfahren liegt darin, statt einer Matrix H die einparametrige Klasse

$$H(\omega) := I - (B(\omega))^{-1}A \tag{2.12}$$

zu betrachten.

Hierbei ist ω so zu wählen, daß $\| H \|$ möglichst klein wird.

Die Bestimmung des optimalen Parameters ω hängt von der Matrix A ab (jedoch nicht von der rechten Seite v) und ist i.a. recht schwierig. Ein Beispiel für die Komplexität der Bestimmung ist in [1] zu finden.

Das bekannteste Relaxationsverfahren ist das **SOR-Verfahren** (**S**uccessive **O**ver **R**elaxation). Bei dem SOR-Verfahren wird

$$B(\omega) := \frac{1}{\omega}D - L$$

gesetzt. Insbesondere ergibt sich für $\omega = 1$ das Einzelschrittverfahren.

Als Iterationsvorschrift erhält man gemäß (2.3)

$$x_{k+1} = \omega D^{-1}Lx_{k+1} + \omega D^{-1}Rx_k + (1-\omega)x_k + \omega D^{-1}v \,. \tag{2.13}$$

Der Bereich, aus dem dieses ω zu wählen ist, ist recht leicht zu verifizieren, denn für den Spektralradius

$$\rho(H) := \max_{\lambda_i \text{ EW von H}} |\lambda_i|$$

gilt:

Satz 2.1 (Kahan) *Für das SOR-Verfahren gilt:*

$$\rho(H(\omega)) \geq |\omega - 1| \ ,$$

wobei das Gleichheitszeichen genau dann gilt, wenn alle Eigenwerte von H auf einem Kreis um $(0,0)$ mit Radius $|\omega - 1|$ liegen.

Beweis: [29].

Andererseits gilt:

Satz 2.2 (Ostrowski, Reich) *Ist A symmetrisch und positiv definit, $\omega \in (0,2)$, dann ist*

$$\rho(H(\omega)) < 1 \ .$$

Beweis: [29].

Weiterhin gilt für beliebige Normen bezüglich eines beliebigen Eigenwertes λ und zugehörigem Eigenvektor x von H:

$$| \lambda | \cdot \| x \| = \| \lambda x \| = \| Hx \| \leq \| H \| \cdot \| x \| \ ,$$

und somit $\| H \| \geq | \lambda |$ für alle Eigenwerte. Dies bedeutet $\| H \| \geq \rho(H)$.

Hieraus folgt, daß für ein beliebiges $\omega \in (0,2)$ das zugehörige SOR-Verfahren konvergiert, falls A symmetrisch und positiv definit ist.

$\omega > 1$ wird als *Überrelaxation* bezeichnet, $\omega < 1$ als *Unterrelaxation.*

Für symmetrische Matrizen gibt es ein verbessertes Verfahren, das die Symmetrieeigenschaft ausnützt, das **SSOR-Verfahren** (Symmetric **SOR**).

Dieses entsteht mit

$$B(\omega) := \frac{1}{\omega(2-\omega)} D(I - \omega D^{-1} L)(I - \omega D^{-1} L^T) \ .$$

Auch dieses Verfahren konvergiert bei symmetrischen und positiv definiten Matrizen stets, falls $\omega \in (0,2)$[18].

2.2 Kondition einer Matrix

In diesem Abschnitt wird der Einfluß von kleinen Änderungen in den Eingangsdaten auf die Lösung diskutiert.

Ändert sich die rechte Seite von (2.1) v zu $\hat{v}$, so gilt für die exakte Lösung des zu lösenden Gleichungssystems $A\hat{x} = \hat{v}$:

$$\hat{x} - x = A^{-1}(\hat{v} - v)\,, \tag{2.14}$$

$$\| v \| = \| Ax \| \leq \| A \| \cdot \| x \| \tag{2.15}$$

$$\Longrightarrow \quad \| x \| \geq \frac{\| v \|}{\| A \|}$$

und mit $\Delta x := \hat{x} - x, \Delta v := \hat{v} - v$

$$\frac{\| \Delta x \|}{\| x \|} \leq \frac{\| A^{-1} \| \cdot \| \hat{v} - v \|}{\| x \|} \leq \underbrace{\| A^{-1} \| \cdot \| A \|}_{\kappa(A)} \cdot \frac{\| \Delta v \|}{\| v \|}\,. \tag{2.16}$$

Der relative Fehler der Eingangsdaten wird also im Lösungsvektor höchstens um den Faktor $\kappa(A)$ verstärkt.

Def 2.2 *Ist A regulär, so heißt $\kappa(A) := \| A^{-1} \| \cdot \| A \|$* **Kondition von A** *(bzgl. der Norm $\| \cdot \|$).*

Für die Kondition gelten folgende Eigenschaften:

$$\kappa(A^{-1}) = \kappa(A) \geq 1\,. \tag{2.17}$$

$$\kappa(\gamma A) = \kappa(A) \quad \forall\, \gamma \in \mathbb{R}\backslash\{0\}\,. \tag{2.18}$$

Ein zweiter Aspekt ist der Einfluß von Störungen der Koeffizientenmatrix A auf die Lösung.

Sei nun die Lösung des gestörten Systems $\hat{A}\hat{x} = v$ gesucht.

Mit $\hat{A} = A + \Delta A$ gilt, falls $\| \Delta A \| \ll \frac{\|A\|}{\kappa(A)}$ [30]:

$$\frac{\| \Delta x \|}{\| x \|} \lessapprox \kappa(A) \cdot \frac{\| \Delta A \|}{\| A \|}\,. \tag{2.19}$$

Wiederum ist die Kondition ein proportionales Maß für die Verstärkung des relativen Eingangsfehlers.

Mit Hilfe des Störungslemmas [23] erhält man für das Gleichungssystem
$(A + \triangle A)(x + \triangle x) = v + \triangle v$ unter den gleichen Voraussetzungen wie bei (2.19):

$$\frac{\| \triangle x \|}{\| x \|} \lessapprox \kappa(A) \left\{ \frac{\| \triangle A \|}{\| A \|} + \frac{\| \triangle v \|}{\| v \|} \right\} \tag{2.20}$$

Beweis:[23].

Dies bedeutet, daß relative Störungen in den Eingangsdaten sich höchstens mit dem Verstärkungsfaktor $\kappa(A)$ auf die Lösung auswirken.

Dabei hängt die Kondition weder von dem spezifischen Lösungsverfahren noch von der rechten Seite des Gleichungssystems ab.

Die hier gezeigten Eigenschaften der Konditionszahl bringen einen erfreulichen Nebenaspekt für die in dieser Arbeit behandelte Vorkonditionierung. Eine Verbesserung der Konditionszahl hat nicht nur, wie im folgenden gezeigt wird, positive Auswirkungen auf die Konvergenzgeschwindigkeit bei Verfahren konjugierter Gradienten, sondern hat zusätzlich einen stabilisierenden Effekt auf das Verfahren bezüglich der Anfälligkeit gegen Rundungsfehler.

2.3 Das Verfahren der konjugierten Gradienten (CG)

2.3.1 Algorithmus

Von Hestenes und Stiefel [21] wurde ein Verfahren zur Lösung von linearen Gleichungssystemen mit symmetrischer positiv definiter (spd) Koeffizientenmatrix A entwickelt, welches auf der Lösung einer quadratischen Optimierungsaufgabe beruht.

Betrachtet werde nun das *Residuum*

$$r(x) = v - Ax$$

als negativer Gradient der Funktion

$$F(x) = \frac{1}{2} r(x)^T A^{-1} r(x) \,. \tag{2.21}$$

Da mit A auch A^{-1} positiv definit ist, gilt $F(x) > 0 \qquad \forall\, r(x) \neq 0$.

Das Minimum von $F(x)$, also $F(x) = 0$, wird für $r(x) = 0$ bzw. $x = A^{-1}v$, der eindeutig bestimmten Lösung des Gleichungssystems (2.1), erreicht.

Somit ist die Lösung des Gleichungssystems transformiert auf die freie Optimierungsaufgabe, das lokale und gleichzeitig globale Minimum der Funktion $F(x)$ zu finden.

Statt $F(x)$ kann auch die Funktion

$$F_0(x) = \frac{1}{2}x^T Ax - x^T v \tag{2.22}$$
$$= F(x) - \frac{1}{2}v^T A^{-1} v$$

betrachtet werden, da sich diese Funktion nur um eine Konstante von F unterscheidet.

Solch eine Optimierungsaufgabe kann iterativ mit der Schrittweite α_k und Suchrichtung p_k nach der Vorschrift

$$x_{k+1} = x_k + \alpha_k p_k \tag{2.23}$$

näherungsweise gelöst werden.

Ist die Suchrichtung vorgegeben, so ist das klassische Prinzip zur Bestimmung von α_k und damit von x_{k+1} die Lösung der eindimensionalen Minimierung

$$\min_{\alpha_k \in \mathbb{R}} F_0(x_k + \alpha_k p_k) \; . \tag{2.24}$$

In dem hier zugrundeliegenden Fall (2.22) gilt nun

$$F_0(x_k + \alpha_k p_k) = F_0(x_k) - \alpha_k r_k^T p_k + \frac{1}{2}\alpha_k^2 {p_k}^T A p_k \; . \tag{2.25}$$

r_k bezeichne hierbei das Residuum der k-ten Iteration, also $r_k = v - Ax_k$. Da A positiv definit ist, hat dieses Polynom 2.Grades sein Minimum gerade im Scheitel bei

$$\alpha_k = \frac{r_k^T p_k}{{p_k}^T A p_k} \; . \tag{2.26}$$

Die Frage der Schrittweite ist somit hinreichend geklärt. Verbleibt noch eine geeignete Wahl der Suchrichtungen.

Verfahren konjugierter Richtungen, deren Spezialfall das Verfahren der konjugierten Gradienten ist, benutzen die folgende Eigenschaft:

Def 2.3 *Sei A spd; $x_0, \ldots, x_{N-1} \in \mathbb{R}^n \backslash \{0\}$ heißen* **paarweise A-konjugiert**, *falls*

$$x_i^T A x_j = 0 \qquad \forall\, 0 \le i < j \le N-1 \; .$$

Insbesondere ergibt sich für A-konjugierte Vektoren die lineare Unabhängigkeit [10] und für I-konjugierte Vektoren die Orthogonalität. Deshalb werden A-konjugierte Vektoren auch häufig als *A-orthogonal* bezeichnet.

Umgekehrt kann man mit dem Gram-Schmidt-Orthogonalisierungsverfahren aus n linear unabhängigen Vektoren stets n paarweise A-konjugierte Vektoren erzeugen, indem man das Skalarprodukt $(x,y)_A := (x,Ay) = x^T Ay$ zugrundelegt.

Dies besagt auch folgender Satz, dessen Beweis analog zum Beweis des Gram-Schmidt-Verfahrens verläuft.

Satz 2.3 *A spd; $y_0,\ldots,y_{N-1}$ linear unabhängig; dann sind die durch*

$$\begin{aligned} p_0 &:= y_0 \\ p_k &:= y_k - \sum_{i=0}^{k-1} \frac{y_k^T A p_i}{p_i^T A p_i} p_i, \qquad k = 1,\ldots,N-1 \end{aligned}$$

definierten Vektoren paarweise A-konjugiert und spannen denselben Teilraum wie $y_0,\ldots,y_{N-1}$ auf.

Ein wesentlicher Vorteil der Verfahren konjugierter Richtungen liegt in der Tatsache, daß die y_k und somit auch die p_k nicht a-priori bekannt sein müssen, sondern beide sukzessiv im Iterationsverfahren berechnet werden können.

Die Verfahren unterscheiden sich in der Wahl der linear unabhängigen, zu orthogonalisierenden y_k.

Motiviert durch das analytische Resultat, daß die beste Abstiegsrichtung der negative Gradient, also das Residuum, ist, wählt man beim Verfahren der konjugierten Gradienten (wegen des engl. Namens **Conjugate Gradients** im folgenden als **CG-Verfahren** bezeichnet) das Residuum der k-ten Stufe als Vektor, den man gemäß Satz 2.3 bezüglich $p_0,\ldots,p_{k-1}$ $A-$konjugiert. Diese Vorgehensweise gibt dem Verfahren seinen Namen. Verbleibt jedoch die Frage, warum die Residuen paarweise linear unabhängig sind. Hierüber hinausgehend wird im folgenden gezeigt, daß die Residuen orthogonal sind.

Mit (2.26) ergibt sich als Iterationsverfahren für bereits festgelegte, bezüglich der $n \times n$ -Matrix A paarweise konjugierte, p_k:

$$\left\{ \begin{aligned} r_k &= v - Ax_k, \\ \alpha_k &= \frac{r_k^T p_k}{p_k^T A p_k}, \\ x_{k+1} &= x_k + \alpha_k p_k \qquad k = 0,\ldots,n-1. \end{aligned} \right. \tag{2.27}$$

Hier gilt:

$$r_k^T p_i = p_i^T r_k = 0, \qquad i = 0,\ldots,k-1, \tag{2.28}$$

denn:

$$r_i - r_k = Ax_k - Ax_i = A \cdot \sum_{j=i}^{k-1} \alpha_j p_j \tag{2.29}$$

und mit

$$p_i^T(r_i - r_k) = \sum_{j=i}^{k-1} \alpha_j p_i^T A p_j = \alpha_i p_i^T A p_i = r_i^T p_i = p_i^T r_i \tag{2.30}$$

folgt die Behauptung.

Folgender Satz liefert die Orthogonalität der Residuen.

Satz 2.4 *Sei A spd; in (2.27) sei $p_0 := r_0$ und p_k entstehe, indem man r_k gemäß Satz 2.3 bezüglich $p_0, \ldots, p_{k-1}$ A-orthogonalisiere.*

Dann gilt:

1. $r_k^T r_j = 0 \qquad$ *für* $0 \leq j < k \leq n$.
2. $\exists N \leq n \qquad$ *mit* $r_N = 0$.
3. $< r_0, \ldots, r_k > = < r_0, Ar_0, \ldots, A^k r_0 > \qquad \forall k \leq N$.

$< M >$ ist der von der Menge M erzeugte Vektorraum, insbesondere heißt $< r_0, Ar_0, \ldots, A^k r_0 >$ der von r_0 durch A erzeugte *Krylov-Unterraum.*

Beweis:
Zu 2): Da höchstens n lin. unabh. Vektoren existieren, ist die Aussage mit 1) bewiesen, da orthogonale Vektoren linear unabhängig sind.

Zu 1) und 3): Dies folgt induktiv über k.

Ist $k = 0$, so ist für 1) nichts zu zeigen bzw. für 3) die Aussage trivial.

Seien $r_0, \ldots, r_{k-1}$ orthogonal, so folgt mit Satz 2.3 und der Induktionsannahme

$$\begin{aligned} < r_0, \ldots, r_{k-1} > &= < p_0, \ldots, p_{k-1} > \\ &= < r_0, \ldots, A^{k-1} r_0 > . \end{aligned}$$

Mit (2.28) folgt: r_k orthogonal zu $< p_0, \ldots, p_{k-1} >$, also auch zu $< r_0, \ldots, r_{k-1} >$ und somit insbesondere auch die lineare Unabhängigkeit. Wegen (2.23) ist

$$\begin{aligned} (2.31)\, r_k = v - Ax_k &= v - Ax_{k-1} - \alpha_{k-1} A p_{k-1} \\ &= \underbrace{r_{k-1}}_{\in < r_0, \ldots, A^{k-1} r_0 >} - \alpha_{k-1} \underbrace{A p_{k-1}}_{\in < Ar_0, \ldots, A^k r_0 >} \end{aligned}$$

und damit $r_k \in < r_0, \ldots, A^k r_0 >$. $\surd$

Die Aussage 2) besagt, daß das Verfahren in höchstens n Schritten die exakte Lösung liefert. Aus diesem Grund ist das CG-Verfahren eigentlich ein direktes Verfahren. Das Ergebnis ist jedoch mehr von theoretischer Bedeutung, denn zum einen ist das Verfahren als Gram-Schmidt-Verfahren anfällig für Rundungsfehler, zum anderen wird in der Praxis gewöhnlich nach deutlich weniger als n Schritten eine gute Näherung der exakten Lösung erreicht.

Das Verfahren verhält sich daher wie ein iteratives Verfahren. Der entscheidende Vorteil liegt darin, daß die berechneten Zwischenresultate schon einen gewissen Optimalitätscharakter haben.

Satz 2.5 *Seien $p_0, \ldots, p_{n-1}$ paarweise A-konjugiert, $x_0 \in \mathbb{R}^n, x_0, \ldots, x_{n-1}$ gemäß (2.27) berechnet, so ist $\forall k < n \quad x_k$ Lösung des Optimierungsproblems*

$$\min_{x \in x_0 + < p_0, \ldots, p_k >} F(x) \ .$$

Beweis:[10].

Die Zwischenresultate sind also bereits Näherungen für die exakte Lösung und werden sukzessiv verbessert.

Der iterative Charakter des CG-Verfahrens hat den Vorteil, daß die Koeffizientenmatrix unverändert bleibt, und im Algorithmus im Unterschied zu anderen Verfahren keine weiteren Matrizen zur Zwischenspeicherung benötigt werden.

Die in Satz 1.3 durchgeführte Orthogonalisierung vereinfacht sich wegen Satz 2.4 auf einen einzigen Summanden, denn, wie im Beweis von Satz 2.4 gezeigt wurde, ist für $j = 0, \ldots, k-1$

$$\begin{aligned} Ap_j \in < Ar_0, \ldots, A^{j+1} r_0 > \ &\subseteq \ < r_0, \ldots, A^{j+1} r_0 > \\ &= \ < p_0, \ldots, p_{j+1} >, \end{aligned}$$

also $Ap_j = \sum_{i=0}^{j+1} \gamma_i p_i$ und wegen (2.28)

$$r_{k+1}^T A p_j = \sum_{i=0}^{j+1} \gamma_i r_{k+1}^T p_i = 0. \tag{2.32}$$

Mit Satz 2.3 ergibt sich mit der Notation

$$\beta_k = -\frac{r_{k+1}^T A p_k}{p_k^T A p_k}\,, \tag{2.33}$$

$$p_{k+1} = r_{k+1} + \beta_k p_k. \tag{2.34}$$

Der vorläufige Algorithmus lautet also:

Geg. x_0, $r_0 := v - Ax_0$, $p_0 := r_0$,

Für $k := 0, \ldots, n-1$:

$$\begin{aligned}
\alpha_k &= \frac{r_k^T p_k}{p_k{}^T A p_k} \\
x_{k+1} &= x_k + \alpha_k p_k \qquad (2.35)\\
r_{k+1} &= v - Ax_{k+1} \qquad (2.36)\\
\beta_k &= -\frac{r_{k+1}^T A p_k}{p_k^T A p_k} \\
p_{k+1} &= r_{k+1} + \beta_k p_k\,. \qquad (2.37)
\end{aligned}$$

Der Arbeitsaufwand besteht im wesentlichen aus zwei Matrix-Vektor-Multiplikationen Ap_k und Ax_{k+1}. Durch Umformungen erreicht man mit (2.35) und (2.36)

$$r_{k+1} = r_k - \alpha_k A p_k\,, \tag{2.38}$$

womit eine Matrix-Vektor-Multiplikation gespart wird. Einige Skalarprodukte werden durch die folgenden Umformungen eingespart.

$$r_k^T p_k = r_k^T (r_k + \beta_{k-1} p_{k-1}) = r_k^T r_k \tag{2.39}$$

$$\alpha_k = \frac{r_k^T r_k}{p_k^T A p_k} \tag{2.40}$$

$$\begin{aligned}
r_{k+1}^T r_{k+1} &= r_{k+1}^T (r_k - \alpha_k A p_k) \\
&= -\alpha_k r_{k+1}^T A p_k \\
&= -\frac{r_k^T r_k}{p_k^T A p_k} \cdot (-\beta_k) \cdot p_k^T A p_k \\
&= \beta_k r_k^T r_k
\end{aligned}$$

$$\Longrightarrow \quad \beta_k = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k} \tag{2.41}$$

Das Ergebnis dieses Abschnitts ist der Algorithmus in Abb. 2.1.

Algorithmus des Verfahrens der konjugierten Gradienten

Geg. $x_0, r_0 := v - Ax_0, p_0 := r_0$

Für $k := 0, 1, \ldots$

$$\alpha_k = \frac{r_k^T r_k}{p_k^T A p_k}$$

(2.42) $$x_{k+1} = x_k + \alpha_k p_k$$

(2.43) $$r_{k+1} = r_k - \alpha_k A p_k$$

$$\beta_k = \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}$$

(2.44) $$p_{k+1} = r_{k+1} + \beta_k p_k$$

Bis das Konvergenzkriterium erfüllt ist.

Abbildung 2.1: Algorithmus des Verfahrens der konjugierten Gradienten

Der verbleibende Aufwand des Verfahrens besteht pro Iterationsschritt somit aus:

1. Speicherplatz für Felder
   - 1 Matrix A
   - 4 Vektoren x_k, r_k, p_k, Ap_k
2. Operationen
   - 1 Matrix-Vektor-Produkt
   - 3 Skalarprodukte
   - 3 Vektoradditionen

Im Hinblick auf die später zu untersuchende parallele Implementierung vereinfacht sich die Bearbeitung, falls alle Skalare zu Beginn berechnet werden können. Durch Umformungen bei der Berechnung von β_k ergibt sich mit (2.40), (2.43) und (2.44)

$$\begin{aligned} \beta_k &= \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k} \\ &= \frac{(r_k - \alpha_k A p_k)^T (r_k - \alpha_k A p_k)}{r_k^T r_k} \end{aligned}$$

$$= 1 - 2\frac{\alpha_k p_k^T A p_k}{r_k^T r_k} + \frac{\alpha_k^2 (Ap_k)^T A p_k}{r_k^T r_k}$$
$$= \alpha_k \frac{(Ap_k)^T A p_k}{p_k^T A p_k} - 1 .$$

Dieses führt zum Algorithmus des modifizierten CG-Verfahrens [6],[7] gemäß Abb. 2.2, der günstigere Parallelisierungseigenschaften als der ursprüngliche besitzt.

Algorithmus des modifizierten CG-Verfahrens

Geg. $x_0, r_0 := v - Ax_0, p_0 := r_0$

Für $k := 0, 1, \ldots$

(2.45) $$\alpha_k = \frac{r_k^T r_k}{p_k^T A p_k}$$

(2.46) $$\beta_k = \frac{\alpha_k (Ap_k)^T A p_k}{p_k^T A p_k} - 1$$

(2.47) $$r_{k+1}^T r_{k+1} = \beta_k r_k^T r_k$$

(2.48) $$x_{k+1} = x_k + \alpha_k p_k$$

(2.49) $$r_{k+1} = r_k - \alpha_k A p_k$$

(2.50) $$p_{k+1} = r_{k+1} + \beta_k p_k .$$

Bis das Konvergenzkriterium erfüllt ist.

Abbildung 2.2: Algorithmus des modifizierten CG-Verfahrens

Bemerkung: Als Lösungsverfahren für die Minimierung von $F(x)$ aus (2.21) kann auch ein anderes Optimierungsverfahren gewählt werden.

So ergibt sich mit dem *Verfahren der sukzessiven Variation der Variablen*, bei dem pro Schritt die Koordinatenrichtungen nacheinander betrachtet werden und die lokalen Schrittweiten sukzessiv als eindimensionale Minimierungsaufgaben in ihrer Koordinatenrichtung exakt berechnet werden, das Einzelschrittverfahren.

Da bei diesem Verfahren die Koordinatenachsen als starre Suchrichtungen verwendet werden, ist die Konvergenzgeschwindigkeit i.a. schlechter als beim CG-Verfahren.

Die Konvergenzeigenschaften des CG-Verfahrens sind nicht wie in Abschnitt 2.1.2 zu verifizieren, da dieses Verfahren nicht in der Form (2.3) darstellbar ist. Deshalb wird das Konvergenzverhalten im folgenden Abschnitt diskutiert.

2.3.2 Konvergenz des CG-Verfahrens

Ist die Norm nicht näher bezeichnet, so ist im folgenden immer für spd–Matrizen die Spektralnorm, also der größte Eigenwert von A, und bei Vektoren die hiermit verträgliche euklidische Norm, also $\| x \| := (x,x)^{\frac{1}{2}} = \sqrt{x^T x}$, gemeint.

Da die im Verfahren der konjugierten Gradienten betrachteten Matrizen positiv definit sind, gilt somit für die Kondition:

$$\kappa(A) = \frac{\lambda_{max}(A)}{\lambda_{min}(A)} , \tag{2.51}$$

wobei λ_{max} den größten, λ_{min} den kleinsten Eigenwert von A bezeichnet.

In Kapitel 2.3.1 ist die Frage nach einem geeignetem Abbruchkriterium noch unbeantwortet geblieben. Wünschenswert wäre ein Kriterium der Form $\| x_k - x^* \| < \varepsilon$, jedoch ist die exakte Lösung x^* unbekannt.

Als Maß geeignet scheint die Norm des Residuums $\| r_k \| = \| v - Ax_k \|$ zu sein, leider ist jedoch das Kriterium eines *kleinen* Residuums sehr schwer zu handhaben, da dem Wert des Residuums nicht anzusehen ist, wie weit die exakte Lösung noch entfernt ist.

Im folgenden bezeichne $\| y \|_A := (y, Ay)^{1/2} = \sqrt{y^T A\, y}$ die von A induzierte euklidische Norm. Für diese Norm gilt mit

$$\gamma := \frac{\sqrt{\kappa(A)} - 1}{\sqrt{\kappa(A)} + 1}$$

$$\| x_k - x^* \|_A \leq 2 \cdot \gamma^k \cdot \| x_0 - x^* \|_A \ . \tag{2.52}$$

Beweis: [11].

Hiermit läßt sich eine Abschätzung für die euklidische Norm des Fehlers der Iterierten angeben, denn es ist

- $A - \lambda_{min}\, I$ positiv semi–definit, d.h. $\forall\, y$ ist $y^T (A - \lambda_{min}\, I)\, y \geq 0$,
- $A - \lambda_{max}\, I$ negativ semi–definit, d.h. $\forall\, y$ ist $y^T (A - \lambda_{max}\, I)\, y \leq 0$,

woraus folgt

$$\lambda_{min} y^T y \leq y^T A y \leq \lambda_{max} y^T y \ , \tag{2.53}$$

also

$$
\begin{aligned}
(2.54) \qquad \| x_k - x^* \|^2 &= (x_k - x^*)^T (x_k - x^*) \\
&\leq \frac{1}{\lambda_{min}} (x_k - x^*)^T A (x_k - x^*) \\
&= \| A^{-1} \| \cdot \| x_k - x^* \|_A^2 \\
&\leq \| A^{-1} \| \cdot (2\gamma^k)^2 \cdot \| x_0 - x^* \|_A^2 \\
&\leq \kappa(A) \cdot (2\gamma^k)^2 \cdot \| x_0 - x^* \|^2 \\
(2.55) \qquad \Rightarrow \quad \| x_k - x^* \| &\leq 2\sqrt{\kappa(A)}\, \gamma^k \| x_0 - x^* \| .
\end{aligned}
$$

Dies spiegelt die Bedeutung der Konditionszahl für das CG-Verfahren wieder, denn somit ist die Kondition nicht nur ein Maß für die Stabilität der Matrix gegenüber Rundungsfehlern, sondern auch ein Konvergenzmaß, da die rechten Seiten in (2.52) und (2.55) streng monoton steigend in $\kappa(A)$ sind.

Aus dieser Formel läßt sich berechnen, wieviele Iterationsschritte maximal nötig sind, um den Fehler des Startvektors um einen Faktor 10^{-m} zu verkleinern:

$$
k_{max} = \left\lceil \frac{m + \log(2\sqrt{\kappa(A)})}{\log(\sqrt{\kappa(A)} + 1) - \log(\sqrt{\kappa(A)} - 1)} \right\rceil^*
$$

Einige dieser Werte sind der Tabelle 2.3.2 zu entnehmen:

	$\kappa(A) = 5$	$\kappa(A) = 10$	$\kappa(A) = 100$	$\kappa(A) = 1000$	$\kappa(A) = 10000$
$m = 1$	4	7	27	102	381
$m = 2$	7	10	38	139	496
$m = 3$	9	14	50	175	611
$m = 5$	14	21	73	248	841

Tabelle 2.1: Einfluß von Konditionszahl und Genauigkeit auf die maximale Anzahl Iterationen

Hieraus ist ersichtlich, daß für die maximale Anzahl der Iterationsschritte die Konditionszahl einen größeren Einfluß hat als die geforderte Genauigkeit der Lösung.

Für den Fehler der Iterierten gilt zusätzlich folgende strenge Monotonie:

$$
(2.56) \qquad \| x_k - x^* \| < \| x_j - x^* \| \qquad \forall k > j,\ x_j \neq x^* \quad .
$$

Beweis:[24].

Bemerkung: In Satz 2.4 wurde gezeigt, daß das CG-Verfahren nach n Iterationen die exakte Lösung liefert. Weitergehend läßt sich für mehrfache Eigenwerte folgendes zeigen:

Satz 2.6 *Besitzt A lediglich $m \leq n$ verschiedene Eigenwerte, so liefert das CG-Verfahren nach maximal m Iterationen die exakte Lösung x^*.*

Beweis:[24].

Dies legt nahe, daß das CG-Verfahren umso besser konvergiert, je mehr die Eigenwerte in Gruppen dicht beieinanderliegen. Dies wird im folgenden als *Clusterung* bezeichnet. Für die Genauigkeit der Fehlerabschätzung bedeutet dies, daß die Abschätzung (2.55) zu grob wird, falls die Eigenwerte weit streuen. So ergibt sich z.B. für eine Matrix A mit einem Eigenwert von 10000 und mit allen anderen Eigenwerten gleich 1 bei einer Genauigkeit von 10^{-3} aus Tabelle 2.1 eine maximale Iterationsanzahl von 611. Das Verfahren liefert jedoch in exakter Arithmetik wegen Satz 2.6 schon nach 2 Iterationen die Lösung.

Kapitel 3

Die Methode der Vorkonditionierung

Wie in Kapitel 2.3.2 gesehen, hängt die Konvergenzgeschwindigkeit des CG-Verfahrens von der Kondition der Ausgangsmatrix ab. In diesem Kapitel wird versucht, durch einen Übergang zu einem äquivalenten Gleichungssystem eine Koeffizientenmatrix mit kleinerer Kondition und somit besseren Konvergenzeigenschaften zu erhalten. Es werden die Theorie der Vorkonditionierung sowie verschiedene Vorkonditionierungsmethoden vorgestellt.

Literatur hierzu findet sich in [10],[11] und [24]. Die ausführlicher behandelte polynomiale Vorkonditionierung stammt aus [2],[3],[4],[22] und [24].

3.1 Das CG-Verfahren mit Vorkonditionierung

Die Idee der Vorkonditionierung des CG-Verfahrens besteht darin, durch Verbesserung der Kondition des zu lösenden Gleichungssystems eine Verbesserung der Konvergenzgeschwindigkeit zu erreichen. Dies geschieht, indem das Gleichungssystem in ein äquivalentes System mit besserer Kondition transformiert wird, das dann mit dem CG-Verfahren gelöst wird. Damit das CG-Verfahren anwendbar bleibt, muß die Koeffizientenmatrix des neuen Gleichungssystems wieder symmetrisch und positiv definit sein.

Im folgenden werden zwei Möglichkeiten vorgestellt, die dies gewährleisten.

Betrachtet wird zunächst das zu (2.1) äquivalente System

$$\underbrace{SAS^T}_{\tilde{A}}\underbrace{S^{-T}x}_{\tilde{x}} = \underbrace{Sv}_{\tilde{v}} \tag{3.1}$$

mit einer regulären Vorkonditionierungsmatrix S. Die Matrix $\tilde{A}$ heißt auch *Kongruenztransformierte* von A durch S.

Die Erhaltung der Symmetrie ergibt sich aus

$$\tilde{A}^T = SA^TS^T = SAS^T = \tilde{A} .$$

Ist A positiv definit, also $x^TAx > 0 \quad \forall\, x \in \mathbb{R}^n\backslash\{0\}$, so gilt, da S regulär ist:

$$x^TSAS^Tx = (S^Tx)^TA(S^Tx) > 0 \qquad \forall\, x \in \mathbb{R}^n\backslash\{0\} ,$$

also ist auch $\tilde{A}$ positiv definit.

Andererseits läßt sich die Kongruenztransformation einsetzen, um das CG-Verfahren anwenden zu können. Der Matrix A ist häufig nicht anzusehen, ob sie positiv definit ist. Oft ist dies nicht gegeben und somit das CG-Verfahren nicht anwendbar. Es ist jedoch $A^TA = A^TIA$ eine Kongruenztransformation der Einheitsmatrix, also spd, falls A regulär ist. Ist also $Ax = v$ zu lösen, mit einer quadratischen Koeffizientenmatrix A, deren Symmetrie oder positive Definitheit nicht gesichert ist, so betrachte $A^TAx = A^Tv$. Auf dieses System ist das CG-Verfahren anwendbar. Für die Kondition gilt: $\kappa(A^TA) = \kappa(A)^2$ [11], so daß für die Konvergenzgeschwindigkeit in (2.52) und (2.55) $\sqrt{\kappa(A)}$ durch $\kappa(A)$ ersetzt werden muß. Dieses Resultat ist nur von theoretischer Bedeutung, denn in der Praxis ist diese Kongruenztransformation zum einen sehr aufwendig, da eine Matrixmultiplikation durchgeführt werden muß und zum anderen i.a. die Besetzungsstruktur der Koeffizientenmatrix zerstört wird, also die Anzahl der Nicht-Nullelemente wächst.

Nach der Anwendung des CG-Verfahrens auf (3.1) muß x als Lösung von $S^{-T}x = \tilde{x}$ rücktransformiert werden, um die Lösung des ursprünglichen Systems zu erhalten.

Im folgenden wird sich zeigen, daß SAS^T nicht in jedem Fall berechnet werden muß, sondern in die Berechnung von Suchrichtungen und Iterierten integriert werden kann. Insbesondere ist häufig nicht einmal die explizite Kenntnis von S notwendig.

Verschiedene Methoden der Vorkonditionierung unterscheiden sich in der Wahl der regulären Matrix S.

Die günstigste Vorkonditionierungsmatrix ist $S = A^{-\frac{1}{2}}$, denn dann ist

$$\kappa(SAS^T) \;=\; \kappa(I) = 1 ,$$

jedoch erfordert die Berechnung von $A^{-\frac{1}{2}}$ bereits genau soviel Aufwand, wie die direkte Lösung von (2.1).

Ein alternativer Ansatz zur Vorkonditionierung ergibt sich aus:

$$\underbrace{(CA)}_{A^*}x \;=\; \underbrace{(Cv)}_{v^*} \tag{3.2}$$

Bei (3.2) ist das Ziel, $\kappa(CA)$ möglichst klein zu halten. Der optimale Wert $C = A^{-1}$ ist auch hier nur von theoretischer Bedeutung.

Hierbei ist die Lösung von (3.2) bereits Lösung des ursprünglichen Systems. Deshalb wird (3.1) im folgenden als Vorkonditionierung *mit* Rücksubstitution, (3.2) als Vorkonditionierung *ohne* Rücksubstitution bezeichnet.

Der Vorteil der ersten Methode ist, daß die Voraussetzungen zur Durchführbarkeit des CG-Verfahrens wieder erfüllt sind. Jedoch ist abschließend eine Rücksubstitution erforderlich, und es müssen zumindest theoretisch 2 Matrixmultiplikationen zur Kongruenztransformation ausgeführt werden.

Bei der zweiten Methode ist nur eine Matrixmultiplikation erforderlich und die Lösung des neuen Systems ist die gleiche wie die des ursprünglichen, jedoch ist es hier notwendig, die Anwendbarkeit des CG-Verfahrens zu gewährleisten. Es sei darauf hingewiesen, daß das Produkt zweier symmetrischer und positiv definiter Matrizen i.a. weder symmetrisch noch positiv definit ist. C muß also so gewählt werden, daß diese Eigenschaften erhalten bleiben.

Im folgenden wird je nach Bedarf (3.1) oder (3.2) zur Vorkonditionierung benutzt. Man erhält (3.2) aus (3.1), falls in (3.1) $S := C$ gesetzt wird und $S^T S^{-T}$ zu I zusammengefaßt wird.

Allgemein ist die Vorkonditionierung ein Vorschaltverfahren zur Verbesserung der Kondition. Dies bedeutet, daß man beliebig viele, auch verschiedene, Vorkonditionierungen vor dem eigentlichen Verfahren durchführen kann.

Zuerst wird eine Vorkonditionierung gemäß (3.1) untersucht. Hierbei läßt sich der Algorithmus weitgehend auf die ursprünglichen Größen zurückführen, so daß die abschließende Rücksubstitution entfällt. Es ergibt sich aus dem Algorithmus des CG-Verfahrens für die entsprechenden Größen:

Für die Initialisierungen:

$$\tilde{x}_0 = S^{-T} x_0 \,, \tag{3.3}$$

$$\tilde{r}_0 = \tilde{v} - \tilde{A}\tilde{x}_0 = S(v - Ax_0) = Sr_0 \,, \tag{3.4}$$

$$\tilde{p}_0 = Sr_0 = Sp_0 = \tilde{r}_0 \tag{3.5}$$

und mit $\hat{p}_0 := S^T S p_0$

$$\tilde{p}_0^T \tilde{A} \tilde{p}_0 = p_0^T S^T S A S^T S p_0 = \hat{p}_0^T A \hat{p}_0 \,. \tag{3.6}$$

Weiterhin

$$\tilde{r}_0^T \tilde{r}_0 = r_0^T S^T S r_0 = \underbrace{(S^T S r_0)}_{\hat{r}_0}{}^T r_0 = \hat{r}_0^T r_0 \,, \tag{3.7}$$

$$\tilde{\alpha}_0 = \frac{\hat{r}_0^T r_0}{\hat{p}_0^T A \hat{p}_0} \,, \tag{3.8}$$

$$\tilde{x}_1 = \tilde{x}_0 + \tilde{\alpha}_0 \tilde{p}_0 \qquad | \cdot S^T$$

$$
\begin{aligned}
(3.9) \qquad \Rightarrow \qquad x_1 &= x_0 + \tilde{\alpha}_0 \hat{p}_0 \ , \\
\tilde{r}_1 &= \tilde{r}_0 - \tilde{\alpha}_0 \tilde{A}\tilde{p}_0 = S(r_0 - \tilde{\alpha}_0 A\hat{p}_0) \qquad | \cdot S^{-1} \\
(3.10) \qquad \Rightarrow \qquad r_1 &= r_0 - \tilde{\alpha}_0 A\hat{p}_0 \ , \\
(3.11) \qquad \tilde{\beta}_0 &= \frac{\tilde{r}_1^T \tilde{r}_1}{\tilde{r}_0^T \tilde{r}_0} = \frac{\hat{r}_1^T r_1}{\hat{r}_0^T r_0}
\end{aligned}
$$

und schließlich

$$
\begin{aligned}
\tilde{p}_1 &= \tilde{r}_1 + \tilde{\beta}_0 \tilde{p}_0 = S r_1 + \tilde{\beta}_0 \tilde{p}_0 \qquad | \cdot S^T \\
(3.12) \qquad \Rightarrow \qquad \hat{p}_1 &= \hat{r}_1 + \tilde{\beta}_0 \hat{p}_0 \ .
\end{aligned}
$$

Der gesamte Algorithmus folgt per Induktion.

Das verbleibende Problem ist die Bestimmung von $\hat{r}_k = (S^T S) r_k$, ansonsten taucht die Vorkonditionierungsmatrix S in den Iterationsschritten nicht mehr auf.

Daß die explizite Kenntnis von S jedoch i.a. nicht nötig ist, zeigt folgende Überlegung. Wäre $S^T S = A^{-1}$ bzw. $A = (S^T S)^{-1}$, so folgt durch Multiplikation von (3.1) mit S^T :

$$
\begin{aligned}
(S^T S) A S^T S^{-T} x &= (S^T S) v \ , \\
\text{also} \qquad x &= A^{-1} v \ .
\end{aligned}
$$

Ziel ist es also, $M = (S^T S)^{-1}$ so zu wählen, daß A durch M möglichst gut approximiert wird und $M\hat{r}_k = r_k$ leicht zu lösen ist. $M = A$ ist hierbei keine geeignete Matrix, da die Lösung von $M\hat{r}_k = r_k$ i.a. genau so aufwendig wie die Lösung des ursprünglichen Systems ist. Bei der Wahl von M muß jedoch die Zerlegbarkeit von $M^{-1} = S^T S$ gewährleistet werden. Diese Forderung ist äquivalent zu der Bedingung, daß M spd ist.

Leicht zu lösendes Gleichungssystem bedeutet hierbei, daß die Lösung ohne hohen Rechenaufwand gefunden werden kann. Dies ist z.B. dann der Fall, wenn die Matrix Diagonalgestalt hat, sie eine obere oder untere Dreiecksmatrix ist oder sie als Produkt einer oberen und unteren Dreiecksmatrix darstellbar ist (Cholesky-Zerlegung).

Da die Formulierung *möglichst gut zu approximieren* schwer handhabbar ist und von der betrachteten Norm abhängt, soll folgender Zusammenhang diese Eigenschaft präzisieren:

Es ist $\tilde{A}$ ähnlich zu $S^T S A = M^{-1} A$ und somit

$$
(3.13) \qquad \kappa(\tilde{A}) = \frac{\lambda_{max}(M^{-1}A)}{\lambda_{min}(M^{-1}A)} \ .
$$

M ist also so zu wählen, daß der Quotient aus größtem und kleinstem Eigenwert von $M^{-1}A$ minimal wird.

Algorithmus des vorkonditionierten CG-Verfahrens

Geg. $x_0, r_0 := v - Ax_0$, Löse $M\hat{r}_0 = r_0$, $\hat{p}_0 := \hat{r}_0$

Für $k := 0, 1, \ldots$

$$\tilde{\alpha}_k = \frac{\hat{r}_k^T r_k}{\hat{p}_k^T A \hat{p}_k}$$

(3.14) $$x_{k+1} = x_k + \tilde{\alpha}_k \hat{p}_k$$

(3.15) $$r_{k+1} = r_k - \tilde{\alpha}_k A \hat{p}_k$$

(3.16) Löse $$M\hat{r}_{k+1} = r_{k+1}$$

(3.17) $$\tilde{\beta}_k = \frac{\hat{r}_{k+1}^T r_{k+1}}{\hat{r}_k^T r_k}$$

(3.18) $$\hat{p}_{k+1} = \hat{r}_{k+1} + \tilde{\beta}_k \hat{p}_k$$

(3.19)

Bis das Konvergenzkriterium erfüllt ist.

Abbildung 3.1: Algorithmus des vorkonditionierten CG-Verfahrens

Es ergibt sich als Fazit dieses Abschnitts der Algorithmus in Abb. 3.1.

Insbesondere ergibt sich für $M = I$ wieder das ursprüngliche CG-Verfahren aus Abbildung 2.1.

3.2 SSOR– und Gesamtschrittvorkonditionierer

Der zusätzliche Aufwand pro Iterationsschritt der Vorkonditionierung besteht also im Lösen des Gleichungssystems (3.16). Hierbei ist M so zu wählen, daß zum einen A durch M möglichst gut approximiert wird, um eine gute Konvergenzbeschleunigung zu erhalten, zum anderen darf der Aufwand zur Lösung von (3.16) nicht zu hoch sein. Ist bereits das Produkt $S^T S$ und somit M^{-1} bekannt, so reduziert sich die Lösung von (3.16) auf ein Matrix-Vektor-Produkt.

Auffallend ist hierbei die Analogie zu a) und b) auf Seite 5, die die Wahl von B zur Konstruktion eines Iterationsverfahrens beschreiben.

Dies kann eine intuitive Hilfe bei der Wahl der Vorkonditionierer sein, da einige der dort verwendeten Matrizen die hier geforderten Eigenschaften besitzen.

Es liefern jedoch nicht alle dort aufgeführten Verfahren geeignete Vorkonditionierer, da die Matrix B, der hier die Matrix M entspricht, symmetrisch und positiv definit sein muß, um die Durchführbarkeit des CG-Verfahrens zu gewährleisten. Die Darstellung von B als Produkt S^TS ist gewährleistet, da für spd–Matrizen gemäß der Cholesky-Zerlegung die Darstellung $B = LL^T$ existiert [11].

Als Resultat ergeben sich hieraus zwei Vorkonditionierungen, die nach den entsprechenden Iterationsverfahren benannt sind (Siehe [10]).

- Gesamtschritt-Vorkonditionierung
  Es ist $M := D$, also $S^{-T} = D^{\frac{1}{2}}$.
  Dieses Verfahren wird auch häufig als **Skalierung mit der Diagonalen** (bzw. engl. Diagonal Scaling) bezeichnet.
  Obwohl dieser Algorithmus formal wie oben beschrieben durchführbar ist, wird in der Praxis wegen des geringeren Aufwandes die Vorkonditionierung dem Verfahren vorgeschaltet und nach Durchführung des Verfahrens rücksubstituiert.
- SSOR-Vorkonditionierung
  Hier ist $M(\omega) := \frac{1}{\omega(2-\omega)} D(I - \omega D^{-1}L)(I - \omega D^{-1}L^T)$,

  also $S^{-T} = \sqrt{\frac{1}{\omega(2-\omega)}} D^{\frac{1}{2}}(I - \omega D^{-1}L^T)$.

  Somit ist S^{-T} eine Dreiecksmatrix, die die gleiche schwache Besetzungsstruktur wie A hat. Die Wahl des optimalen oder wenigstens eines gut geeigneten ω zur Konditionsverbesserung ist schwierig. In der Praxis wird bei der SSOR-Vorkonditionierung ein modifizierter Algorithmus verwendet, der das Lösen des Gleichungssystems umgeht. Ein solcher Algorithmus findet sich in [24].

3.3 Die polynomiale Vorkonditionierung

Im folgenden wird von der Darstellung (3.2) ausgegangen. Eine Vorkonditionierungsmatrix C ist gut geeignet, wenn sie folgende Eigenschaften erfüllt:

1. CA symmetrisch und positiv definit,
2. $\kappa(CA) \ll \kappa(A)$.

Die Idee der polynomialen Vorkonditionierung ist, C als Matrixpolynom vom Grad m in A, also

$C := \mathcal{C}(A) = \sum_{i=0}^{m} \alpha_i A^i$, zu wählen, denn dann gilt bezüglich der Symmetrie

$$(\mathcal{C}(A)A)^T = A^T \sum_{i=0}^{m} \alpha_i (A^i)^T = A \sum_{i=0}^{m} \alpha_i (A^T)^i = (\sum_{i=0}^{m} \alpha_i A^{i+1}) = \mathcal{C}(A)\ A\ .$$

Für die Eigenwerte $\lambda_{min} = \lambda_1 \leq \ldots \leq \lambda_n = \lambda_{max}$ von A gilt:

Lemma 3.1 **a)** *Sei λ_j Eigenwert von A $j = 1, \ldots, n$*
$\Longrightarrow$ *$\mathcal{C}(\lambda_j)$ Eigenwert von $\mathcal{C}(A)$ zum gleichen Eigenvektor x.*
$\Longrightarrow$ *$\mathcal{C}(\lambda_j)\lambda_j$ Eigenwert von $\mathcal{C}(A)A$ zum gleichen Eigenvektor x.*

b) *Gilt für ein Polynom $\mathcal{P}(\xi) := \mathcal{C}(\xi) \cdot \xi > 0 \quad \forall \xi \in [\lambda_{min}, \lambda_{max}]$*
$\Longrightarrow$ *$\mathcal{P}(A)$ ist positiv definit.*

Beweis:
Zu a): Sei $\mathcal{C}(\xi) = \sum_{i=0}^{m} \alpha_i \xi^i$, dann gilt

$$\left(\sum_{i=0}^{m} \alpha_i A^i\right) x = \sum_{i=0}^{m} \alpha_i A^i x = \sum_{i=0}^{m} \alpha_i \lambda_j^i x = \mathcal{C}(\lambda_j) x$$

$$(\sum_{i=0}^{m} \alpha_i A^i) \cdot Ax = (\sum_{i=0}^{m} \alpha_i A^i) \cdot \lambda_j x = \mathcal{C}(\lambda) \lambda_j x$$

zu b): Folgt aus a). $\surd$

Die Matrix CA ist somit symmetrisch und positiv definit, falls $\mathcal{P}(\xi) > 0$
$\forall \xi \in [\lambda_{min}, \lambda_{max}]$ gilt.

Für den Algorithmus ändert sich für das polynomial vorkonditionierte CG-Verfahren wenig, da man das CG-Verfahren auf $\mathcal{C}(A)Ax = \mathcal{C}(A)v$ anwendet. Somit ist im Algorithmus A durch $\mathcal{C}(A)A$ und v durch $\mathcal{C}(A)v$ zu ersetzen und es ergibt sich der Algorithmus in Abb. 3.2. Der Mehraufwand pro Iteration besteht in der Berechnung des Matrix-Vektor-Produktes $\mathcal{C}(A)Ap_k$ statt Ap_k. Dies bedeutet, daß in

Algorithmus des modifizierten CG-Verfahrens mit polynomialer Vorkonditionierung

Geg. $x_0, r_0 := \mathcal{C}(A)(v - Ax_0), p_0 := r_0$

Für $j := 0, 1, \ldots$

$$\begin{aligned}
\alpha_j &= \frac{r_j^T r_j}{p_j^T \mathcal{C}(A) A p_j} \\
\beta_j &= \frac{\alpha_j (\mathcal{C}(A) A p_j)^T \mathcal{C}(A) A p_j}{p_j^T \mathcal{C}(A) A p_j} - 1 \\
r_{j+1}^T r_{j+1} &= \beta_j r_j^T r_j \\
x_{j+1} &= x_j + \alpha_j p_j \\
r_{j+1} &= r_j - \alpha_j \mathcal{C}(A) A p_j \\
p_{j+1} &= r_{j+1} + \beta_j p_j .
\end{aligned}$$

Bis das Konvergenzkriterium erfüllt ist.

Abbildung 3.2: Algorithmus des modifizierten CG-Verfahrens mit polynomialer Vorkonditionierung

jedem Iterationsschritt $m + 1$ Matrix-Vektor-Produkte berechnet werden müssen statt einem.

Für eine geeignete Wahl von $\mathcal{C}(A)$ zur Verbesserung der Kondition hilft folgende Überlegung:

Die optimale Kondition ist gegeben, falls $\mathcal{C}(A)A = I$. Die naheliegende Idee ist nun, die Differenz zwischen $\mathcal{C}(A)A$ und I zu minimieren. Dies führt zu

$$\min_{\mathcal{C} \in P_m} \| I - \underbrace{\mathcal{C}(A)A}_{\mathcal{P}(A)} \| , \tag{3.20}$$

wobei P_m die Menge der Polynome vom Grad m bezeichnet.

Die Lösung dieser Minimierungsaufgabe ist dabei normabhängig.

3.4 CG-Verfahren für indefinite Matrizen

Die hier vorgestellte Methode ermöglicht eine Verallgemeinerung des CG-Verfahrens auf symmetrische Matrizen mit beliebigen reellen Eigenwerten. Für die Anwendbarkeit des Verfahrens auf die vorkonditionierte Matrix $\mathcal{P}(A)$ ist sicherzustellen, daß $\mathcal{P}(\lambda_i) > 0$ für die Eigenwerte von A gilt. An dieser Stelle wird die Eigenschaft, daß A positive Eigenwerte besitzt, nicht benötigt.

Durch Vorkonditionierung mit einem Polynom folgender oder ähnlicher Gestalt kann das CG-Verfahren auf symmetrische negativ definite Matrizen angewandt werden.

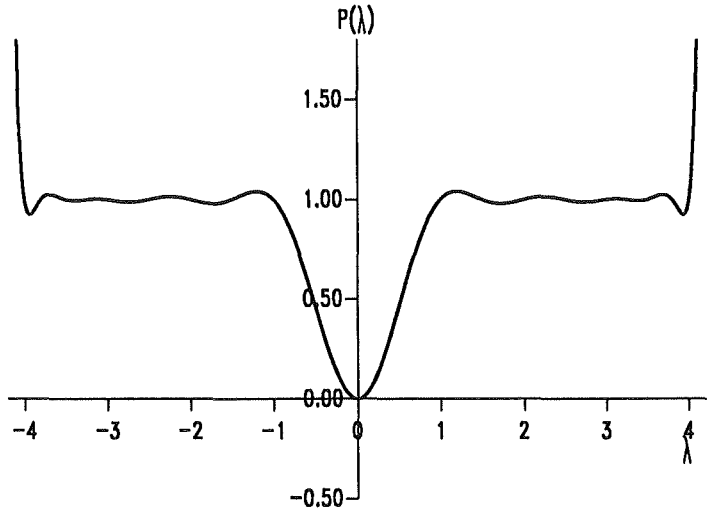


Abbildung 3.3: Vorkonditionierungspolynom für negativ definite Matrizen

Ziel der Vorkonditionierung ist in diesem Fall, die Voraussetzungen für die Anwendbarkeit des CG-Verfahrens zu schaffen.

3.5 Die polynomiale Tschebyscheff-Vorkonditionierung

In diesem Kapitel wird das Problem in (3.20) für eine spezielle Norm gelöst. Anschließend wird eine Vorgehensweise vorgestellt, die die explizite Berechnung der Polynomkoeffizienten umgeht.

3.5.1 Theorie

Wird in (3.20) die Spektralnorm zugrundegelegt, so erhält man, da mit A auch das Matrixpolynom $I - C(A)A$ symmetrisch ist und die Spektralnorm für symmetrische Matrizen der Betrag des betragsgrößten Eigenwertes ist, das folgende Minimax-Problem:

$$\min_{C \in P_m} \left\{ \max_{\lambda_i \text{ EW von A}} |1 - \underbrace{C(\lambda_i) \cdot \lambda_i}_{\mathcal{P}(\lambda_i)}| \right\}$$

$\mathcal{P}$ ist vom Grad $k = m + 1$ und erfüllt $\mathcal{P}(0) = 0$. Es wird also ein Polynom $\mathcal{P}$ k-ten Grades mit $\mathcal{P}(0) = 0$ gesucht, welches alle Eigenwerte von A möglichst nahe zu 1 abbildet. In der Regel sind die Eigenwerte von A nicht bekannt und deren Berechnung zu aufwendig. Es wird stattdessen ein Ersatzproblem betrachtet:

Sei $0 < a \leq \lambda_{min} \leq \lambda_{max} \leq b$ und $a \neq b$, so löse

$$\min_{\substack{\mathcal{P} \in P_k \\ \mathcal{P}(0)=0}} \max_{\lambda \in [a,b]} | 1 - \mathcal{P}(\lambda) | \ . \tag{3.21}$$

Das *Restpolynom* $\mathcal{R}(\lambda) = 1 - \mathcal{P}(\lambda)$ kennzeichnet den Abstand zu 1. Das Restpolynom $\mathcal{R}$ ist vom Grad k, und es gilt $\mathcal{R}(0) = 1$. Somit ist (3.21) äquivalent zu

$$\min_{\substack{\mathcal{R} \in P_k \\ \mathcal{R}(0)=1}} \max_{\lambda \in [a,b]} | \mathcal{R}(\lambda) | \ . \tag{3.22}$$

Die Lösung von (3.22) ist zu einem vorgegebenem Polynomgrad $k \in \mathbb{N}$ explizit gegeben durch [3]:

$$\mathcal{R}(\lambda) = \frac{\mathcal{T}_k(\frac{a+b-2\lambda}{b-a})}{\mathcal{T}_k(\frac{a+b}{b-a})} , \tag{3.23}$$

wobei $\mathcal{T}_k(\xi)$ das k-te Tschebyscheff-Polynom ist.
Wegen der Skalierung und Normierung gilt $\mathcal{R}(0) = 1$ und für $\lambda \in [a, b]$:

$$\frac{a + b - 2\lambda}{b - a} \in [-1, 1]$$

Die Tschebyscheff-Polynome haben folgende Eigenschaften [11], [26]:

Explizite Darstellung:

$$\mathcal{T}_k(\xi) = (-1)^k \cosh(\, k \cdot \operatorname{arcosh}(-\xi)) \qquad \text{für } \xi < -1 \tag{3.24}$$

$$\mathcal{T}_k(\xi) = \cos(k \cdot \arccos \xi) \qquad \text{für } \xi \in [-1,1] \tag{3.25}$$

$$\mathcal{T}_k(\xi) = \cosh(\, k \cdot \operatorname{arcosh}(\xi)) \qquad \text{für } \xi > 1 \tag{3.26}$$

Rekursionsformel:

$$\mathcal{T}_k(\xi) = 2\xi\mathcal{T}_{k-1}(\xi) - \mathcal{T}_{k-2}(\xi) \qquad k \geq 2 \tag{3.27}$$

Orthogonalität:

$$\int_{-1}^{1} \mathcal{T}_i(\xi)\mathcal{T}_j(\xi) \cdot \frac{1}{\sqrt{1-\xi^2}} d\xi = 0 \qquad \text{für } i \neq j \tag{3.28}$$

Spezielle Werte:

$$\mathcal{T}_k(1) = 1, \mathcal{T}_k(-1) = (-1)^k \tag{3.29}$$

(3.30) $\mathcal{T}_k(\xi)$ hat k einfache Nullstellen, diese liegen in $[-1,1]$

$$\text{bei } \xi_i = \cos\left(\frac{2i+1}{k}\frac{\pi}{2}\right) \qquad i = 0, \ldots, k-1 \,,$$

(3.31) und $k+1$ Extrema mit dem Wert $(-1)^k$

$$\text{bei } \xi_i = \cos\left(\frac{i\pi}{k}\right) \qquad i = 0, \ldots, k \,.$$

Die ersten Polynome lauten:

$$\mathcal{T}_0(\xi) = 1, \quad \mathcal{T}_1(\xi) = \xi, \quad \mathcal{T}_2(\xi) = 2\xi^2 - 1 \,.$$

Diese Polynome erhalten für $a > 0$ die spd-Eigenschaft von $\mathcal{P}(A)$, denn:

Lemma 3.2 *Es gilt für* $0 < a \leq \lambda_{min} \leq \lambda_{max} \leq b$, $b \neq a$, $\mathcal{P}(A) = 1 - \mathcal{R}(A)$ *mit* $\mathcal{R}(\lambda)$ *gemäß (3.23):*

$\mathcal{P}(A)$ *ist positiv definit.*

Beweis: Aus (3.26) erhält man für $k \geq 1$: Ist $\xi > 1$, so ist auch $\mathcal{T}_k(\xi) > 1$. Insbesondere ist $\mathcal{T}_k(\frac{b+a}{b-a}) > 1$.

Weiterhin ist wg. (3.31) für $\lambda \in [a,b]$ $\mathcal{T}_k\left(\frac{a+b-2\lambda}{b-a}\right) \in [-1,1]$ und somit

$$\mathcal{P}(\lambda) = 1 - \mathcal{R}(\lambda) = 1 - \frac{\mathcal{T}_k\left(\frac{a+b-2\lambda}{b-a}\right)}{\mathcal{T}_k(\frac{b+a}{b-a})} > 0 \,. \tag{3.32}$$

Mit Lemma 3.1 folgt, daß $\mathcal{P}(A)$ positiv definit ist. √

Für die maximale Differenz der Funktionswerte von $\mathcal{P}(\xi)$ zu 1 auf $[a,b]$, einer oberen Schranke für die Differenz der Eigenwerte von $\mathcal{P}(A)$ zu 1, gilt mit (3.29) und (3.31):

$$\varepsilon_k := \max_{\lambda\in[a,b]} |\mathcal{R}(\lambda)| \;=\; \frac{\max_{\lambda\in[a,b]} |\mathcal{T}_k\left(\frac{a+b-2\lambda}{b-a}\right)|}{\mathcal{T}_k(\frac{b+a}{b-a})} = \frac{1}{\mathcal{T}_k(\frac{b+a}{b-a})} \,, \tag{3.33}$$

also insbesondere $\varepsilon_k < 1$. Der maximale Fehler wird in den Randwerten angenommen, da mit (3.29) gilt: $|\mathcal{T}_k(1)| = |\mathcal{T}_k(-1)| = 1$.

Für die Kondition des vorkonditionierten Systems gilt wg. (3.33):

$$\kappa(\mathcal{P}(A)) \;\leq\; \frac{1+\varepsilon_k}{1-\varepsilon_k} \,. \tag{3.34}$$

Für das Polynom $\mathcal{P}(\lambda) = 1 - \mathcal{R}(\lambda)$ gelten folgende aus den Tschebyscheff-Polynomen leicht herzuleitenden Eigenschaften:

(3.35) $\mathcal{P}(\lambda)$ ist positiv und streng monoton steigend
auf dem Intervall $(0,a]$.

(3.36) $\mathcal{P}(\lambda)$ ist für ungerades k positiv
und streng monoton steigend auf dem Intervall $[b,\infty)$.

(3.37) $\mathcal{P}(\lambda)$ ist für gerades k
streng monoton fallend auf dem Intervall $[b,\infty)$,
positiv auf dem Intervall $[0,b+a]$,
und negativ auf $(b+a,\infty)$.

(3.38) $\mathcal{P}(\lambda)$ hat $k+1$ paarweise verschiedene lokale Extrema
mit dem Wert $1 \pm \varepsilon_k$ auf dem Intervall $[a,b]$.

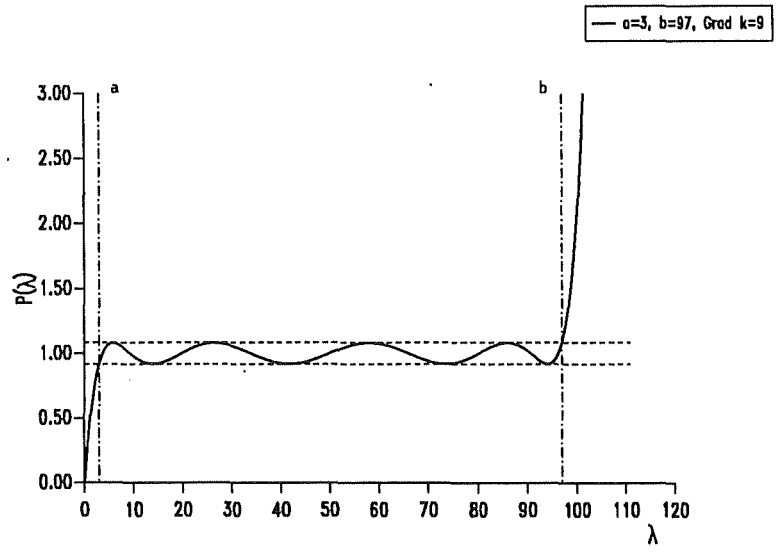


Abbildung 3.4: $\mathcal{P}(\lambda)$ für ungerades k

Zur Verdeutlichung sind in Abb. 3.4 und 3.5 Funktionsverläufe mit ungeradem bzw. geradem Polynomgrad dargestellt.

Aus der Rekursionsformel (3.27) ergibt sich für $\xi > 1$ induktiv die strenge Monotonie $\mathcal{T}_{k+1}(\xi) > \mathcal{T}_k(\xi)$ und aus (3.26) $\lim_{k\to\infty} \mathcal{T}_k(\xi) = \infty$, so daß ε_k streng monoton fallend in k und $\lim_{k\to\infty} \varepsilon_k = 0$ ist.

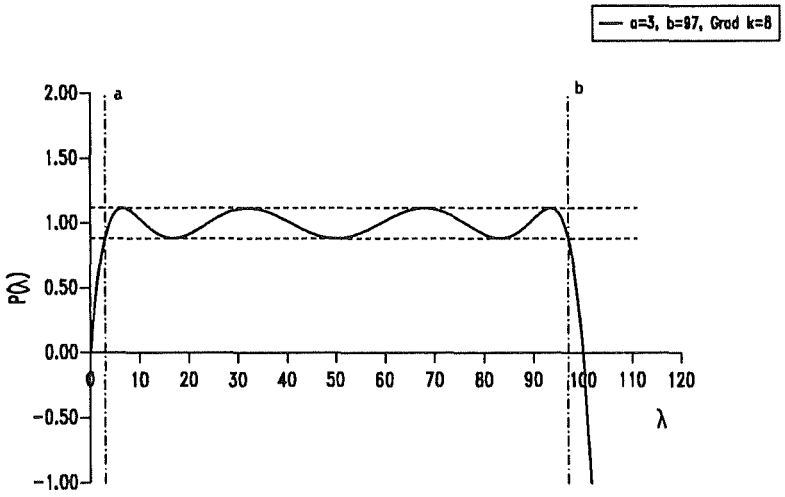


Abbildung 3.5: $\mathcal{P}(\lambda)$ für gerades k

Für den Grenzwert $a \to 0$ konvergiert $\varepsilon_k([a,b])$ gegen 1 $\forall k$. Aus (3.38) folgt, daß dieser Wert über dem Intervall $k+1$ mal angenommen wird. Insbesondere bei hohen Polynomgraden führt dies zu einer starken Oszillation. Es ist also Vorsicht geboten bei kleinen Werten von a. Die Empfindlichkeit der hier entwickelten Polynome $\mathcal{P}(\lambda) = 1 - \mathcal{R}(\lambda)$ für kleine Werte von a ist in Abb. 3.6 dargestellt.

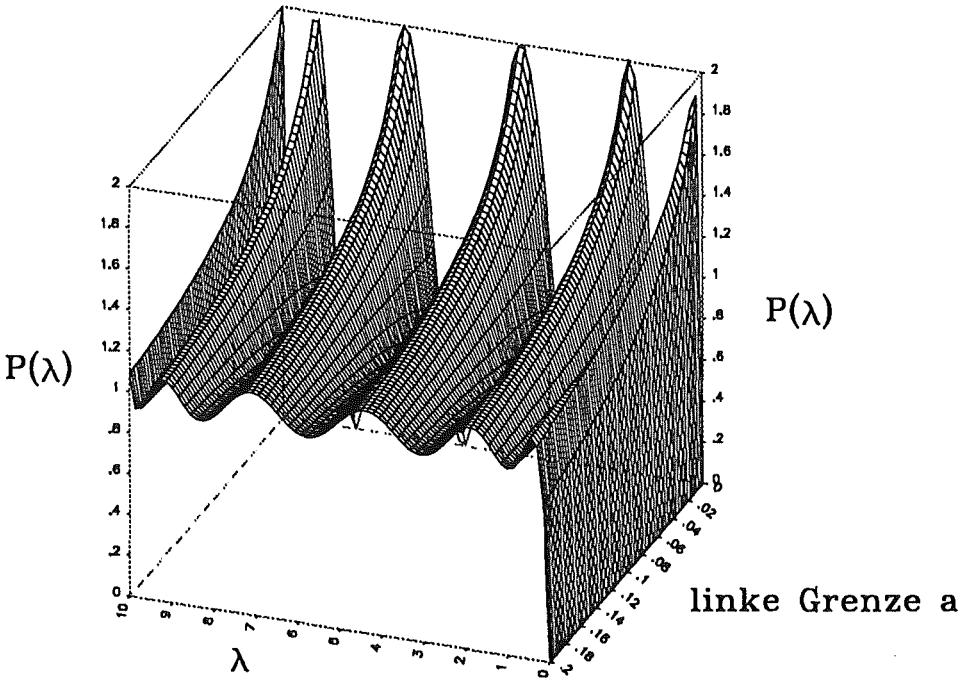


Abbildung 3.6: Abhängigkeit des Tschebyscheff-Vorkonditionierungspolynoms von der linken Intervallgrenze a

3.5.2 Praktische Durchführung

Auf den ersten Blick erscheint es aufwendig, diese Vorkonditionierung anzuwenden, denn es muß bei der Initialisierung u.a. $\mathcal{C}(A)v$ berechnet werden. Dazu müßten zuerst die Koeffizienten des Restpolynoms als Funktion des Tschebyscheff-Polynoms berechnet werden und dann die Koeffizienten von $\mathcal{C}(\lambda) = (1 - \mathcal{R}(\lambda))/\lambda$ bestimmt werden.

Dieses Vorgehen wird vermieden durch den im folgenden beschriebenen Zusammenhang zwischen einer Matrix-Vektor-Multiplikation $\mathcal{C}(A)z$ und einem polynomialen Beschleunigungsverfahren.

Ein Optimierungsverfahren zur Lösung von $Ax = v$, welches als neue Suchrichtung das Residuum der vorhergehenden Iteration verwendet, heißt **Richardson–Iteration**[4],[18]. Formal heißt dies:

$$r_j = v - Ax_j \,, \tag{3.39}$$

$$x_{j+1} = x_j + \tau_j r_j \,, \qquad \tau_j \neq 0 \,. \tag{3.40}$$

Das Residuum wiederum ist der negative Gradient der Funktion $F(x)$ gemäß (2.21); somit ist die Richardson–Iteration das Verfahren des steilsten Abstiegs mit einer Schrittweitenfolge τ_j für die zugrundeliegende Minimierungsaufgabe.

Zur Bestimmung des Richardson–Verfahrens ist es notwendig, die Folge τ_j zu kennen. Bei der praktischen Anwendung werden deshalb k Iterationsschritte mit vorgegebenem $\tau_0, \ldots, \tau_{k-1}$ durchgeführt und dann das Verfahren mit der letzten Iterierten als neuem Startvektor erneut gestartet.

Aus (3.39) und (3.40) ergibt sich

$$\begin{aligned} r_k &= v - A(x_{k-1} + \tau_{k-1} r_{k-1}) \\ &= \underbrace{v - Ax_{k-1}}_{r_{k-1}} - \tau_{k-1} A r_{k-1} \\ &= (I - \tau_{k-1} A) r_{k-1} \,, \end{aligned}$$

und induktiv

$$r_k = \prod_{j=0}^{k-1} (I - \tau_j A) r_0 \,.$$

Insbesondere ergibt eine Permutation der τ_j das gleiche $k-$te Residuum und somit auch die gleiche Iterierte x_k. Betrachtet man nur den letzten Wert eines Zyklus $x_{l \cdot k}$, $l \in \mathbb{N}$, so ist dieser Wert invariant gegenüber Permutationen der τ_j.

Mit der Bezeichnung

$$\mathcal{Q}_k(\lambda) := \prod_{j=0}^{k-1} (1 - \tau_j \lambda)$$

gilt:

$$\mathcal{Q}_k(0) = 1, \; r_k = \mathcal{Q}_k(A) r_0 \,. \tag{3.41}$$

Die Wahl der τ_j bzw. des Polynoms $\mathcal{Q}_k$ ist maßgebend für die Konvergenz des Verfahrens. Deshalb wird dieses Verfahren auch als **polynomiales Beschleunigungsverfahren** bezeichnet. $\mathcal{Q}_k(\lambda)$ heißt Residualpolynom der Richardson–Iteration. Für die Nullstellen μ_j dieses Polynoms gilt: $\mu_j = 1/\tau_j$.

Ist umgekehrt ein Polynom $\mathcal{S}_k(\lambda)$ k-ten Grades mit k reellen von Null verschiedenen Nullstellen gegeben, so wird hierdurch ein Richardson–Verfahren definiert, indem man als τ_j gerade die Inversen der Nullstellen des Polynoms verwendet. Für dieses Polynom gilt

$$\mathcal{S}_k(\lambda) = \mathcal{S}_k(0) \mathcal{Q}_k(\lambda) \,,$$

da $\mathcal{S}_k$ und $\mathcal{Q}_k$ k gleiche Nullstellen haben und für $\lambda = 0$ gilt:

$$\mathcal{S}_k(0)/\mathcal{S}_k(0) = \mathcal{Q}_k(0) = 1 \ .$$

Insbesondere gilt im Falle der Tschebyscheff-Beschleunigung $\mathcal{Q}_k(\lambda) = \mathcal{R}(\lambda)$.

Hieraus ergibt sich nun:

Satz 3.1 *Ist $\mathcal{Q}_k(A) = I - \mathcal{C}(A)A$ ein Polynom k-ten Grades, welches ein polynomiales Beschleunigungsverfahren definiert, so ist die k-te Iterierte des Verfahrens angewandt auf die Lösung des Gleichungssystems $Ax = z$ zum Startvektor $x_0 = 0$ gerade*

$$x_k = \mathcal{C}(A)z$$

Beweis: Es ist $r_0 = z - Ax_0 = z$ und mit (3.41) $r_k = \mathcal{Q}_k(A)z$. Wegen $r_k = z - Ax_k$ ergibt dies

$$x_k = A^{-1}(I - \mathcal{Q}_k(A))z = A^{-1}\mathcal{C}(A)Az = \mathcal{C}(A)z \ .$$

√

Zur Berechnung von $\mathcal{C}(A)z$ wird also das Richardson-Verfahren mit $\tau_j = \frac{1}{\mu_j}$ angewandt, wobei die μ_j gerade die Nullstellen des Restpolynoms sind.

Die Nullstellen der Tschebyscheffpolynome brauchen jedoch nicht explizit berechnet werden, denn es gilt folgende Äquivalenz:

Lemma 3.3 *Die k-te Iterierte des Richardson-Verfahrens, welches als τ_j die Inversen der Nullstellen des Tschebyscheff-Polynoms auf $[a,b]$ mit $0 < a < b$ zum Startvektor $x_0 = 0$ verwendet, ist gleich der k-ten Iterierten des Richardson-Verfahrens mit $\tau_j = 1$ $\forall j$ und anschließender Tschebyscheff-Beschleunigung [11] zum Startvektor $x_0 = 0$ auf dem Intervall $[a_0, b_0]$ mit $a_0 := 1 - b$ und $b_0 := 1 - a$.*

Beweis: Für den Beweis reicht es wegen der Invertierbarkeit von A, die Gleichheit der Residuen der k-ten Stufe zu zeigen. Das Restpolynom hängt gemäß (3.23) von a und b ab und wird deshalb in diesem Beweis zur Verdeutlichung mit diesen als zusätzlichem Index versehen.

Für das zuerst beschriebene Verfahren gilt wie oben gezeigt

$$r_k = \mathcal{R}_{k,a,b}(A)r_0 = \mathcal{R}_{k,a,b}(A)v \ .$$

Im Fall der Tschebyscheff-Beschleunigung auf die k-te Iterierte des Iterationsverfahrens ist

$$\begin{aligned} x_{j+1} &= x_j + r_j \\ &= x_j + v - Ax_j \\ &= Gx_j + v, \qquad G := I - A \ , \end{aligned}$$

und es gilt für die beschleunigte Iterierte y_k [11]:

$$y_k - x^* = -\mathcal{R}_{k,a_0,b_0}(I - A)x^* ,$$

wobei $\mathcal{R}_{k,a_0,b_0}(\lambda)$ das Polynom vom Grad k sei, welches $\mathcal{R}_{k,a_0,b_0}(1) = 1$ erfüllt und die Norm in (3.20) für $\Sigma = [a_0, b_0]$ minimiert. Wegen $a > 0$ gilt $a_0 < b_0 < 1$. Dieses Polynom ist explizit bekannt ([10], Satz 3.6.9):

$$\mathcal{R}_{k,a_0,b_0}(\lambda) = \frac{\mathcal{T}_k(\frac{2\lambda - b_0 - a_0}{b_0 - a_0})}{\mathcal{T}_k(\frac{2 - b_0 - a_0}{b_0 - a_0})} .$$

Es gilt nun, da $A(I-A)^j = (I-A)^j A$, $r_k = v - Ay_k$ und $x^* = A^{-1}v$:

$$\begin{aligned} -y_k &= \mathcal{R}_{k,a_0,b_0}(I - A)A^{-1}v - A^{-1}v , \\ -Ay_k &= \mathcal{R}_{k,a_0,b_0}(I - A)v - v , \\ r_k &= \mathcal{R}_{k,a_0,b_0}(I - A)v . \end{aligned}$$

Bleibt schließlich zu zeigen, daß

$$\mathcal{R}_{k,a,b}(\lambda) = \mathcal{R}_{k,a_0,b_0}(1 - \lambda), \quad \lambda \in [a, b] ,$$

denn hieraus folgt, da Polynome, die auf einem Intervall übereinstimmen, identisch sind,

$$\mathcal{R}_{k,a,b}(A) = \mathcal{R}_{k,a_0,b_0}(I - A) .$$

Es ist

$$\begin{aligned} \mathcal{R}_{k,a_0,b_0}(1 - \lambda) &= \frac{\mathcal{T}_k(\frac{2\cdot(1-\lambda) - b_0 - a_0}{b_0 - a_0})}{\mathcal{T}_k(\frac{2 - b_0 - a_0}{b_0 - a_0})} \\ &= \frac{\mathcal{T}_k(\frac{b + a - 2\lambda}{b - a})}{\mathcal{T}_k(\frac{b+a}{b-a})} = \mathcal{R}_{k,a,b}(\lambda) . \end{aligned} \tag{3.42}$$

√

Es sind also im Algorithmus zur Berechnung von $x = \mathcal{C}(A)v$ k Iterationen der Tschebyschef-Iteration (ausführlich behandelt in [11], Algorithmus 8.10 + Übung), also das Iterationsverfahren $x_{j+1} = (I - A)x_j + v$ zum Startvektor $x_0 = 0$ mit Tschebyscheff-Beschleunigung durchzuführen. Der Schätzwert für den kleinsten Eigenwert der Iterationsmatrix $G := I - A$ ist $1 - b$, wobei b Schätzwert für den größten Eigenwert von A ist. Der Schätzwert für den größten Eigenwert von G ist $1 - a$. Die letzte Iterierte ist das Ergebnis der durchzuführenden Matrix-Vektor-Multiplikation. Der Ausdruck $\mathcal{C}(A)Ap_k$ wird über $A \cdot \mathcal{C}(A)p_k$ analog behandelt.

Der zusätzliche Aufwand einer CG-Iteration besteht in $k - 1$ Matrix–Vektor–Produkten. Das k-te Residuum braucht nicht berechnet zu werden.

x = CHEBY (a,b,z,A,k)

$$\eta = \frac{a+b}{2}, \vartheta = \frac{b-a}{2}, \sigma = \frac{\eta}{\vartheta} ,$$
$$y_0 = 0, \quad y_1 = \eta^{-1} z,$$
$$\rho_1 = 2,$$
$$i = 2, \ldots, k$$

(3.43)
$$\rho_i = \frac{4}{4 - \sigma^2 \rho_{i-1}} ,$$
$$r_{i-1} = z - A y_{i-1} ,$$

(3.44)
$$y_i = \rho_i (y_{i-1} - y_{i-2} + \frac{1}{\eta} r_{i-1}) + y_{i-2} ,$$
$$x = y_k .$$

Abbildung 3.7: Algorithmus der Tschebyscheff-Iteration

Für die Berechnung von $x = C(A)z$ ergibt sich somit der in Abb. 3.7 dargestellte Algorithmus [11].

Hierbei ist (3.44) äquivalent zu

(3.45)
$$y_i = (\rho_i - 1) \underbrace{(y_{i-1} - y_{i-2})}_{dy_{i-1}} + \rho_i/\eta \; r_{i-1} + y_{i-1} ,$$

und somit gilt für die Differenz der y_i folgende Beziehung

$$dy_i = (\rho_i - 1) dy_{i-1} + \frac{\rho_i}{\eta} r_{i-1} .$$

Der zusätzliche Aufwand pro Iterationsschritt liegt im wesentlichen bei m Matrix-Vektor-Produkten und $3m$ Vektoradditionen für r_{i-1}, dy_i und y_i. Liegen Schätzwerte für die extremen Eigenwerte von A vor und ist ein Grad m zur polynomialen Vorkonditionierung vorgegeben, so ergibt sich mit dem modifizierten CG-Algorithmus (Seite 17) der in Abb. 3.8 dargestellte Algorithmus.

3.6 Schätzen der extremen Eigenwerte von A

Das verbleibende Problem liegt in der Bestimmung der Parameter a und b. Hier sind 3 verschiedene Möglichkeiten vorgesehen:

- Manuelle Vorgabe
- Automatische Vorgabe
- Adaptive Anpassung

Algorithmus des modifizierten CG-Verfahrens mit polynomialer Tschebyscheff-Vorkonditionierung

Geg. $x_0, a, b, m, A, \hat{r}_0 := v - Ax_0$
$r_0 = CHEBY(a, b, \hat{r}_0, A, m+1), p_0 := r_0$

Für $j := 0, 1, \ldots$

$$\hat{w}_j = CHEBY(a, b, p_j, A, m+1) \tag{3.46}$$

$$w_j = A\hat{w}_j$$

$$\alpha_j = \frac{r_j^T r_j}{p_j^T w_j} \tag{3.47}$$

$$\beta_j = \frac{\alpha_j w_j^T w_j}{p_j^T w_j} - 1 \tag{3.48}$$

$$r_{j+1}^T r_{j+1} = \beta_j r_j^T r_j \tag{3.49}$$

$$x_{j+1} = x_j + \alpha_j p_j \tag{3.50}$$

$$r_{j+1} = r_j - \alpha_j w_j \tag{3.51}$$

$$p_{j+1} = r_{j+1} + \beta_j p_j. \tag{3.52}$$

Bis das Konvergenzkriterium erfüllt ist.

Abbildung 3.8: Algorithmus des modifizierten CG-Verfahrens mit polynomialer Tschebyscheff-Vorkonditionierung

In (3.21) wurde bereits erwähnt, daß hier ein Ersatzproblem betrachtet wird, welches nicht die Eigenwerte von A, sondern alle Werte $\mathcal{P}(\xi)$ mit $\xi \in [a, b]$ möglichst gut zu 1 clustert. Selbst die exakte Kenntnis der extremen Eigenwerte, d.h. $a = \lambda_{min}$ und $b = \lambda_{max}$, liefert somit i.a. nur suboptimale Ergebnisse [27].

Die manuelle Vorgabe erlaubt es, vorhergehende Rechenläufe mit der gleichen Koeffizientenmatrix zu berücksichtigen oder Parameter durch Kenntnis von Eigenschaften der Matrix A vorzugeben.

Meist ist dies nicht der Fall, so daß eine Automatisierung im Programm vorgenommen werden muß, die dem Benutzer eine einfache Handhabung der polynomialen Vorkonditionierung erlaubt. Dies geschieht mit der automatischen Vorgabe mit

Hilfe der Gerschgorin-Abschätzung für die Eigenwerte bzw. mit der adaptiven Verbesserung der Schätzwerte. Die adaptive Methode ist nur für gerade Vorkonditionierungspolynome zulässig.

3.6.1 Automatische Vorgabe

Für die Koeffizientenmatrix A kann mit Hilfe der Gerschgorin-Abschätzung ein Intervall angegeben werden, welches das Spektrum von A enthält [15].

Diese Schätzwerte sind nicht immer brauchbar, da der Schätzwert für den kleinsten Eigenwert von A immer positiv sein muß. Weiterhin haben Untersuchungen bezüglich des optimalen Parameters a [27] gezeigt, daß eine Wahl von a, die etwas größer als der minimale Eigenwert ist, häufig besser als $a = \lambda_{min}$ ist.

Ist der mit der Gerschgorin-Abschätzung gewonnene Schätzwert für den kleinsten Eigenwert von A positiv, – dies ist insbesondere für streng diagonaldominante Matrizen der Fall – so kann dieser zur Bestimmung des Intervalls herangezogen werden. Ist dieser Wert jedoch negativ, so hilft folgende Vorgehensweise, einen Schätzwert zu gewinnen:

Aus (3.33) ergibt sich, daß zu gegebenen Intervallgrenzen $a > 0$ und b mit $b > a$ ein Grad k des Polynoms gefunden werden kann, so daß die maximale Abweichung der transformierten Eigenwerte zu 1 kleiner als ein beliebiges $\delta := \varepsilon_k$ ist.

Dies motiviert folgende Überlegung:

Zu einem vorgegebenem Polynomgrad k und einem Fehler δ, der auf einem Intervall $[a, b]$ nicht überschritten werden darf, gilt mit (3.26) und (3.33) ausgehend von $\varepsilon_k \leq \delta$ und der Beziehung

$$\frac{a+b}{b-a} = 1 + \frac{2}{\frac{b}{a} - 1} \quad :$$

$$\frac{1}{\delta} \leq \cosh\left(k \cdot \operatorname{arcosh}\left(\frac{a+b}{b-a}\right)\right)$$

$$\frac{b}{a} - 1 \leq \frac{2}{\cosh\left(\frac{1}{k} \operatorname{arcosh}\left(\frac{1}{\delta}\right)\right) - 1}$$

$$\frac{b}{a} \leq \underbrace{1 + \frac{2}{\cosh\left(\frac{1}{k} \operatorname{arcosh}\left(\frac{1}{\delta}\right)\right) - 1}}_{\Phi_{k,\delta}} \tag{3.53}$$

Zur Illustration sollen hier einige Werte von $\Phi_{k,\delta}$ in Tabelle 3.1 angegeben werden.

	$\delta = 0.1$	$\delta = 0.25$	$\delta = 0.5$	$\delta = 0.75$	$\delta = 0.9$
$k = 1$	1.2	1.7	3.0	7.0	19.0
$k = 2$	2.5	4.4	9.9	26.0	74.0
$k = 3$	4.7	9.1	21.4	57.6	165.7
$k = 4$	7.8	15.7	37.6	101.8	294.0
$k = 5$	11.8	24.2	58.3	158.7	458.9
$k = 7$	22.6	46.7	113.7	310.5	898.8
$k = 9$	36.8	76.8	187.5	512.9	1485.2

Tabelle 3.1: Darstellung der Funktion Φ für einige Werte von δ und k.

Liegen Schätzwerte $\hat{a}$ und $\hat{b}$ für die Intervallgrenzen vor, so läßt sich durch Einsetzen in (3.53) überprüfen, ob die Fehlerschranke δ auf dem Intervall eingehalten werden kann.

In der Regel wird dies insbesondere bei kleinen Polynomgraden jedoch nicht der Fall sein. Bestand die bisherige Überlegung darin, den Polynomgrad zu erhöhen, um die Fehlerschranke einzuhalten, so wird nun das Intervall, auf dem der Fehler eingehalten wird, verkleinert.

Für b wählt man den mit der Gerschgorin-Abschätzung gewonnenen Schätzwert für den größten Eigenwert, für a den entsprechenden, falls dieser positiv ist. Falls nicht, so ergibt sich aus (3.53) zu vorgegebenem $k \in \mathbb{N}$ und $0 < \delta < 1$ für $\varepsilon_k = \delta$

$$a = \frac{b}{\Phi_{k,\delta}} . \tag{3.54}$$

Die Eigenwerte im Intervall $[0, a)$ werden bei der Clusterung nicht berücksichtigt. Sie bleiben jedoch positiv und erzeugen, falls i Eigenwerte in diesem Intervall liegen, maximal i zusätzliche Iterationen.

Schlechte Konvergenz ist zu erwarten, falls viele Eigenwerte in diesem Intervall liegen. Deshalb sollte die Fehlerschranke δ nicht zu klein gewählt werden, um nicht zu viele Eigenwerte außerhalb des zur Clusterung benutzten Intervalls zu erhalten. Der Grad des Polynoms k sollte auch nicht zu groß sein, da in jedem Schritt k Matrix-Vektor-Produkte berechnet werden müssen.

Sind die Diagonaleinträge der Matrix A groß im Vergleich zu den Nicht-Diagonalelementen, so führt diese Automatisierung zu guten Ergebnissen.

In vielen praktischen Fällen ist dies jedoch nicht der Fall, so daß die Schätzwerte sehr ungenau sind und ein Verfahren zur besseren Bestimmung der Eigenwerte notwendig ist.

3.6.2 Adaptive Eigenwertschätzung

Ziel dieses Abschnitts ist es, ein Verfahren herzuleiten, welches die Eigenwerte der Matrix A in allgemeineren Fällen gut schätzt und insbesondere im Laufe der Iteration verbesserte Eigenwertschätzungen liefert.

Grundlage des adaptiven Verfahrens ist der Zusammenhang zwischen dem CG-Verfahren und dem Lanzcos-Verfahren zur Lösung des symmetrischen Eigenwertproblems, der es erlaubt, aus den Iterationsparametern α_k und β_k eine Näherung für die Eigenwerte der aktuellen Matrix $\mathcal{P}(A)$ zu gewinnen.

Die Theorie der Lanzcos-Verfahren soll hier nicht vollständig behandelt werden. Es sollen nur einige wesentliche Resultate, die im Zusammenhang mit dem CG-Verfahren stehen, erläutert werden. Ausführliche Darstellungen hierzu finden sich in [11],[15].

Das Lanzcos-Verfahren erzeugt zu einer reellen symmetrischen Matrix M eine Folge (T_j) von symmetrischen $j \times j$ Tridiagonalmatrizen, deren Eigenwerte Näherungen der Eigenwerte von M sind. Gewöhnlich werden zunächst die extremen Eigenwerte von M gut approximiert.

Ausgehend von der Ungleichung (2.53) und der Gleichheit in (2.53) für die Eigenvektoren läßt sich das Eigenwertproblem wie folgt als Optimierungsaufgabe formulieren [11]:

$$\mu_{min} = \min_{y \in \mathbb{R}^n \setminus \{0\}} \frac{y^T M y}{y^T y}, \tag{3.55}$$

$$\mu_{max} = \max_{y \in \mathbb{R}^n \setminus \{0\}} \frac{y^T M y}{y^T y}. \tag{3.56}$$

$\frac{y^T M y}{y^T y}$ wird als *Rayleigh-Quotient* bezeichnet.

Die Minimierung wird nicht über den ganzen $\mathbb{R}^n$ durchgeführt, sondern – ähnlich wie beim CG-Verfahren in Kapitel 2.3 – iterativ über wachsende lineare Teilräume, die Krylov-Unterräume

$$< q_1, Mq_1, \ldots, M^{j-1}q_1 >$$

gelöst, indem man orthonormale Vektoren q_i bestimmt mit $Q_j = (q_1, \ldots, q_j)$, so daß

$$T_j := Q_j^T M Q_j$$

eine Tridiagonalmatrix ist.

Die extremen Eigenwerte von T_j sind gerade die Lösungen der Minimierungs- bzw. Maximierungsaufgabe über $V_j :=< q_1, \ldots, q_j >=< q_1, Mq_1, \ldots, M^{j-1}q_1 >$.

Insbesondere ergeben sich für die Zwischenlösungen aus der Eigenschaft wachsender Teilräume

$$\mu_{min}(M) \leq \mu_{min}(T_j) \leq \mu_{min}(T_{j-1}) \leq \ldots \leq \mu_{max}(T_{j-1}) \leq \mu_{max}(T_j) \leq \mu_{max}(M) .$$

Der Zusammenhang zum CG-Verfahren ist der folgende.

Die Residuen werden im CG-Verfahren paarweise konjugiert und sind somit orthogonal. Zur Normierung wird die euklidische Norm verwendet, und man erhält mit $q_i := \frac{r_i}{\| r_i \|}$, $i = 1, \ldots, j$ eine orthonormale Basis für V_j.

Im folgenden bezeichne $\mathrm{diag}(\xi_1, \ldots, \xi_j)$ eine Diagonalmatrix, deren $i-$tes Diagonalelement ξ_i ist.

Die Vektoren q_i erzeugen durch $Q_j^T M Q_j$ eine Tridiagonalmatrix, denn mit

$$B_j = \begin{pmatrix} 1 & -\beta_0 & 0 & 0 \\ 0 & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & -\beta_{j-1} \\ 0 & \cdots & 0 & 1 \end{pmatrix} ,$$

$$\begin{aligned} R_j &= (r_0, \ldots, r_{j-1}), \\ P_j &= (p_0, \ldots, p_{j-1}), \\ D_j &= \mathrm{diag}(1/\| r_0 \|, \ldots, 1/\| r_{j-1} \|) \end{aligned}$$

gilt wg. $p_j = r_j + \beta_{j-1} p_{j-1}$ für R_j die Darstellung

$$R_j = P_j B_j .$$

Es ist $Q_j = R_j D_j$ und wegen der $M-$Konjugation der p_i

$$P_j^T M P_j = \mathrm{diag}(p_0^T M p_0, \ldots, p_{j-1}^T M p_{j-1}) .$$

Somit ergibt sich für die Matrix T_j:

$$\begin{aligned} T_j &= Q_j^T M Q_j = D_j R_j^T M R_j D_j \\ &= D_j B_j^T P_j^T M P_j B_j D_j \\ &= D_j B_j^T \mathrm{diag}(p_0^T M p_0, \ldots, p_{j-1}^T M p_{j-1}) B_j D_j \end{aligned} \tag{3.57}$$

T_j ist, da nur in B_j eine Nebendiagonale steht und es sich sonst ausschließlich um Diagonalmatrizen handelt, eine symmetrische Tridiagonalmatrix.

Die Diagonalelemente dieser Matrix sind

$$\begin{aligned} T_j(1,1) &= \frac{1}{\alpha_0} , \\ T_j(i,i) &= \frac{1}{\alpha_{i-1}} + \beta_{i-2} \frac{1}{\alpha_{i-2}} \qquad i = 2, \ldots, j . \end{aligned} \tag{3.58}$$

Für die Nebendiagonale gilt

$$(3.59) \qquad T_j(i+1,i) = T_j(i,i+1) \;=\; -\sqrt{\beta_{i-1}}\frac{1}{\alpha_{i-1}} \qquad i = 1,\ldots,j-1\,.$$

Somit ist die Matrix T_j, die im Lanzcos-Algorithmus erzeugt wird, beim CG-Verfahren implizit bekannt. Es brauchen nur die Iterationsparameter α_i und β_i gemäß (3.58) und (3.59) gespeichert zu werden und nach einer vorgegebenen Anzahl von Schritten die Eigenwerte von T_j berechnet zu werden. Diese Eigenwerte sind nun Approximationen für die Eigenwerte von $\mathcal{P}(A)$.

Das verbleibende Problem besteht darin, daß zur Aktualisierung des Intervalls $[a,b]$ Schätzwerte für die Eigenwerte von A benötigt werden.

Im folgenden wird eine Einschränkung des Polynomgrads auf ungerade k für $\mathcal{R}(\lambda)$ bzw. $\mathcal{P}(\lambda)$, also gerades m für die Vorkonditionierungsmatrix $\mathcal{C}(\lambda)$, aus zwei Gründen gemacht [3].

Zum einen gewährleistet dies bei positiven Schätzwerten für die extremen Eigenwerte, die nicht das ganze Spektrum enthalten, die positive Definitheit. Dies zeigt Abb. 3.5 auf Seite 33, da lokale Extrema nur im Intervall $[a,b]$ angenommen werden und mit (3.37) gilt: $\lim_{\lambda\to\infty}\mathcal{P}(\lambda) = +\infty$.

Bei geradem k ist dies, wie Abb. 3.4 zeigt, nicht gewährleistet, da für $\lambda > b+a$ $\mathcal{P}(\lambda)$ negativ wird, wie bereits in (3.37) bemerkt.

Ein weiterer Vorteil ungerader Werte für k liegt darin, daß zu jedem positiven Funktionswert, welcher nicht im Intervall $[1-\varepsilon_k, 1+\varepsilon_k]$ liegt, ein eindeutig bestimmtes Urbild existiert.

Somit läßt sich aus jedem positiven Schätzwert für einen Eigenwert μ von $\mathcal{P}(A)$, der größer als $1+\varepsilon_k$ oder kleiner als $1-\varepsilon_k$ ist, ein Schätzwert des ursprünglichen Eigenwertes rekonstruieren, denn:

Ist $\mu < 1-\varepsilon_k$, so ist $\lambda < a$ und $\frac{a+b-2\lambda}{b-a} > 1$ und es gilt mit (3.26) und (3.32):

$$(3.60) \qquad \mu \;=\; 1-\varepsilon_k \cdot \cosh\left(k \cdot \operatorname{arcosh}\left(\frac{a+b-2\lambda}{b-a}\right)\right)$$

und somit existiert das eindeutige Urbild

$$(3.61) \qquad \lambda = \frac{1}{2}\left[(a+b) - (b-a)\cosh\left(\frac{1}{k}\cdot \operatorname{arcosh}\left(\frac{1-\mu}{\varepsilon_k}\right)\right)\right]$$

und für $\mu > 1+\varepsilon_k$ ist $\lambda > b$ bzw. $\dfrac{a+b-2\lambda}{b-a} < -1$ und es gilt, da k ungerade mit (3.24) und (3.23):

$$\mu = 1 + \varepsilon_k \cdot \cosh\left(k \cdot \operatorname{arcosh}\left(-\frac{a+b-2\lambda}{b-a}\right)\right) \tag{3.62}$$

und somit für diesen Fall

$$\lambda = \frac{1}{2}\left[(a+b) + (b-a)\cosh\left(\frac{1}{k} \cdot \operatorname{arcosh}\left(\frac{\mu-1}{\varepsilon_k}\right)\right)\right] . \tag{3.63}$$

Es ergibt sich für $\mu = 2$ das Urbild $\lambda = a + b$.

Ausgehend von einem Intervall $[a, b]$ mit $\lambda_{min} < a < b < \lambda_{max}$ kann wie folgt vorgegangen werden. Ist ein Lanzcos-Schätzwert für einen Eigenwert von $\mathcal{P}(A)$ kleiner als $1 - \varepsilon_k$, so kann gemäß (3.61) ein Schätzwert für a, der kleiner ist als der bisherige, gewonnen werden und das Intervall $[a, b]$ expandiert werden. Entsprechendes gilt mit (3.63) für b, falls der Lanzcos-Eigenwert größer als $1 + \varepsilon_k$ ist.

Das Verfahren verläuft somit adaptiv. Nach einem vorgegebenem Zyklus werden die extremen Eigenwerte der T_j berechnet und geprüft, ob diese im Intervall $[1-\varepsilon_k, 1+\varepsilon_k]$ liegen. Ist dies der Fall, so wird weiterhin mit den gleichen Werten von a und b gerechnet, ansonsten der Wert aktualisiert.

Das verbleibende Problem besteht in der Initialisierung von a und b. Wegen der Eigenschaft der Expansion des Intervalls wird ein Startintervall benötigt, welches $\lambda_{min} < a < b < \lambda_{max}$ erfüllt. Dieses kann wie folgt erreicht werden. Zu Beginn werden l Iterationsschritte ohne Vorkonditionierung durchgeführt. Dabei wird wiederum die Lanzcos-Matrix aufgestellt und man erhält eine $j \times j$ Matrix deren Eigenwerte innerhalb des Spektrums von A liegen. Dieser Vorlauf zur Eigenwertschätzung ist somit das CG-Verfahren ohne Vorkonditionierung. Die letzte Iterierte kann als verbesserter Startvektor verwendet werden.

Kapitel 4

Implementierung

In diesem Kapitel werden die für das Verständnis wichtigen Begriffe der Parallelverarbeitung und die Parallelrechner iPSC/860 und PARAGON XP/S 10 der Firma Intel vorgestellt. Anschließend werden die in der Implementierung verwendete Speichertechnik sowie die Datenverteilung und das verwendete Kommunikationsschema kurz beschrieben.

4.1 Parallelverarbeitung

Im Gegensatz zur klassischen Zielsetzung, auf immer schnelleren Prozessoren zu rechnen, ist die Vorgehensweise der Parallelverarbeitung, die Anzahl der zur Lösung des Problems verwendeten Prozessoren zu erhöhen.

Dies erfordert eine Zerlegung des Gesamtproblems in Teilaufgaben. Mittlerweile existieren eine Vielzahl von unterschiedlichen Rechnerarchitekturen und Programmiermodellen, wobei in dieser Arbeit keine Übersicht über den aktuellen Stand gegeben werden kann, sondern nur die Merkmale der verwendeten Parallelrechner klassifiziert werden.

4.1.1 Grundlegende Begriffe

Zu Beginn wird eine kurze Übersicht der zum Verständnis notwendigen Begriffe der Parallelverarbeitung gegeben.

Im optimalen Fall läßt sich ein Programm gleichmäßig auf p Prozessoren mit p disjunkten Teilaufgaben aufteilen. Gleichmäßig bedeutet hierbei, daß jeder Prozessor eine etwa gleich lange Bearbeitungszeit für sein Teilproblem hat. Dieses wird mit dem Begriff **Load Balancing** bezeichnet.

In diesem Fall ist zu erwarten, daß sich die Ausführungszeit des parallelen Programms t_p aus der Zeit des sequentiellen Programms t_s ergibt durch

$$\frac{t_s}{p} = t_p \ , \tag{4.1}$$

i.a. gilt jedoch lediglich

$$\frac{t_s}{p} \leq t_p \ , \tag{4.2}$$

da im parallelen Programm zusätzlicher Aufwand durch das Zusammenfassen der Teillösungen entsteht, redundante Berechnungen, d.h. gleiche Berechnungen, die auf jedem Prozessor separat ausgeführt werden, und Datenaustausch benötigt werden.

Aus (4.2) ergibt sich

$$S(p) := \frac{t_s}{t_p} \ \leq \ p \ . \tag{4.3}$$

$S(p)$ ist der **Speedup** des Verfahrens bei p Prozessoren und ein Maß für den proportionalen Zeitgewinn. Wünschenswert ist ein Speedup $S(p)$, der möglichst nahe bei dem Optimum p liegt. Die Größe

$$E = \frac{S(p)}{p} \in (0,1]$$

wird als **Effizienz** bezeichnet.

Die Effizienz wird gewöhnlich in Prozent angegeben. Eine hohe Effizienz bedeutet auch eine hohe Parallelisierbarkeit des Verfahrens für die gewählte Anzahl Prozessoren. Ist die Effizienz für eine beliebige Anzahl Prozessoren hoch, so wird dieser Effekt als **Skalierbarkeit** des Verfahrens bezeichnet.

Bei den hier betrachteten Rechnern handelt es sich um Systeme mit **verteiltem Speicher** (distributed memory). Dies bedeutet, daß es keinen gemeinsamen Adreßraum für die Prozessoren gibt, sondern jeder Prozessor über eine eigene lokale Speichereinheit verfügt. Der Prozessor und sein Speicher werden als logische Einheit (**Knoten**) betrachtet.

Benötigt ein Prozessor Daten eines anderen, so muß ein Datenaustausch – **Kommunikation** – durchgeführt werden. Bei Systemen mit verteiltem Speicher muß dem Prozessor explizit mitgeteilt werden, welche Nachricht er an welchen Prozessor verschicken soll bzw. von welchem Prozessor zu erwarten hat.

Dieses Programmiermodell wird als **Message Passing** bezeichnet.

Message Passing kann auf zwei Arten stattfinden. Bei der **synchronen** bzw. **blockierenden** Kommunikation wartet der Prozessor, bis die Nachricht bei dem empfangenden Prozessor angekommen ist und dieser den Empfang bestätigt. Dies

sichert die korrekte Bearbeitungsreihenfolge der Operationen. Bei der **asynchronen** bzw. **nicht-blockierenden** Kommunikation wird nach dem Verschicken der Nachricht direkt, also ohne Kontrolle, ob diese angekommen ist, weitergerechnet. Die asynchrone Kommunikation benötigt in der Regel weniger Aufwand, birgt jedoch mehr Gefahren für die korrekte Bearbeitung des Algorithmus.

Ein weiteres Charakteristikum eines Parallelrechners ist das Verbindungsnetzwerk der Prozessoren, die **Topologie** des Systems.

4.1.2 Die Rechner iPSC/860 und PARAGON XP/S 10 der Firma Intel

Ausführliche Beschreibungen dieser Rechner finden sich in den Handbüchern des Herstellers zu den Rechnern [34],[35]. Hier sollen die wesentlichen Daten der im Forschungszentrum Jülich installierten Systeme gegenübergestellt werden.

Beide Rechner sind massiv-parallele Systeme mit verteiltem Speicher der Firma Intel.

	iPSC/860	Paragon XP/S 10
Anzahl Prozessoren	32	140
Prozessortyp	Intel i860XR	Intel i860XP
Verbindungsstruktur	Hypercube	2-dim. Gitter
Leistung pro Prozessor	60 MFLOPS	75 MFLOPS
Speicher pro Prozessor	16 MByte	32 MByte
Overall Peak Performance	1.9 GFLOPS	10.5 GFLOPS
Betriebssystem Version	NX/2 3.3.2	OSF/1 1.0.4
Plattenspeicher	2.3 GByte	28 GByte

Tabelle 4.1: Daten der verwendeten massiv-parallelen Rechner

4.2 Allgemeines zur Implementierung

Wie in Kapitel 3.5 und 3.6 dargestellt, ergibt sich die polynomiale Vorkonditionierung als Erweiterung eines implementierten CG–Algorithmus durch wenige Modifikationen des Algorithmus. Diese Modifikationen lassen sich leicht bei jedem bereits implementierten CG–Verfahren durchführen.

Somit ist es nicht entscheidend, welche spezielle Implementierung vorgenommen wurde. Anstatt des Matrix-Vektor-Produkts mit der Koeffizientenmatrix A muß zusätzlich eine Iteration, die im wesentlichen aus einer Folge von Matrix-Vektor-Produkten mit A besteht, integriert werden. Es ist keine Abspeicherung der Matrix $\mathcal{P}(A)$ nötig. Die Aktualisierung der Eigenwerte im adaptiven Fall kann ebenfalls problemlos in den Algorithmus eingefügt werden. Die Eigenwerte der dort entstehenden Tridiagonalmatrix werden mit einer NAG-Routine (F02BFF) bestimmt.

Die hier zugrundeliegende Implementierung des CG-Verfahrens stammt von A. Basermann [7]. Unabhängig von der Parallelität waren hierbei im wesentlichen folgende zwei Fragestellungen zu beantworten:

- Welche Speichertechnik wird für die Matrix A verwendet und wie ergibt sich hieraus das Matrix-Vektor-Produkt ?
- Welche Abbruchkriterien werden bereitgestellt ?

4.2.1 Die eindimensionale Speichertechnik

A. Basermann [7] hat zwei verschiedene Speichertechniken, eine eindimensionale und eine zweidimensionale, implementiert und diese verglichen. Die zweidimensionale erwies sich als in der Regel speicherintensiver, insbesondere bei stark variierender Zahl von Nicht-Nullelementen pro Zeile, die bei einem sehr unregelmäßigen Diskretisierungsgitter entstehen. Von den Ausführungszeiten des Algorithmus erwies sich die eindimensionale Technik der zweidimensionalen als nicht unterlegen. Somit wurde hier die eindimensionale Technik verwendet, die im folgenden kurz vorgestellt wird.

Die Idee dieser Speichertechnik ist die Speicherung der Matrix in drei eindimensionalen Vektoren. Der ersten beiden Vektoren haben soviel Elemente, wie die Matrix von Null verschiedene Einträge hat. Die Elemente des ersten Vektors, dem Werte-Vektor $val(i)$, sind gerade die Einträge der Matrix. Hierbei ist die Matrix zeilenweise abgespeichert. Innerhalb einer Zeile müssen die Komponenten nicht mit aufsteigendem Index geordnet sein. Dies unterstützt die in FE-Anwendungen assemblierten Matrizen.

Der zweite Vektor, der Spaltenindex-Vektor $colind(i)$, enthält die entsprechenden Spaltenindizes der Nicht-Nullelemente.

Schließlich wird ein Vektor benötigt, der den Beginn der einzelnen Zeilen in den ersten beiden Vektoren liefert. Dies ist der Zeilen-Zeiger $rowptr(i)$. Die Länge ist

um eins größer als die Anzahl der Zeilen der Matrix, welches die Bestimmung der Anzahl Nicht-Nullelemente pro Zeile vereinfacht. Der i-te Wert dieses Feldes gibt den Beginn der i-ten Zeile in den ersten beiden Vektoren an. Der letzte Wert des Vektors ist um eins größer als die Anzahl Nicht-Nullelemente, welche im folgenden mit e bezeichnet wird.

$rowptr(i+1) - rowptr(i)$ gibt somit an, wieviele Nicht-Nullelemente die i-te Zeile enthält.

An einem einfachen Beispiel soll diese Speichertechnik kurz erläutert werden. Dieses Beispiel besitzt dieselbe Besetzungstruktur wie das verwendete Beispiel in [7], um die gleiche Datenverteilung und somit das gleiche Kommunikationsschema zu erhalten, jedoch wurde die Diagonale so verändert, daß die Matrix positiv definit ist.

Gegeben sei die 8×8-Matrix

$$A = \begin{pmatrix} 10 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 20 & 9 & 0 & 0 & 0 & 0 & 0 \\ 0 & 9 & 30 & 10 & 0 & 0 & 0 & 0 \\ 0 & 0 & 10 & 90 & 11 & 14 & 12 & 18 \\ 0 & 0 & 0 & 11 & 40 & 0 & 17 & 0 \\ 0 & 0 & 0 & 14 & 0 & 50 & 15 & 0 \\ 0 & 0 & 0 & 12 & 17 & 15 & 60 & 0 \\ 0 & 0 & 0 & 18 & 0 & 0 & 0 & 70 \end{pmatrix} . \tag{4.4}$$

Die Werte können z.B. in der Reihenfolge abgelegt werden, daß pro Zeile das Diagonalelement vorne steht. Dies ergibt für den Wertevektor *val*

(10 | 20 9 | 30 9 10 | 90 10 11 14 12 18 | 40 11 17 | 50 14 15 | 60 12 17 15 | 70 18) .

Zu jedem Wert wird im zweiten Vektor sein Spaltenindex eingetragen, und man erhält

$$colind = (1 \mid 2\ 3 \mid 3\ 2\ 4 \mid 4\ 3\ 5\ 6\ 7\ 8 \mid 5\ 4\ 7 \mid 6\ 4\ 7 \mid 7\ 4\ 5\ 6 \mid 8\ 4\ ) .$$

Schließlich gibt der dritte Vektor an, an welcher Stelle in den ersten beiden Vektoren eine neue Zeile beginnt:

$$rowptr = (1\ 2\ 4\ 7\ 13\ 16\ 19\ 23\ 25\ ) .$$

Mit Hilfe dieser Speichertechnik kann ein Matrix-Vektor-Produkt ausschließlich mit Operationen auf den Nicht-Nullelementen wie folgt beschrieben werden:

Die Lösung w des Matrix-Vektor-Produktes der $n \times n$–Matrix A mit dem Vektor p ergibt sich aus

```
DO i=1,n
   w(i) = 0
   DO j=rowptr(i),rowptr(i+1)-1
      w(i) =  w(i) + val(j) * p(colind(j))
   END DO
END DO
```

4.2.2 Abbruchkriterien

In den hier betrachteten Testbeispielen wurden zwei verschiedene Abbruchkriterien verwendet. Zum einen ist das *Quadrat der euklidischen Norm des Residuums* des j-ten Iterationsschrittes $r_j^T r_j$ im Algorithmus bekannt. Wie bereits erwähnt, ist dieser Norm schwer anzusehen, wie gut die Lösung bereits ist. Häufig ist die Norm noch recht groß, obwohl der Iterationsvektor bereits sehr nahe bei der Lösung liegt oder die Norm bereits sehr klein, obwohl die Lösung noch nicht hinreichend genau ist.

Alternativ hierzu ist ein Abbruchkriterium vorgesehen, welches sich auf die relativen Änderungen im Iterationsvektor bezieht. Von zwei aufeinanderfolgenden Iterierten wird die Komponentendifferenz ermittelt und dieser Wert durch das arithmetische Mittel der Summe der Absolutbeträge dieser Komponenten dividiert. Das Maximum dieser Größen wird im folgenden als *maximal skalierte absolute Differenz* δ_{skal} [8] bezeichnet, also

$$\delta_{skal} = \max_{1 \le i \le n} 2 \frac{|(x_{j+1})_i - (x_j)_i|}{|(x_{j+1})_i| + |(x_j)_i|} ,$$

wobei $(x)_i$ die i-te Komponente des Vektors x bezeichnet. Somit werden Änderungen in kleinen Werten der Komponenten des Vektors stärker gewichtet als Änderungen in großen Werten. Dies ist insbesondere in den Fällen von Bedeutung, in denen die Komponenten des Lösungsvektors sehr unterschiedliche Größenordnung haben.

4.3 Parallelisierung des CG-Verfahrens

Die Frage ergibt sich, welche Teile des CG-Verfahrens sich parallelisieren lassen. Die nicht-skalaren Operationen des CG-Verfahrens sind das Matrix-Vektor-Produkt, die Skalarprodukte und die Vektoradditionen.

Vektoradditionen können sehr gut parallelisiert werden, da jeder Prozessor einen beliebigen Teil der Komponenten aller Vektoren erhält und somit die Additionen lokal ohne Kommunikation ausgeführt werden können.

Auch bei der Berechnung des Skalarprodukts kann ein Großteil der Berechnung lokal ausgeführt werden. Jeder Prozessor berechnet die zu seinen Komponenten gehörende Teilsumme des Skalarproduktes. Anschließend werden die Teilsummen aufaddiert,

und das Resultat ist das gesuchte Skalarprodukt. Da dieses Skalarprodukt von allen Prozessoren benötigt wird, müssen hierbei alle Prozessoren miteinander kommunizieren.

Der Programmier- und Rechenaufwand von Vektoraddition und Skalarprodukt erweist sich als gering. Der Aufwand der Kommunikation ist beim Skalarprodukt unter Umständen sehr hoch, falls viele Prozessoren verwendet werden.

Der wesentliche Aufwand beim CG-Verfahren wie auch bei dessen Parallelisierung liegt in der Berechnung des Matrix-Vektor-Produkts mit der Koeffizientenmatrix A, da sehr viele Operationen ausgeführt werden und die Prozessoren miteinander kommunizieren müssen. In der Regel muß in diesem Fall ein Prozessor nicht mit allen anderen, sondern nur mit einigen Prozessoren kommunizieren. Der Programmierungsaufwand ist bedeutend höher, da wegen der Verteilung der Matrix und der Vektoren auf die Prozessoren ein effizienter Nachrichtenaustausch während des Matrix-Vektor-Produkts stattfinden muß.

4.3.1 Datenverteilung auf die Prozessoren

Bei der Parallelisierung auf einem System mit verteiltem Speicher, wie es hier zugrundeliegt, ist zu Beginn eine effiziente Verteilung der Koeffizientenmatrix und der Vektoren auf die Prozessoren durchzuführen.

Hierzu werden den einzelnen Prozessoren aufeinander folgende vollständige Zeilen der Matrix und die zugehörigen Komponenten der benutzten Vektoren zugewiesen. Die Anzahl der Zeilen, die jeder Prozessor erhält, kann in dieser Implementierung nach unterschiedlichen Kriterien bestimmt werden. Untersucht wurden in [7] die Kriterien

- gleich viele Zeilen,
- gleiche Anzahl Nicht-Nullelemente,
- gleiche Anzahl von Operationen

für jeden Prozessor.

Während die ersten beiden Kriterien sich ausschließlich auf die Struktur der Koeffizientenmatrix beziehen, erlaubt das dritte Kriterium die Einbeziehung der übrigen Operationen des CG-Verfahrens. Für eine effiziente Parallelisierung erweist sich das letzte Kriterium als das beste, da die Arbeit auch für unregelmäßige Besetzungsstrukturen gleichmäßig verteilt wird und somit eine minimale Bearbeitungszeit resultiert.

Dieser Datenverteilung liegt folgende theoretische Betrachtung zugrunde.

Die Anzahl der arithmetischen Operationen eines Matrix-Vektor-Produktes ist proportional zur Anzahl der Nicht-Nullelemente der Matrix. Die übrigen anfallenden Vektoroperationen sind proportional zur Anzahl der Zeilen.

Hat Prozessor k bei der Datenverteilung n_k der n Zeilen erhalten und e_k der insgesamt e Nicht-Nullelemente in diesen Zeilen, so ist sein Anteil a_{op} arithmetischer Operationen an den insgeamt auszuführenden Operationen gegeben durch

$$a_{op} = \frac{e_k + \xi n_k}{e + \xi n} . \tag{4.5}$$

ξ berücksichtigt hierbei das zeitliche Verhältnis des Matrix-Vektor-Produkts zu den übrigen Operationen und ist somit maschinen- und algorithmenabhängig.

Um eine möglichst gleichmäßige Verteilung der arithmetischen Operationen zu erhalten, werden einem Prozessor solange aufeinanderfolgende Zeilen zugewiesen, bis das Kriterium

$$a_{op} = \frac{e_k + \xi n_k}{e + \xi n} \geq \frac{1}{p} \tag{4.6}$$

zum ersten Mal erfüllt ist.
Der Anteil des Matrix-Vektor-Produkts des Verfahrens auf einem Prozessor an dem Gesamtaufwand einer Iteration kann aus

$$a_{MVP} = \frac{e}{e + \xi n} = \frac{1}{1 + \xi / m_z} \tag{4.7}$$

berechnet werden, wobei m_z die mittlere Anzahl Nicht-Nullelemente in den Zeilen, also $m_z = e/n$, bezeichnet. Umgekehrt kann aus dem durch Zeitmessung bestimmten Anteil des Matrix-Vektor-Produkts für das CG-Verfahren ohne Vorkonditionierung der Parameter ξ aus (4.7) näherungsweise bestimmt werden:

$$\xi \approx m_z \cdot \left(\frac{1}{a_{MVP}} - 1 \right) .$$

Weiterhin erlaubt der Parameter ξ eine benutzergesteuerte Datenverteilung bezüglich der anderen oben vorgestellten Kriterien. Wird ξ als freier Parameter betrachtet, so gilt mit (4.5) für $\xi = 0$, daß jeder Prozessor etwa gleich viele Nicht-Nullelemente erhält. Wird der Parameter erhöht, so ergibt sich für den Grenzwert $\xi \to \infty$ die Datenverteilung nach dem Kriterium *gleich viele Zeilen pro Prozessor.*

Beim vorkonditionierten CG-Verfahren ist der Anteil des Matrix-Vektor-Produkts an den gesamten arithmetischen Operationen eine Funktion des Polynomgrades m. Da im vorkonditionierten CG-Verfahren im wesentlichen pro Iterationsschritt zusätzlich m Matrix-Vektor-Produkte berechnet werden, ergibt sich näherungsweise

$$a_{MVP} = \frac{(m+1)\cdot e}{(m+1)\cdot e + \xi n} = \frac{1}{1 + \underbrace{\frac{\xi}{m+1}}_{\tilde{\xi}} \cdot \frac{1}{m_z}}, \tag{4.8}$$

so daß für die Datenverteilung $\tilde{\xi} := \frac{\xi}{m+1}$ verwendet wird.

4.3.2 Kommunikationsschema

Da beim Matrix-Vektor-Produkt auf jedem Prozessor nur Teile der Vektoren vorhanden sind, für die Berechnung einer neuen Komponente jedoch i.a. weitere Komponenten des Vektors benötigt werden, ist ein geeignetes Kommunikationsschema zur Verfügung zu stellen. In [7] werden auch hierzu zwei Varianten vorgeschlagen. In der hier vorgenommenen Implementierung wird diejenige Variante benutzt, die dort die besten Resultate liefert.

Ein wesentlicher Unterschied zum Skalarprodukt besteht darin, daß in der Regel nicht alle Prozessoren mit allen kommunizieren müssen, sondern nur mit denen, die die zu den Nicht-Nullelementen zugehörigen Komponenten des Vektors besitzen. Um diese Prozessoren zu ermitteln, werden die Werte im Vektor *colind* analysiert. Die Indizes, die zu Zugriffen auf andere Prozessoren führen, werden gesammelt und schließlich die Information, welche Daten der Prozessor von welchen Prozessoren benötigt, an alle Prozessoren verschickt.

Die empfangenden Prozessoren bestimmen nun, welche Komponenten im Laufe der Rechnung verschickt werden müssen. Das Kommunikationsschema ist ausschließlich eine Funktion der Besetzungsstruktur der Matrix und muß somit nach dem Einlesen der Matrix vor Beginn des parallelen CG-Verfahrens nur einmal bestimmt werden. Im CG-Verfahren werden dann zur Berechnung des Matrix-Vektor-Produkts zu Beginn die Daten, die von anderen Prozessoren benötigt werden, asynchron verschickt.

Das Matrix-Vektor-Produkt zerfällt hierbei in zwei Teile. Im lokalen Teil werden Elemente des Vektors benutzt, die der Prozessor selbst besitzt. Dieses erfordert keine Kommunikation. Da die Abarbeitung mit den nichtlokalen Daten kommutativ ist, wird dann die Berechnung des Matrix-Vektor-Produkts in der Reihenfolge des Eintreffens der Daten fortgesetzt; die Teilergebnisse werden in den entsprechenden Komponenten des zu berechnenden Vektors aufsummiert.

Während sich also angeforderte Daten auf dem Netz befinden, wird mit lokalen oder bereits eingetroffenen Daten gerechnet. Dies führt zu einer Überlappung von Rechnung und Kommunikation und somit zu Rechenzeitgewinn.

Detailliertere Informationen finden sich in [6],[7],[8].

4.4 Implementierung der Vorkonditionierung

Zielsetzung der vorgenommenen Implementierung ist die Bereitstellung der polynomialen Vorkonditonierung mit den verschiedenen in Kapitel 3.6.2 vorgestellten Eigenwertschätzern sowie des Diagonal Scaling, also der Gesamtschritt-Vorkonditionierung.

Außerdem soll eine Kombination von beiden Vorkonditionierern ermöglicht werden. Hierzu wird das Diagonal Scaling in der Version mit Rücksubstitution implementiert. Dieses wurde in dem Basis-CG-Verfahren von A. Basermann bereits vorgesehen.

4.4.1 Implementierung der Gesamtschritt-Vorkonditionierung (Diagonal Scaling)

Dabei wird die Matrix A umgeformt in $\tilde{A}$ mit

$$\tilde{a}_{ij} = \frac{a_{ij}}{\sqrt{a_{ii} \cdot a_{jj}}}, \tag{4.9}$$

und die rechte Seite v in

$$(\tilde{v})_i = \frac{(v)_i}{\sqrt{a_{ii}}}.$$

Die Lösung x des ursprünglichen Systems erhält man aus der Lösung $\tilde{x}$ des Systems $\tilde{A}\tilde{x} = \tilde{v}$ gemäß der Rücksubstitution

$$(x)_i = \frac{1}{\sqrt{a_{ii}}}(\tilde{x})_i. \tag{4.10}$$

$(x)_i$ bezeichnet hierbei die i-te Komponente des Vektors x.

4.4.2 Parallele Implementierung der polynomialen Vorkonditionierung

Folgende Bestandteile der Vorkonditionierung eignen sich zur Parallelisierung:

- Matrix-Vektor-Produkt mit der Matrix A, wobei hierzu der spezielle Zugriff auf die Matrix in der verwendeten Speichertechnik erfolgen muß,
- Vektoradditionen,
- zur automatischen Eigenwertschätzung benutzte Gerschgorin-Schätzwerte für die extremen Eigenwerte von A.

Die ersten beiden Punkte sind bereits Bestandteil des zugrundeliegenden Basis-CG-Verfahren und somit leicht von dort zu übernehmen. Die Gerschgorin-Schätzwerte γ_{min} und γ_{max} sind definiert durch

$$\gamma_{min} = \max_{1\le i\le n}\left\{a_{ii} + \sum_{\substack{i\neq j\\ j}} |a_{ij}|\right\} \tag{4.11}$$

bzw.

$$\gamma_{max} = \min_{1\le i\le n}\left\{a_{ii} - \sum_{\substack{i\neq j\\ j}} |a_{ij}|\right\} \tag{4.12}$$

und erfüllen

$$\gamma_{min} \le \lambda_{min} \le \lambda_{max} \le \gamma_{max} \ .$$

Diese Werte können zuerst durch Gerschgorin-Abschätzungen, die jeder Prozessor auf den ihm zugeteilten Zeilen durchführen kann, lokal bestimmt werden, und anschließend ergibt das Maximum der Teillösungen der oberen und das Minimum der unteren Werte die gesuchten globalen Werte.

Diese Berechnung ist zu Beginn des Verfahrens einmal durchzuführen.

4.5 Bemerkungen zur Implementierung

Das CG-Verfahren mit Gesamtschritt- und polynomialer Vorkonditionierung wurde auf dem Intel PARAGON XP/S 10 und dem Intel iPSC/860 implementiert.

Als Programmiersprache wurde FORTRAN 77 gewählt, wobei zusätzlich Routinen der NAG Library Mark 15 verwendet werden. Der Compiler auf dem iPSC war der *iPSC Cross Compiler FORTRAN 4.0* und auf PARAGON der *Intel/PARAGON FORTRAN Cross Compiler 4.1.1.* Es wurde die Optimierungsstufe *-O4*, die Optionen zur Vektorisierung *-Mvect* und die zur beschleunigten Floating Point Division *-Knoieee* verwendet. Alle Rechnungen wurden hierbei in doppelter Genauigkeit durchgeführt.

Nach ca. der Hälfte der Bearbeitungszeit dieser Arbeit stand der iPSC/860 nicht mehr zur Verfügung. Dies hatte zur Folge, daß einige auf dem PARAGON erst generierte Testbeispiele nicht auf dem iPSC reproduziert werden konnten.

Wegen einiger Schwierigkeiten bei der asynchronen Kommunikation auf dem PARAGON konnte das iPSC-Programm nicht in seiner vollen Funktionalität portiert werden. Zudem laufen die globalen Kommunikationsroutinen in der aktuell installierten Version noch nicht optimal. Diese Probleme sind Intel bekannt und sollen in naher Zukunft behoben werden.

Aufgrund der Schwierigkeiten bei den globalen Routinen konnten zuverlässige numerische Ergebnisse nur bis Prozessoranzahlen von ca. 70 erzielt werden. Um den Anspruch der Reproduzierbarkeit der Ergebnisse zu gewährleisten, werden in dieser Arbeit auf dem PARAGON maximal 64 Knoten verwendet.

Kapitel 5

Testbeispiele

Die sinnvolle Anwendung eines vorkonditionierten CG-Verfahrens hängt stark vom speziellen Problem ab. Deshalb soll in diesem Kapitel ausführlich auf die untersuchten Problemstellungen eingegangen werden.

Das Zeitverhalten und die numerischen Resultate sind insbesondere Funktionen des Startvektors, der rechten Seite, des verwendeten Abbruchkriteriums, der Kondition und der Besetzungsstruktur der Matrix. Die Kondition wird hierbei aus den Schätzwerten des adaptiven Verfahrens näherungsweise bestimmt, da es hier nicht darauf ankommt, die Werte exakt zu kennen.

Untersucht wurden eine Klasse von schlecht konditionierten Matrizen aus einer eindimensionalen Diskretisierung, einige Matrizen aus Finite Elemente (FE) Problemen des Forschungszentrums Jülich sowie Gleichungssysteme aus dem *Harwell-Boeing Set of Sparse Matrices* [12].

5.1 Tridiagonalmatrix

In [17] findet sich als Beispielmatrix 3.18 folgende Koeffizientenmatrix, wie sie bei der Diskretisierung gewöhnlicher Differentialgleichungen entstehen kann.

$$A = \begin{pmatrix} 2 & -1 & 0 & \dots & \dots & \dots & 0 \\ -1 & 2 & -1 & 0 & \dots & \dots & 0 \\ 0 & -1 & 2 & -1 & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & -1 & 2 & -1 & 0 \\ 0 & \dots & \dots & 0 & -1 & 2 & -1 \\ 0 & \dots & \dots & \dots & 0 & -1 & 2 \end{pmatrix}$$

Diese Matrix dient als einfaches Beispiel, da die Struktur eine Tridiagonalmatrix ist und für die Kondition bzgl. der Spektralnorm gilt

$$\kappa(A) \approx 0.4 \cdot n^2 \ .$$

Somit ergibt eine Erhöhung der Dimension eine Verschlechterung der Kondition.

Weiterhin existieren keine vielfachen Eigenwerte und keine Intervalle, in denen sich die Eigenwerte verdichten, so daß ein schlechtes Konvergenzverhalten zu erwarten ist.

Bei dieser Matrix liefern bereits die mit der Gerschgorin-Methode gewonnenen Eigenwertschätzungen eine sehr gute Näherung für die extremen Eigenwerte.

Setzt man die rechte Seite zu $v = (n+1, 0, \ldots, 0)^T$, so ergibt sich als Lösungsvektor $x^* = (n, n-1, \ldots, 1)^T$.

Als Startvektor wird die Diagonale von A, also $x_0 = (2, \ldots, 2)^T$, genommen.

Im folgenden werden diese Matrizen mit **GC***n* bezeichnet, wobei n die gewählte Dimension angibt. Während die Untersuchungen für verschiedene n durchgeführt wurden, werden nur die Resultate zu **GC5000** präsentiert.

5.2 Matrizen aus FE-Modellen des Forschungszentrums Jülich

5.2.1 Umwelttechnik

Die hier beschriebenen Matrizen stammen aus einem FE-Modell des Instituts ICG-4 des Forschungszentrums Jülich; simuliert wird die Ausbreitung von Schadstoffen in geologischen Systemen [31][32].

Die untersuchten Gleichungssysteme **ICG2d** und **ICG3d** resultieren aus einer zweidimensionalen und einer dreidimensionalen Diskretisierung. Diese Gleichungssysteme haben eine regelmäßige Besetzungsstruktur.

Im Vergleich zu den übrigen hier verwendeten Gleichungssystemen ist die Konditionszahl in beiden Fällen niedrig, so daß hier recht gutartige Testmatrizen vorliegen.

Startvektor und rechte Seite ergeben sich hierbei aus der Problemstellung.

Die Daten der betrachteten Matrizen sind in Tabelle 5.1 aufgeführt.

5.2.2 Strukturmechanik

Diese Matrix mit der Kennung **ISR** aus dem Institut für Sicherheitsforschung und Reaktortechnik des Forschungszentrums Jülich entsteht bei der Berechnung von Wärmespannungen in Materialien. Sie zeichnet sich durch eine im Durchschnitt hohe Zahl Nicht-Nullelemente pro Zeile aus. Die Besetzungsstruktur ist deutlich unregelmäßiger als bei den zuvor betrachteten Matrizen (s. Abb. 5.1).

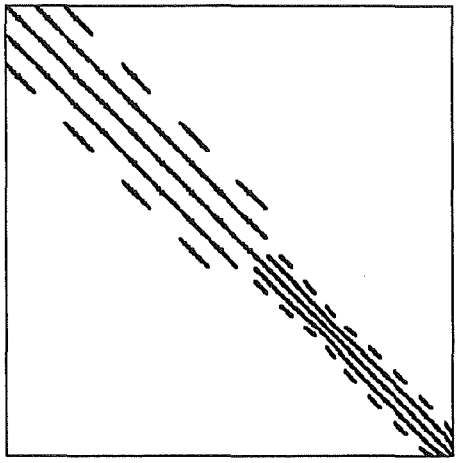


Abbildung 5.1: Matrix aus der Strukturmechanik

Die Daten der Matrix sind in Tabelle 5.1 aufgeführt.

5.2.3 Matrix für ein implizites FE-Modell

Hierbei handelt es sich um eine Matrix aus einem FE-Modell, in dem Verformungen während der Pressung von Motorhauben berechnet werden.

Diese Matrix mit der Bezeichnung **IMPLFE** wurde ausgewählt, da sie bei der Berechnung mit den üblichen Lösungsmethoden Schwierigkeiten bereitet.

Der Lösungsvektor, der mit einem direkten Verfahren ermittelt wurde, enthält ausschließlich kleine Werte, so daß als Startvektor der Nullvektor verwendet wird.

5.3 Beispiele Harwell-Boeing Set

Um Vergleiche zu anderen Arbeiten zu ermöglichen, wurden Tests mit einigen großen, symmetrischen und positiv definiten Matrizen des Harwell-Boeing Set [12] durchgeführt. Hierzu war insbesondere eine Transformation in die oben beschriebene Speichertechnik notwendig.

Sind zu den verwendeten Beispielen keine rechten Seiten oder keine Startvektoren mitgeliefert, so werden die rechten Seiten als Zeilensummen gewählt, so daß sich der Lösungsvektor zu $(1,\ldots,1)^T$ ergibt, und der Startvektor wird ggf. zu $(n, n-1,\ldots,1)^T$ gesetzt.

5.3.1 9-Punkt Laplace-Stern

Ein klassisches Problem ist die Lösung des resultierenden Gleichungssystems, falls ein 9-Punkt Laplace-Stern für die Diskretisierung einer partiellen Differentialgleichung auf einem 2-dimensionalen Gebiet zugrundegelegt wird. Hier wird ein 30×30 Gitter betrachtet. Das resultierende Beispiel hat 900 Zeilen in der Koeffizientenmatrix.
Die Matrix **LAPLACE** verwendet zur Berechnung einer Unbekannten 9 benachbarte Werte, so daß dieses Beispiel im wesentlichen 9 Einträge pro Zeile hat. Nur in den Unbekannten, in denen Randwerte eine Rolle spielen, sind es weniger Einträge. Auch diese Daten sind in Tabelle 5.1 enthalten.

5.3.2 Strukturmechanik

Unter dem Titel **BCSSTRUC2** enthält der Harwell-Boeing Set in der Strukturmechanik verwendete Matrizen zur Statikberechnung mit den Kennungen **BCSSTK***nn*, wobei $nn \in 14, 16, 18$. Die Daten sind in Tab. 5.1 wiedergegeben.

	Ordnung	e	m_z	Bes.grad	$\kappa(A)$
LAPLACE	900	7 744	8	0.9%	200
BCSSTK14	1 806	63 454	35	2.0%	10^9
BCSSTK16	4 884	290 378	59	1.2%	10^6
BCSSTK18	11 948	149 090	12	0.1%	10^{10}
IMPLFE	13 860	661 010	48	0.3%	10^9
ICG2d	17 368	304 000	17	0.1%	2 000
ICG3d	49 392	1 242 814	25	0.05%	3 000
ISR	25 222	3 856 386	153	0.6%	$3 \cdot 10^5$

Tabelle 5.1: Daten der Matrizen

Kapitel 6

Numerische Resultate und Performance

In diesem Kapitel sollen die erzielten Resultate der Implementierung vorgestellt werden. Hierbei sind die numerischen Ergebnisse bis auf Rundungsfehler rechnerunabhängig. Die Performance beschreibt das Zeitverhalten auf den Rechnern.

6.1 Reduktion der Anzahl der Iterationsschritte

Im folgenden bezeichnet it_{cg} die Anzahl Iterationen des Basis-CG-Verfahrens und $it_{vk}(k)$ die des vorkonditionierten CG-Verfahrens mit polynomialer Vorkonditionierung vom Grad $m = k - 1$. it_{ds} und $it_{dsvk}(k)$ bezeichnen die entsprechenden Varianten mit zusätzlichem Diagonal Scaling.

Der Bereich, in dem die Iterationszahl liegt, läßt sich heuristisch wie folgt abschätzen.

Ausgehend von Satz 2.6 läßt sich die Frage beantworten, inwieweit man durch die polynomiale Vorkonditionierung die Iterationszahl zur Berechnung der exakten Lösung bei Verwendung exakter Arithmetik senken kann. Existieren zu der Koeffizientenmatrix A l_A verschiedene Eigenwerte, so ist i.a. $it_{cg} = l_A$. Ein Polynom k-ten Grades hat maximal k reelle Nullstellen und somit auch maximal k Stellen, an denen ein beliebiger Wert angenommen wird.

Somit können maximal k verschiedene Eigenwerte der Matrix A zu einem Eigenwert von $\mathcal{P}(A)$ zusammenfallen. Also gilt für die Anzahl verschiedener Eigenwerte $l_{\mathcal{P}(A)}$ der Matrix $\mathcal{P}(A)$

$$l_{\mathcal{P}(A)} \geq \frac{l_A}{k} .$$

$l_{\mathcal{P}(A)}$ gibt nun die obere Schranke der Iterationszahl für das mit Grad $m = k - 1$ vorkonditionierte CG-Verfahren an.

Dieses Resultat überträgt sich auch auf die Anzahl der Iterationen bis ein bestimmtes Abbruchkriterium erfüllt ist und auch für Eigenwertverteilungen, bei denen die Eigenwerte nicht zusammenfallen, sondern nur geclustert sind.

Diese Überlegungen gelten bei exakter Arithmetik. Die auf den Rechnern erzielten Iterationszahlen weichen in der Regel hiervon nur unwesentlich ab, falls der Einfluß von Rundungsfehlern eine untergeordnete Rolle spielt. Bei einigen Beispielen, den **BCSSTK**-Matrizen und der Matrix **IMPLFE**, sind jedoch die Rundungsfehler dominant, so daß in der Regel mehr als die theoretisch maximale Anzahl der Iterationen n gebraucht wird.

Das für die weiteren Betrachtungen grundlegende Resultat kann hier nur ohne mathematischen Beweis gebracht werden. Ein Gegenbeispiel zu dieser Beobachtung konnte jedoch weder konstruiert werden, noch war dies in der Literatur zu finden.

Hypothese 6.1 *Für die Anzahl der Iterationen des polynomial vorkonditionierten CG-Verfahrens mit Polynomgrad $m = k - 1$ gilt für die Anzahl Iterationsschritte $it_{vk}(k)$ im Vergleich zu der Anzahl Schritte des zugrunde liegenden CG-Verfahrens it_{cg}*

$$it_{vk}(k) \geq \frac{it_{cg}}{k} . \tag{6.1}$$

Bei geeigneter Wahl von a und b gilt darüber hinaus in der Regel

$$it_{vk}(k) \leq it_{cg} . \tag{6.2}$$

Wird zusätzlich das Diagonal Scaling angewandt, so übertragen sich diese Effekte auf die skalierte Matrix, so daß hier i.a. gilt:

$$\frac{it_{ds}}{k} \leq it_{dsvk}(k) \leq it_{ds} . \tag{6.3}$$

Diese Resultate lassen eine genauere Analyse des Aufwandes zu. Hierzu wird die automatische oder manuelle Vorgabe der Eigenwerte betrachtet. Die Gegenüberstellung des Aufwandes findet sich in Tab. 6.1.

Im adaptiven Fall gelten die Resultate fast analog. Die Anzahl durchzuführender Matrix-Vektor-Produkte ist jedoch etwas schwieriger zu berechnen, da ein Vorlaufzyklus zur Berechnung eines Startintervalls der Eigenwerte, sowie eine Reinitialisierung nach der Aktualisierung der Eigenwerte durchgeführt werden muß. Es wird jedoch nur eine beschränkte Anzahl von Aktualisierungen des Eigenwertbereichs zugelassen, so daß die folgenden Betrachtungen auch für dieses Verfahren näherungsweise gelten.

	CG-Verf.	vorkon. CG-Verf. Grad $k-1$
Iterationen	it_{cg}	$it_{vk}(k)$
Skalarprodukte	$3 \cdot it_{cg}$	$3 \cdot it_{vk}(k)$
Matrix-Vektor-Produkte	$it_{cg}+1$	$k \cdot (it_{vk}(k)+1)$

Tabelle 6.1: Aufwand Matrix-Vektor-Produkte und Skalarprodukte mit und ohne polynomiale Vorkonditionierung

Aus der Hypothese 6.1 und (6.2) folgt für die Anzahl Skalarprodukte

$$3 \cdot it_{cg} \geq 3 \cdot it_{vk}(k) \ ,$$

und für die Anzahl Matrix-Vektor-Produkte

$$it_{cg} + 1 \leq k \cdot (it_{vk}(k) + 1) \ .$$

Somit ergibt sich das Resultat, daß sich i.a. beim polynomial vorkonditionierten CG-Verfahren zwar mehr Matrix-Vektor-Produkte als beim Verfahren ohne Vorkonditionierung ergeben, die Anzahl der Skalarprodukte jedoch sinkt.

Ein typischer Verlauf, der den Zusammenhang zwischen Iterationszahl, welche proportional zu den zu berechnenden Skalarprodukten ist, und den zu berechnenden Matrix-Vektor-Produkten über dem verwendeten Polynomgrad aufzeigt, ist in Abb. 6.1 gegeben. Das Verfahren endet, falls die maximal skalierte absolute Differenz 10^{-5} unterschreitet, was für den charakteristischen Verlauf dieser Funktion aber unwesentlich ist.

Kein Polynomgrad (–) bedeutet in diesem Fall das Resultat des CG-Verfahrens ohne Vorkonditionierung, bei dem die Anzahl der Matrix-Vektor-Produkte der Anzahl der Iterationen entspricht. Dieses Beispiel benötigt die theoretisch maximale Anzahl Iterationen 5000. Durch Vorkonditionierung kann die Iterationszahl deutlich gedrückt werden. So ist die Iterationszahl beim Grad 6 ca. 1000.

Dieser Verlauf ist dabei weitgehend unabhängig vom gewähltem Beispiel, Abbruchkriterium und verwendeten Rechner. Die Anzahl Matrix-Vektor-Produkte ist eine monoton wachsende Funktion im gewählten Polynomgrad, wobei hier nur gerade Polynomgrade betrachtet wurden, um den Vergleich zum adaptiven Verfahren zu ermöglichen.

Die numerischen Daten geben jedoch keinen Aufschluß über den optimalen Vorkonditionierungsgrad. Hierzu ist zunächst der Begriff der Optimalität zu klären.
Bei Matrizen mit sehr schlechter Kondition kann es vorkommen, daß das Verfahren ein sehr instabiles Verhalten hat, da die Kondition eine Maßzahl für die Auswirkung von Eingangs- und Rundungsfehlern (vgl. Kap. 2.2) ist. Häufig kann bei solchen Matrizen ein sinnvolles Konvergenzkriterium nicht erfüllt werden oder es führt zu einer

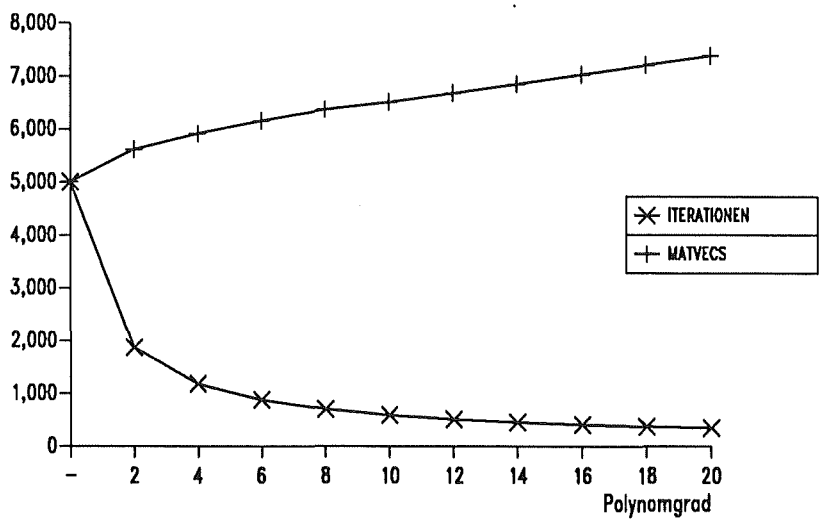


Abbildung 6.1: Matrix-Vektor-Produkte und Iterationen bei verschiedenen Polynomgraden am Beispiel GC5000

falschen Lösung. Ein solches Beispiel ist die Matrix **IMPLFE**, worauf im folgenden noch näher eingegangen wird. Weiterhin liegt die größte numerische Anfälligkeit für Rundungsfehler im CG-Verfahren in der in jedem Schritt durchzuführenden Orthogonalisierung. Diese besteht im wesentlichen aus dem zu berechnenden Skalarprodukt. Eine Verbesserung der Kondition bewirkt somit wegen der Reduktion der Iterationen eine Verminderung der vorzunehmenden Orthogonalisierungen. Dieser Effekt wird i.a. umso ausgeprägter je höher der Polynomgrad gewählt wird, da mit steigendem Polynomgrad die Kondition und die Anzahl durchzuführender Iterationen geringer wird. Deshalb werden in solchen Fällen hohe Polynomgrade gewählt. In der Regel steht jedoch nicht die Stabilität des Verfahrens im Vordergrund. Konvergiert bereits das CG-Verfahren ohne Vorkonditionierung, so besteht die Zielsetzung, durch die Vorkonditionierung die Kosten des Verfahrens zu reduzieren. Diese Kosten werden gemessen an der benötigten Ausführungszeit, wobei in FE-Modellen häufig eine größere Sequenz von Gleichungssystemen zu lösen ist, so daß bereits geringe Zeitgewinne von Bedeutung sind. Die Resultate mit der asynchronen Version auf dem iPSC/860 werden zunächst vorgestellt. Die hier erzielten Effekte kennzeichnen die Parallelisierung der Vorkonditionierung. Die Nachteile dieser Untersuchung sind die Beschränkung auf lediglich 32 Knoten und die, durch den zeitlichen Rahmen bedingt, geringe Zahl der Testbeispiele. Die wesentlichen Aspekte werden jedoch an den Beispielen **GC5000** und **ICG2d** bereits deutlich.

6.2 Resultate auf dem iPSC/860

In den hier dargestellten Untersuchungen wird durch die asynchrone Version des Matrix-Vektor-Produktes die Überlappung von Rechnung und Kommunikation ermöglicht.

Die Testmatrix **GC5000** steht für eine schlecht konditionierte Matrix mit einer gleichmäßigen Eigenwertverteilung, woraus eine hohe Iterationszahl folgt. Weiterhin ist der Aufwand für Kommunikation und Rechnung beim Matrix-Vektor-Produkt wegen der Tridiagonalgestalt und der geringen Zahl Nicht-Nullelemente in einer Zeile gering. Dies ist wegen der erhöhten Zahl von Matrix-Vektor-Produkten beim vorkonditionierten CG-Verfahren günstig. Es sei hier erwähnt, daß aufgrund der Besetzungsstruktur dieses Gleichungssystem deutlich schneller mit einem direkten Verfahren gelöst werden kann. Ziel dieser Untersuchung ist somit nicht eine hohe Performance, sondern eine Analyse des Verhaltens der Vorkonditionierung.

ICG2d hingegen ist eine gut konditionierte Matrix mit einer deutlich höheren Zahl Nicht-Nullelemente pro Zeile.

Falls nicht explizit anders spezifiziert, liegen den Untersuchungen folgende Parameter zugrunde.

- Abbruchkriterium: maximal skalierte absolute Differenz $< 10^{-5}$,
- Adaptives Verfahren
  - Vorlauf: 2 Iterationen zur Berechnung eines Startintervalls für die Initialisierung von a und b,
  - Zyklus: Nach jeweils 5 Iterationen wird das Intervall überprüft und ggf. aktualisiert,
  - Maximal 4 Aktualisierungen des Intervalls
- Gerschgorin: $\delta = 0.9$.

6.2.1 Performance für das Testbeispiel GC5000

Eine wesentliche Frage ist, wie sich die Ausführungszeiten verhalten, falls der Polynomgrad variiert. In Abb. 6.2 sind die Resultate der 3 Methoden zur Eigenwertbestimmung aufgezeigt. Die Messungen wurden dabei auf 4 Prozessoren durchgeführt.

Da die adaptive und die Gerschgorin-Methode i.a. bzgl. der Ausführungszeit nur suboptimale Schätzwerte liefern, wurden für den Polynomgrad 4 durch Ausprobieren die Werte a und b bestimmt, bei denen die Zeit minimal ist. Dieses Vorgehen dient dazu, eine Abschätzung zu erhalten, welche Gewinne bei verbesserten Schätzstrategien noch zu erwarten sind.

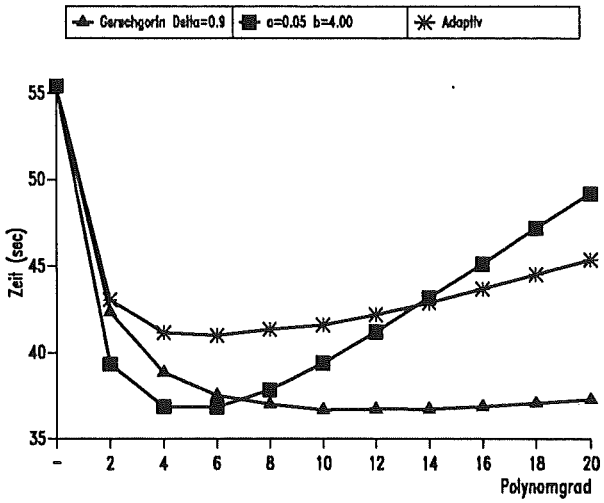


Abbildung 6.2: Ausführungszeiten bei unterschiedlichem Polynomgrad

Das erste hier gewonnene Resultat ist, daß ein Gewinn von ca. 30% bei der Gerschgorin-Methode mit Grad 10 und ca. 25% beim adaptiven Verfahren mit Grad 6 erzielt werden können.

Die erreichten Zeiten liegen dabei schon recht nahe am Optimum. Wie aus Abb. 6.2 ersichtlich ist, sind die für den Grad 6 günstigsten Parameter a und b nicht global optimal. Insbesondere sind die mit dem Gerschgorin-Verfahren ermittelten Parameter für hohe Polynomgrade besser als einmal fest eingestellte Werte. Dabei sei der beobachtete Effekt erwähnt, daß bei höheren Polynomgraden auch das Intervall $[a, b]$ vergrößert werden muß, um die minimale Zeit zu erreichen.

Diese Eigenschaft wird durch die Bestimmung der Eigenwertschätzungen mit der Gerschgorin-Methode in Kapitel 3.6.1 unterstützt, da der Wert a umso kleiner gewählt wird, je höher der Polynomgrad ist.

Dies erklärt den sehr günstigen Verlauf der Gerschgorin-Methode auch bei hohen Polynomgraden. Beim adaptiven Verfahren beeinflußt eine Erhöhung des Polynomgrades die Schätzwerte i.a. nicht. Bedingt durch die Expansionseigenschaft des Intervalls liegt in diesem Fall ein zu kleines Intervall vor. Analog zu den obigen Argumenten ist somit für dieses Verfahren ein kleiner Polynomgrad angebracht.

Der zeitliche Gewinn resultiert, wie im vorigen Kapitel dargestellt, aus der Einsparung zu berechnender Skalarprodukte. Nachteilig beim Skalarprodukt ist weniger der numerische Aufwand, als vielmehr die durchzuführende Kommunikation, da eine globale Synchronisation durchgeführt werden muß. Eine Erhöhung der Prozessorzahl verspricht somit einen zusätzlichen Gewinn für das adaptive Verfahren im Vergleich zum Verfahren ohne Vorkonditionierung, da dort weniger globale Synchronisation durchgeführt werden muß.

Um diesen Effekt zu verdeutlichen, wurden die optimalen Parameter der Version auf 4 Prozessoren – Gerschgorin Grad 10, adaptiv Grad 6 – auf unterschiedlichen Prozessorzahlen gerechnet. Die Ergebnisse sind in Abb. 6.3 zu sehen.

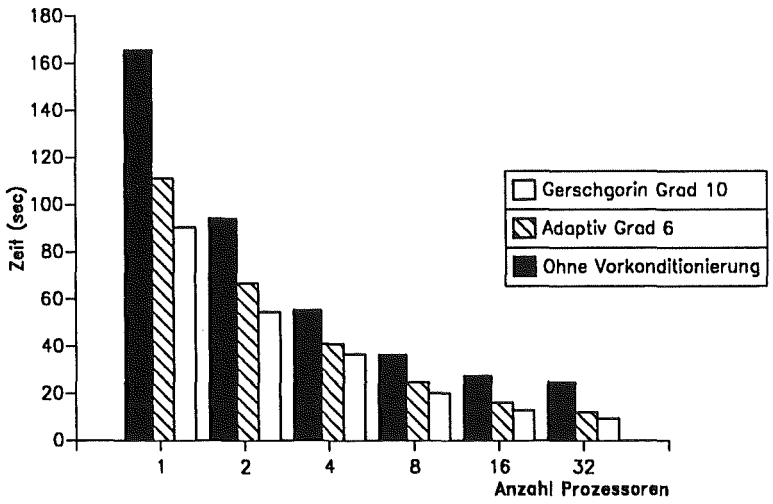


Abbildung 6.3: Ausführungszeiten bei unterschiedlicher Prozessorzahl

Während diese Grafik die absoluten Ausführungszeiten angibt, interessiert für den zusätzlichen Gewinn durch eingesparte Kommunikation der Speedup der Verfahren, also der Quotient aus der Zeit für einen Prozessor und der Zeit für eine bestimmte Anzahl von Prozessoren. Diese Funktion ist in Abb. 6.4 zu sehen.

Neben dem Abb. 6.3 zu entnehmenden Resultat, daß die Zeit beim vorkonditionierten CG bereits auf einem Prozessor signifikant niedriger ist, wird hier deutlich, daß aufgrund der eingesparten Kommunikation das vorkonditionierte Verfahren zusätzlich besser skaliert.

So ist schließlich der Gewinn der beiden vorkonditionierten Verfahren ca. 50%.

Zu beachten sind hier jedoch einige für diesen Testfall spezifischen Eigenschaften. Zum einen liefern die Gerschgorin-Schätzwerte bereits sehr gute Näherungen für die wirklichen Eigenwerte, zum anderen ist diese Matrix aufgrund ihrer Besetzungstruktur und Eigenwertverteilung sehr günstig für die Vorkonditionierung, so daß die für diesen Fall erzielten Ergebnisse die Größenordnung der maximal zu erreichenden Zeitreduktion wiedergeben.

Wegen der einfachen Gestalt dieser Matrix hat das Diagonal Scaling hier keinen Einfluß auf die Iterationszahl und die Ausführungszeit.

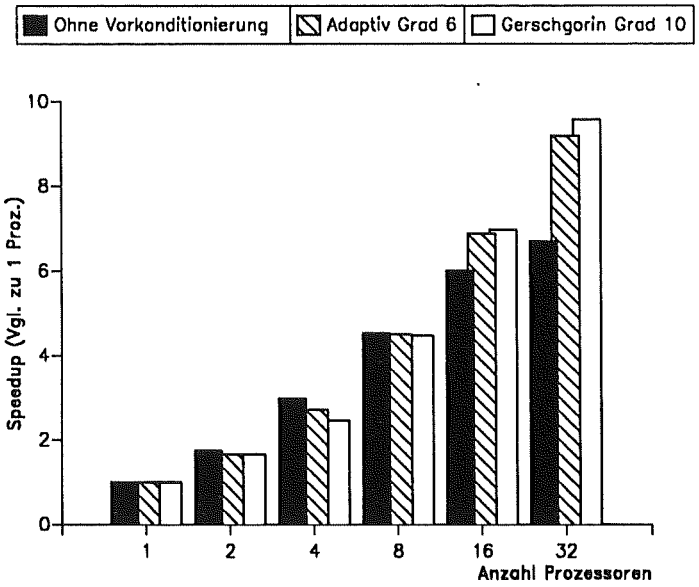


Abbildung 6.4: Speedup auf dem iPSC für GC5000

6.2.2 Performance für das Testbeispiel ICG2d

An diesem Testbeispiel sollen die Grenzen der Anwendbarkeit der polynomialen Vorkonditionierung aufgezeigt werden. Es ist keineswegs so, daß die Vorkonditionierung in jedem Fall zu einer Reduktion der Ausführungszeit führt.

Aufgrund der guten Kondition dieser Matrix liefert das CG-Verfahren ohne Vorkonditionierung bereits nach 161 Iterationen die gewünschte Näherungslösung. Das Diagonal Scaling reduziert diese Iterationszahl auf 110. Es erweist sich als schwierig, diese Iterationszahlen mit der polynomialen Vorkonditionierung weiter zu reduzieren. Hinzu kommt, daß diese Matrix nicht diagonaldominant ist und somit die Gerschgorin-Schätzwerte sehr schlecht werden.

Die Untersuchungen wurden auf 16 Prozessoren durchgeführt. In Tabelle 6.2 sind die optimalen Grade der Polynome in Klammern hinter der Ausführungszeit, welche in Sekunden angegeben ist, aufgeführt.

	ohne Diag. Scal.	mit Diag. Scal.
CG-Verfahren ohne Vorkonditionierung	3.25	2.24
Vorkonditionierung Gerschgorin	3.57 (8)	3.63 (1)
Vorkonditionierung adaptiv	3.85 (2)	2.59 (2)

Tabelle 6.2: Ausführungszeiten in sec bei ICG2d auf 16 Prozessoren

Auffällig ist hierbei, daß die Kombination von Gerschgorin und Diagonal Scaling ein sehr schlechtes Ergebnis liefert. Dies liegt daran, daß durch das Diagonal Scaling die Eigenwerte der Matrix zu 1 geclustert werden. Ist die Matrix nicht diagonaldominant, so ergeben sich auch für die skalierte Matrix schlechte Eigenwertschätzer und somit keine wesentliche Verbesserung durch die polynomiale Vorkonditionierung. Wegen des verhältnismässig hohen Aufwandes – bei Grad 1 doppelt soviele Matrix-Vektor-Produkte wie beim Verfahren ohne Vorkonditionierung – ist mit dieser Kombination kein Zeitgewinn zu erzielen.

Das Problem dieses Testfalles liegt darin, daß wegen der recht geringen Genauigkeitsanforderung durch das grobe Abbruchkriterium die Iterationszahl sehr niedrig ist. Häufig wird in Anwendungen eine höhere Genauigkeit erwartet, was zu einer höheren Iterationszahl führt. Dies wiederum bedeutet einen besseren Ansatzpunkt für die Vorkonditionierung. Die Iterationszahl wächst beim Abbruchkriterium maximal skalierte absolute Differenz $< 10^{-12}$ auf 415 im CG ohne Skalierung bzw. 272 Iterationen im CG mit Skalierung. Die Ausführungszeiten sind Tabelle 6.3 zu entnehmen.

	ohne Skal.	mit Skal.
CG-Verfahren ohne Vorkonditionierung	8.35	5.50
Vorkonditionierung Gerschgorin	8.11 (8)	8.89 (1)
Vorkonditionierung adaptiv	8.74 (2)	5.25 (2)

Tabelle 6.3: Ausführungszeiten in sec bei ICG2d auf 16 Prozessoren mit Abbruch bei Skal. Differenz $< 10^{-12}$

Ein minimaler Gewinn ist bei zusätzlicher adaptiver polynomialer Vorkonditionierung zum Diagonal Scaling zu erzielen. Wenn dies auch kein überragendes Ergebnis ist, so ist es doch der Ansatzpunkt für folgende weitergehende Überlegung.

Ein u.U. schwerer zu erfüllendes Abbruchkriterium ist, daß die euklidische Norm des Residuums einen vorgegebenen Wert ε unterschreitet. Dieses Abbruchkriterium wird ebenfalls häufig in der Praxis verwendet.

Die Iterationszahl erhöht sich beim Kriterium, daß das Verfahren endet, falls die euklidische Norm des Residuums kleiner als 10^{-5} wird, auf 842 Iterationen ohne, bzw. 475 Iterationen mit Skalierung. Dies verbessert die Einsatzmöglichkeit der Vorkonditionierung weiter, wie aus Tab. 6.4 ersichtlich ist.

Das Hauptresultat dieser Untersuchungen ist, daß das Diagonal Scaling in den meisten praktischen Anwendungen bereits bei kleinen Iterationszahlen einen sehr guten Erfolg zeigt. Bei hohen Iterationszahlen bzw. sehr schlecht konditionierten Systemen führt die polynomiale Vorkonditionierung zur Reduktion der Rechenzeit.

Im Beispiel **GC5000** konnte der Effekt beobachtet werden, daß das CG-Verfahren mit Vorkonditionierung zusätzlich besser skaliert.

	ohne Skal.	mit Skal.
CG-Verfahren ohne Vorkonditionierung	16.50	9.48
Vorkonditionierung Gerschgorin	11.64 (8)	13.89 (1)
Vorkonditionierung adaptiv	11.92 (2)	8.66 (2)

Tabelle 6.4: Ausführungszeiten in sec bei ICG2d auf 16 Prozessoren mit Abbruch Norm des Residuums $< 10^{-5}$

6.3 Resultate auf PARAGON XP/S 10

Auf dem PARAGON wurden alle in Kap. 5 vorgestellten Testbeispiele gerechnet. Es sollen sowohl die Resultate der analog zum iPSC/860 gerechneten Daten als auch die Ergebnisse weiterer Beispiele präsentiert werden.

Zu Beginn soll an einem Beispiel der Effekt der Stabilitätserhöhung durch die Vorkonditionierung gezeigt werden.

6.3.1 Erhöhung der Stabilität am Beispiel IMPLFE

Wie bereits erwähnt, wird das CG-Verfahren mit Vorkonditionierung durch die Reduktion der Anzahl Iterationen und somit der Anzahl durchzuführender Skalarprodukte weniger anfällig gegen Rundungsfehler. Das dieses Resultat nicht nur von theoretischem Wert ist, zeigt folgende Untersuchung der Matrix **IMPLFE**.

Wird der Nullvektor als Startvektor gewählt, so ergibt sich der in Abb. 6.5 dargestellte Verlauf für die Norm des Residuums, die zur Übersichtlichkeit logarithmisch aufgetragen ist, über der Anzahl gerechneter Iterationen. Zunächst sinkt die Norm des Residuums, wie dies auch zu erwarten ist. Nach etwa 200 Iterationen steigt die Norm an, so daß keine Konvergenz erzielt werden kann.

Besser sieht der Verlauf des Residuums bei Verwendung des Diagonal Scaling aus, wie Abb. 6.6 zeigt.

In diesem Fall tritt bei einem Wert der euklidischen Norm des Residuums von ca. $10^{-1.5}$ ein negativer Wert bei der Berechnung der Norm auf. Somit führt dies zum Abbruch des CG-Verfahrens, ohne daß das Konvergenzkriterium erfüllt wurde.

Eine weitere Reduktion der Kondition erscheint hier sinnvoll. Durch das Diagonal Scaling kann die Kondition lediglich um einen konstanten Faktor gesenkt werden. Die polynomiale Vorkonditionierung erlaubt hingegen durch Erhöhen des Grades des verwendeten Polynoms eine größere Reduktion.

So liefert die Rechnung mit adaptiver polynomialer Vorkonditionierung mit Grad 30 den in Abb. 6.7 dargestellten Verlauf.

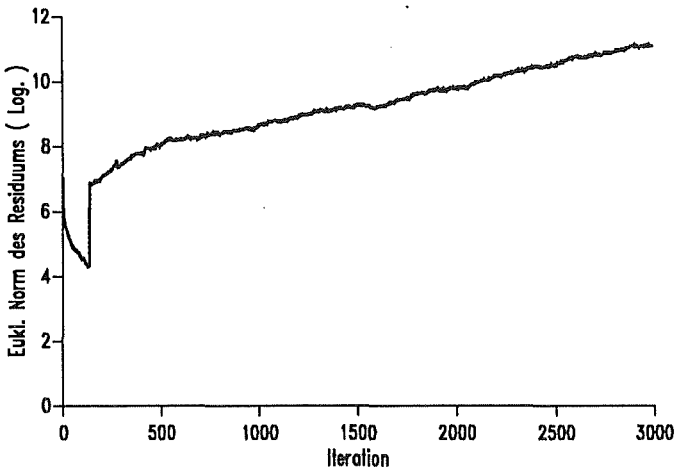


Abbildung 6.5: Verlauf des Residuums (logarithmisch) über der Iterationszahl beim Verfahren ohne Vorkonditionierung

Der Einsatz der polynomialen Vorkonditionierung ermöglicht in diesem Fall zum einen eine genauere Lösung, zum anderen eine erhöhte Stabilität des Verfahrens.

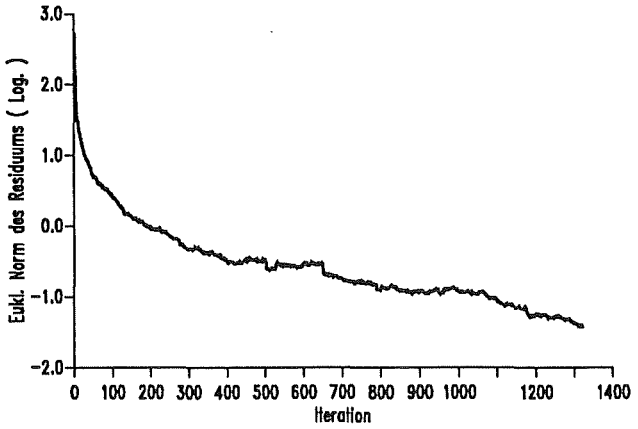


Abbildung 6.6: Verlauf des Residuums (logarithmisch) über der Iterationszahl bei Verwendung des Diagonal Scaling

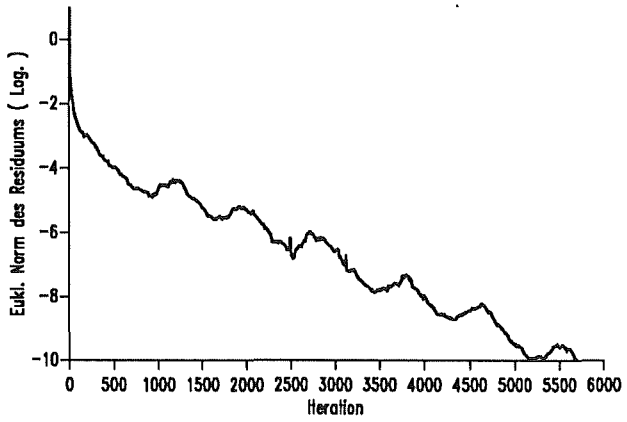


Abbildung 6.7: Verlauf des Residuums (logarithmisch) über der Iterationszahl bei Verwendung der polynomialen Vorkonditionierung

6.3.2 Ergebnisse zu GC5000 und ICG2d

Das Ziel der in diesem Abschnitt durchgeführten Untersuchungen ist der Vergleich zu den auf dem iPSC erzielten Ergebnissen.

Leider konnten die vergleichsweise guten Speedup Werte des iPSC wegen der bereits in dieser Arbeit erwähnten Probleme bei der asynchronen Version des CG-Verfahrens auf PARAGON nicht erzielt werden.

Resultate zu GC5000

Analog zum Kapitel 6.2 soll zunächst die Effizienz der Parallelisierung am Beispiel **GC5000** beschrieben werden. Für die gleichen Polynomgrade ergibt sich der in Abb. 6.8 erzielte Verlauf für die Zeiten auf dem PARAGON. Abb. 6.9 zeigt die Auswirkung auf den Speedup.

Auffällig ist hierbei der Anstieg der Zeit bei höheren Prozessorzahlen. Die Zeit für die Kommunikation ist offensichtlich hoch im Vergleich zur Rechenzeit.

Insbesondere das Skalarprodukt wirkt sich hier negativ auf den Speedup aus. Die Anzahl Kommunikationsschritte ergibt sich aus

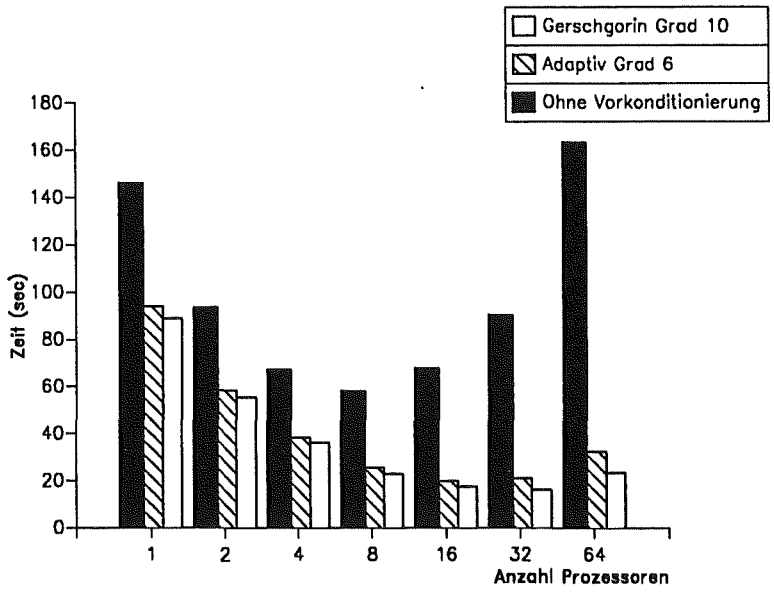


Abbildung 6.8: Zeitverlauf für GC5000 auf PARAGON

- Skalarprodukte: im optimalen Fall $\log_2(p)$ Schritte, derzeit ist jedoch die Anzahl mit einem hohen linearen Faktor versehen,
- Matrix-Vektor-Produkt: jeder Prozessor, bis auf den ersten und letzten, hat zwei Nachbarprozessoren, die Daten benötigen, also müssen von jedem Prozessor zwei Kommunikationsschritte durchgeführt werden.

Die schlechten Zeiten für hohe Prozessorzahlen sind durch die bereits erwähnten Probleme bei lokalen und globalen Kommunikationsroutinen zu erklären.

Wegen der reduzierten Anzahl Skalarprodukte ist der Effekt, daß die Kommunikationszeit die Gesamtzeit verschlechtert, bei den vorkonditionierten Verfahren erst später zu beobachten. Die Gerschgorin-Methode erlaubt recht hohe Polynomgrade und damit verbunden die größte Reduktion der Anzahl der Skalarprodukte. Deshalb zeigt dieses Verfahren den besten Speedup in diesem Beispiel.

Das positive Ergebnis dieser Untersuchung ist, daß die polynomial vorkonditionierten Verfahren auch in der nicht in ihrer vollen Funktionalität übernommenen Version, welche sich negativ auf das Matrix-Vektor-Produkt auswirkt, besser skalieren als das ursprüngliche CG-Verfahren. Dies erlaubt den sinnvollen Einsatz höherer Prozessorzahlen, welche beim ursprünglichen Verfahren zu Zeitverlusten führen.

Der Nachteil der CG-Verfahren, welcher hier deutlich wird, ist, daß auch mit Vorkonditionierung der Speedup nicht beliebig hoch wird. Somit existiert zu jedem Verfahren eine optimale Prozessorzahl. Erst bei wachsender Problemgröße können höhere Prozessorzahlen eingesetzt werden.

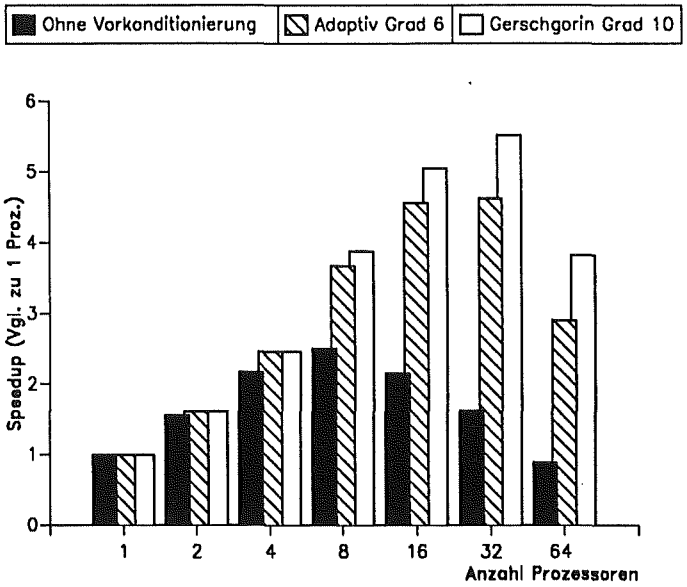


Abbildung 6.9: Speedup für GC5000 auf PARAGON

Die optimale Wahl der Parameter, insbesondere der Prozessorzahl, für ein zu berechnendes Gleichungssystem ist, wie diese Resultate zeigen, eine maschinenabhängige Größe.

Resultate zu ICG2d

Das Beispiel **ICG2d** liefert ein ähnliches Verhalten zum Resultat auf dem iPSC/860. Deshalb sei hier nur auf die zu Tabelle 6.4 analoge Tabelle 6.5 hingewiesen.

	ohne Skal.	mit Skal.
CG-Verfahren ohne Vorkonditionierung	14.4	8.3
Vorkonditionierung Gerschgorin	9.7 (8)	11.7 (1)
Vorkonditionierung adaptiv	9.6 (2)	7.8 (2)

Tabelle 6.5: Ausführungszeiten in sec bei ICG2d auf 16 Prozessoren mit Abbruch Norm des Residuums $< 10^{-5}$ auf PARAGON

Insbesondere ist auffällig, daß die bzgl. der Ausführungszeit optimalen Polynomgrade wiederum die gleichen sind.

6.3.3 Resultate der übrigen Beispiele

Bei der Performance-Messung liegen in diesem Kapitel folgende leicht veränderte Parameter für die adaptive Vorkonditionierung zugrunde:

- Vorlauf: 3 Iterationen,
- max. 3 Aktualisierungen des Intervalls.

Als Abbruchkriterium wurde die Schranke *Euklidische Norm des Residuums* $< 10^{-10}$ verwendet. Weiterhin werden als Prozessoranzahl nur Zweierpotenzen betrachtet, jedoch bereiten andere Prozessorzahlen keine Schwierigkeiten.

Da zum Beispiel **ICG2d** bisher keine Messungen mit unterschiedlichen Prozessorzahlen gemacht wurden, wird dieses Beispiel hier mit aufgeführt.

Folgende Fragestellungen werden in diesem Kapitel für die Testbeispiele beantwortet:

1. Welche Vorkonditionierung liefert die minimale Ausführungszeit im sequentiellen Fall ?
2. Bei welcher Prozessorzahl ist die Ausführungszeit bei einer bestimmten Vorkonditionierung minimal ?
3. Welchen Gewinn bringt die polynomiale Vorkonditionierung ?
4. Wie weit läßt sich die Ausführungszeit des sequentiellen CG-Verfahrens ohne Vorkonditionierung durch Vorkonditionierung und Parallelisierung senken ?

Um die Übersichtlichkeit zu erhalten, werden hier nicht alle erzielten Daten präsentiert. Auch bei diesen Beispielen sind die in den vorherigen Abschnitten beschriebenen Effekte zu beobachten. Es zeigt sich insbesondere, daß die Kombination der polynomialen Vorkonditionierung mit der Gerschgorin-Methode und zusätzlichem Diagonal Scaling sehr schlechte Resultate liefert. Andererseits liefert die Kombination von Diagonal Scaling und adaptiver polynomialer Vorkonditionierung in der Regel ein besseres Resultat als das reine adaptive Verfahren. Die optimalen Polynomgrade variieren nur gering und sind beim adaptiven Verfahren klein (2 oder 4), beim Gerschgorin-Verfahren – falls dieses Verfahren gewinnbringend ist – im Bereich von 6. In praktischen Anwendungen ist der optimale Polynomgrad i.a. nicht bekannt. Der hier vorgeschlagene Grad (Adaptiv Grad 2, Gerschgorin Grad 6) kann vom optimalen Grad abweichen, liefert jedoch nur unwesentlich schlechtere Resultate. Deshalb wird der Polynomgrad für die Messungen konstant gehalten.

Aus diesem Grund werden alle Ergebnisse nur für folgende Konstellationen untersucht:

- keine Vorkonditionierung,
- Diagonal Scaling,
- polynomiale Vorkonditionierung vom Grad 6 mit Gerschgorin-Schätzwerten,
- adaptive polynomiale Vorkonditionierung vom Grad 2 mit zusätzlichem Diagonal Scaling.

Aus Speicherplatzgründen kann das Beispiel **ICG3d** erst ab einer Prozessorzahl von 4 und das Beispiel **ISR** ab 8 Prozessoren gerechnet werden. Somit kann in diesen Beispielen nur der Vergleich zu 4 bzw. 8 Prozessoren angegeben werden. Weiterhin wurde für das Testbeispiel **ISR** die Fehlerschranke *maximal skalierte absolute Differenz* $< 10^{-5}$ gewählt, da dieses Beispiel sehr hohe Iterationszahlen erfordert, sehr viel Kommunikation durchgeführt werden muß und hierdurch bedingt die Ausführungszeit sehr hoch ist. Bei diesem Beispiel kann somit bereits bei einem gröberen Abbruchkriterium das parallele Verhalten der Vorkonditionierung gut beobachtet werden. Für diese Testfälle ergeben sich die in Tab. 6.6 Iterationszahlen.

Kennung	ohne Vorkond.	Diag. Scal.	Gerschgorin Grad 6	Adaptiv Gr. 2 + Diag. Scal.
LAPLACE	84	82	56	74
BCSSTK14	*	851	*	340
BCSSTK16	992	350	140	155
BCSSTK18	*	3220	*	1087
ICG2d	601	325	83	126
ICG3d	1086	165	201	65
ISR	1446	644	351	266

Tabelle 6.6: Iterationszahlen der Testbeispiele

Ein "$*$" bedeutet, daß das Verfahren nicht in der doppelten Anzahl der theoretisch maximalen Iterationszahl n bzw. $n/(m+1)$ bei polynomialer Vorkonditionierung vom Grad m das Abbruchkriterium erfüllt hat. Bei den Matrizen **BCSSTK14** und **BCSSTK18** ist somit ein ähnlicher Effekt wie bei der Matrix **IMPLFE** zu beobachten. Auch hier führt erst die Vorkonditionierung zu einem stabilen CG-Verfahren.

Zunächst sollen hier – soweit möglich – die Resultate der Testbeispiele auf einem Prozessor präsentiert werden. Dieses Vorgehen dient dazu, das sequentielle Verhalten der verschiedenen Verfahren zu beschreiben und den Speedup des optimalen Verfahrens angeben zu können. Die Ergebnisse sind in Tab. 6.7 aufgeführt, wobei die

Konstellation mit minimaler Ausführungszeit gekennzeichnet ist. Die letzte Spalte vergleicht die Resultate des Diagonal Scaling mit denen mit zusätzlicher adaptiver polynomialer Vorkonditionierung.

Kennung	ohne Vorkond.	Diag. Scal.	Gerschgorin Grad 6	Adaptiv Gr. 2 + Diag. Scal.	Gewinn Adap.-Diag. Scal.
LAPLACE	**0.7**	**0.7**	2.3	1.4	–
BCSSTK14	*	**26**	*	28	–
BCSSTK16	120	**42**	109	52	–
BCSSTK18	*	330	*	**295**	10.6%
ICG2d	116	64	90	**63**	1.5%
ICG3d	125	**20**	129	21	–
ISR	279	**125**	442	147	–

Tabelle 6.7: Ausführungszeiten in sec der Testbeispiele auf einem Prozessor (ICG3d auf 4, ISR auf 8 Prozessoren), PARAGON

Die Gerschgorin-Schätzungen für die Eigenwerte sind bei den hier betrachteten Matrizen, außer bei der Matrix **LAPLACE**, so schlecht, daß diese Vorkonditionierung keine Verbesserung bringt. War das Verfahren mit Gerschgorin-Schätzwerten im Beispiel **GC5000** noch sehr effektiv, so erweist es sich hier als ungeeignet. Bei der Matrix **LAPLACE** wäre eine beschleunigte Konvergenz zu erwarten gewesen. Die Matrix **LAPLACE** ist zwar diagonaldominant, jedoch konvergiert das CG-Verfahren ohne Vorkonditionierung bereits nach 84 Iterationen. In diesem Fall ist der Aufwand der Vorkonditionierung zu hoch, um noch einen Zeitgewinn zu erzielen.

Bei der Betrachtung der übrigen Resultate ist zunächst erwähnenswert, daß das Diagonal Scaling schon eine hohe Zeitreduktion bewirkt. Auffällig ist jedoch, daß die polynomiale Vorkonditionierung nur selten besser ist als das einfache Diagonal Scaling. Selbst in diesen Fällen ist der Gewinn gering. Die Betrachtung dieser Resultate rechtfertigt die Ausssage von G. Pini und G. Gambolati in [25] *Is a simple diagonal scaling the best preconditioner for conjugate gradients on supercomputers ?*, die zu dem Ergebnis kommen, daß die Skalierung in den "meisten" Fällen die beste Vorkonditionierung ist. In den übrigen Fällen erreicht sie zumindest die gleiche Größenordnung wie eine andere Vorkonditionierung. Die in den dort angegebenen Rechnungen verwendeten Gleichungssysteme haben jedoch maximal 5000 Zeilen; die Untersuchungen wurden auf einem Cray Vektorrechner mit 4 Prozessoren durchgeführt.

Mag dieses Resultat für Vektorrechner mit einigen Prozessoren und den betrachteten kleinen Problemgrößen zutreffen, so eröffnen die massiv-parallelen Systeme durch die Möglichkeit der Skalierbarkeit auf hohe Prozessorzahlen einen neuen Zugang. Dies belegt Tabelle 6.8.

In dieser Tabelle sind die minimalen Ausführungszeiten, die durch Erhöhen der Prozessorzahlen erreicht werden konnten, angegeben. Dahinter steht in Klammern die verwendete Prozessorzahl.

Kennung	ohne Vorkond.	Diag. Scal.	Gerschgorin Grad 6	Adaptiv Gr. 2 + Diag. Scal.	Gewinn Adap.- Diag. Scal.
LAPLACE	**0.6** (4)	**0.6** (4)	1.1 (16)	1.0 (4)	–
BCSSTK14	*	11 (16)	*	**7.5** (32)	32%
BCSSTK16	19 (16)	7.1 (16)	8.2 (64)	**5.7** (32)	20%
BCSSTK18	*	69 (32)	*	**52** (16)	25%
ICG2d	19 (32)	10 (32)	9.4 (64)	**8.0** (64)	20%
ICG3d	56 (32)	8.7 (32)	34 (64)	**5.6** (64)	35%
ISR	89 (64)	40 (64)	91 (64)	**31** (64)	28%

Tabelle 6.8: Ausführungszeiten in sec der Testbeispiele auf der optimalen Prozessoranzahl auf PARAGON

Offensichtlich kann die höhere Ausführungszeit des adaptiven Verfahrens in vielen sequentiellen Fällen durch ein günstigeres Zeitverhalten im parallelen Fall ausgeglichen werden. Es ist erkennbar, daß in allen Fällen, außer bei der Matrix **LAPLACE**, die Kombination von Diagonal Scaling und adaptiver polynomialer Vorkonditionierung das optimale Verfahren liefert. Die Gewinne liegen hierbei zwischen 20 und 35 Prozent. Die Methode mit den Gerschgorin-Werten skaliert zwar besser als die anderen Verfahren, ist jedoch wegen der hohen sequentiellen Ausführungszeit hier nicht sinnvoll. Bei großen Matrizen, wie dem Beispiel **ISR**, wird die kürzeste Ausführungszeit bei der maximal verwendeten Prozessorzahl 64 erreicht. Auch für das Gerschgorin-Verfahren ist mit höherer Prozessorzahl ein weiterer Zeitgewinn zu erwarten, während die Verfahren ohne Vorkonditionierung bzw. mit Diagonal Scaling die kürzesten Ausführungszeiten meist bei weniger als 64 Prozessoren erreichen.

Das einzige Beispiel, bei dem die polynomiale Vorkonditionierung nicht zum Erfolg führt, ist das Beispiel **LAPLACE**. Für kleine gutartige Systeme kann es also vorkommen, daß die Methode keinen Gewinn bringt.

Zusammenfassend läßt sich sagen, daß durch gemeinsame Anwendung von Parallelisierung, Diagonal Scaling und polynomialer Vorkonditionierung die Ausführungszeit des CG-Verfahrens in den meisten Fällen deutlich gesenkt werden kann.

Kapitel 7

Zusammenfassung und Ausblick

In diesem Kapitel werden die Resultate dieser Arbeit zusammengefaßt und schließlich mögliche Erweiterungen der Implementierung vorgestellt.

7.1 Zusammenfassung

In dieser Arbeit werden verschiedene Methoden der Vorkonditionierung für die Methode der konjugierten Gradienten zur Lösung von großen dünnbesetzten Gleichungssystemen auf Parallelrechnern mit verteiltem Speicher behandelt.

Hierzu werden das Diagonal Scaling und die polynomiale Vorkonditionierung mit Tschebyscheff-Polynomen und verschiedenen Eigenwertschätzern untersucht. Das letztgenannte Verfahren erweist sich als eine einfache und effiziente Erweiterung eines CG-Verfahrens auf Parallelrechnern. Die polynomiale Vorkonditionierung benötigt Eigenwertschätzungen der extremen Eigenwerte der Koeffizientenmatrix. Hier werden zwei Methoden behandelt, zum einen die Verwendung von Gerschgorin-Schätzwerten, zum anderen die unter Ausnutzung des Zusammenhangs zwischen dem CG-Verfahren und dem Lanzcos-Verfahren für das symmetrische Eigenwertproblem resultierende adaptive Schätzung. Die polynomiale Vorkonditionierung führt zu einer erhöhten Stabilität des Verfahrens und in der Kombination mit dem Diagonal Scaling und der adaptiven Eigenwertschätzung zu einer hohen Reduktion der Ausführungszeit.

Ein wesentlicher Vorteil dieses Verfahrens für den Anwender liegt in der einfachen Implementierung – auch auf einem Rechner mit Message-Passing-Programmiermodell – und in der einfachen Handhabung. Neben einigen voreingestellten Parametern muß nur der Polynomgrad vorgegeben werden.

Ein hoher Polynomgrad erweist sich als günstig, falls das Verfahren ohne Vorkonditionierung nicht oder sehr schlecht konvergiert. In diesem Fall kann durch einen

hohen Grad des Polynoms die Stabilität des Verfahrens deutlich verbessert werden. Die Reduktion der Kondition der Koeffizientenmatrix kann über den Polynomgrad, die Eigenwertschätzungen und die übrigen Parameter, wie Vorlaufzeit und Aktualisierungszyklus, gesteuert werden. Da jedoch ein hoher Polynomgrad zu einer erhöhten Anzahl arithmetischer Operationen führt, sind die Ausführungszeiten im adaptiven Verfahren in der Regel mit dem Grad 2 optimal. Die günstigste Wahl der Parameter ist stark von dem Zeitaufwand der Kommunikation und somit von der Anzahl verwendeter Prozessoren und den rechnerspezifischen Ausführungszeiten für den Nachrichtenaustausch abhängig.

In jedem Iterationsschritt müssen beim mit einem Polynom m-ten Grades vorkonditionierten Verfahren $m+1$ Matrix-Vektor-Produkte berechnet werden; die Anzahl der Skalarprodukte bleibt gleich. Der wesentliche Effekt der polynomialen Vorkonditionierung liegt in der Tatsache, daß die Gesamtanzahl der Matrix-Vektor-Produkte beim Verfahren mit Vorkonditionierung steigt, die Anzahl der Skalarprodukte jedoch sinkt. Dies bedeutet für die Parallelisierung einen erhöhten Anteil lokaler Rechnungen und weniger globale Kommunikation, die für Skalarprodukte erforderlich ist.

Zeigt die polynomiale Vorkonditionierung im sequentiellen Fall nur bescheidenen Erfolg, so skaliert dieses Verfahren im parallelen Fall deutlich besser als das Verfahren ohne Vorkonditionierung. Der Grund ist die erhöhte Anzahl lokaler Berechnungen. So ist schließlich bei fast allen untersuchten Beispielen das CG-Verfahren mit Diagonal Scaling und adaptiver polynomialer Vorkonditionierung das optimale Verfahren.

Wird zur Diskretisierung ein Differenzenverfahren mit einem Laplace-Stern verwendet, das auf eine große, diagonaldominante Matrix mit schlechter Kondition führt, so erweist sich die Methode der Gerschgorin-Schätzungen für die extremen Eigenwerte als sinnvoll und sehr effizient, da hohe Polynomgrade verwendet werden können. Da in Finite-Elemente-Diskretisierungen derartige Matrizen in der Regel nicht entstehen, ist i.a. das adaptive Verfahren vorzuziehen.

Zusammenfassend läßt sich sagen, daß sich die adaptive polynomiale Vorkonditionierung sehr gut für parallele CG-Verfahren eignet. Zum einen ist dieser Vorkonditionierer einfach zu implementieren, da nur Bestandteile des CG-Verfahrens benötigt werden, zum zweiten ist für die Parallelisierung die hohe Skalierbarkeit des Verfahrens von Vorteil; schließlich legt auch das sehr gutartige Verhalten in Verbindung mit dem Diagonal Scaling die gemeinsame Anwendung dieser Vorkonditionierer nahe. Weiterhin eröffnet das adaptive Verfahren durch die flexible Reduktion der Rundungsfehler über den Polynomgrad neue Möglichkeiten zur Bestimmung einer genaueren Lösung.

Das CG-Verfahren mit adaptiver Vorkonditionierung wird somit den gewachsenen Ansprüchen bezüglich des Einsatzes hoher Prozessorzahlen, erhöhter Zuverlässigkeit der Konvergenz und der deutlichen Reduktion der Rechenzeit gegenüber dem zugrundeliegenden CG-Verfahren ohne Vorkonditionierung gerecht.

7.2 Ausblick

Sicherlich ist das Thema *polynomiale Vorkonditionierung* hier nicht abschließend behandelt worden. Die erzielten Resultate lassen jedoch eine optimistische Prognose dieser Verfahren für massiv-parallele Rechnersysteme zu. Es existiert noch eine ganze Reihe von Verbesserungsmöglichkeiten, von denen einige kurz erläutert werden sollen.

Der größte Performancegewinn ist durch verbesserte Kommunikationsroutinen zu erwarten, da die Möglichkeiten der Hardware beim PARAGON bisher nicht ausgeschöpft wurden. Dies führt u.a. zum sinnvollen Einsatz höherer Prozessorzahlen. Da auf dieses Problem jedoch der Anwender keinen Einfluß hat, soll darauf hier nicht näher eingegangen werden.

Für die Implementierung ergeben sich z.B. folgende Möglichkeiten, die zum Teil ausgetestet wurden:

Bandbreitenminimierung

Durch eine Umnumerierung der Variablen kann die Bandbreite der Koeffizientenmatrix verkleinert werden. Ein häufig angewandtes Verfahren ist die Methode *Reverse Cuthill-McKee*. Die Reduktion der Bandbreite führt zu einer erhöhten Anzahl lokaler Berechnungen und somit zu einer Einsparung von Kommunikation [9].

Die Minimierung der Bandbreite ist nicht das primäre Ziel, sondern die Minimierung der Kommunikation. Somit müßte eine Umnumerierung derart stattfinden, daß möglichst wenig Kommunikation durchgeführt werden muß. Solche Algorithmen sind u.a. abhängig von der verwendeten Datenverteilung.

Alternative Aktualisierungsstrategien für das Eigenwertintervall

Falls sich die Schätzwerte für die Eigenwerte ändern, wird in der vorliegenden Implementierung in jedem Fall das Intervall aktualisiert. Es muß eine Reinitialisierung des CG-Verfahrens durchgeführt werden. Alternativ hierzu kann man eine der folgenden Aktualisierungsstrategien verwenden, die zu einer in der Regel verminderten Anzahl Aktualisierungen führen:

1. Aktualisierung nur, falls sich a um $p\%$ verkleinert bzw. b vergrößert,
2. Verwenden nicht der extremen Eigenwerte der Lanczos-Tridiagonalmatrix, sondern Weglassen der l kleinsten bzw. größten Eigenwerte,
3. Einführung eines asymptotischen Konvergenzfaktors [3]; die Aktualisierung wird nur durchgeführt, falls sich dieser Faktor verbessert.

Alle diese Strategien können in speziellen Fällen zu einer beschleunigten Konvergenz führen; jedoch erweisen sich die in dieser Arbeit ermittelten Schätzwerte bereits als gut geeignet.

Verwendung anderer Polynome

Zur Verwendung von Tschebyscheff-Polynomen zur Vorkonditionierung werden Schätzwerte der extremen Eigenwerte benötigt. Es wird die maximale Differenz aller Werte in demjenigen Intervall, welches durch die extremen Eigenwerte gegeben ist, zu 1 minimiert. Es findet somit eine gleichmäßige Behandlung aller Bereiche dieses Intervalls statt. Da die Lanczos-Matrizen jedoch auch Informationen über die Eigenwertverteilung der Koeffizientenmatrix enthalten, sollten Bereiche, in denen sich Eigenwerte verdichten, stärker berücksichtigt werden. Dies kann z.B. durch gewichtete least-squares Polynome geschehen. Vermutlich wird sich die hohe Skalierbarkeit auf diese und andere Polynome, die aus einer anderen zur Minimierung verwendeten Norm entstehen, übertragen. Insbesondere kann im Algorithmus von Seite 39 die Tschebyscheff-Iteration durch eine andere Iteration gleicher Struktur, d.h. eine Folge von Matrix-Vektor-Produkten, ersetzt werden. Es ist ferner interessant, das Verhalten des Verfahrens für indefinite symmetrische Matrizen (s. Kap.3.4) zu untersuchen.

Untersuchung des Polynomgrades bei höheren Prozessorzahlen

Die Untersuchung größerer Beispiele macht den Einsatz höherer Prozessorzahlen sinnvoll. Insbesondere ist von Interesse, wie sich der optimale Polynomgrad bei steigender Prozessorzahl verhält. Lag der günstigste Grad in den hier betrachteten Fällen bei 2, so ist es denkbar, daß bei höheren Prozessoranzahlen die Bedeutung der Lokalität zunimmt und somit höhere Polynomgrade zu besseren Ergebnissen führen.

Literaturverzeichnis

[1] L.M. Adams, R. J. Leveque, D.M. Young *Analysis of the SOR-Iteration for the 9-point Laplacian*, SIAM J. Numer. Anal., 26:152-168 (1988)

[2] St. Ashby *Adaptive Polynomial Preconditioning for Hermitian Indefinite Linear Systems*, BIT,29:583-609 (1989)

[3] St. Ashby, Th. Manteuffel, J. Otto *A Comparison of Adaptive Chebyshev and Least Squares Polynomial Preconditioning for Hermitian Positive Definte Linear Systems*, UCRL-JC-106726 (1991)

[4] St. Ashby *Minimax Polynomial Preconditioning for Hermitian Linear Systems*, SIAM J.Matrix Anal. Appl.,12:766-789 (1991)

[5] St. Ashby, Th. Manteuffel, P. Saylor *A Taxonomy for Conjugate Gradient Methods*, SIAM J. Numer. Anal.,27:1542-1568 (1990)

[6] C. Aykanat, F. Özgüner, D.S. Scott *Vectorization and parallelization of the conjugate gradient algorithm on hypercube-connected vector processors* , Microprocessing and Microprogramming, 29:67–82 (1990)

[7] A. Basermann *Datenverteilungs- und Kommunikationsmodelle für parallele CG-Verfahren zur Lösung von Gleichungssystemen mit dünnbesetzter Koeffizientenmatrix aus FE-Anwendungen*, Aachener Informatik-Berichte Nr.93-7, Graduiertenkolleg Informatik und Technik, Fachgruppe Informatik der RWTH Aachen (1993)

[8] A. Basermann *Conjugate Gradients parallelized on the Hypercube*, Int. Journal of Modern Physics C, Vol.4, No. 6:1295-1306 (1993)

[9] A. Basermann *Data Distribution and Communication Schemes for Solving Sparse Systems of Linear Equations from FE Applications by Parallel CG Methods*, Forschungszentrum Jülich, Interner Bericht KFA-ZAM-IB-9323 (1993)

[10] W. Bunse, A. Bunse-Gerstner *Numerische Lineare Algebra*, Teubner (1988)

[11] P. Deuflhard, A. Hohmann *Numerische Mathematik*, de Gruyter (1991)

[12] I.S. Duff *User's Guide for the Harwell-Boeing Sparse Matrix Collection, Release I*,Technical Report TR/PA/92/86, CERFACS, Toulouse Cedex (1992)

[13] G. Engeln-Müllges, F. Reutter *Numerische Mathematik für Ingenieure*, BI Wissenschaftsverlag (1985)

[14] A. Frommer *Lösung linearer Gleichungssysteme auf Parallelrechnern*, Vieweg (1990)

[15] G.H. Golub, Ch.F. van Loan *Matrix Computations*, 2. Auflage, John Hopkins University Press (1989)

[16] G.H. Golub, M.D. Kent *Estimates of Eigenvalues of Iterative Methods*, Math. of Comp., 53:619-626 (1989)

[17] R.T. Gregory, D.L. Karney *A Collection of Matrices for testing Computational Algorithms*, Robert E. Krieger Publishing Co. (1978)

[18] W. Hackbusch *Iterative Solution of Large Sparse Systems of Equations* , Springer (1994)

[19] W.W. Hager *Applied Numerical Algebra* , Prentice-Hall International (1988)

[20] L.A. Hageman, D.M. Young *Applied Iterative Methods* , Academic Press (1981)

[21] M.R. Hestenes, E. Stiefel *Methods of conjugate gradients for solving linear systems*, Journal of Research of the National Bureau of Standards, 49:409-436 (1952)

[22] O.G. Johnson, Ch.A. Micchelli, G. Paul *Polynomial Preconditioners for Conjugate Gradient Calculations*, SIAM J. Numer. Anal., 20:363-376 (1983)

[23] A. Kielbasinski, H. Schwetlick *Numerische lineare Algebra*, Harri Deutsch (1988)

[24] J. M. Ortega *Introduction to parallel and vector solution of linear systems*, Plenum Press New York, London (1988)

[25] G. Pini, G. Gambolati *Is a simple diagonal scaling the best preconditioner for conjugate gradients on supercomputers ?*, Adv. Water Resources, 13:147-153 (1990)

[26] T.J. Rivlin *The Chebyshev Polynomials* , John Wiley (1974)

[27] Y. Saad *Practical Use of Polynomial Preconditioning for the Conjugate Gradient Method*, SIAM J. Sci. Stat. Comput.,Vol.6, 4:865-881 (1985)

[28] J. Stoer *Numerische Mathematik I*, 2. Auflage, Springer (1976)

[29] J. Stoer, R. Bulirsch *Numerische Mathematik II*, 2. Auflage, Springer (1976)

[30] W. Törnig, P. Spellucci *Numerische Mathematik für Ingenieure und Physiker, Band 1, Numerische Methoden der Algebra* , 2. Auflage, Springer (1988)

[31] H. Vereecken, G. Lindenmayr, A. Kuhr, D.H. Welte, A. Basermann *Numerical modelling of field scale transport in heterogeneous variably saturated porous media*, KFA/ICG-4 Internal Report No. 500393 m (1993)

[32] G.T. Yeh *3DFEMWATER: A Three-Dimensional Finite Element Model of Water Flow through saturated-unsaturated Media*, Oak Ridge National Laboratory ORNL-6386 (1987)

[33] D. Young *A Historical Overview of Iterative Methods*, Computer Physics Communications, 53:1-17 (1989)

[34] *iPSC/860 System User's Guide*, Intel Corporation, Order Number 312312-001 (1992)

[35] *PARAGON OSF/1 User's Guide*, Intel Corporation, Order Number 312489-001 (1993)

[36] *SMART Benutzerhandbücher, Institut für Statik und Dynamik der Luft- und Raumfahrtkonstruktionen der Universität Stuttgart*, ISD-Berichte (1976-1992)

Dank

Die vorliegende Arbeit ist im Zentralinstitut für Angewandte Mathematik (ZAM) des Forschungszentrums Jülich erstellt worden. Ich möchte dem Institutsleiter Herrn Prof. Dr. F. Hoßfeld für die Möglichkeit, die Arbeit in seinem Institut anzufertigen und die Betriebsmittel zu nutzen, herzlich danken.

Herrn Prof. Dr. S. Filippi vom Lehrstuhl für Numerische Mathematik der Justus Liebig Universität Gießen danke ich für die Themenstellung und sein Interesse an der Arbeit.

Bei Herrn Dr. P. Weidner, dem Leiter der Abteilung *Mathematik* im ZAM, möchte ich mich für die vielen fruchtbaren Gespräche und Hilfestellungen bedanken.

Mein besonderer Dank gilt Herrn Dipl.-Ing. A. Basermann. Neben der intensiven fachlichen Beratung und den vielen wertvollen Anregungen sei hier insbesondere das sehr angenehme konstruktive Arbeitsklima erwähnt, welches diese Arbeit positiv beeinflußt hat. Weiterhin danke ich ihm für die Bereitstellung seiner Programme.

Herrn Dipl.-Ing. M. Bücker danke ich für die zahlreichen Diskussionen und gemeinsamen Problembewältigungen sehr herzlich.

Für die flexible Bereitstellung der Rechenanlage möchte ich mich bei Frau J. Docter bedanken.

An dieser Stelle möchte ich mich auch bei all denen bedanken, ohne deren Mithilfe die Durchführung meines Studiums nicht denkbar gewesen wäre. Zu allererst sei hier meine Frau Elke erwähnt, deren Hilfe wesentlich zum Gelingen meines Studiums beigetragen hat.
Für die geleistete Unterstützung möchte ich mich auch herzlich bei meinen Eltern Rudi und Karola Schelthoff sowie bei Elli und Karl-Heinz Voß bedanken.