Iterative Methoden zur Lösung von Gleichungssystemen

Matthias Heming

29. Januar 2009

Inhaltsverzeichnis

[1] Veröffentlich als LaTeX-Quelltext und PDF-Datei unter http://blog.familie-heming.de

1 Einleitung

1.1 Einordnung des Arbeit

Im Rahmen des Seminars *Schwachbesetzte Systeme* soll die vorliegende Arbeit eine groben Überblick über verschiedene iterative Methoden zur Lösung linearer Gleichungssysteme bieten. Dabei wird zwischen stationären und nicht stationären Verfahrensweisen unterschieden. Als Beispiel für stationäre Verfahren werden das Jacobi- und Gauss-Seidel-Verfahren vorgestellt, ohne jedoch auf Details zum Konvergenznachweis einzugehen.

Um den Bezug zu Sparse-Matrizen herzustellen, wurde der Algorithmus des Jacobi-Verfahren in zwei verschiedenen Codevarianten in Matlab implementiert. Je nach Schreibweise werden matlabintern verschiedenen Methoden verwendet, die eine starke Optimierung bei der Rechnung mit dünn besetzten Matrizen ermöglichen.

Als Beispiel für ein nicht stationäres Verfahren wird das Verfahren der *konjugierten Gradienten (CG)* näher erläutert. Die Verfahren *Steepest Decent* und *Conjugate Directions* werden dabei erklärt, um ein besonders intuitives Verständnis des CG-Verfahrens zu ermöglichen (Shewchuk (1994)). Anhand des Implementierungsbeispiels soll offensichtlich werden, dass eine Optimierung für Sparse-Matrizen nicht notwendig ist, da diese implizit durch Anwendung von Matrix-Vektor-Multiplikationen realisiert wird.

Auf verschiedene andere Verfahren, die keine speziellen Vorbedingungen wie das CG-Verfahren benötigen, wird nur im Rahmen einer zitierten Aufzählung mit minimaler Beschreibung eingegangen.

1.2 Erste Mathematik

Gelöst werden soll das Gleichungssystem

$$Ax = b \qquad A \in \mathbb{R}^{n \times n} \quad x, b \in \mathbb{R}. \tag{1}$$

Im Rahmen dieser Arbeit soll der Fall eines singulären A nicht betrachtet werden. Gleichung 1 sei daher unter der Bedingung der Regularität eindeutig lösbar.

Das Ziel bei der Entwicklung von iterativen Methoden ist dabei die Aufstellung einer Iterationsvorschrift, die als einzigen Fixpunkt x die Lösung aus Gleichung 1 beinhaltet. Dabei unterscheidet man Methoden, die in jedem Iterationsschritt die gleiche Funktion anwenden (stationär), und Methoden, deren Iterationsfunktion vom aktuellen Iterationsschritt abhängt (nicht-stationär).

2 Stationäre Methoden

Um aus Gleichung 1 eine Iterationsvorschrift zu erstellen, nutzt man eine Zerlegung $A = M - N$, so dass sich die folgenden Umformungen ergeben:

$$\begin{aligned} & (M - N)x = b \\ \Leftrightarrow \quad & Mx = Nx + b \\ \Leftrightarrow \quad & x = M^{-1}(Nx + b) = M^{-1}Nx + M^{-1}b. \end{aligned} \tag{2}$$

Dabei muss natürlich beachtet werden, dass nur eine Zerlegung mit regulärem M genutzt werden kann. Ein erster Gedanke ist nun, Gleichung 2 einfach naiv als Iterationsvorschrift zu interpretieren:

$$x^{(i+1)} = M^{-1}(Nx^{(i)} + b) = M^{-1}Nx^{(i)} + M^{-1}b. \tag{3}$$

Ob dabei tatsächlich ein sinnvolles Verfahren entsteht, hängt unmittelbar von der Iterationsmatrix $M^{-1}N$ und damit von der Zerlegung ab. Neben dem Nachweis der Konvergenz ist es ebenfalls wichtig, dass die Invertierung der Matrix M bzw. die Lösung des Gleichungssystems $My = Nx^{(i)}$ nach y mit sehr geringem Aufwand möglich ist.

2.1 Fehlerbetrachtung

Auch wenn die Verfahrenskonvergenz nicht einzeln gezeigt wird, so soll ein möglicher Ansatz angedeutet werden. Dazu betrachtet man den Fehler im i-ten Iterationsschritt

$$e^{(i)} = x^{(i)} - x$$

mit x als eindeutige Lösung der Gleichung 1. Subtrahiert man Gleichung 2 von Gleichung 3, so ergibt sich die folgende rekursive Berechnungsvorschrift für die Fehlerterme:

$$e^{(i+1)} = x^{(i+1)} - x = M^{-1}N(x^{(i)} - x) = M^{-1}Ne^{(i)}.$$

Auch wenn die Fehlerterme konkret nicht berechenbar sind – dazu muss die Lösung bekannt sein – so kann auf diese Weise analysiert werden, welche Auswirkungen die Iteration auf sie hat. Passend dazu kann gezeigt werden, dass ein solches Iterationsverfahren dann konvergiert, wenn der Spektralradius der Iterationsmatrix kleiner als 1 ist,

$$\varrho(M^{-1}N) < 1.$$

2.2 Abbruchbedingung

Der tatsächliche Fehler aus dem vorherigen Abschnitt kann nicht berechnet werden. Also ich es prinzipiell ebenfalls nicht möglich, festzustellen, wie genau die aktuelle Iterierte der tatsächlichen Lösung angenähert ist. Eine Möglichkeit, eine sinnvolle Abbruchbedingung herzustellen, ist die Untersuchung des Residuums (vgl. (Quarteroni u. Saleri, 2006, S. 138))

$$r^{(i)} = b - Ax^{(i)}. \tag{4}$$

Für die tatsächliche Lösung gilt $r = 0$. Einen Schätzer für den relativen Fehler bekommt man bei vorgegebener Genauigkeit $\varepsilon > 0$ mit

$$||r^{(i)}|| \leq \varepsilon ||b||.$$

Man beachte jedoch, dass eine geringe Residuumsnorm nur bedeutet, dass $Ax^{(i)}$ und b sich recht ähnlich sind. Tatsächliche Lösung und Iterierte sind nur im Falle einer gut Konditionierten Matrix ebenfalls ähnlich, für schlecht Konditionierte Matrizen gilt dieses nicht. Es kann keine Auskunft über die Genauigkeit gegeben werden – man hofft also, dass man die Näherungslösung dann doch irgendwie gebrauchen kann..

2.3 Das Jacobi-Verfahren

Beim Jacobiverfahren wird die Zerlegung $A = D - B$ gewählt, wobei D der Diagonalanteil der Matrix A und $B = D - A$ den Rest darstellt. Damit ist die Iterationsmatrix

$$M^{-1}N = D^{-1}(D - A) = I - D^{-1}A. \tag{5}$$

Zur Leistungsfähigkeit sagt Tessarek u. a. (1998):

- Besonders einfach in der Verwendung, aber nur langsam konvergierend. Außer bei starker Diagonaldominanz von A ist das Jacobi-Verfahren nur als vorkonditionierendes Verfahren für nichtstationäre Methoden zu empfehlen.
- Parallelisierung ist sehr einfach durchzuführen.

An verschiedenen Stellen wird das Verfahren beschrieben, indem die Formel zur Berechnung der einzelnen Komponenten der Iterierten angegeben wird. Dabei ergibt sich für die $(k+1)$-te Iterierte mit Komponenten $i = 1, \ldots, n$ die Formel

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j \neq i} a_{ij} x_j^k \right).$$

Obwohl mathematisch zwar vollkommen richtig, ist die naive Implementation nach dieser Formel (Algorithmus 1) in Matlab recht langsam. Dies wird klar, wenn man bedenkt, dass bei dieser Formulierung auf alle Komponenten der Matrix bzw. des Vektors explizit zugegriffen wird und keinerlei optimierte Sparse-Matrix-Operationen verwendet werden können. Man betrachte daher die Iteration in ihrem Ursprung als Matrix-Vektor-Multiplikation:

$$x^{(i+1)} = D^{-1}(Bx^{(i)} + b).$$

Eine weiter vereinfachte Schreibweise ist möglich, wenn man die umgeformte Zerlegung $B = D - A$ einsetzt, und das Residuum (vgl. Gleichung 4) ins Spiel bringt:

$$\begin{aligned} x^{(i+1)} &= D^{-1}(Dx^{(i)} - Ax^{(i)} + b) \\ &= x^{(i)} + D^{-1}(b - Ax^{(i)}) \\ &= x^{(i)} + D^{-1}r^{(i)}. \end{aligned} \tag{6}$$

Ein in dieser Form implementierter Algorithmus (Algorithmus 2) hat ein wesentlich besseres Laufzeitverhalten. Die Berechnung von D^{-1} ist – unter der Bedingung, dass alle Diagonalelemente ungleich null sind – sehr einfach. Da das Residuum für die Abbruchbedingung bereits berechnet werden muss, kann es nun direkt weiterverwendet werden und es entsteht kein zusätzlicher Arbeitsaufwand.

Algorithm 1: Jacobi-Iteration mit schlechtem Laufzeitverhalten in Matlab

Eingabe: Matrix A, rechte Seite b und Startvektor x
$imax$: maximale Anzahl der Iterationsschritte
tol: Genauigkeit, bei der die Iteration abbricht
Ausgabe: x: berechnete Näherungslösung

Beginn
  $iter := 0$;
  $bnorm = norm(b)$;
  $resnorm := norm(b - A \cdot x)$;
  $n = dim(A)$;
  solange *(iter < imax) und (resnorm/bnorm > tol)* **tue**
    für $i := 1$ *bis* n **tue** `%Nächste Iterierte Berechnen`
      $summe = 0$;
      für $j := 1$ *bis* n **tue**
        wenn $(i \neq j)$ **dann** $summe = summe + A_{ij}x_j$;
      $xneu_i = (b_i - summe)/a_{ii}$;
    `%Abbruchbedingung aktualisieren`
    $resnorm = norm(b - A \cdot xneu)$;
    $x = xneu$;
    $iter = iter + 1$;
Ende

Algorithm 2: Jacobi-Iteration mit verbessertem Laufzeitverhalten in Matlab

Eingabe: Matrix A, rechte Seite b und Startvektor x
$imax$: maximale Anzahl der Iterationsschritte
tol: Genauigkeit, bei der die Iteration abbricht
Ausgabe: x: berechnete Näherungslösung

Beginn
  $iter := 0$;
  $res = b - A \cdot x$;
  $resnorm = norm(res)$;
  $bnorm := norm(b)$;
  $D := diag(A)$ `%Vektor mit Diagonalelementen`
  solange *(iter < imax) und (resnorm/bnorm > tol)* **tue**
    $x = x - res./D$ `%Komponentenweise Division`
    $res = b - A \cdot x$;
    $resnorm = norm(res)$;
    $iter = iter + 1$;
Ende

2.4 Das Gauß-Seidel-Verfahren

Seien L und U der untere bzw. obere Dreiecksanteil (ohne Diagonale) der Matrix A. Bei der Gauß-Seidel-Iteration wird nicht nur die Diagonale, sondern ebenfalls der untere Dreiecksanteil als Matrix M extrahiert, es ist $M = D + L$ und damit $N = -U = (D + L) - A$. Damit lautet die Iterationsmatrix

$$M^{-1}N = (D + L)^{-1} \cdot (D + L - A) = I - (D + L)^{-1}A.$$

Es ergibt sich die Iterationsvorschrift

$$\begin{aligned} x^{(i+1)} &= x^{(i)} - (D + L)^{-1}Ax^{(i)} + (D + L)^{-1}b \\ &= x^{(i)} + (D + L)^{-1}(b - Ax^{(i)}) \\ &= x^{(i)} + (D + L)^{-1}r^{(i)}. \end{aligned} \tag{7}$$

Da $D + L$ eine Dreiecksmatrix ist, ist die Lösung der Gleichung $(D + L)y = r^{(i)}$ – welche der Invertierung von $(D + L)$ entspricht – durch einfache Vorwärtssubstitution möglich. Da eine Vorwärtssubstitution nicht direkt in Matlab implementiert ist und eine for-Schleife als eigene Implementierung viel zu langsam ist, muss ein anderer Weg zur Berechnung gefunden werden. Da die Gaußelimination bei einer Matrix eine Dreiecksform herstellt und danach entsprechend substituiert, kann man hoffen, dass der \-Operator die bereits vorliegende Dreiecksform erkennt und nur die notwendige Substitution vornimmt. Das Laufzeitverhalten von Algorithmus 3 ist zumindest recht gut und spricht für die Erfüllung dieser Hoffnung.

Algorithm 3: Gauß-Seidel-Iteration

Eingabe: Matrix A, rechte Seite b und Startvektor x
$imax$: maximale Anzahl der Iterationsschritte
tol: Genauigkeit, bei der die Iteration abbricht
Ausgabe: x: berechnete Näherungslösung

Beginn
  $iter := 0$;
  $res = b - A \cdot x$;
  $resnorm = norm(res)$;
  $bnorm := norm(b)$;
  $DL := tril(A)$;
  solange *(iter < imax) und (resnorm/bnorm > tol)* **tue**
    $x = x + DL \backslash res$;
    $res = b - A \cdot x$;
    $resnorm = norm(res)$;
    $iter = iter + 1$;
Ende

Es ist hier wieder Tessarek u. a. (1998) zitiert, der zur Leistungsfähigkeit des Gauß-Seidel-Verfahrens sagt:

- Schnellere Konvergenz als beim Jacobi-Verfahren, aber trotzdem in vielen Fällen nicht konkurrenzfähig gegenüber den nichtstationären Verfahren.
- Anwendbar auf strikt diagonaldominante oder symmetrische positiv definite Matrizen.
- Eignung zur Parallelisierung ist von der Struktur von A abhängig. Unterschiedliche Anordnungen der Unbekannten bewirken verschiedene Grade der Parallelisierbarkeit.

2.5 Weitere Verfahren

Ob ein Verfahren für eine bestimmte Matrix konvergiert bzw. wie schnell es konvergiert, kann ebenfalls durch einen sogenannten Relaxationsfaktor beeinflusst werden. Mit Hilfe des Relaxationsparameter wird nur der Informationsanteil aus der Matrix A mit einem Faktor $\omega \in \mathbb{R}$ gewichtet. Die Zerlegung des relaxierten Jacobi-Verfahren ist nun $M = \omega D$ bzw. $N = \omega D - A$ und die Iterationsmatrix lautet:

$$M^{-1}N = \frac{1}{\omega}D^{-1} \cdot (\omega D - A) = I - \frac{1}{\omega}D^{-1}A.$$

Die Matrix M beim Gauß-Seidel-Verfahren könnte mit Hilfe eines Relaxationsparamters auch wie folgt gewählt werden:

$$M = \frac{1}{\omega} D + L.$$

Für $\omega > 1$ spricht man in diesem Fall von einer *Überrelaxation*. Dieser Name gab dem Verfahren, welches hauptsächlich mit $\omega \in (0, 2)$ angewendet wird, den neuen Namen *Successive Overrelaxation (SOR)*.

Wie bereits angedeutet, können je nach Wahl der Zerlegung sehr verschiedene Verfahren erzeugt werden, auf diese hier nicht näher eingegangen wird.

3 Das CG-Verfahren

Zu allererst eine erste Einschränkung: Das CG-Verfahren funktioniert von der Idee her nur mit symmetrisch positiv definiten Matrizen. Obwohl zum Verständnis des Verfahrens ein langer Umweg gemacht werden muss, so sollte die Notwendigkeit dieser Bedingung bereits im ersten Schritt klar werden. Dazu wird das Problem des Lösen der Gleichung 1 auf ein etwas höhergelegenes Niveau transferiert.

3.1 Das Minimierungsproblem

Betrachtet man die quadratische Form

$$f : \mathbb{R}^n \to \mathbb{R} \qquad f(x) = \frac{1}{2} x^T A x - b^T x + c, \quad c \in \mathbb{R}, \tag{8}$$

so hat ein findiger Mensch festgestellt, dass für den Gradient von $f(x)$ gilt:

$$f'(x) = \begin{pmatrix} \frac{\partial}{\partial x_1} f(x) \\ \frac{\partial}{\partial x_2} f(x) \\ \cdots \\ \frac{\partial}{\partial x_n} f(x) \end{pmatrix} = \frac{1}{2} A^T x + \frac{1}{2} A x - b = A x - b. \tag{9}$$

Im letzten Schritt von Gleichung 9 ist dabei die Bedingung der Symmetrie von A eingegangen. Es kann damit das Ursprungsproblem der Lösung von $Ax = b$ aufgefasst werden als die Suche nach einem Extremwert der Gleichung 8.

Weiterhin kann nun gezeigt werden, dass für positiv definites A nur ein Extremwert, nämlich ein globales Minimum, existiert[2].

Shewchuk (1994) zeigt Abbildung 1 als Illustration für das Beispielproblem

$$A = \begin{pmatrix} 3 & 2 \\ 2 & 6 \end{pmatrix} \qquad b = \begin{pmatrix} 2 \\ -8 \end{pmatrix} \qquad c = 0. \tag{10}$$

[2]Zwar nicht trivial, aber auch weniger kompliziert, sondern eher aufwendig und daher nicht Teil dieser Arbeit, kann in (Shewchuk, 1994, S. 5) nachgelesen werden.

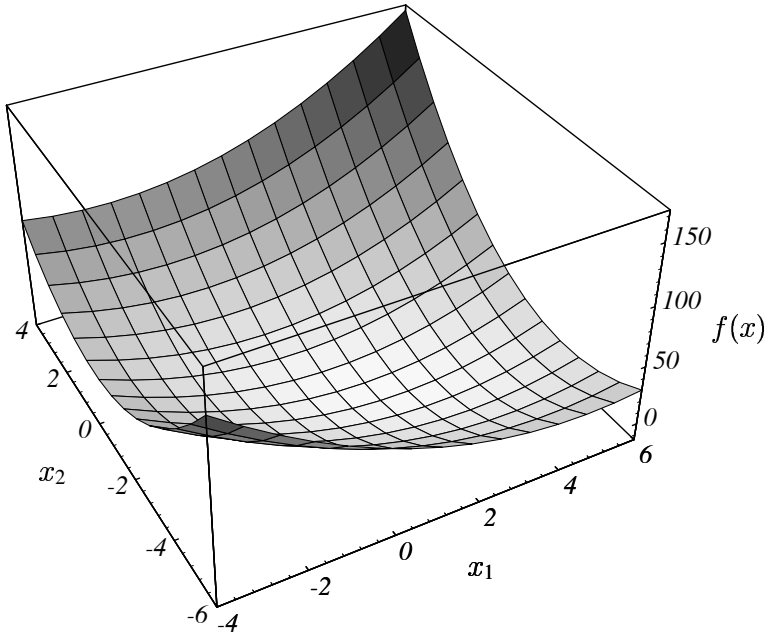

Abbildung 1: Graph der quadratischen Form 10, (Shewchuk, 1994, S. 3)

3.2 Steepest Decent

Für gegebenes $x_{(i)}$ stellt man fest, dass das in Gleichung 4 eingeführte Residuum $r_{(i)}$ dem negativen des Gradienten $f'(x_{(0)}) = Ax_{(0)} - b$ entspricht. Da der Gradient in die Richtung des höchsten Anstieges zeigt, zeigt der Residuenvektor damit in die Richtung des stärksten Gefälles. Für die Suche nach einem globalen Minimum gerade passend. Es stellt sich nun die Frage, wie weit in dieser Richtung gegangen werden soll – welches α in Gleichung 11 benutzt werden soll – nicht, dass man aus Versehen an dem Minimum vorbeiläuft.

$$x_{(i+1)} = x_{(i)} + \alpha r_{(i)} \tag{11}$$

Setzt man $x_{(i+1)}$ in f ein und sucht das Minimum mit Hilfe der Nullstellen der Ableitung nach α, so erhält man den Ausdruck $f'(x_{(i+1)})r_{(i)}$ (Kettenregel). *Man beachte also, dass die einzelnen Residuenvektoren orthogonal zueinander stehen!* Mit Hilfe verschiedener Umformungsschritte[3] kann der optimale Wert für α konkret berechnet werden:

$$\alpha = \frac{r_{(i)}^T r_{(i)}}{r_{(i)}^T A r_{(i)}}. \tag{12}$$

Mit dem bereits bekannten Rahmen für die Berücksichtigung des Abbruchkriteriums aus Abschnitt 2.2 lassen sich die Gleichungen 11 und 12 zu Algorithmus 4 zusammenfassen.

Algorithm 4: Iteration nach dem Prinzip des *Steepest Decent (steilster Abstieg)*

Eingabe: Matrix A, rechte Seite b und Startvektor x
$imax$: maximale Anzahl der Iterationsschritte
tol: Genauigkeit, bei der die Iteration abbricht
Ausgabe: x: berechnete Näherungslösung

Beginn
  $iter = 0$;
  $res = b - A \cdot x$;
  $resnorm = norm(res)$;
  $bnorm = norm(b)$;
  solange *(iter < imax) und (resnorm/bnorm > tol)* **tue**
    $alpha = (res' \cdot res)/(res' \cdot A \cdot res)$;
    $x = x + alpha \cdot res$;
    $res = b - A \cdot x$;
    $resnorm = norm(res)$;
    $iter = iter + 1$;
Ende

[3] Nachzulesen bei (Shewchuk, 1994, S. 6).

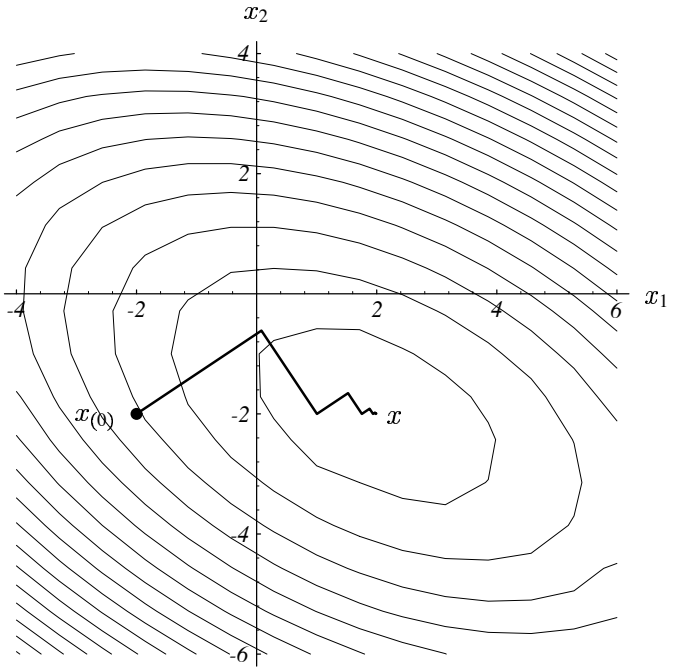

Abbildung 2: Verlauf einer *Steepest Decent* Iteration, (Shewchuk, 1994, S. 8)

3.3 Conjugate Directions

Es lassen sich bei der Methode des steilsten Abstieges zwei Dinge feststellen. Im allgemeinen Fall kann es passieren, dass in unterschiedlichen Iterationsschritten mehrfach in der gleichen Richtung korrigiert wird – vergleiche Abbildung 2. In dem speziellen Fall, dass die Eigenwerte der Matrix gleich sind (die quadratische Form ist nicht elliptisch, sondern kreisförmig), oder der Fehlervektor einem Eigenwert enstpricht (Fehlervektor parallel zu einer Ellipsenachse) kommt die Iteration nach sofort bei der exakten Lösung an – vergleiche Abbildung 3.

Teilt man den Fehlervektor zum Startwert in orthogonale Komponenten und legt man die Iteration so an, dass jeder Iterationsschritt jeweils einen dieser Komponenten eliminiert – vergleiche Abbildung 4 –, dann könnte man immerhin in n Schritten bei der exakten Lösung ankommen. Und es wäre natürlich praktisch, wenn diese sehr schnelle Konvergenz ebenfalls nicht auf bestimmte Sonderfälle beschränkt ist.

Ein solches optimiertes Verfahren, welches für eine Menge orthogonaler Suchvektoren pro Suchrichtung so weit läuft, dass die zu dieser Suchrichtung passende Fehlerkomponente vollständig eliminiert wird, muss der Bedingung genügen, dass bei der Richtung $d_{(i)}$ der verbleibende Fehlervektor $e_{(i+1)}$ orthogonal zu $d_{(i)}$ ist.

Einziges Problem dabei ist, dass der verbleibende Fehlervektor $e_{(i+1)}$ leider unbekannt ist. Wenn man aber jetzt anstelle der normalen Orthogonalität eine A-Orthogonalität[4]

[4]Zwei Vektoren, die A-orthogonal sind, werden auch *konjugiert* genannt, woher der Namen des Verfahrens herkommt.

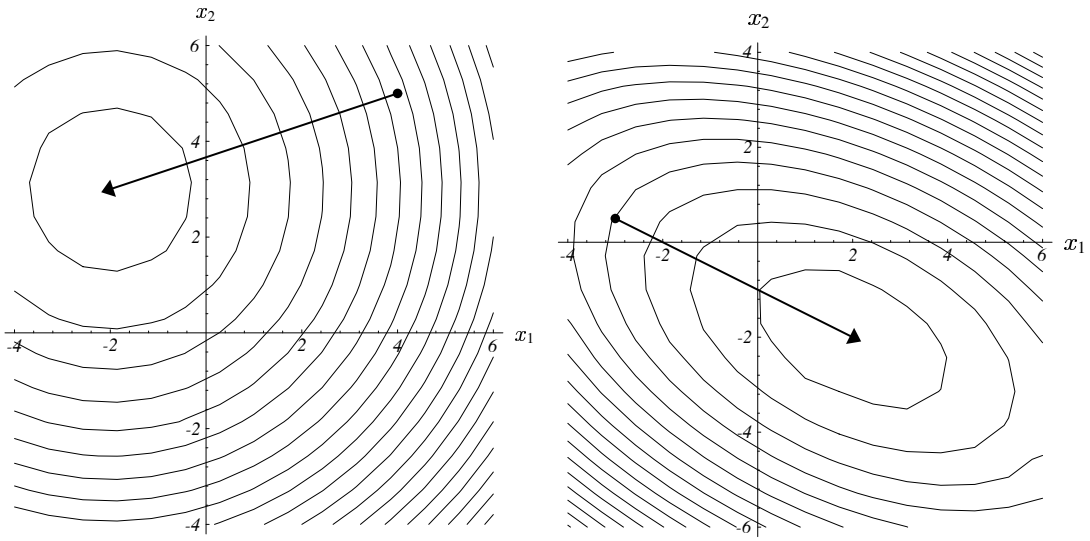

Abbildung 3: Iteration unter optimalen Bedingungen, (Shewchuk, 1994, S. 15,16)

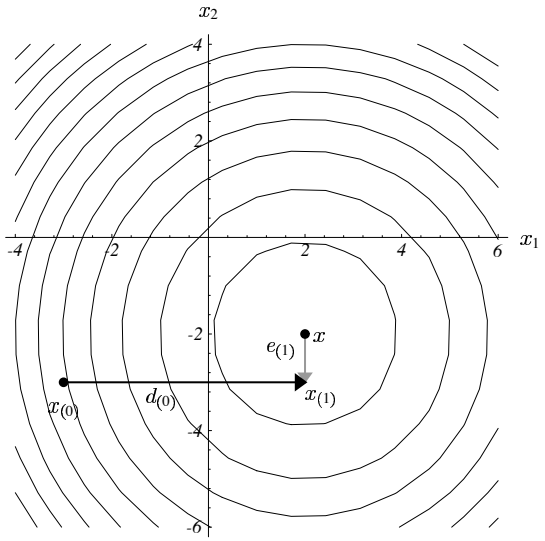

Abbildung 4: Iteration, die die Lösung bereits kennt, (Shewchuk, 1994, S. 22)

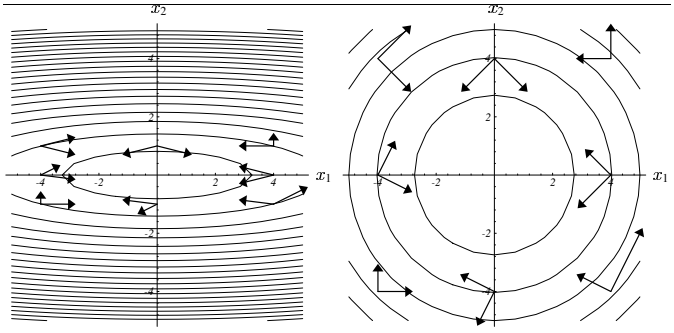

Abbildung 5: A-orthogonale Vektoren (links) werden durch *Ziehen und Zerren* zu orthogonalen Vektoren (rechts) modifiziert, (Shewchuk, 1994, S. 23)

nutzt, wird das Problem auf einmal einfacher. Zur einfacheren Vorstellung betrachte man die quadratische Form als auf *Gummipapier* gedruckt. Abbildung 5 zeigt, wie die Anwendung der A-Orthogonalität die elliptische Form (links) durch *Ziehung und Zerren* in eine kreisförmige Form (rechts) bringt.

Warum wird das Problem einfacher? Nimmt man einmal an, es wäre $e_{(i+1)}$ bekannt, dann könnte man für die Iteration $x_{(i)} + \alpha d_{(i)}$ durch die Bedingung der Orthogonalität von $d_{(i)}$ und $e_{(i+1)}$ analog zur Methode des steilsten Abstiegs das optimale α berechnen[5].

$$\alpha = -\frac{d_{(i)}^T e_{(i)}}{d_{(i)}^T d_{(i)}} \tag{13}$$

Wenn man nun aber konjugierte Suchvektoren nutzt, dann wird Gleichung 13 zu Gleichung 14 und diese ist berechenbar, da $Ae_{(i)} = r_{(i)}$ berechenbar ist.

$$\alpha = -\frac{d_{(i)}^T A e_{(i)}}{d_{(i)}^T A d_{(i)}} = -\frac{d_{(i)}^T r_{(i)}}{d_{(i)}^T A d_{(i)}} \tag{14}$$

Damit ist eigentlich alles geklärt, man braucht nur noch passende A-orthogonale Vektoren. Dort liegt jedoch das Problem. Nimmt man nämlich dazu z. B. das Gram-Schmidt-Verfahren, dann hat dieses den gleichen Aufwand wie eine Gauß-Elimination, womit der Vorteil eines Iterationsverfahren – der geringe Aufwand – dahinschwindet.

Und jetzt endlich kommt das Verfahren der *konjugierten Gradienten* ins Spiel!

3.4 Conjugate Gradients

Bei dem Gram-Schmidt-Verfahren nimmt man n linear unabhängige Vektoren $u_0, \ldots, u_n$. Es kann ohne weiteres $d_{(0)} = u_0$ gesetzt werden, die weiteren A-orthogonalen Richtungen $d_{(1)}, \ldots, d_{(n)}$ werden nun Stück für Stück konstruiert, indem von jedem Vektor u_i die

[5] Vergleiche wieder einmal Shewchuk (1994), diesmal Seite 22.

Anteile subtrahiert werden, die nicht orthogonal zu den früheren Vektoren sind:

$$d_{(i)} = u_i - \sum_{j=0}^{i-1} \beta_{ij} d_{(j)}.$$

Die Berechnung der jeweiligen Koeffizienten β_{ij} geschieht dabei wie folgt:

$$\beta_{ij} = -\frac{u_i^T A d_{(j)}}{d_{(j)}^T A d_{(j)}}.$$

Ein findiger Mathematikzauberer hat erwirkt[6], dass das neue Residuum $r_{(i+1)}$ bereits orthogonal zu allen vorherigen Residuen. Damit ist es sinnvoll, die Residuen als Basisvektoren u_i des Gram-Schmidt-Verfahrens zu nutzen. Ebenfalls hat er erwirkt, dass das Residuum $r_{(i+1)}$ mit Ausnahme von d_i ebenfalls A-orthogonal zu allen vorherigen Suchrichtungen ist. Faszinierenderweise werden damit die Koeffizienten des Gram-Schmidt-Verfahrens wesentlich einfacher, denn es ist $\beta_{ij} = 0$ für $i > j + 1$. Jetzt noch ein wenig Zaubersalz bzw. ein paar mathematische Umformungen und es ergibt sich:

$$\beta_{(i)} = \beta_{i,i-1} = \frac{r_{(i)}^T r_{(i)}}{r_{(i-1)}^T r_{(i-1)}}. \tag{15}$$

Fasst man nun das Verfahren der *conjugate gradients* zusammen, so ergeben sich folgende Schritte:

$$\begin{aligned}
d_{(0)} &= r_{(0)} = b - Ax_{(0)} \quad \text{(Initialisierung)} \\
\alpha_{(i)} &= \frac{r_{(i)}^T r_{(i)}}{d_{(i)}^T A d_{(i)}} \\
x_{(i+1)} &= x_{(i)} + \alpha_{(i)} d_{(i)} \\
r_{(i+1)} &= b - Ax_{(i)} \\
\beta_{(i+1)} &= \frac{r_{(i+1)}^T r_{(i+1)}}{r_{(i)}^T r_{(i)}} \\
d_{(i+1)} &= r_{(i+1)} + \beta_{(i+1)} d_{(i)}
\end{aligned}$$

Als finale Zusammenfassung wird aus diesen Zeilen wird nun Algorithmus 5 kreiert.

Es sei zum Schluss gesagt, dass die hier vorgestellten Algorithmen nur auf Matrix-Vektor- bzw. Vektor-Vektor-Multiplikationen aufbauen und daher sowohl für vollbesetzte als auch für dünn besetze Matrizen gleichermaßen funktionieren, solange die Multiplikationen entsprechen implementiert sind.

[6] Dazu braucht Shewchuk (1994) immerhin mehrere Abschnitte, also wirklich nicht trivial.

Algorithm 5: Das Verfahren der konjugierten Gradienten

Eingabe: Matrix A, rechte Seite b und Startvektor x
$imax$: maximale Anzahl der Iterationsschritte
tol: Genauigkeit, bei der die Iteration abbricht
Ausgabe: x: berechnete Näherungslösung

Beginn
  $iter = 0$;
  $res = b - A \cdot x$;
  $resnorm = norm(res)$;
  $bnorm = norm(b)$;
  $d = res$;
  solange *(iter < imax) und (resnorm/bnorm > tol)* **tue**
    $alpha = (res' \cdot res)/(d' \cdot A \cdot d)$;
    $x = x + alpha \cdot d$;
    `%Vorbereitung für nächste Iteration: Berechnung von neuem` d
    $resneu = b - A \cdot x$;
    $beta = (resneu' \cdot resneu)/(res' \cdot res)$;
    $d = resneu + beta \cdot d$;
    `%Abbruchbedingung berücksichtigen`
    $res = resneu$;
    $resnorm = norm(res)$;
    $iter = iter + 1$;
Ende

3.5 Ausblick

Nach dieser doch recht komplexen Einführung in das Verfahren der konjugierten Gradienten ist nicht darauf eingegangen worden, dass das Verfahren auf Krylow-Unterräumen operiert, diese können recht gut zum Nachweis der Residuenmagie für Gleichung 15 genutzt werden.

Weiterhin wurde nicht eingegangen auf verschiedene andere nicht stationäre Verfahren, welche auch unter weniger strikten Voraussetzungen wie Symmetrie und positive Definitheit ihre Anwendung finden. Dazu zählen z. B. *BiCG*, *CGS*, *BiCGSTAB* oder *GMRES*.

Literatur

[Quarteroni u. Saleri 2006] Quarteroni, Alfio ; Saleri, Fausto: *Wissenschaftliches Rechnen mit MATLAB*. Springer, 2006. – Teile verfügbar unter `http://books.google.com/books?id=04FdFoP8hwkC`

[Shewchuk 1994] Shewchuk, Jonathan R.: *An Introduction to the Conjugate Gradient Method Without the Agonizing Pain*. online. `http://www.cs.cmu.edu/~quake-papers/painless-conjugate-gradient.ps`. Version: August 1994, Abruf: 04. Dezember 2009

[Tessarek u. a. 1998] Tessarek, Helmut ; Rosenauer, Gregor ; Spannocchi, Raphael ; Thiery, Christian ; Gruber, Martin ; Thomas Dorn, Dominik P. ; Phuong, Anh T. ; Trocker, Reinhold: *Numerische Mathematik, interaktiv*. online. `http://www.dorn.org/uni/sls/kap09/i09_00.htm`. Version: 1998, Abruf: 04. Dezember 2009. – Kapitel 9: Schwach besetzte Systeme – Abschnitt 9: Auswahl eines Iterativen Verfahrens