Vertiefung der Numerischen Mathematik

Lars Grüne
Lehrstuhl für Angewandte Mathematik
Mathematisches Institut
Universität Bayreuth
95440 Bayreuth
lars.gruene@uni-bayreuth.de
num.math.uni-bayreuth.de

Vorlesungsskript
Erste Auflage
Sommersemester 2019

Vorwort

Dieses Skript ist im Rahmen einer gleichnamigen Vorlesung entstanden, die ich im Sommersemester 2019 an der Universität Bayreuth gehalten habe. Dabei ist Kapitel 1 eine Neufassung eines bestehenden Kapitels aus meinem Skript "Einführung in die Numerische Mathematik", während die weiteren Kapitel neu geschrieben wurden. Dabei konnte ich auf ein Skript von Prof. Anton Schiela zurückgreifen, das bei der Erstellung dieses Skripts an vielen Stellen ausführlich genutzt wurde. Dafür möchte ich mich an dieser Stelle sehr herzlich bedanken. Darüberhinaus dienten die Numerik-Lehrbücher von Deuflhard und Hohmann [3], Köckler und Schwarz [5] sowie Bornemann [1] als Grundlage.

Ich möchte mich an dieser Stelle wie üblich bei meinen Mitarbeiterinnen und Mitarbeiter n— insbesondere bei Tobias Sproll — sowie bei allen Teilnehmerinnen und Teilnehmern dieser Vorlesung bedanken, die durch ihr aufmerksames Lesen während der Vorlesung oder bei der Prüfungsvorbereitung Fehler im Skript gefunden und mitgeteilt haben.

Bayreuth, Juli 2019 Lars Grüne

Inhaltsverzeichnis

Kapitel 1

Eigenwerte und Eigenvektoren

Eigenwerte von Matrizen spielen in vielen Anwendungen eine Rolle. Gesucht sind dabei diejenigen $\lambda \in \mathbb{C}$, für die die Gleichung

$$Av = \lambda v$$

für einen *Eigenvektor* $v \in \mathbb{C}^n$ erfüllt ist. In vielen Anwendungen ist man darüberhinaus an den zugehörigen Eigenvektoren v interessiert.

Im letzten Kapitel haben wir bei der Betrachtung iterativer Verfahren gesehen, dass die Eigenwerte der Matrix $M^{-1}N$ Auskunft über die Konvergenz dieser Verfahren geben. Dies ist ein generelles Prinzip linearer Iterationen (ähnlich ist dies bei linearen Differentialgleichungen) und ein wichtiges Beispiel für eine Problemklasse, bei der die Kenntnis der Eigenwerte einer Matrix wichtig ist. Weitere Anwendungen sind z.B. das Seitenranking von Google, bei dem Eigenvektoren eine wichtige Rolle spielen (vgl. die Erläuterungen in der Vorlesung oder den Artikel auf der E-Learning Seite zur Vorlesung) oder Anwendungen in der Bildverarbeitung (vgl. das 7. Übungsblatt).

Wir werden in diesem relativ kurzen Kapitel einige Algorithmen zur Berechnung von Eigenwerten und zugehörigen Eigenvektoren für spezielle Matrizen (z.B. symmetrische Matrizen) kennen lernen. Bevor wir mit konkreten Algorithmen beginnen, wollen wir uns allerdings mit der Kondition des Eigenwertproblems beschäftigen.

1.1 Kondition des Eigenwertproblems

Wie üblich beschreibt die Kondition die Stärke der Änderung des Ergebnisses in Abhängigkeit von den Eingabedaten. Diese sind hier die Einträge der Matrix A. Wir müssen also untersuchen, wie sich ein Eigenwert $\lambda_0(A)$ in Abhängigkeit von Änderungen in A verändert, also die Größe

$$\frac{|\lambda_0(A) - \lambda_0(A + \Delta A)|}{\|\Delta A\|}$$

berechnen. Da die Eigenwerte nicht linear von der Matrix A abhängen, ist dieser Ausdruck im Allgemeinen schwer zu berechnen; wie allgemein bei nichtlinearen Problemen üblich, beschränken wir uns daher auf eine lineare Approximation der Änderung von $\lambda_0(A)$, welche

gerade durch die Ableitung $D\lambda_0(A)$ gegeben ist. Diese Ableitung ist dabei für festes A als lineare Abbildung $D\lambda_0(A) : \mathbb{C}^{n\times n} \to \mathbb{C}$ aufzufassen. Wir betrachten für kleine $\|\Delta A\|$ also die Näherung

$$\lambda_0(A + \Delta A) = \lambda_0(A) + D\lambda_0(A)\Delta A + R(\Delta A),$$

wobei der Restterm $R(\Delta A)$ die Bedingung $R(\Delta A)/\|\Delta A\| \to 0$ für $\|\Delta A\| \to 0$ erfüllt.

Die explizite Berechnung von $D\lambda_0(A)$ ist recht kompliziert; viel einfacher ist es, $D\lambda_0(A)$ nur in einer geeigneten Norm abzuschätzen. Wir wählen dazu die durch die $\|\cdot\|_2$-Norm induzierte Operatornorm, also die Verallgemeinerung der bekannten Matrixnorm.

Definition 1.1 Die (absolute) Kondition der Berechnung eines Eigenwerts $\lambda_0(A)$ für eine Matrix $A \in \mathbb{C}^{n\times n}$ ist definiert durch

$$\kappa_{\text{abs}} \;:=\; \|D\lambda_0(A)\|_2 \;:=\; \max_{\substack{\Delta A\in\mathbb{C}^{n\times n}\\ \|\Delta A\|_2=1}} |D\lambda_0(A)\Delta A| \;=\; \sup_{\Delta A\in\mathbb{C}^{n\times n}\setminus\{0\}} \frac{|D\lambda_0(A)\Delta A|}{\|\Delta A\|_2},$$

mit der Konvention $\kappa_{\text{abs}} := \infty$, falls $\lambda_0(A)$ nicht differenzierbar ist. □

Beachte, dass κ_{abs} von A und λ_0 abhängt. Insbesondere kann diese Ableitung für ein und dieselbe Matrix und verschiedene Eigenwerte unterschiedliche Werte annehmen. Das folgende Beispiel zeigt, dass $\lambda_0(A)$ tatsächlich nicht differenzierbar sein kann.

Beispiel 1.2 Betrachte die Matrizen

$$A = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \qquad \Delta A = \begin{pmatrix} 0 & 0 \\ \delta & 0 \end{pmatrix}$$

Dann hat A den doppelten Eigenwert $\lambda_1(A) = \lambda_2(A) = 0$ und $A + \Delta A$ die Eigenwerte $\lambda_{1/2}(A + \Delta A) = \pm\sqrt{\delta}$. Zudem gilt $\|\Delta A\|_2 = \delta$ und

$$\lambda_1(A + \Delta A) - \lambda_1(A) = \sqrt{\delta}.$$

Falls $\lambda_1(A)$ differenzierbar ist, folgt damit

$$D\lambda_1(A)\Delta A = \sqrt{\delta} - R(\Delta A),$$

also

$$|R(\Delta A)| \geq \sqrt{\delta} - |D\lambda_1(A)\Delta A| \geq \sqrt{\delta} - \|D\lambda_1(A)\|_2\|\Delta A\|_2 = \sqrt{\delta} - \|D\lambda_1(A)\|_2\delta \geq \sqrt{\delta}/2$$

für alle hinreichend kleinen $\delta > 0$, was der Beschränktheit $|R(\Delta A)| \leq c\|\Delta A\|^2$ widerspricht. Tatsächlich gilt hier

$$\frac{|\lambda_1(A) - \lambda_1(A + \Delta A)|}{\|\Delta A\|} = \frac{\sqrt{\delta}}{\delta} = \frac{1}{\sqrt{\delta}} \to \infty \text{ für } \delta \to 0,$$

weswegen es sinnvoll ist, in diesem Fall $\kappa_{\text{abs}} = \infty$ zu setzen. □

Um das Problem der Nicht-Differenzierbarkeit zu vermeiden, beschränken wir uns im folgenden Satz auf *einfache* Eigenwerte, d.h. Eigenwerte $\lambda_0 \in \mathbb{C}$, die einfache Nullstellen des charakteristischen Polynoms $\chi_A(\lambda) = \det(A - \lambda \mathrm{Id})$ sind. Für diese gilt der folgende Satz.

Satz 1.3 Die (absolute) Kondition der Berechnung eines einfachen Eigenwertes $\lambda_0(A)$ einer Matrix $A \in \mathbb{C}^{n\times n}$ gemessen in der 2-Norm ist gegeben durch

$$\kappa_{\mathrm{abs}} = \|D\lambda_0(A)\|_2 = \frac{\|x_0\|_2 \, \|y_0\|_2}{|\langle x_0, y_0\rangle|},$$

wobei x_0 ein Eigenvektor von A zum Eigenwert λ_0 und y_0 ein *adjungierter Eigenvektor* ist, d.h. ein Eigenvektor von $\overline{A}^T$ zum Eigenwert $\bar{\lambda}_0$, also $\overline{A}^T y_0 = \bar{\lambda}_0 y_0$. Hierbei bezeichnet $\langle x_0, y_0\rangle$ das euklidische Skalarprodukt im $\mathbb{C}^n$, also $\langle x_0, y_0\rangle = \bar{x}_0^T y_0$.

Beweis: Wir zeigen zunächst die Gleichung

$$|D\lambda_0(A)C| = \frac{|\langle Cx_0, y_0\rangle|}{|\langle x_0, y_0\rangle|} \tag{1.1}$$

für beliebige Matrizen $C \in \mathbb{C}^{n\times n}$ und die Eigenvektoren x_0 und y_0 aus dem Satz.

Sei $\chi_D(\lambda)$ das charakteristische Polynom einer Matrix $D \in \mathbb{C}^{n\times n}$. Wir betrachten die Matrix $A + tC$ für $t \in \mathbb{R}$ und definieren eine Abbildung $g : \mathbb{C} \times \mathbb{R} \to \mathbb{C}$ mittels $g(\lambda, t) = \chi_{A+tC}(\lambda)$. Da λ_0 eine einfache Nullstelle des Polynoms $\chi_A(\lambda)$ ist, folgt

$$\frac{\partial}{\partial\lambda} g(\lambda, t)|_{\lambda=\lambda_0,\, t=0} = \chi'_A(\lambda_0) \neq 0.$$

Nach dem impliziten Funktionensatz gibt es deshalb für hinreichend kleines $\varepsilon > 0$ eine differenzierbare Abbildung $\lambda : (-\varepsilon, \varepsilon) \to \mathbb{C}$ mit $\lambda(0) = \lambda_0$, $0 = g(\lambda(t), t) = \chi_{A+tC}(\lambda(t))$ und $0 \neq \frac{\partial}{\partial\lambda} g(\lambda, t)|_{\lambda=\lambda(t)}$ für $t \in (-\varepsilon, \varepsilon)$. Also ist $\lambda(t)$ für $t \in (-\varepsilon, \varepsilon)$ ein einfacher Eigenwert von $A + tC$. Da für die Eigenvektoren $x(t)$ zu den einfachen Eigenwerten $\lambda(t)$ eine explizite Formel existiert, hängen diese differenzierbar von t ab. Damit gilt

$$(A + tC)x(t) = \lambda(t)x(t) \text{ für } t \in (-\varepsilon, \varepsilon).$$

Leiten wir diese Gleichung an der Stelle $t = 0$ nach t ab, so folgt

$$Cx_0 + Ax'(0) = \lambda_0 x'(0) + \lambda'(0)x_0,$$

wobei $'$ für die Ableitung nach t steht. Bilden wir nun für alle Terme das Skalarprodukt $\langle \cdot, y_0\rangle$ und stellen die Skalare vor die Skalarprodukte (beachte, dass diese im ersten Argument stehen und deswegen konjugiert werden müssen), so folgt

$$\langle Cx_0, y_0\rangle + \langle Ax'(0), y_0\rangle = \bar{\lambda}_0 \langle x'(0), y_0\rangle + \overline{\lambda'(0)}\langle x_0, y_0\rangle.$$

Mit

$$\langle Ax'(0), y_0\rangle = \langle x'(0), \overline{A}^T y_0\rangle = \langle x'(0), \bar{\lambda}_0 y_0\rangle = \bar{\lambda}_0 \langle x'(0), y_0\rangle$$

und $\lambda'(0) = D\lambda(A)C$ folgt $\langle Cx_0, y_0\rangle = \overline{\lambda'(0)}\langle x_0, y_0\rangle$, also auch

$$|\langle Cx_0, y_0\rangle| = |\overline{\lambda'(0)}| \, |\langle x_0, y_0\rangle| = |\lambda'(0)| \, |\langle x_0, y_0\rangle|$$

und damit die behauptete Gleichung (1.1).

Zum Beweis des Satzes betrachten wir die Matrix $C = y_0\bar{x}_0^T$. Für diese gilt

$$\frac{\|Cz\|_2}{\|z\|_2} = \frac{\|y_0\bar{x}_0^T z\|_2}{\|z\|_2} \leq \frac{\|x_0\|_2\|y_0\|_2\|z\|_2}{\|z\|_2} = \|x_0\|_2\|y_0\|_2$$

für beliebige $z \in \mathbb{C}^n$ und

$$\frac{\|Cx_0\|_2}{\|x_0\|_2} = \frac{\|y_0\bar{x}_0^T x_0\|_2}{\|x_0\|_2} = \frac{\big\|y_0\|x_0\|_2^2\big\|_2}{\|x_0\|_2} = \|y_0\|_2\|x_0\|_2,$$

also ist $\|C\|_2 = \|x_0\|_2\|y_0\|_2$.

Für beliebige $C \in \mathbb{C}^{n\times n}$ liefert die Cauchy-Schwarz Ungleichung

$$|\langle Cx_0, y_0\rangle| \leq \|Cx_0\|_2\|y_0\|_2 \leq \|C\|_2\|x_0\|_2\|y_0\|_2. \tag{1.2}$$

Für $C = y_0\bar{x}_0^T$ gilt hierbei

$$|\langle Cx_0, y_0\rangle| = |\langle y_0, Cx_0\rangle| = |\bar{y}_0^T Cx_0| = |\bar{y}_0^T y_0\bar{x}_0^T x_0| = \|y_0\|_2^2\|x_0\|_2^2 = \|C\|_2\|y_0\|_2\|x_0\|_2, \tag{1.3}$$

also Gleichheit. Aus (1.1) und (1.2) folgt also für beliebige $C \in \mathbb{C}^{n\times n}$

$$\|D\lambda_0(A)\|_2 = \sup_{C\in\mathbb{C}^{n\times n}\setminus\{0\}} \frac{|D\lambda_0(A)C|}{\|C\|_2} = \sup_{C\in\mathbb{C}^{n\times n}\setminus\{0\}} \frac{|\langle Cx_0, y_0\rangle|}{\|C\|_2|\langle x_0, y_0\rangle|} \leq \frac{\|x_0\|_2\,\|y_0\|_2}{|\langle x_0, y_0\rangle|},$$

wobei für $C = y_0\bar{x}_0^T$ wegen (1.3) Gleichheit gilt. Dies zeigt die Behauptung. □

Bemerkung 1.4 (i) Der Ausdruck für κ_{abs} hat eine geometrische Interpretation: Für reelle Vektoren x_0, $y_0 \in \mathbb{R}^n$ gilt nämlich

$$\frac{\|x_0\|_2\|y_0\|_2}{\langle x_0, y_0\rangle} = \frac{1}{\cos\varphi(x_0, y_0)},$$

wobei $\varphi(x_0, y_0)$ den Winkel zwischen den Vektoren x_0 und y_0 bezeichnet.

(ii) Besonders gut konditioniert sind Eigenwertprobleme für *normale Matrizen*, also Matrizen $A \in \mathbb{C}^{n\times n}$ für die $\overline{A}^T A = A\overline{A}^T$ gilt. Für diese lässt sich zeigen, dass für jeder Eigenvektor x_0 von A gleich dem zugehörigen adjungierten Eigenvektor ist. Der Winkel $\varphi(x_0, x_0)$ ist offenbar gleich 0, der Kosinus also gleich 1, weswegen hier $\kappa_{\text{abs}} = 1$ gilt. □

Beispiel 1.5 Wir illustrieren die Kondition an der 2×2-Dreiecksmatrix

$$A = \begin{pmatrix} a & c \\ 0 & b \end{pmatrix}$$

mit $a, b, c \in \mathbb{R}$. Man rechnet leicht nach, dass die Matrix genau dann normal ist, wenn $c = 0$ gilt. Die Eigenwerte lauten a und b und die zugehörigen Eigenvektoren von A bzw. A^T sind

$$x_0 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} \frac{c}{b-a} \\ 1 \end{pmatrix}, \qquad y_0 = \begin{pmatrix} 1 \\ \frac{c}{a-b} \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

Für den Eigenwert a ergibt sich also

$$\frac{\|x_0\|_2\|y_0\|_2}{\langle x_0, y_0\rangle} = \sqrt{1 + \frac{c^2}{(b-a)^2}}.$$

Die Kondition wird bei festem a und b also um so schlechter, je größer der Betrag des Nichtdiagonaleintrags c wird. □

Bevor wir zu numerischen Verfahren zur Berechnung von Eigenwerten kommen, wollen wir kurz die vielleicht naheliegendste Methode untersuchen, nämlich die Berechnung der λ über die Nullstellen des charakteristischen Polynoms. Diese Methode ist numerisch äußerst instabil (unabhängig von der Kondition der Eigenwertberechnung selbst) und bereits kleinste Rundungsfehler können sehr große Fehler im Ergebnis nach sich ziehen. Als Beispiel betrachte das Polynom

$$P(\lambda) = (\lambda - 1)(\lambda - 2)\cdots(\lambda - 20)$$

mit den Nullstellen $\lambda_i = i$ für $i = 1, \ldots, 20$.[1] Wenn dieses Polynom als charakteristisches Polynom einer Matrix berechnet wird (z.B. ist es gerade das charakteristische Polynom $\chi_A(\lambda)$ der Matrix $A = \mathrm{diag}(1, 2, \ldots, 20)$), liegt es üblicherweise nicht in der obigen "Nullstellen"-Darstellung sondern in anderer Form vor, z.B. ausmultipliziert. Wenn man das obige $P(\lambda)$ ausmultipliziert, ergeben sich Koeffizienten zwischen 1 (für λ^{20}) und $20! \approx 10^{20}$ (der konstante Term). Stört man nun den Koeffizienten vor λ^{19} (der den Wert 210 hat) mit dem sehr kleinen Wert $\varepsilon = 2^{-23} \approx 10^{-7}$, so erhält man die folgenden Nullstellen für das gestörte Polynom $\widetilde{P}(\lambda) = P(\lambda) - \varepsilon\lambda^{19}$:

$$\begin{aligned}
\lambda_1 &= 1.000\,000\,000 & \lambda_{10/11} &= 10.095\,266\,145 \pm 0.643\,500\,904\,i\\
\lambda_2 &= 2.000\,000\,000 & \lambda_{12/13} &= 11.793\,633\,881 \pm 1.652\,329\,728\,i\\
\lambda_3 &= 3.000\,000\,000 & \lambda_{14/15} &= 13.992\,358\,137 \pm 2.518\,830\,070\,i\\
\lambda_4 &= 4.000\,000\,000 & \lambda_{16/17} &= 16.730\,737\,466 \pm 2.812\,624\,894\,i\\
\lambda_5 &= 4.999\,999\,928 & \lambda_{18/19} &= 19.502\,439\,400 \pm 1.940\,330\,347\,i\\
\lambda_6 &= 6.000\,006\,944 & \lambda_{20} &= 20.846\,908\,101\\
\lambda_7 &= 6.999\,697\,234 & & \\
\lambda_8 &= 8.007\,267\,603 & & \\
\lambda_9 &= 8.917\,250\,249 & &
\end{aligned}$$

Die winzige Störung bewirkt also beachtliche Fehler, insbesondere sind 10 Nullstellen durch die Störung komplex geworden.

1.2 Vektoriteration

Die einfachste Möglichkeit der Berechnung von Eigenwerten ist die Vektoriteration, die sich entweder als *direkte Iteration* (auch bekannt als *von Mises-Iteration* oder *power iteration*) oder als *inverse Iteration* (auch *inverse power iteration*) durchführen lässt.

[1] Das Beispiel stammt von dem englischen Numeriker James H. Wilkinson (1919–1986), der die Entdeckung dieses Polynoms angeblich als "the most traumatic experience in my career as a numerical analyst" bezeichnet hat.

Gegeben sei eine reelle Matrix $A \in \mathbb{R}^{n\times n}$. Die Idee der direkten Iteration beruht darauf, für einen beliebigen Startwert $x^{(0)} \in \mathbb{R}^n$ die Iteration

$$x^{(i+1)} = Ax^{(i)} \tag{1.4}$$

durchzuführen. Dass dieses einfache Verfahren tatsächlich unter gewissen Bedingungen einen Eigenwert liefert, zeigt der folgende Satz. Wir erinnern hierbei daran, dass symmetrische reelle Matrizen nur reelle Eigenwerte besitzen.

Satz 1.6 Sei $A \in \mathbb{R}^{n\times n}$ eine symmetrische Matrix und $\lambda_1 = \lambda_1(A)$ ein einfacher Eigenwert, für den die Ungleichung

$$|\lambda_1| > |\lambda_2| \geq |\lambda_3| \geq \ldots \geq |\lambda_n|$$

für alle anderen Eigenwerte $\lambda_j = \lambda_j(A)$ gilt. Sei weiterhin $x^{(0)} \in \mathbb{R}^n$ ein Vektor, für den $\langle x^{(0)}, v_1\rangle \neq 0$ für den zu $\lambda_1(A)$ gehörigen (normierten) Eigenvektor v_1 gilt. Dann konvergiert die Folge $y^{(i)} := x^{(i)}/\|x^{(i)}\|$ für $x^{(i)}$ aus (1.4) gegen $\pm v_1$, also gegen einen normierten Eigenvektor zum Eigenwert λ_1. Insbesondere konvergiert damit der sogenannte *Rayleigh'sche Quotient*

$$\lambda^{(i)} := \frac{\langle Ax^{(i)}, x^{(i)}\rangle}{\langle x^{(i)}, x^{(i)}\rangle} = \langle Ay^{(i)}, y^{(i)}\rangle$$

gegen den Eigenwert λ_1.

Beweis: Wegen der Symmetrie von A existiert eine Orthonormalbasis von Eigenvektoren $v_1, \ldots, v_n$ von A. Damit gilt

$$x^{(0)} = \sum_{j=1}^{n} \alpha_j v_j \quad \text{mit } \alpha_i = \langle x^{(0)}, v_i\rangle,$$

insbesondere also $\alpha_1 \neq 0$. Daraus folgt

$$x^{(i)} = A^i x^{(0)} = \sum_{j=1}^{n} \alpha_j \lambda_j^i v_j = \alpha_1 \lambda_1^i \underbrace{\left(v_1 + \sum_{j=2}^{n} \frac{\alpha_j}{\alpha_1} \left(\frac{\lambda_j}{\lambda_1}\right)^i v_j \right)}_{=:z^{(i)}}.$$

Da $|\lambda_i| < |\lambda_1|$ ist für $i = 2, \ldots, n$, gilt $\lim_{i\to\infty} z^{(i)} = v_1$ und damit

$$y^{(i)} = \frac{x^{(i)}}{\|x^{(i)}\|} = \pm \frac{z^{(i)}}{\|z^{(i)}\|} \to \pm v_1.$$

Die Konvergenz $\lambda^{(i)} \to \lambda_1$ folgt, indem wir $y^{(i)} = v_1 + r^{(i)}$ mit $r^{(i)} \to 0$ schreiben. Dann gilt

$$\begin{aligned} \langle Ay^{(i)}, y^{(i)}\rangle &= \langle A(v_1 + r^{(i)}), v_1 + r^{(i)}\rangle \\ &= \langle Av_1, v_1\rangle + \underbrace{\langle Ar^{(i)}, v_1\rangle + \langle Av_1, r^{(i)}\rangle + \langle Ar^{(i)}, r^{(i)}\rangle}_{\to 0} \\ \to \langle Av_1, v_1\rangle &= \langle \lambda_1 v_1, v_1\rangle = \lambda_1 \|v_1\| = \lambda_1. \end{aligned}$$

□

Beachte, dass die Symmetrie der Matrix A hier nur eine hinreichende aber keine notwendige Bedingung für die Konvergenz des Verfahrens ist. So ist z.B. in [2] bewiesen, dass das Verfahren auch für die (nicht symmetrische) Matrix aus dem Seitenranking funktioniert.

Dieses einfache Verfahren hat mehrere Nachteile: Erstens erhalten wir nur den betragsmäßig größten Eigenwert $|\lambda_1|$ und den zugehörigen Eigenvektor, zweitens hängt die Konvergenzgeschwindigkeit davon ab, wie schnell die Terme $|\lambda_j/\lambda_1|^i$, also insbesondere $|\lambda_2/\lambda_1|^i$ gegen Null konvergieren. Falls also $|\lambda_1| \approx |\lambda_2|$ und damit $|\lambda_2/\lambda_1| \approx 1$ gilt, ist nur sehr langsame Konvergenz zu erwarten.

Die *inverse Vektoriteration* vermeidet diese Nachteile. Sei A wiederum eine reelle symmetrische Matrix. Wir setzen voraus, dass wir einen Schätzwert $\tilde{\lambda} \in \mathbb{R}$ für einen Eigenwert $\lambda_j = \lambda_j(A)$ kennen, für den die Ungleichung

$$|\tilde{\lambda} - \lambda_j| < |\tilde{\lambda} - \lambda_k| \text{ für alle } k = 1, \ldots, n,\ k \neq j$$

mit $\lambda_k = \lambda_k(A)$ gilt. Dann betrachten wir die Matrix $\widetilde{A} = (A - \tilde{\lambda}\mathrm{Id})^{-1}$. Diese besitzt die Eigenwerte $1/(\lambda_k - \tilde{\lambda})$ für $k = 1, \ldots, n$, also ist $1/(\lambda_j - \tilde{\lambda})$ der betragsmäßig größte Eigenwert.

Die inverse Vektoriteration ist nun gegeben durch

$$x^{(i+1)} = (A - \tilde{\lambda}\mathrm{Id})^{-1} x^{(i)}. \tag{1.5}$$

Aus Satz 1.6 (angewendet auf $(A - \tilde{\lambda}\mathrm{Id})^{-1}$ an Stelle von A) folgt, dass diese Iteration gegen einen normierten Eigenvektor v_j von $(A - \tilde{\lambda}\mathrm{Id})^{-1}$ zum Eigenwert $1/(\lambda_j - \tilde{\lambda})$ konvergiert. Wegen

$$\begin{aligned} & (A - \tilde{\lambda}\mathrm{Id})^{-1} v_j &&= 1/(\lambda_j - \tilde{\lambda}) v_j \\ \Leftrightarrow \quad & (\lambda_j - \tilde{\lambda}) v_j &&= (A - \tilde{\lambda}\mathrm{Id}) v_j \\ \Leftrightarrow \quad & \lambda_j v_j &&= A v_j \end{aligned}$$

ist dies gerade ein Eigenvektor von A zum Eigenwert λ_j. Die Konvergenzgeschwindigkeit ist bestimmt durch den Term

$$\max_{\substack{k=1,\ldots,n \\ k \neq j}} \frac{|\lambda_j - \tilde{\lambda}|}{|\lambda_k - \tilde{\lambda}|}.$$

Je kleiner dieser Term ist, d.h. je besser der Schätzwert ist, desto schneller wird die Konvergenz.

Die tatsächliche Implementierung der Iteration (1.5) ist hier etwas komplizierter als bei der direkten Iteration (1.4). Während dort in jedem Schritt eine Matrix-Vektor Multiplikation mit Aufwand $O(n^2)$ durchgeführt werden muss, geht hier die Inverse $(A - \tilde{\lambda}\mathrm{Id})^{-1}$ ein. In der Praxis berechnet man nicht die Inverse (weil dies numerisch sehr aufwändig ist), sondern löst das lineare Gleichungssystem

$$(A - \tilde{\lambda}\mathrm{Id}) x^{(i+1)} = x^{(i)}, \tag{1.6}$$

wobei bei der Verwendung eines direkten Verfahrens die Matrix $(A - \tilde{\lambda}\mathrm{Id})$ nur einmal am Anfang der Iteration faktorisiert werden muss und dann in jedem Iterationsschritt einmal Vorwärts- bzw. Rückwärtseinsetzen durchgeführt werden muss. Der Aufwand $O(n^3)$

der Zerlegung kommt also hier zum Aufwand des Verfahrens dazu, die einzelnen Iterationsschritte haben bei diesem Vorgehen allerdings keinen höheren Rechenaufwand als bei der direkten Iteration, da das Vorwärts- bzw. Rückwärtseinsetzen wie die Matrix-Vektor Multiplikation den Aufwand $O(n^2)$ besitzen.

Für sehr gute Schätzwerte $\tilde{\lambda} \approx \lambda_j$ wird die Matrix $(A-\tilde{\lambda}\mathrm{Id})$ "fast" singulär (für $\tilde{\lambda} = \lambda_j$ wäre sie singulär), weswegen die Kondition von $(A-\tilde{\lambda}\mathrm{Id})$ sehr groß wird. Wegen der besonderen Struktur des Algorithmus führt dies hier aber nicht auf numerische Probleme, da zwar die Lösung $x^{(i+1)}$ des Gleichungssystems (1.6) mit großen Fehlern behaftet sein kann, sich diese Fehler aber in der hier eigentlich wichtigen *normierten Lösung* $x^{(i+1)}/\|x^{(i+1)}\|$ nicht auswirken. Wir wollen dies an einem Beispiel illustrieren.

Beispiel 1.7 Betrachte

$$A = \begin{pmatrix} -1 & 3 \\ -2 & 4 \end{pmatrix}$$

mit den Eigenwerten $\lambda_1(A) = 2$ und $\lambda_2(A) = 1$. Wir wählen $\tilde{\lambda} = 1 - \varepsilon$ für ein sehr kleines $\varepsilon > 0$. Dann ist die Matrix

$$(A - \tilde{\lambda}\mathrm{Id}) = \begin{pmatrix} -2+\varepsilon & 3 \\ -2 & 3+\varepsilon \end{pmatrix} \quad \text{mit} \quad (A - \tilde{\lambda}\mathrm{Id})^{-1} = \frac{1}{\varepsilon(\varepsilon+1)} \underbrace{\begin{pmatrix} 3+\varepsilon & -3 \\ 2 & -2+\varepsilon \end{pmatrix}}_{=:B}$$

fast singulär und man sieht leicht, dass für die Kondition z.B. in der Zeilensummennorm die Abschätzung $\mathrm{cond}_\infty(A-\tilde{\lambda}\mathrm{Id}) > 1/\varepsilon$ gilt. Die Inverse besitzt aber eine typische spezielle Struktur: die großen Einträge, die der Grund für die große Kondition sind, entstehen lediglich durch einen skalaren Vorfaktor. Daher ist die Berechnung von $y^{(i+1)} = x^{(i+1)}/\|x^{(i+1)}\|$ mittels (1.6) nicht stark anfällig für Rundungsfehler, denn es gilt

$$y^{(i+1)} = \frac{(A - \tilde{\lambda}\mathrm{Id})^{-1}x^{(i)}}{\|(A - \tilde{\lambda}\mathrm{Id})^{-1}x^{(i)}\|} = \frac{Bx^{(i)}}{\|Bx^{(i)}\|_2},$$

d.h. der ungünstige große Faktor $1/(\varepsilon(\varepsilon+1))$ kürzt sich heraus. Ein Zahlenbeispiel illustriert dies noch einmal: Betrachten wir beispielsweise $x^{(i)} = (1,0)^T$ und $\varepsilon = 1/1000$, so erhält man

$$x^{(i+1)} = (A - \tilde{\lambda}\mathrm{Id})^{-1}x^{(i)} = \begin{pmatrix} 2998.001998 \\ 1998.001998 \end{pmatrix} \text{ und } y^{(i+1)} = \frac{x^{(i+1)}}{\|x^{(i+1)}\|_2} = \begin{pmatrix} 0.832135603 \\ 0.554572211 \end{pmatrix}$$

während man für die gestörte Lösung mit $\tilde{x}^{(i)} = (1.1, -0.1)^T$

$$\tilde{x}^{(i+1)} = (A - \tilde{\lambda}\mathrm{Id})^{-1}\tilde{x}^{(i)} = \begin{pmatrix} 3597.502498 \\ 2397.502498 \end{pmatrix} \text{ und } \tilde{y}^{(i+1)} = \frac{\tilde{x}^{(i+1)}}{\|\tilde{x}^{(i+1)}\|_2} = \begin{pmatrix} 0.832117832 \\ 0.554598875 \end{pmatrix}$$

erhält. Die Störung in der ersten Nachkommastelle der rechten Seite bewirkt in $\tilde{x}^{(i+1)}$ zwar einen Fehler von etwa 600 und verstärkt sich daher sichtlich. In dem für den Algorithmus eigentlich wichtigen Vektor $\tilde{y}^{(i+1)}$ ist der Effekt der Störung hingegen erst in der fünften Nachkommastelle der Lösung sichtbar. □

1.3 Der QR-Algorithmus

Zwar kann man mit der inversen Vektoriteration im Prinzip alle Eigenwerte berechnen, benötigt dafür aber geeignete Schätzwerte.

Wir wollen daher nun einen Algorithmus betrachten, der in einer einzigen Rechnung alle Eigenwerte einer Matrix approximiert. Wie bereits bei linearen Gleichungssystemen wird auch hier eine Faktorisierung mittels orthogonaler Matrizen eine wichtige Rolle spielen, weswegen der Algorithmus ebenfalls QR-Algorithmus genannt wird.

Wir werden diesen Algorithmus für reelle symmetrische Matrizen herleiten und am Ende kurz auf die Verallgemeinerung auf allgemeine Matrizen eingehen.

Die Grundidee für reelle symmetrische Matrizen besteht darin, dass für solche Matrizen eine Orthonormalbasis aus Eigenvektoren v_1, $v_2, \ldots,$ v_n besteht, so dass für die orthogonale Matrix $Q = (v_1, \ldots, v_n) \in \mathbb{R}^{n \times n}$ die Gleichung

$$Q^T A Q = \Lambda = \operatorname{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)$$

gilt, wobei die λ_i gerade die Eigenwerte von A sind. Wenn wir also eine orthogonale Matrix Q finden können, mit der A auf Diagonalgestalt konjugiert werden kann, so können wir die Eigenwerte direkt aus der resultierenden Diagonalmatrix Λ und die zugehörigen Eigenvektoren aus Q ablesen. Leider ist eine direkte Transformation auf Diagonalgestalt nicht möglich; dies geht nur mit einem iterativen Algorithmus. Zwar kann man z.B. orthogonale Householder-Matrizen dazu verwenden, Matrizen auf obere Dreiecksgestalt zu bringen, allerdings lässt sich dies nicht zur Konjugation auf Diagonalgestalt verwenden, da die positiven Effekte durch die Multiplikation von links mit H bei der Multiplikation von rechts mit H^T wieder zunichte gemacht werden.

Allerdings ist es möglich, eine Matrix mit Hilfe von Householder-Matrizen auf Tridiagonalform zu transformieren. Dazu wird bei jeder Spiegelung eine Null weniger erzeugt, was dazu führt, dass die Multiplikation von rechts mit H^T die bereits erzeugten Nullen nicht wieder entfernt. Diese Transformation auf Tridiagonalform ist sinnvoll, weil dadurch der Rechenaufwand der nachfolgenden iterativen Transformation auf Diagonalform deutlich verringert wird. Das folgende Lemma zeigt, wie die Householder-Matrizen für diese Tridiagonalformtransformation konstruiert werden müssen.

Lemma 1.8 Sei A eine reelle symmetrische Matrix der Form

$$A = \begin{pmatrix}
a_{1\,1} & a_{1\,2} & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\
a_{2\,1} & a_{2\,2} & a_{2\,3} & \ddots & & & & \vdots \\
0 & \ddots & \ddots & \ddots & \ddots & & & \vdots \\
\vdots & \ddots & \ddots & a_{k-1\,k-1} & a_{k-1\,k} & 0 & \cdots & 0 \\
\vdots & & \ddots & a_{k\,k-1} & a_{k\,k} & \cdots & \cdots & a_{k\,n} \\
\vdots & & & 0 & \vdots & & & \vdots \\
\vdots & & & \vdots & \vdots & & & \vdots \\
0 & \cdots & \cdots & 0 & a_{n\,k} & \cdots & \cdots & a_{n\,n}
\end{pmatrix}$$

und bezeichne $w = (w_1, \ldots, w_n)^T = (0, \cdots, a_{k-1\,k}, \ldots, a_{n\,k})^T \in \mathbb{R}^n$ die k-te Spalte dieser Matrix. Sei $H = H(v) = \mathrm{Id} - \frac{2vv^T}{v^T v}$ die Householder-Matrix mit

$$\begin{aligned} c &= \mathrm{sgn}(w_{k+1})\sqrt{w_{k+1}^2 + w_{k+2}^2 + \cdots + w_n^2} \in \mathbb{R} \\ v &= (0, \ldots, 0, c + w_{k+1}, w_{k+2}, \ldots, w_n)^T, \end{aligned}$$

$\mathrm{sgn}(a) = 1$, falls $a \geq 0$, $\mathrm{sgn}(a) = -1$, falls $a < 0$ und $H = \mathrm{Id}$ falls $v = 0$. Dann gilt

$$HAH^T = \begin{pmatrix} a_{1\,1} & a_{1\,2} & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ a_{2\,1} & a_{2\,2} & a_{2\,3} & \ddots & & & & \vdots \\ 0 & \ddots & \ddots & \ddots & \ddots & & & \vdots \\ \vdots & \ddots & \ddots & a_{k\,k} & \tilde{a}_{k\,k+1} & 0 & \cdots & 0 \\ \vdots & & \ddots & \tilde{a}_{k+1\,k} & \tilde{a}_{k+1\,k+1} & \cdots & \cdots & \tilde{a}_{k+1\,n} \\ \vdots & & & 0 & \vdots & & & \vdots \\ \vdots & & & \vdots & \vdots & & & \vdots \\ 0 & \cdots & \cdots & 0 & \tilde{a}_{n\,k+1} & \cdots & \cdots & \tilde{a}_{n\,n} \end{pmatrix}$$

Beweis: Wir betrachten zunächst das Produkt HA. Es bezeichne $a_{\cdot\,j}$ die j-te Spalte von A. Aus Lemma 2.19 der Einführung in die Numerik wissen wir, dass

$$Ha_{\cdot\,j} = a_{\cdot\,j}, \quad \text{für } j = 1, \ldots, k-1,$$

also für die ersten $k-1$ Spalten der Matrix A. Ebenfalls nach diesem Lemma gilt für die k-te Spalte

$$Ha_{\cdot\,k} = (0, \ldots, 0, a_{k-1\,k}, a_{k\,k}, -c, 0, \ldots, 0)^T.$$

Für beliebige Vektoren $x \in \mathbb{R}^n$ und $i = 1, \ldots, k$ folgt aus der Form von v sofort

$$[Hx]_i = x_i - \underbrace{v_i}_{=0} \frac{2v^T x}{v^T v} = x_i,$$

also gilt für die Spalten $a_{\cdot\,j}$ für $j = k+1, \ldots, n$

$$Ha_{\cdot\,j} = (0, \ldots, 0, a_{k\,j}, *, \ldots, *)^T.$$

Insgesamt gilt also

$$HA = \begin{pmatrix} a_{1\,1} & a_{1\,2} & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ a_{2\,1} & a_{2\,2} & a_{2\,3} & \ddots & & & & \vdots \\ 0 & \ddots & \ddots & \ddots & \ddots & & & \vdots \\ \vdots & \ddots & \ddots & a_{k-1\,k-1} & a_{k-1\,k} & 0 & \cdots & 0 \\ \vdots & & \ddots & a_{k\,k-1} & a_{k\,k} & \cdots & \cdots & a_{k\,n} \\ \vdots & & & 0 & -c & * & \cdots & * \\ \vdots & & & \vdots & 0 & \vdots & & \vdots \\ \vdots & & & \vdots & \vdots & \vdots & & \vdots \\ 0 & \cdots & \cdots & 0 & 0 & * & \cdots & * \end{pmatrix}$$

Da A symmetrisch ist, gilt $(HAH^T)^T = (H^T)^T A^T H^T = HAH^T$, d.h., HAH^T ist ebenfalls symmetrisch. Mit dem gleichen Argument wie oben folgt für Zeilenvektoren y die Gleichung $[yH^T]_i = y_i$ für $i = 1, \ldots, k$, weswegen die ersten k Spalten von HA und HAH^T übereinstimmen. Die behauptete Form ergibt sich damit aus der Symmetrie von HAH^T. □

Die folgende Aussage ist nun ein einfaches Korollar.

Korollar 1.9 Sei A eine reelle symmetrische Matrix. Dann existiert eine orthogonale Matrix P, so dass $P^T AP$ symmetrisch und in Tridiagonalgestalt ist.

Beweis: Durch Anwendung von Lemma 1.8 für $k = 1, \ldots, n-2$ erhält man Matrizen $H^{(1)}, \ldots, H^{(n-2)}$, für die $P^T = H^{(n-2)} \cdots H^{(1)}$ die gewünschte Eigenschaft besitzt. □

Der QR-Zerlegungsalgorithmus 2.20 aus der Einführung in die Numerik lässt sich leicht zur Berechnung dieser Transformationsmatrix P abändern, indem man den Index j in Schritt (1) nur von 2 bis $n-1$ laufen lässt und alle *Spaltenindizes* in den folgenden Berechnungen um 1 erniedrigt, also $a_{j\,j}$ und $a_{i\,j}$ durch $a_{j\,j-1}$ bzw. $a_{i\,j-1}$ ersetzt. Die Berechnung von $H^{(j)}y^{(j)}$ macht hier natürlich keinen Sinn, die Berechung von $H^{(j)}A^{(j)}$ kann falls gewünscht auf die Berechnung von $H^{(j)}A^{(j)}{H^{(j)}}^T$ erweitert werden.

Die Iteration zur iterativen Transformation von A in Diagonalform ist nun durch den folgenden Algorithmus gegeben.

Algorithmus 1.10 (QR-Algorithmus zur Eigenwertberechnung)
Eingabe: symmetrische, reelle Matrix $A \in \mathbb{R}^{n \times n}$

(0) Setze $A^{(1)} := P^T AP$ mit P aus Korollar 1.9, $i := 1$

(1) Berechne eine QR-Zerlegung $A^{(i)} = Q^{(i)} R^{(i)}$

(2) Setze $A^{(i+1)} := R^{(i)} Q^{(i)}$, $i := i + 1$ und fahre fort bei (1)

Ausgabe: Approximation aller Eigenwerte von A als Diagonaleinträge von $A^{(i)}$ und der Eigenvektoren als Spalten der Matrix $PQ^{(1)} \cdots Q^{(i)}$, für Details siehe Bemerkung 1.14 (iii).

□

Natürlich müssen wir in der praktischen Implementierung noch ein geeignetes Abbruchkriterium einführen, das sich aus der folgenden Konvergenzanalyse ergeben wird.

Wir werden den Algorithmus 1.10 nun analysieren. Wir beginnen mit einigen Struktureigenschaften der Matrizen $A^{(i)}$.

Lemma 1.11 Für alle $i \geq 1$ gilt:

(i) $A^{(i)}$ ist konjugiert zu A. Genauer gilt $A^{(i+1)} = Q_i^T P^T A P Q_i$ mit $Q_i = Q^{(1)} \cdots Q^{(i)}$ und P aus Korollar 1.9.

(ii) $A^{(i)}$ ist symmetrisch

(iii) $A^{(i)}$ ist eine Tridiagonalmatrix

Beweis: (i) Es gilt

$$A^{(i+1)} = R^{(i)} Q^{(i)} = \underbrace{(Q^{(i)})^T Q^{(i)}}_{=\mathrm{Id}} R^{(i)} Q^{(i)} = (Q^{(i)})^T A^{(i)} Q^{(i)}.$$

Durch induktives Fortfahren erhält man so

$$A^{(i+1)} = (Q^{(i)})^T \cdots (Q^{(1)})^T A^{(1)} Q^{(1)} \cdots Q^{(i)},$$

und damit $A^{(i+1)} = (Q^{(i)})^T \cdots (Q^{(1)})^T P^T A P Q^{(1)} \cdots Q^{(i)}$, also die Behauptung.

(ii) Nach (i) gilt

$$(A^{(i)})^T = (Q_i^T P^T A P Q_i)^T = Q_i^T P^T A^T P Q_i = Q_i^T P^T A P Q_i = A^{(i)},$$

da A symmetrisch ist.

(iii) Sei $A^{(i)}$ tridiagonal. Durch explizites Ausrechnen der QR-Zerlegung mit Hilfe von Lemma 2.19 aus der Einführung in die Numerik sieht man, dass $(Q^{(i)})^T$ tridiagonal ist, also auch $Q^{(i)}$. Also ist $A^{(i+1)} = R^{(i)} Q^{(i)}$ von der Form

$$A^{(i+1)} = \begin{pmatrix} * & \cdots & \cdots & \cdots & * \\ * & \ddots & & & \vdots \\ 0 & \ddots & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & * & * \end{pmatrix}$$

Da $A^{(i+1)}$ nach (ii) aber symmetrisch ist, müssen die Elemente oberhalb der oberen Nebendiagonalen aus Symmetriegründen gleich Null sein, also ist $A^{(i+1)}$ eine Tridiagonalmatrix. Da $A^{(1)} = A$ eine Tridiagonalmatrix ist, folgt diese Eigenschaft also für alle $A^{(i)}$. □

Ein weiteres vorbereitendes Lemma zeigt eine Eigenschaft der QR-Zerlegung.

Lemma 1.12 Sei $A \in \mathbb{R}^{n\times n}$ eine invertierbare Matrix und seien $A = Q_1R_1$ und $A = Q_2R_2$ QR-Zerlegungen von A. Dann gilt $Q_1 = Q_2D$ und $R_1 = DR_2$ mit $D = \operatorname{diag}(\mu_1, \ldots, \mu_n)$ und $\mu_k = \pm 1$. Insbesondere existiert eine eindeutige QR-Zerlegung bei der alle Diagonalelemente von R positiv sind.

Beweis: Aus $Q_1R_1 = Q_2R_2$ folgt $Q_2^TQ_1 = R_2R_1^{-1}$ (beachte, dass R_1 und R_2 invertierbar sind, da A invertierbar ist). Die Inverse einer oberen Dreiecksmatrix ist wieder eine obere Dreiecksmatrix, ebenso das Produkt zweier oberer Dreiecksmatrizen. Also ist $R = R_2R_1^{-1}$ eine obere Dreiecksmatrix, die wegen $R = Q_2^TQ_1$ orthogonal ist. Damit ist insbesondere $R^{-1} = R^T$, weswegen R^T zugleich eine obere Dreiecksmatrix (als Inverse einer oberen Dreiecksmatrix) und eine untere Dreiecksmatrix (als Transponierte einer oberen Dreiecksmatrix) ist. Folglich muss R eine Diagonalmatrix sein. Aus der Orthogonalitätsbedingung $\langle Rx, Ry\rangle = \langle x, y\rangle$ leitet man nun für $x = y = e_i$ her, dass die Einträge von R nur die Werte ± 1 annehmen können. Damit folgt wegen $R^{-1} = R$, dass $D = R$ die gesuchte Matrix ist. □

Der folgende Satz zeigt die Konvergenzeigenschaften des Algorithmus. Um den Beweis zu vereinfachen, beschränken wir uns auf paarweise verschiedene Eigenwerte.

Satz 1.13 Sei $A \in \mathbb{R}^{n\times n}$ eine symmetrische Matrix mit den Eigenwerten $\lambda_1, \ldots, \lambda_n$, für die die Ungleichung

$$|\lambda_1| > |\lambda_2| > \ldots > |\lambda_n| > 0$$

gelte. Seien $A^{(i)}$, $Q^{(i)}$ und $R^{(i)}$ aus Algorithmus 1.10. Dann gilt

$$\lim_{i\to\infty} A^{(i)} = \Lambda = \operatorname{diag}(\lambda_1, \ldots, \lambda_n).$$

Beweis: O.B.d.A. nehmen wir an, dass A bereits in Tridiagonalgestalt ist, also $A^{(1)} = A$ gilt. Wir zeigen zunächst per Induktion die Gleichung

$$A^i = Q_iR_i \tag{1.7}$$

für die i-te Potenz von A, mit $Q_i = Q^{(1)}\cdots Q^{(i)}$ und $R_i = R^{(i)}\cdots R^{(1)}$. Für $i = 1$ ist (1.7) klar, denn

$$A^1 = A = A^{(1)} = Q^{(1)}R^{(1)} = Q_1R_1.$$

Für den Schritt $i \to i+1$ verwenden wir $A^{(i+1)} = Q_i^TAQ_i$ aus Lemma 1.11(i), aus dem wegen $Q^{(i+1)}R^{(i+1)} = A^{(i+1)}$ die Gleichung $AQ_i = Q_iQ^{(i+1)}R^{(i+1)}$ folgt. Damit und mit der Induktionsannahme $A^i = Q_iR_i$ folgt

$$A^{i+1} = AA^i = AQ_iR_i = Q_iQ^{(i+1)}R^{(i+1)}R_i = Q_{i+1}R_{i+1},$$

also (1.7). Beachte, dass Q_i orthogonal und R_i eine obere Dreiecksmatrix ist, weswegen Q_iR_i eine QR-Zerlegung von A^i darstellt.

Sei nun Q die orthogonale Matrix aus den Eigenvektoren von A, für die $Q^TAQ = \Lambda$ gilt. Dann gilt

$$A^i = Q\Lambda^iQ^T \text{ und } \Lambda^i = \operatorname{diag}(\lambda_1^i, \ldots, \lambda_n^i).$$

Zur Vereinfachung der Beweisführung nehmen wir an, dass Q^T eine LR-Zerlegung $Q^T = LR$ besitzt (falls nicht, findet man immer eine Zerlegung von PQ^T für eine geeignete Permutationsmatrix P, die in allen folgenden Gleichungen berücksichtigt werden muss, was zwar prinzipiell geht, die Beweisführung aber unübersichtlicher macht). Hierbei können wir L so wählen, dass auf der Diagonalen nur 1-Elemente stehen (dies ist die Form, die die Gauß-Elimination liefert). Dann gilt

$$A^i = Q\Lambda^i LR = Q(\Lambda^i L\Lambda^{-i})(\Lambda^i R). \tag{1.8}$$

Die Einträge von $\Lambda^i L\Lambda^{-i}$ lassen sich mit $L = (l_{ij})$ explizit als

$$(\Lambda^i L\Lambda^{-i})_{kj} = l_{kj}\left(\frac{\lambda_k}{\lambda_j}\right)^i$$

berechnen. Wegen der unteren Dreiecksstruktur von L und der Annahme an die Eigenwerte gilt $|\lambda_k/\lambda_j| < 1$ für alle $k \neq j$ mit $l_{kj} \neq 0$. Also folgt $(\lambda_k/\lambda_j)^i \to 0$ und damit

$$\Lambda^i L\Lambda^{-i} = \mathrm{Id} + E_i \text{ mit } E_i \to 0 \text{ für } i \to \infty.$$

Aus (1.8) erhalten wir so

$$A^i = Q(\mathrm{Id} + E_i)(\Lambda^i R).$$

Sei nun $\widetilde{Q}_i\widetilde{R}_i$ die eindeutige QR-Zerlegung von $\mathrm{Id} + E_i$ mit positiven Vorzeichen der Diagonalelemente von $\widetilde{R}_i$. Wegen der Eindeutigkeit dieser Zerlegung konvergieren diese Matrizenfolgen gegen die QR-Zerlegung von $\mathrm{Id} = \overline{Q}\,\overline{R}$ mit positiven Diagonalelementen von $\overline{R}$, welche gerade durch $\mathrm{Id} = \mathrm{Id} \cdot \mathrm{Id}$ gegeben ist[2], also gilt

$$\widetilde{Q}_i \to \mathrm{Id} \text{ und } \widetilde{R}_i \to \mathrm{Id} \text{ für } i \to \infty$$

und damit auch

$$\widetilde{Q}_i^T \to \mathrm{Id} \text{ und } \widetilde{R}_i^{-1} \to \mathrm{Id} \text{ für } i \to \infty.$$

Die Matrizen $Q\widetilde{Q}_i$ und $\widetilde{R}_i\Lambda^i R$ bilden nun wegen (1.8) und

$$A^i = Q(\Lambda^i L\Lambda^{-i})(\Lambda^i R) = Q\widetilde{Q}_i\widetilde{R}_i\Lambda^i R$$

eine QR-Zerlegung von A^i. Da $A^i = Q_iR_i$ eine weitere QR-Zerlegung ist, folgt mit Lemma 1.12 die Existenz von $D_i = \mathrm{diag}(\pm 1, \ldots, \pm 1)$, so dass

$$Q_i = Q\widetilde{Q}_iD_i \text{ und } R_i = D_i\widetilde{R}_i\Lambda^i R$$

gilt. Aus $Q^{(i)} = Q_{i-1}^T Q_i$ und $R^{(i)} = R_iR_{i-1}^{-1}$ sowie $D_i^{-1} = D_i$ und $D_i^T = D_i$ folgt daher

$$\begin{aligned} A^{(i)} &= Q^{(i)}R^{(i)} = Q_{i-1}^T Q_iR_iR_{i-1}^{-1} \\ &= D_{i-1}\widetilde{Q}_{i-1}^T Q^T Q\widetilde{Q}_iD_iD_i\widetilde{R}_i\Lambda^i RR^{-1}\Lambda^{-(i-1)}\widetilde{R}_{i-1}^{-1}D_{i-1} \\ &= D_{i-1}\widetilde{Q}_{i-1}^T\widetilde{Q}_i\widetilde{R}_i\Lambda\widetilde{R}_{i-1}^{-1}D_{i-1}. \end{aligned}$$

Da jede der Matrizen $\widetilde{Q}_{i-1}^T$, $\widetilde{Q}_i$, $\widetilde{R}_i$ und $\widetilde{R}_{i-1}^{-1}$ gegen Id konvergiert, konvergiert das Produkt also gegen $D_{i-1}\Lambda D_{i-1} = \Lambda$, womit die Behauptung gezeigt ist. □

[2] Genauer erhält man wegen der Beschränktheit der Matrixnormen $\|\widetilde{Q}_i\|_2$ und $\|\widetilde{R}_i\|_2$ zunächst konvergente Teilfolgen, aus denen man wegen der Eindeutigkeit der Zerlegung dann auf einen eindeutigen Grenzwert für alle Teilfolgen schließt.

Bemerkung 1.14 (i) Tatsächlich ist die Tatsache, dass $A^{(1)}$ in Tridiagonalform vorliegt, nicht wesentlich für den korrekten Ablauf des Algorithmus, sie vereinfacht aber die QR-Zerlegung in Schritt (1) erheblich: Man sieht leicht, dass die in Algorithmus 2.20 der Einführung in die Numerik auftretenden Summen in diesem Fall nur aus maximal 3 Summanden bestehen, weswegen sich der Aufwand eines Schrittes des Algorithmus auf $O(n^2)$ reduziert. Die Transformation von A auf Tridiagonalgestalt besitzt zwar den Aufwand $O(n^3)$, ohne die vorhergehende Tridiagonalisierung wäre allerdings der Aufwand *jedes Schrittes* gleich $O(n^3)$.

(ii) Eine genauere Analyse zeigt, dass der Algorithmus auch für mehrfache Eigenwerte konvergiert. Existieren allerdings Eigenwerte λ_i und λ_j mit $\lambda_i = -\lambda_j$ so kann in $A^{(i)}$ ein 2×2-Block stehen bleiben.

(iii) Die approximativen Eigenwerte von A liest man bei diesem Algorithmus einfach aus der Diagonalen von $A^{(i)}$ ab. Die zugehörigen Eigenvektoren werden approximiert durch die Spalten der Transformationsmatrix $\overline{Q}_i$, für die $\overline{Q}_i^T A \overline{Q}_i = A^{(i)}$ gilt. Diese ist nach Lemma 1.11(i) gerade durch $\overline{Q}_i = PQ_{i-1} = PQ^{(1)} \cdots Q^{(i-1)}$ gegeben.

(iv) Als Abbruchkriterium des Algorithmus empfiehlt sich die Größe der Nicht-Diagonaleinträge von $A^{(i)}$. Zerlegen wir die Matrix $A^{(i)}$ in ihren Diagonalanteil $\Lambda^{(i)}$ und den Nicht-Diagonalanteil $B^{(i)}$, so gilt für die Eigenwerte von A (die gleich den Eigenwerten von $A^{(i)}$ sind)

$$\lambda_j(A) = \lambda_j(A^{(i)}) = \lambda_j(\Lambda^{(i)} + B^{(i)}) \approx \lambda_j(\Lambda^{(i)}) + \|B^{(i)}\|_2,$$

wobei die letzte "ungefähre" Gleichung aus der Konditionsanalyse des Eigenwertproblems folgt. Für Diagonalmatrizen (bzw. allgemeiner für normale Matrizen) ist die Kondition des Eigenwertproblems bezüglich der 2-Norm gemäß Bemerkung 1.4(ii) gerade gleich 1. Die Matrixnorm $\|B^{(i)}\|_2$ lässt sich aber durch die Größe der Einträge von $B^{(i)}$, also der Nicht-Diagonaleinträge von $A^{(i)}$ abschätzen.

Eine Analyse der Matrizen $\widetilde{Q}_i$ und $\widetilde{R}_i$ sowie ihrer Inversen zeigt, dass die Größe der Nicht-Diagonaleinträge durch die Einträge der im Beweis vorkommenden Matrix E_i bestimmt ist und durch $C \max_{j<k}(\lambda_k/\lambda_j)^{i-1}$ für ein $C > 0$ abgeschätzt werden kann. □

Die Beobachtung in Punkt (iv) zeigt, dass der Algorithmus langsam konvergiert, wenn zwei Eigenwerte λ_j und λ_{j+1} existieren, die betragsmäßig nahe beieinander liegen.

Zur Beschleunigung des Algorithmus verwendet man hier sogenannte *Shift-Strategien*, von denen wir hier den *expliziten Shift* kurz erläutern wollen. Die Grundidee ist, die Eigenwerte von A so zu verschieben, dass der Quotient $|\lambda_{j+1}/\lambda_j|$ kleiner wird. Dazu führt man in der Iteration einen (möglicherweise vom aktuellen Iterationsschritt i abhängigen) *Shift-Parameter* σ_i ein und ändert den Algorithmus wie folgt ab.

Algorithmus 1.15 (QR-Algorithmus zur Eigenwertberechnung mit Shift)
Eingabe: symmetrische, reelle Matrix A, Folge von Shift-Parametern σ_i

(0) Setze $A^{(1)} := P^T A P$ mit P aus Korollar 1.9, $i := 1$

(1) Berechne eine QR-Zerlegung $A^{(i)} - \sigma_i \mathrm{Id} = Q^{(i)} R^{(i)}$

(2) Setze $A^{(i+1)} := R^{(i)} Q^{(i)} + \sigma_i \mathrm{Id}$, $i := i + 1$ und fahre fort bei (1)

Ausgabe: Approximation aller Eigenwerte von A in der Diagonalen von $A^{(i)}$ □

Analog zur einfachen Version des Algorithmus gilt

$$A^{(i+1)} = (Q^{(i)})^T A^{(i)} Q^{(i)} \text{ und } (A - \sigma_i \mathrm{Id}) \dots (A - \sigma_1 \mathrm{Id}) = Q^{(1)} \cdots Q^{(i)} R^{(i)} \cdots R^{(1)}.$$

Hier konvergieren die Nicht-Diagonaleinträge von $A^{(i)}$ gegen Null mit der oberen Schranke

$$\max_{j<k} C \left(\frac{|\lambda_k - \sigma_1|}{|\lambda_j - \sigma_1|} \cdots \frac{|\lambda_k - \sigma_{i-1}|}{|\lambda_j - \sigma_{i-1}|} \right).$$

Die σ_i sollten also möglichst nahe an dem Eigenwert λ_{j+1} liegen, für den $|\lambda_{j+1}/\lambda_j|$ maximal wird.

Wie man dies in der Praxis erreichen kann, ist ein bis heute ungelöstes Problem und immer noch Gegenstand aktueller Forschung. Es gibt allerdings eine Reihe heuristischer Kriterien, die oft gute Ergebnisse zeigen. Eine von J.H. Wilkinson vorgeschlagene Strategie z.B. besteht darin, die Eigenwerte der 2×2 Matrix am unteren Ende der Tridiagonalmatrix $A^{(i)}$ zu berechnen, und σ_i als den kleineren dieser Werte zu wählen.

Bemerkung 1.16 Für nichtsymmetrische Matrizen transformiert man A in Schritt (0) zunächst auf die sogenannte *Hessenbergform*

$$A^{(1)} = \begin{pmatrix} * & \cdots & \cdots & \cdots & * \\ * & \ddots & & & \vdots \\ 0 & \ddots & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & * & * \end{pmatrix}$$

statt auf Tridiagonalgestalt. Der QR-Algorithmus berechnet dann — unter geeigneten Voraussetzungen — ausgehend von der Hessenberg-Form iterativ eine obere Dreiecksmatrix, deren Diagonaleinträge auf Grund des Satzes von Schur (siehe Schwarz/Köckler [5], Satz 5.12) die Eigenwerte approximieren. Für komplexe Eigenwerte $\lambda_j = a + ib$ erhält man dabei einen 2×2-Block der Form

$$\begin{pmatrix} c_1 & c_2 \\ c_3 & c_4 \end{pmatrix},$$

der — als 2×2-Matrix aufgefasst — gerade die Eigenwerte $a \pm ib$ besitzt. □

1.4 Algorithmen für schwach besetzte Matrizen: Arnoldi und Lanczos

Der rechnerische Hauptaufwand der beschriebenen Verfahren liegt für sehr große Matrizen in der Reduktion der ursprünglichen Matrix A auf Tridiagonal- oder Hessenbergform. Ist diese einmal erreicht, hat jeder QR-Schritt nur noch die Ordnung $O(n^2)$, vgl. Bemerkung 1.14(i).

Für schwach (oder auch dünn) besetzte Matrizen — also Matrizen, in denen viele Einträge Null sind — ist es in der Regel nicht effizient, für die Transformation auf Tridiagonal- oder Hessenbergform Householder-Transformationen mit Aufwand $O(n^3)$ zu verwenden. Deswegen stellen wir hier alternative Verfahren vor und beginnen mit dem allgemeinen Fall, in dem A nicht symmetrisch ist.

Das dafür geeignete Verfahren ist das **Arnoldi-Verfahren**. Dies beruht auf der Idee, die Gleichung

$$QA^{(1)} = AQ$$

explizit aufzuschreiben. Bezeichnen wir die Spalten von Q mit q_j und die Einträge von $A^{(1)}$ mit $b_{ij} = q_i^T A q_j$, so erhalten wir

$$\sum_{i=1}^{j+1} b_{ij} q_i = A q_j$$

und daraus

$$b_{j+1,j} q_{j+1} = A q_j - \sum_{i=1}^{j} b_{ij} q_i =: v$$

Gibt man die erste Spalte q_1 von Q vor, kann man damit die weiteren q_i rekursiv bestimmen.

Algorithmus 1.17 (Arnoldi-Verfahren)

Eingabe: $A \in \mathbb{R}^{n\times n}$, $q_1 \in \mathbb{R}^n$ mit $\|q_1\|_2 = 1$

(1) Für $j = 1, 2, \ldots, n$

(2) $v := Aq_j$
für $i = 1, \ldots, j$
$b_{ij} := q_i^T v$
$v := v - b_{ij} q_i$
Ende der i-Schleife

if $j \neq n$

(3) if $v \neq 0$
$b_{j+1,j} := \pm\|v\|$
$q_{j+1} := v / b_{j+1,j}$

(4) else
$b_{j+1,j} := 0$
wähle $q_{j+1} \perp \operatorname{span}\{q_1, \ldots, q_j\}$ beliebig mit $\|q_{j+1}\|_2 = 1$
end if
end if

Ende der j-Schleife

Ausgabe: $A^{(1)} = (b_{ij})_{i,j=1,\ldots,n}$, $Q = (q_1, \ldots, q_n)$ □

Die Rechnungen im Schritt (2) stellen dabei sicher, dass v (und damit q_{j+1}) senkrecht auf $q_1, \ldots, q_j$ steht: Für $j = 1$ gilt

$$v^T q_1 = \left(Aq_1 - q_1^T (Aq_1) q_1\right)^T q_1 = q_1^T A^T q_1 - q_1^T A^T q_1 \underbrace{q_1^T q_1}_{=1} = 0$$

Für $j > 1$ fährt man induktiv fort. Tatsächlich findet hier eine Variante der Gram-Schmidt Orthogonalisierung statt (diese werden wir später in Abschnitt 2.8 genauer betrachten). Die Wahl von $b_{j+1,j}$ stellt $\|q_{j+1}\|_2 = 1$ sicher. Beachte, dass das Vorzeichen hier frei gewählt werden kann. Der Fall $v = 0$ wird als "Zusammenbruch" (engl. breakdown) des Verfahrens bezeichnet, die Wahl von q_{j+1} in (4) als "Neustart". Dieser wird oft auch im Fall $v \approx 0$ durchgeführt. Ein Breakdown tritt auf, wenn die Vektoren $q_1, \ldots, q_k$ einen invarianten Unterraum für A aufspannen, wenn also $A\text{span}\{q_1, \ldots, q_k\} = \text{span}\{q_1, \ldots, q_k\}$ gilt.

Im symmetrischen Fall $A^T = A$ ist auch $A^{(1)}$ symmetrisch und damit tridiagonal. In diesem Fall können wir die Schleife in (2) ersetzen durch

$$v := v - b_{jj}q_j - b_{j-1,j}q_{j-1}.$$

Das so modifizierte Verfahren heißt **Lanczos-Verfahren**. Falls wir nur an den Eigenwerten und nicht an den Eigenvektoren interessiert sind, müssen die q_j nicht gespeichert werden.

Vorteile des Verfahrens sind, dass die Anzahl der Rechenoperationen i.W. nur von der Anzahl der nicht-Null Einträge von A abhängt, da nur diese beim Auswerten von Aq_j verwendet werden. Tatsächlich braucht A gar nicht explizit bekannt zu sein; es reicht für den Algorithmus, eine Routine zu haben, die Aq_j ausrechnen kann. Nachteil des Verfahrens ist, dass es numerisch weniger stabil als die Variante über die QR-Zerlegung ist.

1.5 Das unvollständige Lanczos-Verfahren

Die Eigenwerte von A könnte man nun mit dem gerade beschriebenen Lanczos-Verfahren und nachfolgendem QR-Verfahren bestimmen. Wenn A sehr groß ist, kann aber auch diese sehr effiziente Methode sehr lange dauern. Wir untersuchen daher nun, ob man noch sinnvolle Ergebnisse erhalten kann, wenn man das Lanczos-Verfahren frühzeitig, also nach $m \ll n$ Schritten abbricht.

Wir betrachten also eine symmetrische Matrix $A \in \mathbb{R}^{n\times n}$ und nummerieren ihre Eigenwerte $\lambda_i(A)$ in der Form

$$\lambda_1(A) \geq \ldots \geq \lambda_n(A).$$

Die Matrix Q enthalte eine Orthonormalbasis aus Eigenvektoren, also $A = Q^T\Lambda Q$, wobei $Q = \text{diag}(\lambda_1(A), \ldots, \lambda_n(A))$. Wir erinnern zudem daran, dass die Eigenwerte immer reell sind und für die Kondition des Eigenwertproblems $\kappa_{\text{abs}} = 1$ gilt (dies folgt aus Bemerkung 1.4, weil symmetrische Matrizen immer normal sind).

Wir betrachten nun zunächst drei vorbereitende Sätze.

Satz 1.18 Sei $A^T = A$, $\|x\|_2 = 1$ und $\|(A - \mu\text{Id})x\|_2 = \varepsilon$ für ein $\mu \in \mathbb{R}$ und $\varepsilon > 0$. Dann besitzt A einen Eigenwert λ mit $|\lambda - \mu| \leq \varepsilon$.

Beweis: Es gilt

$$\begin{aligned}\max_i \left\{\frac{1}{\lambda_i(A) - \mu}\right\} &= \|(A - \mu\text{Id})^{-1}\|_2 = \sup_{z\in\mathbb{R}^n} \frac{\|(A - \mu\text{Id})^{-1}(z)\|_2}{\|z\|_2} \\ &\geq \frac{\|x\|_2}{\|(A - \mu\text{Id})x\|_2} = \frac{1}{\varepsilon}.\end{aligned}$$

Durch Bilden des Kehrwerts folgt die Behauptung. □

Als nächstes betrachten wir eine Möglichkeit, Eigenwerte als Maxima und Minima zu charakterisieren. Für $k \in \{1, \ldots, n\}$ definieren wir dazu für die Orthonormalbasis aus Eigenvektoren $q_1, \ldots, q_n$.

$$Q_k := \operatorname{span}\{q_1, \ldots, q_k\} \quad \text{und} \quad \overline{Q}_k := \operatorname{span}\{q_k, \ldots, q_n\}.$$

Es gilt der folgende Satz.

Satz 1.19 (Courant-Fischer) Sei $A \in \mathbb{R}^{n\times n}$ symmetrisch. Dann gilt für $k \in \{1, \ldots, n\}$:

$$\begin{aligned} \lambda_k(A) &= \max_{\dim S = k} \min_{\substack{y\in S \\ \|y\|_2=1}} y^T A y = \min_{\substack{y\in Q_k \\ \|y\|_2=1}} y^T A y \\ &= \min_{\dim S = n-k+1} \max_{\substack{y\in S \\ \|y\|_2=1}} y^T A y = \max_{\substack{y\in \overline{Q}_k \\ \|y\|_2=1}} y^T A y. \end{aligned}$$

Beweis: Zunächst einmal gilt

$$\max_{\dim S = k} \min_{\substack{y\in S \\ \|y\|_2=1}} y^T A y \geq \min_{\substack{y\in Q_k \\ \|y\|_2=1}} y^T A y = q_k^T A q_k = \lambda_k(A).$$

Außerdem gilt für jeden k-dimensionalen Teilraum $S \subset \mathbb{R}^n$ aus Dimensionsgründen $S \cap \overline{Q}_k \neq \{0\}$. Für alle $y_* \in S \cap \overline{Q}_k$ mit $\|y_*\|_2 = 1$ gilt

$$\min_{\substack{y\in S \\ \|y\|_2=1}} y^T A y \leq y_*^T A y_* \leq \lambda_k(A)$$

und weil dies für alle k-dimensionalen Unterräume $S \subset \mathbb{R}^n$ gilt, gilt die Ungleichung auch für das Maximum über S. Die anderen Gleichungen folgen mit analogen Argumenten. □

Als Spezialfälle folgen aus dem Satz

$$\lambda_1 = \max_{\|y\|_2=1} y^T A y \quad \text{und} \quad \lambda_n = \min_{\|y\|_2=1} y^T A y.$$

Verwendet man die Matrizen Q_m anstelle der Matrizen Q aus dem Lanczos-Verfahren, so ist

$$B_m := Q_m^T A Q_m$$

für $m < n$ eine kleinere Matrix als $Q^T A Q$, deren Eigenwerte (die auch *Ritz-Werte* genannt werden) demnach schneller berechnet werden können. Es sei V der von Q_m aufgespannte Unterraum. Der folgende Satz zeigt, wie die Eigenwerte von B_m mit denen von A zusammenhängen.

Satz 1.20 (Interlacing) Es gilt

$$\lambda_{k+n-m}(A) \leq \lambda_k(B_m) \leq \lambda_k(A).$$

Beweis: Mit Satz 1.19 und wegen $S := Q_m S' \subset V$ für alle Unterräume $S' \subset \mathbb{R}^n$ gilt

$$\lambda_k(B_m) = \max_{\substack{S \subset V \\ \dim S = k}} \min_{y \in S} y^T A y \leq \max_{\substack{S \subset \mathbb{R}^n \\ \dim S = k}} \min_{y \in S} y^T A y = \lambda_k(A)$$

und

$$\begin{aligned} \lambda_k(B_m) &= \min_{\substack{S \subset V \\ \dim S = m-k+1}} \max_{y \in S} y^T A y \geq \min_{\substack{S \subset \mathbb{R}^n \\ \dim S = m-k+1}} \max_{y \in S} y^T A y \\ &= \lambda_{n-(m-k+1)+1}(A) = \lambda_{k+n-m}(A). \end{aligned}$$

□

Insbesondere gilt also

$$\lambda_n(A) \leq \lambda_m(B_m) \leq \lambda_1(B_m) \leq \lambda_1(A)$$

und für $m = n - 1$

$$\lambda_n(A) \leq \lambda_{n-1}(B_{n-1}) \leq \lambda_{n-1}(A) \leq \ldots \leq \lambda_1(B_{n-1}) \leq \lambda_1(A).$$

Die schon erwähnte Idee ist nun, die Matrizen Q_m und damit die $m \times m$-Tridiagonalmatrix $B_m = Q_m^T A Q_m$ durch $m \ll n$ Schritte des Lanczos-Verfahrens zu berechnen. Beachte dabei, dass die q_i aus dem Lanczos-Verfahren keine Eigenvektoren sind, der Algorithmus also nicht die gleichen Matrizen B_m wie in Satz 1.20 berechnet. Satz 1.20 liefert daher keinen Beweis für die Genauigkeit des Verfahrens (diese werden wir im Anschluss separat beweisen), aber die Motivation für den Ansatz.

Algorithmus 1.21 (Lanczos-Verfahren mit m Schritten)

Eingabe: $A \in \mathbb{R}^{n \times n}$, $q_1 \in \mathbb{R}^n$ mit $\|q_1\|_2 = 1$

(1) Für $j = 1, 2, \ldots, m$

(2) $v := A q_j$
für $i = \max\{1, j-1\}, \ldots, j$
$b_{ij} := q_i^T v$
$v := v - b_{ij} q_i$
Ende der i-Schleife

(3) if $v \neq 0$
$b_{j+1,j} := \|v\|$
$q_{j+1} := v / b_{j+1,j}$

(4) else
Ende mit Fehlermeldung: Breakdown
end if

Ende der j-Schleife

Ausgabe: $B_m = (b_{ij})_{i,j=1,\ldots,m}$, $Q_m = (q_1, \ldots, q_m)$ □

Die Eigenwerte von B_m können dann mit dem QR-Verfahren berechnet werden. Für diese gilt

Satz 1.22 Sei (μ, w) ein Eigenpaar von B_m mit $\|w\|_2 = 1$. Dann besitzt A einen Eigenwert λ mit $|\lambda - \mu| \leq b_{m+1,m}|w_m|$, wobei $b_{m+1,m}$ und w_m die entsprechenden Einträge von B_m und w sind.

Beweis: Es gilt $AQ_n = QB_n$ mit $B_n = A^{(1)}$. Betrachtet man nur die ersten m Spalten dieser Matrixgleichung, so erhält man wegen der Tridiagonalform von B_n

$$AQ_m = (QB_n)_{1..m} = Q_m B_m + b_{m+1,m} q_{m+1} e_m^T,$$

wobei wir verwendet haben, dass B_n als Tridiagonalmatrix die Blockstruktur

$$B_n = \begin{pmatrix} B_m & * \\ C_m & * \end{pmatrix} \quad \text{mit} \quad C_m = \begin{pmatrix} 0 & 0 & b_{m+1,m} \\ 0 & 0 & 0 \\ \vdots & \vdots & \vdots \\ 0 & 0 & 0 \end{pmatrix}$$

besitzt. Für $x = Q_m w$ gilt $\|x\|_2 = 1$ und damit

$$(A - \mu \mathrm{Id})x = AQ_m w - Q_m \mu w = (AQ_m - Q_m B_m)w = b_{m+1,m} q_{m+1} e_m^T w.$$

Daraus folgt $\|(A - \mu \mathrm{Id})x\|_2 \leq b_{m+1,m}|w_m|$ und die Behauptung folgt aus Satz 1.18. □

1.6 Weitere Verfahren für symmetrische Matrizen

Abschließend wollen wir einige Alternativen zum QR-Verfahren für tridiagonale, symmetrische Matrizen besprechen, die in gewissen Situationen Vorteile haben können.

Bisektion

Für tridiagonale und symmetrische Matrizen nimmt das charakteristische Polynom eine sehr einfache Form an, weswegen es u.U. eine sinnvolle Alternative für die Eigenwertberechnung sein kann. Bezeichnen wir die Diagonalelemente mit a_i und die Nebendiagonalelemente mit b_i. Sei zudem A_k die obere linke $k \times k$-Submatrix von A. Dann gilt für das charakteristische Polynom aus der Determinantenentwicklung die Rekursionsformel

$$p_{A_k}(\lambda) = \det((A - \lambda \mathrm{Id})_k) = (a_k - \lambda) p_{A_{k-1}}(\lambda) - b_{k-1} p_{A_{k-2}}(\lambda).$$

Wenn wir nun a und b mit $p(a)p(b) < 0$ finden können, können wir eine Nullstelle, also einen Eigenwert, relativ einfach per Bisektion ausrechnen. Das Problem ist aber, dass wir dann nicht genau wissen, welchen Eigenwert man so findet.

Dieses Problem kann man aber beheben: Da die Determinante gerade das Produkt der Eigenwerte ist, ist das Vorzeichen von $\det((A-\lambda \mathrm{Id})_k)$ gerade dann positiv, wenn die Anzahl der Eigenwerte $< \lambda$ gerade ist. Zudem folgt aus dem Interlacing-Satz 1.20

$$\lambda_{j+1}((A - \lambda \mathrm{Id})_{k+1}) \leq \lambda_j((A - \lambda \mathrm{Id})_k) \leq \lambda_j((A - \lambda \mathrm{Id}_{k+1}).$$

Berechnen wir also rekursiv die Determinanten $\det((A-\lambda \mathrm{Id})_k)$ für $k = 1, 2, \ldots$, so bedeutet jeder Vorzeichenwechsel von k nach $k+1$, dass der neu hinzugekommene Eigenwert $< \lambda$ ist. Damit können wir rekursiv bestimmen, wie viele Eigenwerte kleiner und wie viele größer als λ sind, d.h. die Zahl

$$j_+(\lambda) = \max\{j \,|\, \lambda_j(A) > \lambda\}.$$

Damit können wir nun in der Bisektion gezielt den k-ten Eigenwert aus der Liste der Eigenwerte zwischen a und b wie folgt ausrechnen

Algorithmus 1.23 (**Bisektionsalgorithmus für k-ten Eigenwert**)

Eingabe: $a < b$ mit $p(a)p(b) < 0$, $\varepsilon > 0$

solange $|b - a| > \varepsilon$

$c = a + (b - a)/2$

if $(j_+(c) > k)$ then $b = c$ else $a = c$ end if

Ausgabe: Näherung c des k-ten Eigenwerts $\lambda_k(A)$, falls dieser am Anfang in $[a, b]$ enthalten ist. Falls $\lambda_k(A) < a$, konvergiert c gegen a und falls $\lambda_k(A) > b$ konvergiert c gegen b. □

Dass der Algorithmus tatsächlich das behauptete Konvergenzverhalten aufweist, folgt daraus, dass im Fall $\lambda_k(A) \in [a, b]$ im verlauf des Algorithmus stets $j_+(a) \leq k$ und $j^+(b) \geq k$ gilt.

Wenn die Matrix A groß ist, man nur wenige Eigenwerte berechnen will und geeignete Startwerte a und b leicht zu bestimmen sind, kann die Bisektion besser sein als der QR-Algorithmus.

Jacobi-Iteration

Die Jacobi-Iteration benötigt symmetrische, aber nicht notwendigerweise tridiagonale Matrizen A als Eingabe. Ihre Idee liegt darin, für $1 \leq i < j \leq n$ die Submatrix

$$\widehat{A}_{ij} = \begin{pmatrix} a_{ii} & a_{ij} \\ a_{ij} & a_{jj} \end{pmatrix}$$

zu betrachten und eine Givens-Rotation G_{ij} (vgl. dazu (1.9)) mit Submatrix $\widehat{G}_{ij}$ zu konstruieren, so dass $\widehat{B}_{ij} = \widehat{G}_{ij}\widehat{A}_{ij}\widehat{G}_{ij}^T$ eine Diagonalmatrix wird. Dann setzen wir

$$B := G_{ij} A G_{ij}^T.$$

Die Einträge c und s der Givens-Rotationsmatrix (1.9) wählt man dazu als

$$s := 1/\sqrt{1+\tau^2}, \quad c = s\tau$$

mit

$$\tau := \frac{1}{\Theta + \mathrm{sgn}(\Theta)\sqrt{1+\Theta^2}}, \quad \Theta = \frac{a_{jj} - a_{ii}}{2a_{ij}}.$$

Das folgende Beispiel illustriert diese Rechnung

Beispiel 1.24 Betrachte die Matrix

$$A = \begin{pmatrix} 1 & 3 & 4 \\ 3 & 2 & 0 \\ 4 & 0 & 3 \end{pmatrix}.$$

Dann ist

$$\widehat{A}_{13} = \begin{pmatrix} 1 & 2 \\ 2 & 3 \end{pmatrix}$$

und gemäß den Rechnungen nach (1.9) ergibt sich (alle Zahlen auf sechs Stellen gerundet)

$$\widehat{G}_{13} = \begin{pmatrix} 0.61541 & 0.78821 \\ -0.78821 & 0.61541 \end{pmatrix}.$$

Damit gilt

$$G_{13} A G_{13}^T = \begin{pmatrix} 4.23606 & 2.10292 & 0 \\ 2.10292 & 2 & -3.40260 \\ 0 & -3.40260 & -0.23607 \end{pmatrix}.$$

Es sind also tatsächlich an den gewünschten Stellen Nulleinträge entstanden, dafür sind aber an anderen Stellen aus Nulleinträgen Nicht-Nulleinträge geworden. □

Tatsächlich können sich beim Übergang von A zu B alle Elemente in der i-ten und j-ten Zeile und Spalte von A verändern. Daher ist zunächst nicht offensichtlich, dass diese Operation die Nichtdiagonalelemente von A insgesamt verkleinert. Im obigen Beispiel ist das für einzelne Elemente offensichtlich auch nicht der Fall. Betrachtet man allerdings die Summe der quadrierten Nichtdiagonalelemente, so lautet diese für A $2 \cdot 3^2 + 2 \cdot 4^2 = 18 + 32 = 50$ und für die transformierte Matrix $2 \cdot 2.10292^2 + 2 \cdot 3.40260^2 \approx 32$. Die Summe der Quadrate nimmt also offenbar ab und dies lässt sich auch allgemein beweisen. Wir betrachten dazu

$$S(A) := \sum_{\substack{i,j=1 \\ i \neq j}}^{n} a_{ij}^2 = \|A\|_F^2 - \|\mathrm{diag}(A)\|_F^2,$$

wobei $\|A\|_F := \sqrt{\sum_{i,j=1}^n a_{ij}^2}$ die Frobenius-Norm von A bezeichnet und $\mathrm{diag}(A) := \mathrm{diag}(a_{11}, a_{22}, \ldots, a_{nn})$ gilt. Da $\|A\|_F^2$ die Summe der quadrierten Längen der Spaltenvektoren in A ist, ist diese Größe invariant unter orthogonalen (also längentreuen) Transformationen. Es gilt also $\|B\|_F = \|A\|_F$ und damit

$$\begin{aligned} S(B) - S(A) &= \|B\|_F^2 - \|\mathrm{diag}(B)\|_F^2 - \|A\|_F^2 + \|\mathrm{diag}(A)\|_F^2 \\ &= \|\mathrm{diag}(A)\|_F^2 - \|\mathrm{diag}(B)\|_F^2 \quad = \quad a_{ii}^2 + a_{jj}^2 - b_{ii}^2 - b_{jj}^2, \end{aligned}$$

da dies die einzigen Nichtdiagonalelemente unter den veränderten Elementen sind. Wiederum auf Grund der Längenerhaltung der Givens-Rotation folgt nun $b_{ii}^2 + b_{jj}^2 + 2b_{ij}^2 = a_{ii}^2 + a_{jj}^2 + 2a_{ij}^2$ sowie $b_{ij} = b_{ji} = 0$. Damit folgt

$$S(B) = S(A) - 2a_{ij}^2.$$

Die Summe der Nichtdiagonalelemente nimmt also in jedem Schritt ab. Die Elemente a_{ij} werden nun entweder so ausgewählt, dass ihr Betrag maximal ist (was den größten Abstieg von $S(A)$ bewirkt aber eine aufwändige Suche nötig macht), oder es werden einfach zyklisch alle Elemente a_{ij} durchlaufen. Die erste Variante heißt *klassisches Jacobi-Verfahren*, die zweite *zyklisches Jacobi-Verfahren*. Bezeichnet man die so erhaltenen Matrizen mit $A^{(k)}$, so kann man in beiden Fällen beweisen, dass $S(A^{(k)}) \to 0$ konvergiert für $k \to \infty$.

Divide and Conquer

Kehren wir noch einmal zum tridiagonalen (und symmetrischen) Fall zurück. Dann kann die Matrix A aufgeteilt werden als

$$A = \begin{pmatrix} A_1 & & & \\ & \beta & & \\ & \beta & & \\ & & A_2 & \end{pmatrix} = \begin{pmatrix} \widehat{A}_1 & & & \\ & & & \\ & & & \\ & & & \widehat{A}_2 \end{pmatrix} + \beta \begin{pmatrix} & & & \\ & 1 & 1 & \\ & 1 & 1 & \\ & & & \end{pmatrix}.$$

Nun kann man mit geeigneten Algorithmen die Eigenwerte von A aus denen von $\widehat{A}_1$ und $\widehat{A}_2$ berechnen und so das Problem in kleinere Teilprobleme aufspalten. Details dazu finden sich z.B. in dem Buch *Grundlagen der Numerischen Mathematik und des Wissenschaftlichen Rechnens* von Martin Hanke-Bourgeois, Vieweg und Teubner, 2009.

1.7 Singulärwertzerlegung

Wir haben bereits in der Einführung in die Numerik die Singulärwertzerlegung einer nichtquadratischen Matrix kennengelernt. Hier beweisen wir zunächst deren Existenz, womit wir auch den Zusammenhang zu den Eigenwerten klären.

Satz 1.25 Zu jeder Matrix $A \in \mathbb{R}^{m\times n}$ existieren zwei orthogonale Matrizen $U \in \mathbb{R}^{m\times m}$ und $V \in \mathbb{R}^{n\times n}$ (also mit $U^TU = \mathrm{Id}$, $V^TV = \mathrm{Id}$) sowie eine Matrix $\Sigma \in \mathbb{R}^{m\times n}$ gibt, bei der nur die Diagonalelemente σ_i Werte ungleich Null annehmen können, so dass

$$A = U\Sigma V^T$$

gilt. Die Diagonaleinträge von Σ sind dabei eindeutig (bis auf Vertauschung), stimmen mit den Wurzeln der Eigenwerte der Matrix A^TA überein und heißen die *Singulärwerte* von A.

Beweis: Da A^TA symmetrisch ist, existiert eine orthogonale Matrix $V \in \mathbb{R}^{m\times m}$ mit

$$V^TA^TAV = \mathrm{diag}(\lambda_1, \ldots, \lambda_m).$$

Dabei seien die λ_j betragsmäßig absteigend sortiert, also $|\lambda_{j+1}| \le |\lambda_j|$. Für die Spalten v_j von V gilt $A^TAv_j = \lambda_j v_j$ und damit

$$\lambda_j = \lambda_j v_j^T v_j = v_j^T A^T A v_j = \|Av_j\|_2^2 \ge 0.$$

Also können wir $\sigma_i := \sqrt{\lambda_i}$ setzen und erhalten reelle Werte.

Seien nun $\sigma_1, \ldots, \sigma_r \neq 0$ und $\sigma_{r+1}, \ldots, \sigma_m = 0$. Wir setzen $u_i = \sigma_i^{-1} A v_i \in \mathbb{R}^n$ für $i = 1, \ldots, r$. Dann ist

$$u_i^T u_i = \lambda_i^{-1} (Av_i)^T A v_i = \lambda_i^{-1} v_i^T A^T A v_i = v_i^T v_i = 1$$

und

$$u_i^T u_j = \lambda_i^{-1/2} \lambda_j^{-1/2} (Av_i)^T A v_j = \lambda_i^{-1} v_i^T A^T A v_j = \lambda_i^{-1/2} \lambda_j^{1/2} v_i^T v_j = 0$$

für $i \neq j$. Folglich können die $u_1, \ldots, u_r$ zu einer Orthonormalbasis $(u_1, \ldots, u_n)$ des $\mathbb{R}^n$ ergänzt werden. Aus $\sigma_{r+1}, \ldots, \sigma_m = 0$ folgt $\lambda_{r+1}, \ldots, \lambda_m = 0$ und damit $Av_j = 0$ für $j = r+1, \ldots, m$. Für $U = (u_1, \ldots, u_n) \in \mathbb{R}^{n \times n}$ gilt daher $U\Sigma = (Av_1, \ldots, Av_r, 0, \ldots, 0) = (Av_1, \ldots, Av_m) = AV$ und folglich

$$U \Sigma V^T = AVV^T = A.$$

□

Die Singulärwertzerlegung ist z.B. wichtig, um sehr große Matrizen A durch Matrizen A_k zu ersetzen, die mit weniger Platz gespeichert werden können. Dazu definiert man U_k als die Matrix, die die ersten k Spalten von U enthält, V_k als die Matrix, die die ersten k Zeilen von V enthält und Σ_k als die obere linke $k \times k$-Untermatrix von Σ und setzt $A_k := U_k \Sigma_k V_k$. Beachte, dass A_k und A die gleiche Dimension besitzen. Dann kann man beweisen, dass die Differenz zwischen Ax und $A_k x$ in der 2-Norm von der Größe der in Σ_k fehlenden Singulärwerte abhängt. Besitzt A also viele Singulärwerte nahe 0, so kann man k klein wählen und viele Singulärwerte "abschneiden", ohne einen großen Fehler zu machen, wodurch U_k Σ_k und V_k mit insgesamt deutlich weniger Speicherbedarf als das ursprüngliche A gespeichert werden können. Auch kann $A_k x$ dann schneller ausgewertet werden als Ax.

Eine erste Idee zur Berechnung der Singulärwertzerlegung wäre nun, einfach einen Eigenwertalgorithmus auf $A^T A$ anzuwenden. Ähnlich wie bei den Normalengleichungen kann dies aber numerisch instabil werden, weil die Kondition von $A^T A$ sehr groß werden kann. Besser ist es, direkt mit der Matrix A zu arbeiten.

Dazu bringen wir B zunächst durch geeignete Householder-Transformationen von links und rechts in Bidiagonalform, um die Anzahl der Rechenoperationen der nachfolgenden QR-Iterationen zu verringern.

Algorithmus 1.26
Eingabe: $A_0 := A \in \mathbb{R}^{m \times n}$
für $j = 1, \ldots, n-1$
    bestimme die Householder-Matrix H_j, die die j-te Spalte von A_{j-1}
        nach span$\{e_1, \ldots, e_j\}$ spiegelt
    Setze $\widetilde{A}_j := H_j A_{j-1}$
    Falls $j \leq n-2$
        bestimme die Householder-Matrix K_j, die die j-te Zeile von $\widetilde{A}_j$
            nach span$\{e_1, \ldots, e_{j+1}\}$ spiegelt
    Setze $A_j := \widetilde{A}_j K_j$
Ausgabe: $B = \widetilde{A}_{n-1} = H_{n-1} \cdot \ldots \cdot H_1 A K_1 \cdot \ldots K_{n-2}$ in Bidiagonalform □

Alternativ kann man für große schwach besetzte Matrizen eine Variante des Lanczos-Verfahrens verwenden.

Die Matrix B ist jetzt in der Form

$$B = \begin{pmatrix} b_{11} & b_{12} & & \\ & \ddots & \ddots & \\ & & \ddots & b_{n-1,n} \\ & & & b_{nn} \end{pmatrix}.$$

Die Idee ist nun, direkt auf der Matrix B eine Operation durchzuführen, die einem Schritt des QR-Algorithmus auf der Matrix B^TB entspricht und die Bidiagonalstruktur von B erhält. Dies geht allerdings nicht mit Housholder-Transformationen sondern nur, wenn die Matrix Q über Givens-Rotationen berechnet wird. Dabei wird die orthogonale Matrix Q nicht durch Spiegelungen sondern durch Drehungen erzeugt. Eine Givens-Rotation ist definiert als Matrix der Form

$$G_{kl} := \begin{pmatrix} 1 & & & & & & & & & \\ & \ddots & & & & & & & & \\ & & 1 & & & & & & & \\ & & & c & & & s & & & \\ & & & & 1 & & & & & \\ & & & & & \ddots & & & & \\ & & & & & & 1 & & & \\ & & & -s & & & & c & & \\ & & & & & & & & 1 & \\ & & & & & & & & & \ddots \\ & & & & & & & & & & 1 \end{pmatrix} \tag{1.9}$$

mit $s = \sin\alpha$ und $c = \cos\alpha$ für einen Winkel $\alpha \in \mathbb{R}$. Diese Einträge stehen dabei in Zeile bzw. Spalte k und l.

Angewendet auf einen Vektor x ergibt sich

$$G_{kl}x = \begin{cases} cx_k + sx_l, & \text{für } i = k \\ -sx_k + cx_l, & \text{für } i = l \\ x_i, & \text{sonst} \end{cases} \tag{1.10}$$

Die Matrix G_{lk} definiert also eine Rotation oder Drehung um den Winkel α in der Ebene $\text{span}\{e_k, e_l\}$. Zur Erzeugung der oberen Dreiecksmatrix $R = Q^TA$ aus einer Matrix A müssen wir nun die Einträge unterhalb der Diagonalen durch passende Drehungen zu Null machen. Sei dazu x die k-te Spalte der Matrix A und $l > k$ der Zeilenindex des Elements, das den Wert 0 erhalten soll. Durch Einsetzen in (1.10) sieht man, dass man dies mit der Wahl

$$\tau := x_k/x_l, \quad s := 1/\sqrt{1+\tau^2}, \quad c = s\tau$$

für $|x_l| > |x_k|$ bzw.

$$\tau := x_l/x_k, \quad c := 1/\sqrt{1+\tau^2}, \quad s = c\tau$$

andernfalls, gerade erreicht. Man kann also die einzelnen Einträge der Matrix A unterhalb der Diagonalen wir bei der Gauß-Elimination sukzessive zu Null machen. Aufmultiplizieren der dazu benötigten G_{kl}'s liefert dann die Matrix Q^T.

Für voll besetzte Matrizen sind für die QR-Zerlegung mittels Givens-Rotationen ähnlich viele Operationen wie bei Householder nötig. Für Matrizen in spezieller Form — z.B. in Hessenberg-Form aus Bemerkung 1.16 — reduziert sich die Anzahl der benötigten Operationen hingegen deutlich, ähnlich wie bei Householder, vgl. Bemerkung 1.14.

Für unsere Aufgabe, die QR-Zerlegung von B^TB auf Basis von B zu berechnen, haben die Givens-Rotationen gegenüber den Housholder-Spiegelungen den entscheidenden Vorteil, dass ihre Auswirkungen auf die Einträge in den anderen Spalten günstiger sind. Angewendet auf

$$B^TB = \begin{pmatrix} * & * & 0 & 0 & 0 \\ * & * & * & 0 & 0 \\ 0 & * & * & * & 0 \\ 0 & 0 & * & * & * \\ 0 & 0 & 0 & * & * \end{pmatrix}$$

muss zunächst G_{12} angewendet werden, um die erste Spalte in die Form $(*, 0, 0, 0, 0, 0)^T$ zu bringen. Dadurch ist BG_{12}^T von der Form

$$\widetilde{B}_1 := BG_{12}^T = \begin{pmatrix} * & * & 0 & 0 & 0 \\ \oplus & * & * & 0 & 0 \\ 0 & 0 & * & * & 0 \\ 0 & 0 & 0 & * & * \\ 0 & 0 & 0 & 0 & * \end{pmatrix}.$$

Dieser zusätzliche Eintrag kann aber durch eine weitere Givens-Rotation $\widetilde{G}_{12}$ angewendet auf $\widetilde{B}_1$ wieder entfernt werden. Wir erhalten so

$$B_1 := \widetilde{G}_{12}\widetilde{B}_1 = \widetilde{G}_{12}BG_{12}^T = \begin{pmatrix} * & * & \oplus & 0 & 0 \\ 0 & * & * & 0 & 0 \\ 0 & 0 & * & * & 0 \\ 0 & 0 & 0 & * & * \\ 0 & 0 & 0 & 0 & * \end{pmatrix}.$$

Wegen

$$B_1^TB_1 = (\widetilde{G}_{12}BG_{12}^T)^T(\widetilde{G}_{12}BG_{12}^T) = (BG_{12}^T)^T(\widetilde{G}_{12}BG_{12}^T) = \widetilde{B}_1^T\widetilde{B}_1$$

hat die Matrix $\widetilde{G}_{12}$ hierbei keinen Einfluss auf die Matrix, die wir QR-zerlegen wollen.

Der jetzt neu entstandene nicht-Null-Eintrag an der Stelle $(1, 3)$ — der sogenannte Fill-in — wird nun durch die nächste Givens-Rotation G_{23} angewendet auf $B_1^TB_1$ wieder entfernt, wie ein kleine Rechnung zeigt. Da dies diese Rotation eindeutig definiert, kann diese nun auch direkt auf Basis von B_1 berechnet werden. Die neue Matrix $\widetilde{B}_2$ besitzt nun einen Fill-in an der Stelle $(3, 2)$, der durch ein geeignetes $\widetilde{G}_{23}$ entfernt werden muss. Führt man dies Schritt für Schritt weiter durch, rutscht der Fill-in immer weiter nach rechts unten und verschwindet zuletzt. Aufmultiplizieren der $\widetilde{G}_{ij}$ zu einem $\widetilde{G}$ und der G_{ij} zu einem G liefert dann die bidiagonale Matrix $B^{(1)} = \widetilde{G}BG^T$, wobei $(B^{(1)})^TB^{(1)}$ gerade der Matrix entspricht, die nach einem QR-Schritt aus B^TB hervorgeht. Auf diese Weise kann der QR-Algorithmus 1.10 direkt auf B statt auf $A = B^TB$ ausgeführt werden.

Kapitel 2

Iterative Verfahren für lineare Gleichungssysteme

Wir haben bereits in der Einführung in die Numerik einige iterative Verfahren zur Lösung linearer Gleichungssystems

$$Ax = b$$

kennengelernt. Diese Verfahren sind insbesondere dann effizient, wenn die Matrix A sehr groß ist, aber nur wenige Einträge ungleich Null besitzt. Die Matrix A heißt dann schwach besetzt oder dünn besetzt. Solche Matrizen speichert man nicht in einem zweidimensionalen Array, sondern man speichert jeden Eintrag $a_{ij} \neq 0$ einzeln als Tripel (i, j, d) mit $d = a_{ij}$ ab. Wird eine Matrix durch N solche Tripel (i_l, j_l, d_l) mit $l = 1, \ldots, N$ beschrieben, kann man z.B. die Multiplikation von A mit einem Vektor x mittels

$$y = 0, \qquad y_{i_l} := y_{i_l} + d_l x_{j_l},\ l = 1, \ldots, N$$

mit Ordnung $O(N)$ realisieren. Der Rechenaufwand hängt also nicht von der Größe der Matrix sondern von der Anzahl der nicht-Null-Einträge ab.

Es gibt durchaus Ansätze, direkte Verfahren wie die Gauß-Elimination oder das Choleski-Verfahren zur Lösung linearer Gleichungssysteme mit solchen Matrizen zu verwenden. Das Problem ist aber, dass im Zuge des Eliminationsverfahrens viele vorherige Null-Einträge Werte ungleich Null erhalten, so dass die dünn besetzte Struktur verloren geht und der Rechenaufwand drastisch ansteigt. Verfahren dieser Art sind daher nur für ganz spezielle Strukturen geeignet, d.h. wenn die nicht-Null-Einträge nach speziellen Mustern angeordnet sind. Wir wollen diese Verfahren hier nicht betrachten und uns ganz auf iterative Verfahren beschränken. Wir beginnen mit einer sehr allgemeinen Idee, in der wir insbesondere einige Verfahren aus der Einführung in die Numerik als Spezialfälle wieder finden können.

2.1 Grundidee

Das einfachste Verfahren zur iterativen Lösung ist die Richardson-Iteration, die nichts anderes als die lineare Version der Fixpunkt-Iteration ist: wir schreiben das Nullstelenproblem

$0 = f(x) := b - Ax$ um in ein Fixpunktproblem $0 = g(x) = x + f(x) = x + (b - Ax) = (\mathrm{Id} - A)x + b$ und verwenden g als Iterationsvorschrift.

Algorithmus 2.1 Richardson-Iteration

(0) Wähle einen Startwert $x^{(0)} \in \mathbb{R}^n$

(1) Berechne iterativ $x^{(k+1)} := (\mathrm{Id} - A)x^{(k)} + b$ □

Korollar 2.2 Die Richardson-iteration konvergiert gegen die Lösung des Gleichungssystems $Ax = b$, wenn $\Sigma(A) \subset B_\Theta(1)$ für ein $\Theta < 1$ gilt. Dabei bezeichnet $\Sigma(A) = \{\lambda \in \mathbb{C} \,|\, \lambda \text{ ist Eigenwert von } A\}$ das Spektrum von A und $B_\Theta(1)$ den Ball mit Radius Θ um 1 in $\mathbb{C}$.

Beweis: Es folgt aus Korollar 5.3 aus der Einführung in die Numerik, dass die Iteration konvergiert, wenn $\rho(\mathrm{Id}-A) < 1$ ist (Erinnerung: $\rho(A) = \max\{|\lambda| : \lambda \in \mathbb{C} \text{ ist Eigenwert von } A\}$ bezeichnet den Spektralradius von A).

Da die Eigenwerte von $\mathrm{Id} - A$ gerade durch $1 - \lambda$ mit $\lambda \in \Sigma(A)$ gegeben sind, ist dies äquivalent zu der angegebenen Bedingung $\Sigma(A) \subset B_\Theta(1)$ für ein $\Theta < 1$. □

2.2 Vorkonditionierung

Offenbar funktioniert dieses einfache Verfahren nur für eine eingeschränkte Klasse von Matrizen A. Um das Verfahren zu erweitern, schreiben wir es in der Form

$$\Delta x^{(k)} := b - Ax^{(k)}, \qquad x^{(k+1)} := x^{(k)} + \Delta x^{(k)}.$$

Die Idee zur Erweiterung liegt nun darin, die Definition von $\Delta x^{(k)}$ durch Multiplikation mit einer invertierbaren Matrix $M \in \mathbb{R}^{n\times n}$ zu verallgemeinern. Dies führt auf das Verfahren

$$M\Delta x^{(k)} = b - Ax^{(k)}, \qquad x^{(k+1)} := x^{(k)} + \Delta x^{(k)}. \tag{2.1}$$

Die Matrix M heißt hierbei Vorkonditionierer und für $M = \mathrm{Id}$ erhalten wir die einfache Richardson-Iteration. Die Multiplikation von $\Delta x^{(k)}$ mit M ist äquivalent zur Multiplikation von A und b mit M^{-1}, d.h. formal skaliert der Präkonditionierer das Gleichungssystem lediglich, woraus sofort ersichtlich ist, dass sich die Lösung nicht ändert. Die linke Gleichung in (2.1) muss hierbei i.A. noch gelöst werden (wie man das macht, hängt von der Form von M ab), deswegen steht dort ein "=" und nicht ein ":=". Die Iteration (2.1) kann man alternativ als

$$x^{(k+1)} := x^{(k)} + M^{-1}(b - Ax^{(k)}) = (\mathrm{Id} - M^{-1}A)x^{(k)} = M^{-1}(M - A)x^{(k)} + M^{-1}b$$

schreiben, woran man noch einmal sieht, dass der Vektor $x^* = A^{-1}b$ — unabhängig von der Wahl von M — ein Fixpunkt ist.

Das folgende Korollar folgt völlig analog zu Korollar 2.2.

Korollar 2.3 Die Iteration (2.1) konvergiert gegen die Lösung des Gleichungssystems $Ax = b$, wenn $\Sigma(M^{-1}A) \subset B_\Theta(1)$ oder äquivalent $\rho(M^{-1}(M - A)) < \Theta$ für ein $\Theta < 1$ gilt.

Wir haben in der Einführung der Numerik bereits gesehen, dass der Banach'sche Fixpunktsatz, der hinter diesen Konvergenzresultaten steckt, auch Fehlerabschätzungen angibt, allerdings im Allgemeinen in einer unbekannten Norm (siehe Bemerkung 2.30 in der Einführung der Numerik). Mit den folgenden Überlegungen können wir eine konstruktive Variante dieser Aussage erhalten.

Für eine symmetrische und positiv definite (im Folgenden kurz "spd") Matrix $M \in \mathbb{R}^{n\times n}$ definieren wir dazu das (komplexe) Skalarprodukt

$$\langle x, y\rangle_M := x^H My.$$

Eine Matrix B heißt selbstadjungiert bzgl. M, falls

$$\langle Bx, y\rangle_M = \langle x, By\rangle_M \quad \forall\, x, y \in \mathbb{R}^n$$

gilt, was äquivalent zu $B^T M = MB$ ist. In diesem Fall sinde die Eigenwerte von B reell und es gibt eine M-orthogonale Basis von Eigenvektoren, d.h. $\langle v_i, v_j\rangle_M = 0$ für $i \neq j$. Für $V = (v_1, \dots, v_n)$ ist dies äquivalent zu $V^T MV = \operatorname{diag}(d_1, \dots, d_n)$ mit $d_i = \langle v_i, v_i\rangle_M^2$.

Für die Konvergenz in Korollar 2.3 sind die Eigenwerte von $M^{-1}A$ entscheidend. Diese sind durch die äquivalenten Gleichungen

$$M^{-1}Av = \lambda v \quad \Leftrightarrow \quad Av = \lambda Mv$$

charakterisiert, die als verallgemeinertes Eigenwertproblem bezeichnet werden. Dabei haben die Matrizen $M^{-1}A$ und AM^{-1} die gleichen Eigenwerte, weil sie wegen $AM^{-1} = M(M^{-1}A)M^{-1}$ ähnlich zueinander sind. Zudem ist ist $M^{-1}A$ selbstajungiert bzgl. M.

Analog können wir das reelle Skalarprodukt

$$\langle x, y\rangle_M := x^T My$$

mit zugehöriger Norm $\|x\|_M = \sqrt{\langle x, x\rangle_M}$ definieren. Für die obige Eigenwertbasis V und $\Lambda = \operatorname{diag}(\lambda_1, \dots, \lambda_n)$ gilt dann $V^T MV = D$ sowie

$$M^{-1}AV = V\Lambda \quad \Rightarrow \quad V^T AV = V^T MV\Lambda = D\Lambda.$$

Zudem sind die Einträge d_i von D positiv. Daraus folgt, dass wir eine positive Diagonalmatrix D und eine invertierbare Matrix V finden können mit

$$V^T MV = D \text{ und } V^T AV = D\Lambda.$$

Durch Skalierung der v_i können wir $D = \mathrm{Id}$ wählen. Dann gilt $\|x\|_M = \|V^{-1}x\|_2$ für das skalierte V. Falls A spd ist, können wir analog $\langle x, y\rangle_A$ und $\|x\|_A$ definieren. Skalieren wir dann die v_i so, dass $D = \Lambda^{-1}$ gilt, folgt $\|x\|_A = \|V^{-1}x\|_2$ für das skalierte V.

Damit erhalten wir das folgende Resultat.

Korollar 2.4 Falls A und M spd sind und $\Sigma(M^{-1}A) \in [1-\Theta, 1+\Theta]$ gilt für ein $\Theta < 1$, dann gelten für die Iteration (2.1) die Abschätzungen

$$\|x^{(k+1)} - x^*\|_M \leq \Theta \|x^{(k)} - x^*\|_M$$
$$\|x^{(k+1)} - x^*\|_A \leq \Theta \|x^{(k)} - x^*\|_A.$$

Beweis: Unter Verwendung der obigen Gleichungen gilt $M^{-1}(M-A) = \mathrm{Id} - M^{-1}A = \mathrm{Id} - V\Lambda V^{-1}$. Daraus folgt

$$\begin{aligned} \|M^{-1}(M-A)x\|_M &= \|x - V\Lambda V^{-1}x\|_M = \|V^{-1}(x - V\Lambda V^{-1}x)\|_2 \\ &= \|(\mathrm{Id} - \Lambda)V^{-1}x\|_2 \leq \underbrace{\|\mathrm{Id} - \Lambda\|_2}_{\leq \Theta} \|V^{-1}x\|_2 \leq \Theta \|x\|_M. \end{aligned}$$

Die gleiche Ungleichung beweist man analog für $\|M^{-1}(M-A)x\|_A$.

Aus diesen Ungleichungen folgt, dass $x \mapsto M^{-1}(M-A)$ eine Kontraktion bzgl. der angegebenen Normen mit Konstante Θ ist. Daraus folgt sofort die Behauptung. □

Das Resultat sagt, das die Kontraktionseigenschaft nicht nur für eine i.A. unbekannte Norm sondern für zwei bekannte (und berechenbare) Normen gilt.

2.3 Klassische Vorkonditionierer

Wir betrachten nun eine Reihe unterschiedlicher Vorkonditionierer.

$M = A$: In diesem Fall erhalten wir für beliebiges $x^{(0)}$ nach einem Schritt $x^{(1)} = x^*$ (bis auf Rundungsfehler). Die Wahl führt aber auf kein praktisches Verfahren, da wir zur Implementierung ja $M^{-1} = A^{-1}$ kennen müssten.

$M = \mathrm{diag}(a_{11}, \ldots, a_{nn})$: In diesem Fall stimmt die Iterationsvorschrift $x^{(k+1)} = M^{-1}(M - A)x^{(k)} + M^{-1}b$ gerade mit dem **Jacobi-Verfahren** aus Abschnitt 2.9 der Einführung in die Numerik überein.

$M =$ "unterer Dreiecksanteil von A": Damit erhalten wir gerade das **Gauß-Seidel-Verfahren** aus Abschnitt 2.9 der Einführung in die Numerik.

Die Konvergenz dieser Verfahren hatten wir bereits in der Einführung in die Numerik untersucht, weswegen wir dies hier nicht wiederholen.

Man kann diese klassischen Verfahren modifizieren, indem man Blöcke entlang der Diagonalen oder ganze Nebendiagonalen zu M hinzufügt; so erhält man Block- oder Band-Jacobi bzw. -Gauß-Seidel-Verfahren.

Für symmetrische Matrizen A ist es naheliegend, auch den Vorkonditionieren M im Gauß-Seidel-Verfahren symmetrisch zu wählen. Dazu wählt man D als den Diagonalanteil von A und L als den unteren Dreiecksanteil von A ohne die Diagonale und setzt

$$M = (L + D)D^{-1}(L + D)^T.$$

Dieses **symmetrische Gauß-Seidel-Verfahren** hat die folgende Konvergenzeigenschaft.

Satz 2.5 Sei A spd. Dann gilt $\Sigma(M^{-1}A) \in (0,1)$, womit die symmetrische Gauß-Seidel-Iteration konvergiert.

Beweis: Es gilt

$$M = D + L + L^T + LD^{-1}L^T = A + LD^{-1}L^T.$$

Die Eigenwerte λ von $M^{-1}A$ erfüllen die Gleichung

$$Av = \lambda Mv = \lambda(A + LD^{-1}L^T)v.$$

Da beide Matrizen spd sind, sind alle Eigenwerte reell. Wenn wir die Gleichung von links mit v^T multiplizieren erhalten wir

$$\lambda = \frac{v^T Av}{v^T Av + v^T LD^{-1}L^T c} \in (0,1),$$

weil A und $LD^{-1}L^T$ spd und damit alle Summanden positiv sind. □

Weitere Varianten zur Wahl von M sind unvollständige LR- oder Choleski-Zerlegungen.

2.4 Relaxation

Die Relaxation oder Dämpfung ist uns ebenfalls bereits aus der Einführung in die Numerik bekannt. Die Idee besteht darin, die Iteration (2.1) durch

$$M\Delta x^{(k)} := b - Ax^{(k)}, \qquad x^{(k+1)} := x^{(k)} + \omega\Delta x^{(k)}. \tag{2.2}$$

zu ersetzen, wobei $\omega > 0$ der Relaxations- oder Dämpfungsparanmeter ist. Dies kann man alternativ durch Ersetzen von M durch $\omega^{-1}M$ erreichen, weswegen Relaxation nur eine andere Art ist, den Vorkonditionierer M zu wählen. Dies können wir für die Konvergenzanalyse ausnutzen.

Eine geschickte Wahl von $\omega > 0$ gibt uns die Möglichkeit, Konvergenz zu erhalten, auch wenn die Bedingung an $\Sigma(M^{-1}A)$ aus Korollar 2.3 nicht erfüllt ist. Nehmen wir an, dass $\Sigma(M^{-1}A) \subset B_K(c)$ liegt für $c \in \mathbb{R}$ und $R < |c|$. Dann gilt für $\omega = 1/c$ die Inklusion

$$\Sigma(\omega M^{-1}A) = \frac{1}{c}\Sigma(M^{-1}A) \subset B_{R/c}(1) \subset B_\Theta(1)$$

für $\Theta = R/c < 1$.

Für den Fall, dass $\Sigma(\omega M^{-1}A)$ reell ist, können wir dies noch verfeinern. Nehmen wir an, dass $\Sigma(M^{-1}A) \subset [\gamma, \Gamma]$ gilt für reelle Zahlen $0 < \gamma \leq \Gamma$. Dann gilt $\Sigma(M^{-1}A) \subset B_R(c)$ für $R = (\Gamma - \gamma)/2$ und $c = (\gamma + \Gamma)/2$. Für

$$\omega = \frac{1}{c} = \frac{2}{\gamma + \Gamma}$$

ergibt sich so

$$\Sigma(\omega M^{-1}A) \subset [1 - \Theta, 1 + \Theta], \qquad \Theta = \frac{\Gamma - \gamma}{\gamma + \Gamma} = \frac{\kappa_{A,M} - 1}{\kappa_{A,M} + 1},$$

wobei

$$\kappa_{A,M} = \Gamma/\gamma \tag{2.3}$$

die Kondition von A relativ zu M genannt wird. Die Werte Γ und γ sind dabei gerade der maximale bzw. minimale Eigenwert des verallgemeinerten Eigenwertproblems

$$Av = \lambda Mv.$$

Für den Fall, dass M und A spd sind, ergibt sich eine besonders schöne Aussage.

Satz 2.6 Falls M und A spd sind, gibt es $0 < \gamma \leq \Gamma$ mit $\Sigma(M^{-1}A) \subset [\gamma, \Gamma]$ und für die Wahl von ω wie oben gilt

$$\|x^{(k+1)} - x^*\|_M \leq \Theta \|x^{(k)} - x^*\|_M$$
$$\|x^{(k+1)} - x^*\|_A \leq \Theta \|x^{(k)} - x^*\|_A.$$

mit $\Theta = (\kappa_{A,M} - 1)/(\kappa_{A,M} + 1)$ und der in (2.3) definierten Kondition $\kappa_{A,M}$ von A relativ zu M.

Beweis: Sei $\lambda \in \Sigma(M^{-1}A)$ mit Eigenvektor v. Dann sind λ und v reell und es gilt

$$\lambda = \frac{v^T Av}{v^T Mv} > 0.$$

Damit ist auch γ als kleinster Eigenwert positiv. Die Behauptung folgt nun aus Korollar 2.4. □

2.5 Das vorkonditionierte Gradientenverfahren

Eine naheliegende Idee ist nun, den Ralaxationsparameter ω in jedem Iterationsschritt anzupassen, also im k-ten Schritt ω_k zu verwenden. Eine Möglichkeit, dies zu machen, bietet Korollar 2.4. Aus dem Korollar wissen wir, dass der Ausdruck $\|x^{(k)} - x^*\|$ monoton fällt. Die Idee ist nun, ω_k so zu wählen, dass dieser Ausdruck möglichst klein wird. In Anlehnung an einen Begriff aus der Optimierung wird der zu minimierende Ausdruck als Merit-Funktion definiert. Hier lautet die Merit-Funktion

$$m(x) = \frac{1}{2}\|x - x^*\|_A^2$$

(das Quadrat und der Faktor $1/2$ ändern an der Minimalstelle des Problems nichts, erleichtern aber die Rechnungen).

Ein Problem bei diesem Ansatz ist, dass x^* bekannt sein muss, damit m ausgewertet werden kann. Dieses lässt sich jedoch lösen, denn

$$m(x) = \frac{1}{2}(x - x^*)^T A(x - x^*) = \frac{1}{2}x^T Ax - (Ax^*)^T x + \frac{1}{2}(x^*)^T Ax^* = \frac{1}{2}x^T Ax + b^T x + c$$

mit $c = \frac{1}{2}(x^*)^T Ax^*$. Da die Minimalstelle des Problems unabhängig von c ist, kann diese Term, der als einziger den unbekannten Wert x^* enthält, einfach weggelassen werden. Statt m können wir also die Funktion

$$q(x) = \frac{1}{2}x^T Ax - b^T x$$

minimieren. Die notwendige Optimalitätsbedingung dafür lautet

$$0 = Dq(x) = (Ax - b)^T,$$

und liefert stets einen Minimierer, weil $D^2q(x) = A$ positiv definit ist. Tatsächlich kann auch die Vorschrift zur Berechnung von $\Delta x^{(k)}$ als notwendige Optimalitätbedingung interpretiert werden. Der Vektor $\Delta x^{(k)}$ minimiert gerade den Ausdruck

$$g_k(\Delta x) = \frac{1}{2}\Delta x^T M \Delta x + Dq(x^{(k)})\Delta x.$$

Für gegebenes $\Delta x^{(k)}$ ist nun ω_k so zu bestimmen, dass

$$\omega \mapsto q(x^{(k)} + \omega_k \Delta x^{(k)})$$

minimiert wird. Dies führt mit der Kettenregel auf die Bedingung

$$\begin{aligned} 0 &= Dq(x^{(k)} + \omega_k \Delta x^{(k)})\Delta x^{(k)} = \Big(A(x^{(k)} + \omega_k \Delta x^{(k)}) - b\Big)^T \Delta x^{(k)} \\ &= (A(x^{(k)}) - b)^T \Delta x^{(k)} + \omega_k (\Delta x^{(k)})^T A \Delta x^{(k)}, \end{aligned}$$

die durch die Wahl

$$\omega_k = -\frac{(A(x^{(k)}) - b)^T \Delta x^{(k)}}{(\Delta x^{(k)})^T A \Delta x^{(k)}} = \frac{(\Delta x^{(k)})^T M \Delta x^{(k)}}{(\Delta x^{(k)})^T A \Delta x^{(k)}}$$

erfüllt wird, wobei die zweite Gleichung aus der Iterationsvorschrift (2.1) folgt. Die Konvergenz des Verfahrens stellt nun der folgende Satz sicher.

Satz 2.7 Falls A und M spd sind, so gilt für das Gradientenverfahren

$$\|x^{(k+1)} - x^*\|_A \leq \Theta \|x^{(k)} - x^*\|$$

mit $\Theta = (\kappa_{A,M} - 1)/(\kappa_{A,M} + 1)$ und der in (2.3) definierten Kondition $\kappa_{A,M}$ von A relativ zu M.

Beweis: Für gegebenes $x^{(k)}$ wird $\Delta x^{(k)}$ genau wie bei dem in Satz 2.6 betrachteten Verfahren gewählt. Sei $\hat{x}^{(k+1)} = x^{(k)} + \omega \Delta x^{(k)}$ die nächste Iterierte aus dem dortigen Verfahren

Da die Wahl von ω_k die Merit-Funktion m in $x^{(k+1)} = x^{(k)} + \omega_k \Delta x^{(k)}$ minimiert, folgt

$$m(x^{(k+1)}) \leq m(\hat{x}^{(k+1)})$$

und damit

$$\|x^{(k+1)} - x^*\|_A \leq \|\hat{x}^{(k+1)} - x^*\|_A \leq \Theta \|x^{(k)} - x^*\|_A,$$

also die Behauptung. □

2.6 Krylovräume

Um zu besseren Verfahren als den bisher genannten zu kommen, ist es sinnvoll, sich die Unterräume anzuschauen, in denen die Näherungen $x^{(1)}, \ldots, x^{(k)}$ liegen. Dazu definieren wir die sogenannten Krylovräume.

Definition 2.8 Für $B \in \mathbb{R}^{n\times n}$ und $v \in \mathbb{R}^n$ bezeichnet

$$\mathcal{K}_k(B, v) := \operatorname{span}\{v, Bv, B^2v, \ldots, B^{k-1}v\} = \operatorname{span}\{B^i v \,|\, i = 0, \ldots, k-1\}$$

den Krylovraum der Ordnung $k \in \mathbb{N}$ zu B und v. □

Der Zusammenhang dieser Räume mit den bisherigen Verfahren lässt sich z.B. an Hand der relaxierten Iteration

$$x^{(k+1)} := x^{(k)} + \omega_k M^{-1}(b - Ax^{(k)})$$

erläutern: Für $x^{(k)} - x^*$ gilt dabei

$$x^{(k+1)} - x^* = x^{(k)} - x^* + \omega_k M^{-1}A(x^* - x^{(k)}) = (\mathrm{Id} - \omega_k M^{-1}A)(x^{(k)} - x^*),$$

woraus per Induktion und mit Ausmultiplizieren

$$x^{(k+1)} - x^* = \prod_{i=0}^{k+1}(\mathrm{Id} - \omega_k M^{-1}A)(x^{(0)} - x^*) = \left(\mathrm{Id} + \sum_{i=1}^{k+1} c_i (M^{-1}A)^i\right)(x^{(0)} - x^*)$$

folgt, wobei die c_i von den ω_k abhängen. Definiert man das Polynom $Q(\lambda) = \sum_{i=0}^{k+1} c_i \lambda^i$ mit $c_0 = 1$ (also mit $Q(0) = 1$), so kann man dies kurz schreiben als

$$x^{(k+1)} - x^* = Q(M^{-1}A)(x^{(0)} - x^*). \tag{2.4}$$

Andererseits gilt wegen

$$\left(\mathrm{Id} + \sum_{i=1}^{k} c_i (M^{-1}A)^i\right)(x^{(0)} - x^*) = x^{(0)} - x^* + \sum_{i=1}^{k+1} c_i (M^{-1}A)^i (x^{(0)} - x^*)$$

die Gleichung

$$x^{(k+1)} = x^{(0)} + \sum_{i=1}^{k} c_i (M^{-1}A)^i (x^{(0)} - x^*) = x^{(0)} - \sum_{i=1}^{k+1} c_i (M^{-1}A)^{i-1} \underbrace{M^{-1}(b - Ax^{(0)})}_{=\Delta x^{(0)}}. \tag{2.5}$$

Definiert man nun den Krylovraum

$$\begin{aligned} V_{k+1} := \mathcal{K}_{k+1}(M^{-1}A, \Delta x_0) &= \operatorname{span}\{(M^{-1}A)^i \Delta x^{(0)} \,|\, i = 0, \ldots, k\} \\ &= \operatorname{span}\{\Delta x^{(i)} \,|\, i = 0, \ldots, k\} \\ &= \operatorname{span}\{(M^{-1}A)^i (x^{(0)} - x^*) \,|\, i = 1, \ldots, k+1\}, \end{aligned}$$

so gilt

$$x^{(k+1)} \in x^{(0)} + V_{k+1}.$$

Die Idee ist nun, die beste Approximationen der exakten Lösung x^* in diesem Raum zu finden, also die Aufgabe

$$\text{minimiere } \|x - x^*\| \text{ über } x \in x^{(0)} + V_{k+1} \tag{2.6}$$

zu lösen. Nach den bisherigen Überlegungen ist dies äquivalent dazu, die Koeffizienten c_i bzw. das Polynom Q optimal zu wählen. Schreibt man die Menge der dafür in Frage kommenden Polynome als

$$\mathcal{P}_{k+1}^1 := \{Q \in \mathcal{P}_{k+1} \,|\, Q(0) = 1\},$$

so ist dies gerade die Aufgabe

$$\text{minimiere } \|Q(M^{-1}A)(x^{(0)} - x^*)\| \text{ über } Q \in \mathcal{P}_{k+1}^1. \tag{2.7}$$

Eine exakte Lösung dieses Minimierungsproblems ist ohne Kenntnis von x^* i.A. nicht möglich. Es gibt aber verschiedene Ansätze, das Problem näherungsweise zu lösen, die sich z.B. in der Wahl der Norm in (2.6) bzw. (2.7) unterscheiden. Einige davon werden wir im Folgenden besprechen.

2.7 Tschebyscheff Semi-Iteration

Zur Motivation dieses Ansatzes nehmen wir an, dass eine Norm $\|\cdot\|_m$ existiert, für die

$$\rho(M^{-1}A) = \|M^{-1}A\|_m$$

gilt. Betrachtet man den Beweis von Lemma 2.29 aus der Einführung in die Numerik, so sieht man, dass dies gilt, wenn $M^{-1}A$ diagonalisierbar ist. Dann gilt für $Q \in \mathbb{P}_k^1$ die Gleichung

$$\rho(Q(M^{-1}A)) = Q(\rho(M^{-1}A)) = Q(\|M^{-1}A\|_m) = \|Q(M^{-1}A)\|_m.$$

und damit

$$\begin{aligned} \|x^{(k+1)} - x^*\|_m &= \|Q(M^{-1}A)(x^{(0)} - x^*)\|_m \le \|Q(M^{-1}A)\|_m \|x^{(0)} - x^*\|_m \\ &= \rho(Q(M^{-1}A))\|x^{(0)} - x^*\|_m. \end{aligned}$$

Es ist also sinnvoll, zur näherungsweisen Lösung von (2.7) den Spektralradius $\rho(Q(M^{-1}A))$ zu minimieren, also das Problem

$$\min_{Q \in \mathcal{P}_{k+1}^1} \rho(Q(M^{-1}A)) = \min_{Q \in \mathcal{P}_{k+1}^1} \max_{\lambda \in \Sigma(M^{-1}A)} |Q(\lambda)|$$

zu lösen.

Auch dieses Problem können wir i.A. nicht exakt lösen, denn dafür müssten wir das Spektrum $\Sigma(M^{-1}A)$ kennen, dessen Berechnung selbst wieder aufwändig ist. Wir beschränken

uns daher auf den Fall, dass $\Sigma(M^{-1}A)$ reell ist und wir $0 < \gamma \leq \Gamma$ kennen, so dass $\Sigma(M^{-1}A) \subseteq [\gamma, \Gamma]$ gilt. Dann lautet die Minimierungsaufgabe

$$\min_{Q \in \mathcal{P}_{k+1}^1} \max_{\lambda \in [\gamma,\Gamma]} |Q(\lambda)|. \tag{2.8}$$

Solch eine ähnliche Aufgabe haben wir bereits in der Einführung in die Numerik behandelt, nämlich in Satz 3.26,. Dort wurde gezeigt, dass das normierte Tschebyscheff-Polynom $T_{k+1}/2^k$ die Aufgabe

$$\min_{\substack{Q \in \mathcal{P}_{k+1} \\ Q \text{ normiert}}} \|Q\|_\infty = \min_{\substack{Q \in \mathcal{P}_{k+1} \\ Q \text{ normiert}}} \max_{x \in [-1,1]} |Q(x)|$$

löst. Wegen $T_{k+1}(x) = \cos((k+1)\arccos(x))$ für $x \in [-1,1]$ folgt $|T_{k+1}(x)| \leq 1$ für $x \in [-1,1]$. Zudem erfüllen die Tschebyscheff-Polynome die Dreiterm-Rekursion

$$T_{k+1}(x) = 2xT_k(x) - T_{k-1}(x), \quad T_0(x) = 1,\ T_1(x) = x.$$

Um nun (2.8) zu lösen, müssen wir die T_k nur geeignet transformieren und den Wert in $x = 0$ auf 1 normieren. Dies führt auf

$$Q_k(\lambda) := \frac{T_k(t(\lambda))}{T_k(t(0))} \qquad \text{mit} \qquad t(\lambda) = \frac{(\Gamma + \gamma) - 2\lambda}{\Gamma - \gamma}.$$

Beachte, dass $t(\lambda) \in [-1,1]$ gilt für alle $\lambda \in [\gamma, \Gamma]$. Mit der bereits im letzten Abschnitt eingeführten Kondition $\kappa_{A,M} = \Gamma/\gamma$ ergibt sich

$$t_0 := t(0) = \frac{\Gamma + \gamma}{\Gamma - \gamma} = \frac{\kappa_{A,M} + 1}{\kappa_{A,M} - 1} > 1.$$

Mit $\tau_k := T_k(t_0)$ erhalten wir

$$\rho(Q_k(M^{-1}A)) \leq \max_{\lambda \in [\gamma,\Gamma]} |Q_k(\lambda)| = \frac{1}{|\tau_k|} \max_{\lambda \in [\gamma,\Gamma]} |T_k(\lambda)| = \frac{1}{|\tau_k|}. \tag{2.9}$$

Satz 2.9 Das Polynom Q_k erfüllt die Abschätzung

$$\rho(Q_k(M^{-1}A)) \leq 2\left(\left(\frac{\sqrt{\kappa_{A,M}} - 1}{\sqrt{\kappa_{A,M}} + 1}\right)^k + \left(\frac{\sqrt{\kappa_{A,M}} + 1}{\sqrt{\kappa_{A,M}} - 1}\right)^k\right)^{-1} \leq 2\left(\frac{\sqrt{\kappa_{A,M}} - 1}{\sqrt{\kappa_{A,M}} + 1}\right)^k$$

Beweis: Mit Hilfe der Dreiterm-Rekursion prüft man nach, dass die Gleichung

$$T_k(x) = \frac{1}{2}\left((x + \sqrt{x^2 - 1})^k + (x - \sqrt{x^2 - 1})^k\right).$$

gilt. Für $t_0 = (\kappa_{A,M} - 1)/(\kappa_{A,M} + 1)$ gilt dann

$$t_0 \pm \sqrt{t_0^2 - 1} = \left(\frac{\sqrt{\kappa_{A,M}} + 1}{\sqrt{\kappa_{A,M}} - 1}\right)^{\pm 1}$$

und damit

$$|\tau_k| = T_k(t_0) = \frac{1}{2}\left(\left(\frac{\sqrt{\kappa_{A,M}} - 1}{\sqrt{\kappa_{A,M}} + 1}\right)^k + \left(\frac{\sqrt{\kappa_{A,M}} + 1}{\sqrt{\kappa_{A,M}} - 1}\right)^k\right).$$

Einsetzen in (2.9) liefert dann erste Ungleichung. Die zweite folgt durch eine einfache Rechnung. □

Bemerkung 2.10 Alternativ kann man $T_k(x)$ für $x \geq 1$ abschätzen durch

$$T_k(x) = \frac{1}{2}\left((x+\sqrt{x^2-1})^k + (x-\sqrt{x^2-1})^k\right) \geq x^k.$$

Dies folgt mit $a = \sqrt{x^2-1} \geq 0$ aus

$$(x+a)^k + (x-a)^k = \sum_{j=0}^{k}\binom{k}{j}x^{k-j}a^j + \sum_{j=0}^{k}\binom{k}{j}x^{k-j}(-a)^j = 2x^k + \sum_{j=1}^{k}\underbrace{\binom{k}{j}x^{k-j}(a^j \pm a^j)}_{\geq 0}.$$

Dadurch erhält man die Abschätzung

$$\rho(Q_k(M^{-1}A)) \leq \frac{1}{t_0^k} = \left(\frac{\kappa_{A,M}-1}{\kappa_{A,M}+1}\right)^k,$$

welche zeigt, dass das Tschebyscheff-Verfahren mindestens so schnell wie die Richardson-Iteration konvergiert. □

Um das Verfahren zu implementieren, müssen wir Q_k berechnen. Mit $\tau_k = T_k(t_0)$ ergeben sich aus der Dreiterm-Rekursion für T_k die folgenden Dreiterm-Rekursionen für Q_k und τ_k:

$$\begin{aligned} Q_{k+1}(\lambda) &= 2t(\lambda)\frac{\tau_k}{\tau_{k+1}}Q_k(\lambda) - \frac{\tau_{k-1}}{\tau_{k+1}}Q_{k-1}(\lambda) \\ \tau_{k+1} &= 2t_0\tau_k - \tau_{k-1}. \end{aligned}$$

Mit

$$\rho_k = 2t_0\frac{\tau_k}{\tau_{k+1}} \Rightarrow (1-\rho_k) = -\frac{\tau_{k-1}}{\tau_{k+1}} \text{ und } \omega = \frac{2}{\Gamma+\gamma} \Rightarrow \frac{t(\lambda)}{t_0} = -\omega\lambda + 1$$

vereinfacht sich die Rekursion für Q_k zu

$$Q_{k+1}(\lambda) = \rho_k(1-\omega\lambda)Q_k(\lambda) + (1-\rho_k)Q_{k-1}(\lambda)$$

und die Startwerte lauten mit $Q_0(\lambda) = 1$ und $Q_1(\lambda) = 1-\omega\lambda$. Aus $(\tau_{k+1}+\tau_{k-1})/\tau_k = 2t_0$ folgt $2t_0/\rho_k + \rho_{k-1}/(2t_0) = 2t_0$. Daraus erhalten wir für ρ_k die Rekursionsformel

$$\rho_{k+1} = \frac{4t_0^2}{4t_0^2 - \rho_k}$$

mit Startwert

$$\rho_0 = 2t_0\frac{\tau_0}{\tau_1} = 2t_0\frac{T_0(t_0)}{T_1(t_1)} = 2t_0\frac{1}{t_0} = 2.$$

Die Iterationsvorschrift für die $x^{(k)}$ ergibt sich nun aus der Formel (2.4)

$$x^{(k+1)} - x^* = Q_{k+1}(M^{-1}A)(x^{(0)} - x^*).$$

Daraus folgt

$$\begin{aligned} x^{(k+1)} &= Q_{k+1}(M^{-1}A)(x^{(0)} - x^*) + x^* \\ &= \rho_k(\mathrm{Id} - \omega M^{-1}A)(x^{(k)} - x^*) + (1-\rho_k)(x^{(k-1)} - x^*) + x^* \\ &= \rho_k(x^{(k)} - \omega M^{-1}A(x^{(k)} - x^*)) + (1-\rho_k)x^{(k-1)} \\ &= \rho_k(x^{(k)} - \omega \underbrace{M^{-1}(Ax^{(k)} - b)}_{=-\Delta x^{(k)}}) + (1-\rho_k)x^{(k-1)}. \end{aligned}$$

In der üblichen Form aufgeschrieben, ergibt sich also der Rekursionsschritt

$$M\Delta x^{(k)} = b - Ax^{(k)}, \qquad x^{(k+1)} := \rho_k(x^{(k)} + \omega\Delta x^{(k)}) + (1-\rho_k)x^{(k-1)}.$$

Beachte, dass in jeder Iteration zwei Vorgängerwerte $x^{(k)}$ und $x^{(k-1)}$ benötigt werden. Daher werden auch zwei Startwerte benötigt, um die Rekursion starten zu können. Dabei wird $x^{(0)}$ frei gewählt und $x^{(1)}$ ist gemäß (2.5) gegeben durch

$$x^{(1)} := x^{(0)} - c_1(M^{-1}A)^0\Delta x^{(0)} = x^{(0)} + \omega\Delta x^{(0)},$$

weil $Q_1(\lambda) = 1 - \omega\lambda$ und damit $c_1 = -\omega$ ist.

Der vollständige Algorithmus sieht damit wie folgt aus.

Algorithmus 2.11 (Tschebyscheff Semi-Iteration)
Eingabe: $x^{(0)} \in \mathbb{R}^n$, $\Gamma > \gamma > 0$, $\varepsilon > 0$

(0) Setze $\theta := 4t_0^2 = 4\left(\frac{\Gamma+\gamma}{\Gamma-\gamma}\right)^2$, $\omega := \frac{2}{\Gamma+\gamma}$, $\rho_0 := 2$

(1) Löse $M\Delta x^{(0)} = b - Ax^{(0)}$ und setze $x^{(1)} := x^{(0)} + \omega\Delta x^{(0)}$; setze $k := 1$

(2) Setze $\rho_k := \frac{\theta}{\theta - \rho_{k-1}}$, löse $M\Delta x^{(k)} = b - Ax^{(k)}$
und setze $x^{(k+1)} := \rho_k(x^{(k)} + \omega\Delta x^{(k)}) + (1-\rho_k)x^{(k-1)}$

(3) Falls $\|\Delta x^{(k)}\| < \varepsilon$, Ende; sonst setze $k := k+1$ und gehe zu (2)

Ausgabe: $x^{(k+1)} \approx x^*$ □

Die Konvergenzeigenschaften des Algorithmus klärt der folgende Satz. Darin ist $\|\cdot\|$ eine beliebige Norm, also nicht unbedingt die oben betrachtete Norm $\|\cdot\|_m$.

Satz 2.12 Sei $\Sigma(M^{-1}A) \subset [\gamma, \Gamma]$ mit $\gamma > 0$. Dann konvergiert die Tschebyscheff Semi-Iteration mit der Abschätzung

$$\|x^{(k)} - x^*\| \le C\Theta_k\|x^{(0)} - x^*\|$$

mit $\Theta_k = 2\left(\frac{\sqrt{\kappa_{A,M}}-1}{\sqrt{\kappa_{A,M}}+1}\right)^k$. Falls M und A spd sind, gilt zudem

$$\begin{aligned} \|x^{(k+1)} - x^*\|_M &\le \Theta_k\|x^{(0)} - x^*\|_M \\ \|x^{(k+1)} - x^*\|_A &\le \Theta_k\|x^{(0)} - x^*\|_A. \end{aligned}$$

Bemerkung 2.13 Wir haben mit der Aufgabenstellung begonnen, das beste $x^{(k+1)}$ in $x^{(0)} + V_{k+1}$ zu finden. Dieses ist i.A. eine Linearkombination aus allen Vektoren, die V_{k+1} aufspannen. Da dabei alle $x^{(0)}, \dots, x^{(k)}$ eingehen, wäre es prinzipiell möglich, dass alle diese Werte in die Berechnung von $x^{(k+1)}$ eingehen. Tatsächlich gehen aber — bedingt durch den Lösungsansatz — nur die letzten beiden Werte $x^{(k)}$ und $x^{(k-1)}$ ein. Das ist für die Implementierung sehr vorteilhaft, denn eine Iteration, die auf immer mehr Vorgängerwerten aufbaut, wird in jedem Schritt aufwändiger auszuwerten und benötigt für jede weitere Iteration mehr Speicher. □

2.8 Das CG-Verfahren

Wir wollen nun die Idee des Gradientenverfahrens aus Abschnitt 2.5 auf die Krylovräume übertragen. Dabei nehmen wir weiterhin an, dass A und M spd sind. Beim Gradientenverfahren hatten wir ausgenutzt, dass die Probleme

$$\min q(x) = \frac{1}{2}x^T Ax - b^T x \quad \text{und} \quad \min m(x) = \frac{1}{2}\|x - x^*\|_A$$

den gleichen Minimierer besitzen. Diese Tatsache wollen wir jetzt bei der Lösung von (2.6) ausnutzen.

Satz 2.14 Seien A und M spd und $x^{(k+1)}$ die Lösung des Problems

$$\min_{x \in x^{(0)} + V_{k+1}} q(x) \tag{2.10}$$

und $\hat{x}^{(k+1)}$ die Iterierte des Tschebyscheff-Verfahrens mit gleichem $x^{(0)}$ und V_{k+1} und exakten Parametern $\Gamma \geq \gamma > 0$. Dann gilt die Abschätzung

$$\|x^{(k+1)} - x^*\|_A \leq \|\hat{x}^{(k+1)} - x^*\|_A \leq 2\left(\frac{\sqrt{\kappa_{A,M}} - 1}{\sqrt{\kappa_{A,M}} + 1}\right)^k \|x^{(0)} - x^*\|_A$$

Beweis: Da die Minimierer von q und m übereinstimmen, gilt per Definition von $x^{(k+1)}$ die Ungleichung $\|x^{(k+1)} - x^*\|_A \leq \|\hat{x}^{(k+1)} - x^*\|_A$. Die Aussage folgt damit aus Satz 2.12. □

Bemerkung 2.15 Wir werden später in Bemerkung 2.18 noch beweisen, dass, wenn wir alle $x^{(k)}$ für $k \geq 1$ gemäß (2.10) wählen und keine Rundungsfehler auftreten, die Gleichheit $x^{(n)} = x^*$ gilt. □

Zunächst aber überlegen wir uns, wie wir das Problem (2.10) algorithmisch lösen. Dazu überlegen wir uns, wie wir aus dem Minimierer $x^{(k)}$ von q über $x^{(0)} + V_k$ den Minimierer $x^{(k+1)}$ von q über $x^{(0)} + V_{k+1}$ berechnen können.

Für den Minimierer gilt

$$Dq(x^{(k)})v = 0 \text{ für alle } v \in V_k,$$

was wir kurz als

$$Dq(x^{(k)})V_k = 0$$

schreiben. Für $d_k := Dq(x^{(k)})^T = Ax^{(k)} - b$ gilt dann

$$d_k^T V_k = 0.$$

Wir machen nun den Ansatz $x^{(k+1)} = x^{(k)} + \alpha p_k$, wobei wir $\alpha \in \mathbb{R}$ so wählen, dass $r(\alpha) := q(x^{(k)} + \alpha p_k)$ minimiert wird. Dann ist einerseits $V_{k+1} = \text{span}\{V_k, p_k\}$ (beachte, dass der Krylovraum V_{k+1} gerade von allen $\Delta x^{(i)}$, $i = 0, \ldots, k$ aufgespannt wird und $\Delta x^{(k)}$ ein Vielfaches von p_k ist) und andererseits

$$d_{k+1} := Ax^{(k+1)} - b = A(x^{(k)} + \alpha p_k) - b = d_k + \alpha A p_k.$$

Aus der Wahl von α folgt zudem

$$0 = r'(\alpha) = Dq(x^{(k)} + \alpha p_k)p_k = (A(x^{(k)} + \alpha p_k) - b)^T p_k = (d_k + \alpha A p_k)^T p_k = d_{k+1}^T p_k$$

und damit $\alpha = -d_k^T p_k / (p_k^T A p_k)$ sowie $d_{k+1}^T p_k = 0$.

Damit nun $x^{(k+1)}$ die Funktion q über $x^{(0)} + V_{k+1}$ minimiert, muss $d_{k+1}^T V_{k+1} = 0$ gelten. Dies ist aber äquivalent zu

$$\forall \mu \in \mathbb{R} : \; 0 = d_{k+1}^T(\mu p_k + V_k) = d_{k+1}^T V_k = (d_k + \alpha A p_k)^T V_k = \alpha p_k^T A V_k$$

Wir suchen also eine Richtung p_k, die die Bedingung $p_k^T A V_k = 0$ erfüllt. Dies schreiben wir kurz als

$$p_k \perp_A V_k$$

bezeichnen (gesprochen: p_k steht A-senkrecht auf V_k oder p_k ist A-orthgonal zu V_k).

Zur Konstruktion von p_k erinnern wir an das Gram-Schmidt Orthogonalisierungsverfahren: Seien $w_1, \ldots, w_k \in \mathbb{R}^n$. Dann erhalten wir A-orthogonale Vektoren $v_1, \ldots, v_k$ mit $\text{span}\{v_1, \ldots, v_k\} = \text{span}\{w_1, \ldots, w_k\}$ durch die Vorschrift

$$\begin{aligned}
v_1 &:= w_1 \\
v_2 &:= w_2 - \frac{v_1^T A w_2}{v_1^T A v_1} v_1 \\
v_3 &:= w_3 - \frac{v_1^T A w_3}{v_1^T A v_1} v_1 - \frac{v_2^T A w_3}{v_2^T A v_2} v_2 \\
&\vdots \\
v_k &:= w_k - \sum_{j=0}^{k-1} \frac{v_j^T A w_k}{v_j^T A v_j} v_j.
\end{aligned}$$

Die Idee ist nun, zunächst einmal eine Richtung g_k zu bestimmen, die sich leicht berechnen lässt und die bereits ein guter Kandidat für eine Abstiegsrichtung ist. Daraus berechnen wir dann mit Gram-Schmidt die eigentliche Richtung p_k mit $p_k \perp_A V_k$. Das folgende Lemma beschreibt die Details.

Lemma 2.16 Es sei $g_k \in \mathbb{R}^n$ das Minimum der Abbildung $v \mapsto \frac{1}{2} v^T M v + Dq(x^{(k)})v$, die durch $g_k = -M^{-1} d_k$ gegeben ist. Es sei $v_1, \ldots, v_{k+1}$ die aus einer Basis $w_1, \ldots, w_k$ von V_k und $w_{k+1} = g_k$ berechnete A-orthogonale Basis und $p_k := v_{k+1}$. Dann gilt $g_k \perp_M V_k$, $p_k \perp_A V_k$ und $V_{k+1} := \text{span}\{V_k, p_k\} = \text{span}\{V_k, g_k\}$. Falls $x^{(k)} \neq x^*$ ist zudem $g_k \neq 0$ und $p_k \neq 0$.

Beweis: Dass $g_k = M^{-1}d_k$ die angegebene Abbildung minimiert, folgt durch Ausrechnen und Nullsetzen der Ableitung. Aus $g_k^T M V_k = -d_k^T V_k = 0$ folgt $g_k \perp_M V_k$. Aus der Gram-Schmidt Konstruktion folgt: Falls $g_k \in V_k$ gilt, ist $p_k = 0$, andernfalls ist p_k von der Form $g_k + v$ mit $v \in V_k$. In beiden Fällen folgt $\operatorname{span}\{V_k, p_k\} = \operatorname{span}\{V_k, g_k\}$. Falls $x^{(k)} \neq x^*$, ist $d_k = Ax^{(k)} - b \neq 0$ und damit auch $g_k = -M^{-1}d_k \neq 0$. Da $g_k \perp_M V_k$ gilt und M spd ist, folgt aus $g_k \neq 0$ auch $g_k \notin V_k$. Dann liefert Gram-Schmidt aber auch ein $p_k \neq 0$. □

Die im Lemma beschriebene Konstruktion von p_k führt auf die folgende, erste Version der sogenannten CG-Verfahrens. "CG" steht dabei für "conjugate gradients", also "konjugierte Gradienten" und beschreibt, dass die Suchrichtungen p_k gerade aus Gradienten gewonnen werden, die A-senkrecht aufeinander stehen.

Algorithmus 2.17 (CG-Verfahren, vorläufige Version)

Eingabe: $A, M \in \mathbb{R}^{n\times n}$ spd, $x^{(0)} \in \mathbb{R}^n$, $\varepsilon > 0$

(0) Setze $d_0 := Dq(x^{(0)}) = Ax^{(0)} - b$, löse $Mg_0 = -d_0$, setze $p_0 := g_0$ und $k := 0$

(1) Setze $x^{(k+1)} := x^{(k)} + \alpha_k p_k$, $d_{k+1} := d_k + \alpha_k A p_k$ mit $\alpha_k = -\frac{d_k^T p_k}{p_k^T A p_k}$
Löse $Mg_{k+1} = -d_{k+1}$
Setze $p_{k+1} := g_{k+1} - \sum_{i=0}^{k} r_{i,k+1} p_i$ mit $r_{i,k+1} = \frac{g_{k+1}^T A p_i}{p_i^T A p_i}$

(2) Falls $\|x^{(k+1)} - x^{(k)}\| < \varepsilon$, Ende; sonst setze $k := k+1$ und gehe zu (1) □

Beachte, dass die $p_0, \ldots, p_k$ per Konstruktion stets eine A-Orthogonalbasis von V_{k+1} bilden, weswegen das Gram-Schmidt-Verfahren nur auf den neu berechneten Vektor g_{k+1} angewendet werden muss.

Bemerkung 2.18 Per Konstruktion findet das CG-Verfahren immer die beste Näherung $x^{(k)}$ von x^* (im Sinne der Norm $\|\cdot\|_A$) in V_k. Gilt also $x^* \in V_k$, so ist $x^{(k)} = x^*$. Gilt andererseits $x^* \notin V_k$, so gilt nach Lemma 2.16 auch $p_k \neq 0$ und damit $\dim V_{k+1} = \dim V_k + 1$. Da $\dim V_{k+1} \leq n$ gilt, muss also spätestens für $k = n$ der Vektor x^* in V_n liegen, woraus $x^{(k)} = x^*$ folgt.

Gilt nun $x^{(k)} = x^*$, so ist g_k gleich 0, woraus $x^{(k+1)} = x^*$ und damit per Induktion $x^{(n)} = x^*$ folgt. Da $x^{(k)} = x^*$ für irgendein $k \leq n$ gilt, folgt also stets $x^{(n)} = x^*$. Diese exakte Gleichheit gilt aber selbstverständlich nur, wenn keine Rundungsfehler auftreten. □

Der große Nachteil dieser Variante des CG-Verfahrens ist, dass alle $p_0, \ldots, p_k$ zur Berechnung von p_{k+1} gespeichert werden müssen — im Gegensatz zur Tschebyscheff Semi-Iteration. Wir können dieses Problem aber beheben. Dazu fassen wir zunächst noch einmal die wichtigsten Eigenschaften der bisherigen Konstruktion zusammen. Es gilt

$$V_k = \operatorname{span}\{p_i \,|\, i = 0, \ldots, k-1\} = \operatorname{span}\{g_i \,|\, i = 0, \ldots, k-1\}$$

sowie

$$g_k \perp_M V_k \quad \text{und} \quad p_k \perp_A V_k.$$

Die Vereinfachung von Algorithmus 2.17 beruht nun auf dem folgenden Lemma.

Lemma 2.19 Im CG-Verfahren gilt $g_{k+1} \perp_A V_k$ und somit $r_{i,k+1} = 0$ für $i = 0, \ldots, k-1$. Der letzte Schritt des CG-Verfahrens vereinfacht sich also zu $p_{k+1} := g_{k+1} - r_{k,k+1} p_k$. Dabei gilt $r_{k,k+1} =: -\beta_{k+1} = -d_{k+1}^T g_{k+1} / (d_k^T g_k)$.

Beweis: Die erste Aussage folgt aus

$$g_{k+1}^T A p_i = g_{k+1}^T \alpha_i^{-1} (d_{i+1} - d_i) = \alpha_i^{-1} g_{k+1}^T M (g_i - g_{i+1}) = 0,$$

was für alle $i = 0, \ldots, k-1$ gilt. Wegen $p_k - g_k \in V_k$ und $d_k^T V_k = 0$ gilt nun $d_k^T p_k = d_k^T g_k$. Zudem gilt $g_{k+1}^T d_k = -g_{k+1} M g_k = 0$, weil $g_k \in V_{k+1}$ und $g_{k+1} \perp_M V_{k+1}$. Damit folgt

$$r_{k,k+1} = \frac{g_{k+1}^T A p_k}{p_k^T A p_k} = \frac{1}{\alpha_k} \frac{g_{k+1}(d_{k+1} - d_k)}{p_k^T A p_k} = -\frac{d_{k+1}^T g_{k+1}}{d_k^T p_k} = -\frac{d_{k+1}^T g_{k+1}}{d_k^T g_k}.$$

□

Dies führt auf die folgende vereinfachte (und in der Praxis übliche) Variante des CG-Algorithmus.

Algorithmus 2.20 (CG-Verfahren, endgültige Version)

Eingabe: $A, M \in \mathbb{R}^{n \times n}$ spd, $x^{(0)} \in \mathbb{R}^n$, $\varepsilon > 0$

(0) Setze $d_0 := Dq(x^{(0)}) = Ax^{(0)} - b$, löse $Mg_0 = -d_0$, setze $p_0 := g_0$ und $k := 0$

(1) Setze $x^{(k+1)} := x^{(k)} + \alpha_k p_k$, $d_{k+1} := d_k + \alpha_k A p_k$ mit $\alpha_k = -\frac{d_k^T p_k}{p_k^T A p_k}$
Löse $M g_{k+1} = -d_{k+1}$
Setze $p_{k+1} := g_{k+1} + \beta_{k+1} p_k$ mit $\beta_{k+1} = \frac{d_{k+1}^T g_{k+1}}{d_k^T g_k}$

(2) Falls $\|x^{(k+1)} - x^{(k)}\| < \varepsilon$, Ende; sonst setze $k := k+1$ und gehe zu (1) □

Bemerkung 2.21 (i) Schreibt man die Vektoren $p_0, \ldots, p_k$ und $g, \ldots, g_k$ in Matrizen G_{k+1} und P_{k+1}, so kann die letzte Zeile in Schritt (1) des CG-Verfahrens als $G_{k+1} = P_{k+1} R_{k+1}$ geschrieben werden, wobei R_{k+1} eine obere Dreiecksmatrix mit den Einträgen $r_{i,j}$ ist. Lemma 2.19 zeigt dann

$$R_{k+1} = \begin{pmatrix} 1 & -\beta_1 & & \\ & \ddots & \ddots & \\ & & \ddots & -\beta_{k+1} \\ & & & 1 \end{pmatrix}.$$

(ii) Lemma 2.19 gilt nicht mehr, wenn M von Schritt zu Schritt verändert wird. Will man variierende M verwenden, so muss man die aufwändigere erste Version des Algorithmus verwenden oder man verwendet die vereinfachte Variante mit der alten Formel für $\beta_k = r_{k,k+1} = \frac{g_{k+1}^T A p_k}{p_k^T A p_k}$. Dabei nimmt man aber einen Fehler in Kauf, der die Konvergenzgeschwindigkeit deutlich verringern kann. □

Für nicht-spd Matrizen gibt es eine Variante des CG-Verfahrens, das sogenannte CGN-Verfahren. Es beruht darauf, das CG-Verfahren auf die präkonditionierte Normalengleichung

$$A^T M^{-1} A x - A^T M^{-1} b = 0$$

anzuwenden. Dadurch löst man das Problem

$$\min_{x \in x^{(0)} + V_k} \|x - x^*\|^2_{A^T M^{-1} A} = \|A(x - x^*)\|^2_{M^{-1}} = \|Ax - b\|^2_{M^{-1}}$$

auf dem Krylovraum $V_k = \mathcal{K}_k(\widetilde{M}^{-1}\widetilde{A}, \widetilde{M}^{-1}\widetilde{A}(x - x^*))$, wobei $\widetilde{A} = A^T M^{-1} A$ und $\widetilde{M}$ der Präkonditionierer im CG-Verfahren, also eine weitere spd Matrix ist. Man minimiert also das sogenannte vorkonditionierte Residuum $\|Ax - b\|_{M^{-1}}$.

Dieses Verfahren ist sehr leicht zu implementieren, zudem ist die Konvergenzanalyse einfach: im Falle $M = \widetilde{M} = \mathrm{Id}$ ist $\widetilde{A} = A^T A$ und die Kondition (und damit die Konvergenzgeschwindigkeit) ist gegeben durch

$$\kappa_{M,\widetilde{A}} = \sqrt{\frac{\lambda_{\max}(A^T A)}{\lambda_{\min}(A^T A)}} = \frac{\sigma_1(A)}{\sigma_n(A)},$$

also durch das Verhältnis des größten zum kleinsten Singulärwert. Ein Nachteil des Verfahrens ist ist, dass $\widetilde{A}$ i.d.R. deutlich schlechtere Kondition als A hat (wie üblich, wenn man die Normalengleichung verwendet), wodurch das CGN-Verfahren dann nur langsam konvergiert.

Ein Vorteil bei allen bisher betrachteten Verfahren war, dass A nicht explizit bekannt sein muss; es genügt, eine Routine zur Berechnung von Ax für gegebenes x zu haben. Dies ist beim CGN-Verfahren nicht mehr so: hier muss man auch $A^T A$ oder $A^T M^{-1} A$ auswerten können, was schwierig sein kann, wenn A nicht explizit vorliegt. Dies ist ein weiterer Nachteil dieses Verfahrens.

2.9 Das GMRES-Verfahren

Das GMRES-Verfahren ist ein weiteres Verfahren für nicht-symmetrische Matrizen. Es beruht auf dem Arnoldi-Verfahren, das wir bereits in Abschnitt 1.4 kennengelernt haben und das Matrizen A in Hessenbergform $A^{(1)}$ transformiert. Dies kann zum Lösen von linearen Gleichungssystemen verwendet werden, indem die Matrix $A^{(1)}$ in einem nachfolgenden Schritt QR-zerlegt wird, was dann nur noch den Aufwand $O(n^2)$ besitzt. Der Aufwand wird weiter verringert, indem das Arnoldi-Verfahren nur unvollständig ausgeführt wird.

Um den Präkonditionierer M in die Rechnung einzubeziehen, stellen wir zunächst eine Variante des Arnoldi-Verfahrens vor, bei dem die Spalten q_j von Q nicht orthonormal sondern M-orthonormal sind. Für das resultierende $Q = (q_1, \ldots, q_n)$ mit $Q^T A Q = A^{(1)}$ gilt dann also nicht $Q^T Q = \mathrm{Id}$ sondern $Q^T M Q = \mathrm{Id}$.

Algorithmus 2.22 (Arnoldi-Verfahren mit Präkonditionierer M)

Eingabe: $A \in \mathbb{R}^{n \times n}$, $q_1 \in \mathbb{R}^n$ mit $\|q_1\|_M = 1$

(1) Für $j = 1, 2, \ldots, k-1$

(2) $w := Aq_j$
Löse $Mv = w$
für $i = 1, \ldots, j$
$b_{ij} := q_i^T w \;\; (= q_i^T M v)$
$v := v - b_{ij} q_i$
Ende der i-Schleife

(3) if $v \neq 0$
$b_{j+1,j} := \sqrt{v^T w} \;\; (= \sqrt{v^T M v})$
$q_{j+1} := v / b_{j+1,j}$

(4) else
Ende mit Fehlermeldung: Breakdown
end if

Ende der j-Schleife

Ausgabe: $B_k = (b_{ij})_{i,j=1,\ldots,k}$, $Q_k = (q_1, \ldots, q_k)$ □

Hierbei dient der Vektor w in der i-Schleife in (2) und in (3) dazu, Multiplikationen mit M zu vermeiden. Falls diese leicht implementiert werden können, kann w dort durch Mv ersetzt werden. Der Vektor w müsste in der i-Schleife eigentlich mittels $w := w - b_{ij} M q_i$ aktualisiert werden. Weil die q_i M-orthonormal sind und daher bei den Termen, in denen w auftritt, sowieso wegfallen würden, kann man dieses Update aber weglassen, sofern dies keine Probleme bei der numerischen Stabilität nach sich zieht.

Im Falle $k = n$ läuft der Algorithmus vollständig durch und wir erhalten $B_{n-1} = A^{(1)}$. Falls wir den Algorithmus nach $k - 1 < n - 1$ Schritten vorzeitig abbrechen, erhalten wir eine $k \times k$ Hessenbergmatrix B_k sowie die M-orthogonale Matrix Q_k mit

$$B_k = Q_k^T A Q_k.$$

Dabei bilden die Spalten von Q_k eine Basis des Krylovraums $V_k = \mathcal{K}_k(M^{-1}A, q_1)$.

Wie im CGN-Verfahren ist die Idee von GMRES nun die Minimierung des vorkonditionierten Residuums

$$\min_{x \in x^{(0)} + V_k} \|x - x^*\|^2_{A^T M^{-1} A} = \|A(x - x^*)\|^2_{M^{-1}} = \|Ax - b\|^2_{M^{-1}}, \tag{2.11}$$

wobei M spd und A invertierbar aber i.A. nicht symmetrisch ist. Wie im CG-Verfahren (und anders als im CGN-Verfahren) ist dabei

$$V_k = \mathcal{K}_k(M^{-1}A, v_0) \quad \text{mit} \quad v_0 = M^{-1}(b - Ax_0).$$

Satz 2.23 Sei $x^{(k)}$ der Minimierer von (2.11). Dann gilt für $v_k := M^{-1}(b - Ax^{(k)})$ die Abschätzung

$$\|v_k\|_M \leq \min_{p \in \mathcal{P}_k^1} \|p(M^{-1}A)\|_M \|v_0\|_M.$$

Falls $M^{-1}A$ diagonalisierbar ist mit $M^{-1}A = V\Lambda V^{-1}$, so gilt zudem

$$\|v_k\|_M \leq \min_{p\in\mathcal{P}_k^1} \max_{\lambda\in\Sigma(M^{-1}A)} |p(\lambda)|\kappa_M(V)\|v_0\|_M,$$

wobei $\kappa_M(V) := \|V\|_{2,M}\|V^{-1}\|_{M,2}$ und $\|V\|_{p,q} := \sup_{x\neq 0} \|Vx\|_q/\|x\|_p$.

Beweis: Die erste Abschätzung folgt sofort aus der Darstellung von $x^{(0)} + V_k$ mittels der Polynome $\mathcal{P}_k^1$ und der Tatsache, dass $x^{(k)}$ ein Minimierer ist. Die zweite Abschätzung folgt aus

$$\|p(M^{-1}A)\|_M = \|Vp(\Lambda)V^{-1}\|_M \leq \|V\|_{2,M}\|p(\Lambda)\|_2\|V^{-1}\|_{M,2} = \kappa_M(V) \max_{\lambda\in\Sigma(M^{-1}A)} |p(\lambda)|.$$

□

Falls V M-orthogonal ist, gilt $\|Vx\|_M^2 = \langle Vx, MVx\rangle = x^T \underbrace{V^TMV}_{=\mathrm{Id}} = x^Tx = \|x\|_2^2$ und analog $\|V^{-1}x\|_2^2 = \|x\|_M^2$. Daraus folgt $\kappa_M(V) = 1$.

Satz 2.23 sagt uns, wie gut der Minimierer $x^{(k)}$ das Gleichungssystem löst, d.h. wie schnell das Verfahren für wachsende k konvergiert. Dabei spielt im diagonalisierbaren Fall offensichtlich die Größe

$$m_k := \min_{p\in\mathcal{P}_k^1} \max_{\lambda\in\Sigma(M^{-1}A)} |p(\lambda)|$$

eine entscheidende Rolle. Wir betrachten drei Fälle, in denen diese abgeschätzt werden kann:

- Falls $\Sigma(M^{-1}A) \subset [\gamma, \Gamma]$ mit $\gamma > 0$, so erhalten wir, indem wir p als Tschebyscheff-Polynom wählen,
$$m_k \leq 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^k$$
mit $\kappa := \Gamma/\gamma$. Im Falle positiver Eigenwerte erhalten wir also eine Konvergenz ähnlich wie im CG-Verfahren, allerdings in einer anderen Norm.

- Falls $\Sigma(M^{-1}A) \subset B_r(c)$ mit $r < |c|$, so können wir $p(z) = ((c-z)/c)^k$ wählen und erhalten
$$m_k \leq \left(\frac{r}{|c|}\right)^k.$$
Die Konvergenz ist also schnell, wenn die Eigenwerte nahe beisammen und weit weg von der 0 liegen.

- Falls $\Sigma(M^{-1}A) \subset [\Gamma_-, \gamma_i] \cup [\gamma_+, \Gamma_+]$ mit $\gamma_- < 0 < \gamma^+$ und $\gamma_- - \Gamma_- = \Gamma_+ - \Gamma_+$, so können wir ein passendes $p(z)$ mit Hilfe von Tschebyscheff-Polynome definieren und erhalten
$$m_{2k} \leq 2\left(\frac{\kappa-1}{\kappa+1}\right)^k$$
mit $\kappa = \sqrt{\Gamma_-\Gamma_+/(\gamma_-\gamma_+)}$. Dies ist i.A. eine deutlich schlechtere Abschätzung als in den ersten beiden Fällen.

Wir müssen nun noch erklären, wie wir Algorithmus 2.22 nutzen können, um das Problem (2.11) zu lösen. Wie bereits erwähnt, bilden die Spalten von Q_k gerade die Basis von V_k und es gilt $Q_k^T M Q_k = \mathrm{Id}$ und $B_k = Q_k^T A Q_k$ ist eine $k \times k$ Hessenbergmatrix. Zudem ist $Q_k Q_k^T M$ gerade die M-orthogonale Projektion von $\mathbb{R}^n$ nach V_k. Wir definieren nun

$$\widetilde{B}_k := Q_{k+1}^T A Q_k \in \mathbb{R}^{(k+1)\times k},$$

was eine (nicht quadratische) Hessenbergmatrix ist. Für $x \in V_k$, $Q_k y = x - x^{(0)}$ und $v = M^{-1}(Ax - b) \in V_{k+1}$ gilt dann

$$\widetilde{B}_k y - Q_{k+1}^T(b - Ax^{(0)}) = Q_{k+1}^T(AQ_k y + Ax^{(0)} - b) = Q_{k+1}^T M M^{-1}(Ax - b) = Q_{k+1}^T M v.$$

Daraus folgt

$$\|\widetilde{B}_k y - Q_{k+1}^T(b - Ax^{(0)})\|_2^2 = \|Q_{k+1}^T M v\|_2^2 = v^T M Q_{k+1} Q_{k+1}^T M v = v^T M v = \|v\|_M^2. \quad (2.12)$$

Zudem gilt

$$Q_{k+1}^T(b - Ax^{(0)}) = Q_{k+1}^T M v_0 = Q_{k+1}^T M q_1 \|v_0\|_M = e_1 \|v_0\|_M$$

wobei $e_1 \in \mathbb{R}^{k+1}$ den ersten Einheitsvektor bezeichnet.

Insgesamt erhalten wir so

$$\min_{x \in x^{(0)} + V_k} \|M^{-1}(b - Ax)\|_M^2 \quad \Leftrightarrow \quad \min_{y \in \mathbb{R}^k} \|\widetilde{B}_k y - \|v_0\|_M e_1\|_2^2.$$

Das Problem auf der rechten Seite ist nun ein lineares Ausgleichsproblem mit Hessenbergmatrix, das mittels einer QR-Zerlegung $\widetilde{B}_k = U_k R_k$ mit Aufwand $O(k^2)$ gelöst werden kann.

Es geht aber noch effizienter: Wir setzen dazu

$$u^{(k)} := U_k^T \|v_0\|_M e_1 \quad \text{und} \quad \tilde{u}^{(k)} = (u_1^{(k)}, \ldots, u_k^{(k)})^T.$$

Dann ist die optimale Lösung und der zugehörige Zielfunktionswert gegeben durch

$$y_k := R_k^{-1} \tilde{u}^{(k)} \quad \text{und} \quad \|\widetilde{B}_k y_k - \|v_0\|_M e_1\|_2 = u_{k+1}^{(k)},$$

vgl. die Rechnungen vor Algorithmus 2.21 in der Einführung in die Numerik. Die Matrix $\widetilde{B}_{k+1}$ entsteht nun aus der Matrix $\widetilde{B}_k$ durch Anfügen einer weiteren Spalte b_{k+1}. Verwenden wir für die QR-Zerlegung Givens-Rotationen, so kann man diese Spalte transformieren, indem man alle für die QR-Zerlegung von $\widetilde{B}_k$ durchgeführten Givens-Rotationen auf b_{k+1} anwendet (was der Multiplikation mit U_k^T entspricht) und dann noch eine weitere Rotation G_{k+1} anwendet, um das Element auf der unteren Nebendiagonalen zu Null zu machen. Da diese die bereits bestehenden Spalten in R_k nicht verändert, muss nur die neue Spalte r_{k+1} auf diese Weise berechnet werden, was lediglich ein Aufwand der Ordnung $O(k)$ ist. In Formeln ausgedrückt gilt dann $U_{k+1} = (U_k, u^{(k+1)})$ und $R_{k+1} = (R_k, r_{k+1})$ mit

$$u^{(k+1)} = G_{k+1} \begin{pmatrix} u^{(k)} \\ 0 \end{pmatrix} \quad \text{und} \quad r_{k+1} = U_{k+1}^T b_{k+1}.$$

Da die Rotation G_{k+1} nur die Zeilen $k+1$ und $k+2$ verändert, gilt zudem $u_j^{(k+1)} = u_j^{(k)}$ für $j = 1, \ldots, k$. Diese wird üblicherweise nur einmal am Ende der Iteration berechnet. Abgebrochen wird der Algorithmus, wenn das Residuum

$$\|M^{-1}(b - Ax^{(k)})\|_M = \|\widetilde{B}_k y_k - \|v_0\|_M e_1\|_2 = u_{k+1}^{(k)}$$

hinreichend klein ist. Die Lösung x_k des ursprünglichen Problems ergibt sich dann als $x^{(k)} = x^{(0)} + Q_k y_k$.

Varianten von GMRES

Ein Nachteil von GMRES ist, dass die immer größer werdende Matrix $Q_k \in \mathbb{R}^{n\times k}$ gespeichert werden muss, wobei n in der Regel sehr groß ist. So bald diese Matrix zum Speichern zu groß wird, wird üblicherweise neu gestartet. Die Veriante des GMRES-Verfahrens, in der nach jeweils l Schritten mit neuem Startwert $x^{(0)} := x^{(l)}$ neu gestartet wird, wird mit GMRES(l) bezeichnet.

Eine weitere Möglichkeit, die Größe der Matrix Q_k zu beschränken, ist die Verwendung eines unvollständigen Arnoldi-Algorithmus, in dem die Schleife in (2) nur von $j - s + 1$ bis j für ein $s \in \mathbb{N}$ läuft. Dadurch kann der Algorithmus so modifiziert werden, dass nur die letzten s Spalten von Q_k gespeichert werden müssen. Der Preis, den man für diese Vereinfachung zahlt, ist, dass Q_k dann nicht mehr orthogonal ist und die Minimierungsprobleme in (2.11) nur noch näherungsweise übereinstimmen. Dies bringt einen Fehler in das Verfahren, der geeignet abgeschätzt werden muss, um einen akzeptablen Wert für s zu erhalten.

GMRES kann auch auf symmetrische Matrizen A angewendet werden. In diesem Fall kann das Arnoldi-Verfahren durch den folgenden Lanczos-Algorithmus ersetzt werden.

Algorithmus 2.24 (Lanczos-Verfahren mit Präkonditionierer M)

Eingabe: $A \in \mathbb{R}^{n\times n}$, $q_1 \in \mathbb{R}^n$ mit $\|q_1\|_2 = 1$

(1) Für $j = 1, 2, \ldots, k-1$

(2)
$w := Aq_j$
Löse $Mv = w$
für $i = \max\{1, j-1\}, \ldots, j$
$b_{ij} := q_i^T w \;\; (= q_i^T M v)$
$v := v - b_{ij} q_i$
Ende der i-Schleife

(3)
if $v \neq 0$
$b_{j+1,j} := \sqrt{v^T w} \;\; (= \sqrt{v^T M v})$
$q_{j+1} := v / b_{j+1,j}$

(4)
else
Ende mit Fehlermeldung: Breakdown
end if

Ende der j-Schleife

Ausgabe: $B_k = (b_{ij})_{i,j=1,\ldots,k}$, $Q_k = (q_1, \ldots, q_k)$ □

Der Unterschied zum Arnoldi-Algorithmus liegt dabei darin, dass die Schleife in (2) nur von $j-1$ bis j läuft, also jeweils nur zwei Durchläufe durchgeführt werden. Dies könnte man tatsächlich auch ohne Schleife direkt als Dreiterm-Rekursion implementieren (so hatten wir das in Abschnitt 1.4 gemacht). Die GMRES-Variante für symmetrische Matrizen mit Lanczos wird MINRES genannt. Beachte, dass im symmetrischen Fall stets $\kappa_M(V) = 1$ in Satz 2.23 gilt.

Kapitel 3

Kontinuierliche Optimierung

Dieses Kapitel vertieft das Kapitel 5 aus der Einführung in die Numerische Mathematik, in dem wir mit dem Gauß-Newton-Verfahren bereits ein Verfahren zur Lösung eines — recht speziellen — Optimierungsproblems behandelt haben. Hier werden wir allgemeinere Probleme betrachten.

Allgemein lautet die Problemstellung

$$\min_{x \in X} f(x),$$

wobei X eine Teilmenge von $\mathbb{R}^n$ ist, die Menge der zulässigen Werte. Hierbei nehmen wir Differenzierbarkeit von f an.

3.1 Unbeschränkte nichtlineare Optimierung

Bei der unbeschränkten Optimierung gilt $X = \mathbb{R}^n$, d.h., es wird keine Einschränkung bei der Auswahl von x gemacht.

3.1.1 Das SQP-Verfahren

SQP steht für "sequentielle quadratische Programmierung"; der Ausdruck "Programmierung" darin ist ein etwas altmodischer Begriff für die Lösung einer Optimierungsproblems. Die Grundlage des SQP-Verfahrens ist, dass quadratische unbeschränkte Optimierungsprobleme, also Probleme der Form

$$\min_{x \in \mathbb{R}^n} \frac{1}{2} x^T A x - x^T b$$

sehr leicht zu lösen sind, wenn A spd ist. Es genügt dann, das Gleichungssystem $Ax = b$ zu lösen, da dieses eine Nullstelle des Gradienten liefert, die wegen der positiven Definitheit von A die gesuchte Minimalstelle ist.

Die Idee des SQP-Verfahrens liegt nun darin, eine quadratische Approximation an f zur Minimierung zu nutzen. Dazu approximieren wir

$$f(x+v) \approx f(x) + v^T \nabla f(x) + \frac{1}{2} a_x(v,v) =: q_x(v) \tag{3.1}$$

mit $a_x(v,v) = v^T A_x v$ und minimieren q_x über v. Dazu müssen natürlich q_x und v zumindest für kleine v hinreichend gut übereinstimmen, d.h.

$$r_x(v) := f(x+v) - q_x(v) \tag{3.2}$$

muss für kleine v hinreichend klein sein. Die Funktion q_x wird hierbei als "quadratisches Modell" bezeichnet. Haben wir eine Möglichkeit, solch ein quadratisches Modell q_x zu finden (Genaueres dazu besprechen wir im Anschluss), können wir den folgenden Algorithmus formulieren.

Algorithmus 3.1 (SQP mit Liniensuche)

Eingabe: $x^{(0)} \in \mathbb{R}^n$, $\varepsilon > 0$

(0) Setze $k := 0$

(1) Bestimme $\Delta x^{(k)} := \operatorname{argmin}_{x \in \mathbb{R}^n} q_{x^{(k)}}(x)$
Bestimme ein geeignetes $\alpha_k > 0$ und setze $x^{(k+1)} := x^{(k)} + \alpha_k \Delta x^{(k)}$

(2) Falls $\|x^{(k+1)} - x^{(k)}\| < \varepsilon$ beende den Algorithmus;
sonst setze $k := k+1$ und gehe zu (1) □

Zu klären ist nun, wie wir die quadratische Approximation q_x und die Schrittweite $\alpha_k \in \mathbb{R}$ bestimmen.

Zur Wahl von q_x

Zur Wahl von q_x ist der quadratische Term $a_x(v,v) = v^T A_x v$ in

$$q_x(v) := f(x) + v^T \nabla f(x) + \frac{1}{2} a_x(v,v)$$

zu bestimmen. Der Minimierer ist dann als Lösung des Gleichungssystems $Ax = b$ mit $A = A_x$ und $b = -\nabla f(x)$ gegeben, d.h. $\Delta x = -A_x^{-1} \nabla f(x)$.

Für die Wahl von a_x bzw A_x gibt es mehrere Möglichkeiten.

- $a_x(v,v) = v^T v \;\Leftrightarrow\; A_x = \mathrm{Id}$. In diesem Fall erhalten wir

  $$\Delta x = -\nabla f(x),$$

  d.h. die Suchrichtung ist der negative Gradient, also die Richtung des steilsten Abstiegs. Dieses Verfahren heißt *Gradientenverfahren* oder *Methode des steilsten Abstiegs*.

- $a_x(v,v) = v^T M v \;\Leftrightarrow\; A_x = M$ mit einem Vorkonditionierer M. In diesem Fall erhalten wir

  $$\Delta x = -M^{-1} \nabla f(x)$$

  und das Verfahren heißt *vorkonditioniertes Gradientenverfahren*.

- $a_x(v,v) = v^T H_f(x)v \quad \Leftrightarrow \quad A_x = H_f(x)$, wobei $H_f(x)$ die Hesse-Matrix von f im Punkt x, also die zweite Ableitung von f bezeichnet. In diesem Fall erhalten wir

  $$\Delta x = -H_f(x)^{-1}\nabla f(x).$$

  Falls f zwei mal stetig differenzierbar ist, ist $H_f(x)$ symmetrisch. In diesem Fall gilt für $g(x) := \nabla f(x)$ gerade $Dg(x) = H_f(x)$ und damit

  $$\Delta x = -Dg(x)^{-1}g(x).$$

  Für $\alpha_k = 1$ erhalten wir also gerade das Newton-Verfahren zur Suche einer Nullstelle von $g = \nabla f$. Für variables α_k heißt das Verfahren daher *Newton-Verfahren mit variabler Schrittweite*. Beachte, dass dieses Verfahren nur dann anwendbar ist, wenn $H_f(x)$ invertierbar ist und nur dann ein Minimum von q_x liefert, wenn $H_f(x)$ positiv definit ist. In diesem Fall konvergiert das Verfahren lokal quadratisch, falls die Voraussetzungen von Satz 5.15 aus der Einführung in die Numerik erfüllt sind.

- $a_x(v,v) = \gamma_x v^T M v + v^T H_f(x)v \quad \Leftrightarrow \quad A_x = \gamma_x M + H_f(x)$, wobei $\gamma_x > 0$, M ein Vorkonditionierer und $H_f(x)$ die Hesse-Matrix von f im Punkt x ist. Hierbei wird γ_x gerade so groß gewählt, dass A_x spd ist, um Konvergenz und das Finden eines Minimums von q_x sicher zu stellen. Falls $H_f(x^*)$ im Minimierer x^* spd ist, kann superlineare Konvergenz des Verfahrens bewiesen werden.

- Eine weitere Methode ist, A_x während der Laufzeit des Verfahrens zu konstruieren. Ein Verfahren dieser Art ist die BFGS-Methode, die wir später genauer besprechen werden.

Das folgende Lemma zeigt, dass das SQP-Verfahren stets eine Abstiegsrichtung findet, d.h. dass die Richtungsableitung von f in Richtung Δx immer negativ ist, falls A_x spd ist. Zudem gibt es eine Schranke für die Stärke des Abstiegs an.

Lemma 3.2 Es seien A_x symmetrisch und $0 < \gamma < \Gamma$ so, dass für das quadratische Modell die Ungleichung

$$\gamma\|v\|^2 \leq a_x(v,v) \leq \Gamma\|v\|^2$$

für alle $v \in \mathbb{R}^n$ erfüllt ist. Dann gilt

$$Df(x)\Delta x \leq -\gamma\|\Delta x\|^2 \leq -\frac{\gamma}{\Gamma^2}\|\nabla f(x)\|^2.$$

Beweis: Die Ungleichungen folgen aus der Tatsache, dass $\nabla f(x) = -A_x\Delta x$. Damit gilt zum einen

$$Df(x)\Delta x = \nabla f(x)^T\Delta x = -\Delta x^T A_x \Delta x = -a_x(\Delta x, \Delta x) \leq -\gamma\|\Delta x\|^2,$$

also die erste Ungleichung. Zum anderen erhalten wir aus der Annahme $\|A_x\| \leq \Gamma$ und damit

$$\|\nabla f(x)\| \leq \|A_x\|\,\|\Delta x\| \leq \Gamma\|\Delta x\|,$$

also $-\|\Delta x\|^2 \leq \|\nabla f(x)\|^2/\Gamma^2$, woraus die zweite Ungleichung folgt. □

Wählt man die α_k analog zu den ω_k im vorkonditionierten Gradientenverfahrens, so kann man die Konvergenz des Verfahrens mit einem ähnlichen Beweis wie in Satz 2.7 beweisen. Dies ist anschaulich klar, denn so bald man sich in der Nähe des Minimums befindet ist das Verfahren bis auf Terme höherer Ordnung identisch zu diesem Verfahren. Die genaue Ausarbeitung der zugehörigen Abschätzungen ist aber sehr technisch. Man erhält so

$$\|x^{(k+1)} - x^*\|_A \leq \Theta_k \|x^{(k)} - x^*\|_A$$

mit $\Theta_k = (\kappa_{H_f(x^*),A_{x^{(k)}}} - 1)/(\kappa_{H_f(x^*),A_{x^{(k)}}} + 1)$ und der in (2.3) definierten Kondition $\kappa_{H_f(x^*),A^{(k)}}$ von $H_f(x^*)$ relativ zu $A_{x^{(k)}}$.

Im Fall $\Theta_k \to 0$ erhalten wir superlineare Konvergenz. Dies ist der Fall, falls $A_{x^{(k)}} \to H_f(x^*)$ gilt, weil dann $\kappa_{H_f(x^*),A_{x^{(k)}}} \to 1$ gilt, weil alle Eigenwerte von $A^{-1}_{x^{(k)}} H_f(x^*)$ gegen Eigenwerte von Id, also gegen 1 konvergieren.

Zur Wahl von α_k: Die Armijo-Regel

Als nächstes betrachten wir eine Möglichkeit, die Schrittweite α_k zu bestimmen, so dass wir Konvergenz des Gesamtverfahrens basierend auf Lemma 3.2 beweisen können. Aus der Taylor-Entwicklung erhalten wir die Gleichung

$$f\left(x^{(k)} + \alpha_k \Delta x^{(k)}\right) = f\left(x^{(k)}\right) + \alpha_k Df\left(x^{(k)}\right)\Delta x^{(k)} + s_{x^{(k)}}\left(\alpha_k \Delta x^{(k)}\right)$$

erhalten. Die Idee der Armijo-Schrittweite besteht nun darin, ein $\eta \in (0,1)$ vorzugeben und α_k so zu wählen, dass

$$f\left(x^{(k)} + \alpha_k \Delta x^{(k)}\right) \leq f\left(x^{(k)}\right) + \eta\alpha_k Df\left(x^{(k)}\right)\Delta x^{(k)} \tag{3.3}$$

gilt. Das folgende Lemma zeigt, dass dies immer möglich ist.

Lemma 3.3 Es seien die Voraussetzungen von Lemma 3.2 in $x = x_k$ mit $\gamma = \gamma_k$ erfüllt und für den Restterm aus (3.2) existiere ein C_k mit

$$s_{x^{(k)}}(v) \leq C_k \|v\|^2.$$

Dann ist (3.3) für alle $\alpha_k \in (0, \bar{\alpha}_k)$ erfüllt mit $\bar{\alpha}_k = (1-\eta)\gamma_k/C_k$.

Beweis: Wir lassen im Beweis die Indizes k bzw (k) weg. Es gilt

$$\begin{aligned} f(x+\alpha\Delta x) - f(x) &= \alpha Df(x)\Delta x^(k) + r_x(\alpha\Delta x) \\ &\leq (\eta + 1 - \eta)Df(x) + C\alpha^2\|\Delta x\|^2 \\ &\leq \eta\alpha Df(x)\Delta x - (1-\eta)\alpha\gamma\|\Delta x\|^2 + C\alpha^2\|\Delta x\|^2 \\ &= \eta\alpha Df(x)\Delta x + \alpha(C\alpha - (1-\eta)\gamma)\|\Delta x\|. \end{aligned}$$

Für $\alpha \leq \bar{\alpha}$ ist der letzte Term gerade ≤ 0, woraus die Behauptung folgt. □

Bemerkung 3.4 Falls Df global Lipschitz-stetig ist, also $L > 0$ existiert mit

$$\|Df(x) - Df(y)\| \leq L\|x - y\|,$$

so folgt aus dem Hauptsatz der Differential- und Integralrechnung

$$\begin{aligned} f(x+v) - f(x) - Df(x)v &= \int_0^1 (Df(x+tv) - Df(x))v dt \\ &\leq \int_0^1 tL\|v\|^2 dt \leq \frac{L}{2}\|v\|^2, \end{aligned}$$

woraus die Voraussetzung von Lemma 3.3 mit $C_k \leq L/2$ folgt. □

Der folgende Satz zeigt nun Konvergenz des SQP-Verfahrens mit der Armijo-Schrittweite, falls wir geeignete Gleichmäßigkeit (also Unabhängigkeit von k) aller betrachteter Schranken annehmen.

Satz 3.5 Für die vom SQP-Verfahren in Algorithmus (3.1) erzeugte Folge $x^{(k)}$ gelten die folgenden Annahmen:

- Es existiere $\underline{f} \in \mathbb{R}$ mit $f(x^{(k)}) \geq \underline{f}$ für alle $k \in \mathbb{N}$.
- Die Voraussetzungen von Lemma 3.2 seien erfüllt gleichmäßig in k, d.h. es existieren $0 < \underline{\gamma} \leq \overline{\Gamma}$ mit $\underline{\gamma} \leq \gamma_k \leq \Gamma_k \leq \overline{\Gamma}$ für alle $k \in \mathbb{N}$.
- Es existiere $\underline{\alpha} > 0$ mit $\underline{\alpha} \leq \overline{\alpha}_k$ für $\overline{\alpha}_k$ aus Lemma 3.3 und alle $k \in \mathbb{N}$. Zudem sei α_k so gewählt, dass $\hat{\alpha} := \inf_{k\in\mathbb{N}} \alpha_k > 0$.

Dann gilt $Df(x^{(k)}) \to 0$, d.h. $x^{(k)}$ konvergiert gegen einen kritischen Punkt von f.

Beweis: Per Konstruktion gilt $f(x^{(k+1)}) \leq f(x^{(k)})$, zudem gilt $\underline{f} - f(x^{(0)}) \leq f(x^{(j+1)} - f(x^{(0)}) = \sum_{k=0}^{j} f(x^{(k+1)} - f(x^{(k)})$. Damit folgt

$$f - f(x^{(0)}) \leq \sum_{k=0}^{\infty} f(x^{(k+1)} - f(x^{(k)})$$

und daher $|f(x^{(k+1)} - f(x^{(k)})| \to 0$ für $k \to \infty$. Aus Lemma 3.2 und der Annahme erhalten wir nun

$$|f(x^{(k+1)} - f(x^{(k)})| \geq \eta\alpha_k |Df(x^{(k)})\Delta x^{(k)}| \geq \eta\alpha_k\gamma_k\|\Delta x\| \geq \frac{\eta\hat{\alpha}\underline{\gamma}}{\overline{\Gamma}^2}\|Df(x^{(k)})\|^2$$

und damit $Df(x^{(k)}) \to 0$. □

Bemerkung 3.6 Sofern $Df(x^{(k)}) \neq 0$ ist, ist $f(x^{(k)})$ nach Konstruktion streng monoton fallend in k. Daher kann der kritische Punkt, gegen den $x^{(k)}$ konvergiert, kein lokales Maximum sein. Es könnte aber durchaus ein Sattelpunkt sein. □

Eine einfache Methode, eine Armijo-Schrittweite zu berechnen, die die Bedingungen des Satzes erfüllt, stellt der folgende Algorithmus dar.

Algorithmus 3.7 (einfache Armijo-Schrittweitensuche)

Eingabe: $x, \Delta x \in \mathbb{R}^n$,

(0) Setze $\alpha := 1$

(1) Falls $f(x + \alpha\Delta x) \leq f(x) + \eta\alpha Df(x)\Delta x$, beende den Algorithmus

(2) Setze $\alpha := \alpha/2$ und gehe zu (1) □

3.1.2 Das BFGS-Verfahren

Wir haben im letzten Abschnitt gesehen, dass das SQP-Verfahren besonders schnell konvergiert, wenn A_x als Hesse-Matrix $H_f(x)$ gewählt wird, weil wir dann ein Newton-Verfahren für die Nullstelle des Gradienten erhalten. Allerdings ist die Berechnung der Hesse-Matrix in jedem Schritt aufwändig, daher möchte man deren Berechnung durch eine einfachere Konstruktion ersetzen. Das BFGS-Verfahren (benannt nach dessen Erfindern Broyden, Fletcher, Goldfarb und Shanno) liefert solch eine Konstruktion. Die Idee liegt darin, die Matrix A_x Schritt für Schritt zu aktualisieren und dabei Informationen über gewisse Einträge der Hesse-Matrix durch Differenzen der ersten Ableitungen zu approximieren.

Das quadratische Modell ist jetzt von der Form

$$q_x(v) = f(x) - Df(x)v + \frac{1}{2}b_k(v, v)$$

mit einem vom Iterationsschritt k abhängigen quadratischen Term. Der Ansatz zur Wahl der aktualisierten quadratischen Form b_{k+1} ist nun

$$b_{k+1}(x^{(k+1)} - x^{(k)}, w) = (Df(x^{(k+1)}) - Df(x^{(k)}))w$$

für alle $w \in \mathbb{R}^n$ (beachte: für die Hesse-Matrix würde diese Gleichung näherungsweise gelten, allerdings nicht nur für $x^{(k+1)}$ und $x^{(k)}$ sondern für alle x und y). Wir schreiben diese Gleichung kurz als

$$b_{k+1}(\Delta x^{(k)}, w) = \Delta f_k' w \tag{3.4}$$

mit $\Delta f_k' = Df(x^{(k+1)}) - Df(x^{(k)})$. Wir wollen (3.4) erreichen, indem wir die eine Bilinearform u_k von Rang 1 zu b_k addieren. Solche Bilinearformen sind von der Form $b(v, w) = v^T x x^T v$ für ein $x \in \mathbb{R}^n$; diese sind günstig, weil nur der Vektor x gespeichert werden muss und nicht wie im allgemeinen Fall $b(v, w) = v^T B w$ eine volle Koeffizientenmatrix B (d.h. der Speicherbedarf beträgt n statt n^2 Gleitkommazahlen). Ein weiterer Vorteil dieses Updates ist, dass die Inverse der aktualisierten Matrix leicht berechnet werden kann, was wir im Abschnitt über die Implementierung unten sehen werden.

Die Aktualisierung bzw. das Update wird dabei gewählt als

$$u_k(v, w) := \frac{v^T(\Delta f_k')^T \Delta f_k' w}{\Delta f_k' \Delta x^{(k)}}.$$

Diese Bilinearform ist positiv semidefinit, falls

$$\Delta f_k' \Delta x^{(k)} = (Df(x^{(k+1)}) - Df(x^{(k)}))(x^{(k+1)} - x^{(k)}) > 0 \tag{3.5}$$

gilt. Diese Bedingung muss bei der Liniensuche berücksichtigt werden (im nächsten Absatz sehen wir, warum). Im positiv semidefiniten Fall gilt $u_k(v,w) = v^T x x^T v$ für $x = (\Delta f_k')^T / \sqrt{\Delta f_k' \Delta x^{(k)}}$.

Um (3.4) zu erhalten, benötigen wir noch einen weiteren Korrekturterm, der als

$$r_k(v,w) = \frac{b_k(\Delta x^{(k)}, v) b_k(\Delta x^{(k)}, w)}{b_k(\Delta x^{(k)}, \Delta x^{(k)})}$$

gewählt wird. Eine einfache Rechnung zeigt dann, dass

$$b_{k+1}(v,w) := (b_k + u_k - r_k)(v,w) = b_k(v,w) + \frac{v^T (\Delta f_k')^T \Delta f_k' w}{\Delta f_k' \Delta x^{(k)}} - \frac{b_k(\Delta x^{(k)}, v) b_k(\Delta x^{(k)}, w)}{b_k(\Delta x^{(k)}, \Delta x^{(k)})} \tag{3.6}$$

tatsächlich (3.4) erfüllt. Aus der Cauchy-Schwarz-Ungleichung folgt dabei für alle $v \in \mathbb{R}^n$ die Ungleichung $b_k(v,v) - r_k(v,v) \geq 0$, weswegen die Bilinearform $b_k - r_k$ positiv semidefinit ist. Dabei gilt $b_k(v,v) - r_k(v,v) = 0$ gerade für $v = \beta \Delta x^{(k)}$, $\beta \in \mathbb{R}$. Für diese v ist aber $u_k(v,v) > 0$ falls $\beta \neq 0$, weswegen b_{k+1} insgesamt positiv definit ist.

Die Gleichung (3.6) heißt BFGS Update-Schritt. Für einen vollständigen Algorithmus muss nun noch die erste Bilinearform b_0 festgelegt werden. Diese kann man als

$$b_0(v,w) = v^T H_f(x) w$$

wählen, wenn die Hesse-Matrix $H_f(x)$ spd und leicht zu invertieren ist. Alternativ setzt man

$$b_0(v,w) = v^T M w$$

für eine der Matrizen, die nach Algorithmus 3.1 besprochen wurden.

Implementierung

Die Implementierung folgt Algorithmus 3.1. Darin muss in jedem Schritt ein lineares Gleichungssystem der Form $A_x x = b$ gelöst werden. Hier ist die Matrix A_x gerade die Matrix B_k, für die $b_k(v,w) = v^T B_k w$ gilt. Diese ist gegeben durch die Updatevorschrift

$$B_{k+1} = B_k + \frac{(\Delta f_k')^T \Delta f_k'}{\Delta f_k' \Delta x^{(k)}} - \frac{B_k \Delta x^{(k)} (\Delta x^{(k)})^T B_k}{(\Delta x^{(k)})^T B_k \Delta x^{(k)}}.$$

Die spezielle Form des BFGS-Updates erlaubt es nun, die Inverse B_{k+1}^{-1} leicht explizit aus B_k^{-1} zu berechnen. Für $H_k := B_k^{-1}$ gilt

$$H_{k+1} = \left(\mathrm{Id} - \frac{\Delta x^{(k)} \Delta f_k'}{\Delta f_k' \Delta x^{(k)}} \right) H_k \left(\mathrm{Id} - \frac{(\Delta f_k')^T (\Delta x^{(k)})^T}{\Delta f_k' \Delta x^{(k)}} \right) + \frac{\Delta x^{(k)} (\Delta x^{(k)})^T}{\Delta f_k' \Delta x^{(k)}} \tag{3.7}$$

Mit etwas Rechnung prüft man nach, dass für diese Matrix tatsächlich $H_{k+1} = B_{k+1}^{-1}$ gilt. Am Anfang des Algorithmus muss hierbei einmal die Matrix B_0 invertiert werden. Durch Verbindung mit Algorithmus 3.1 führt dies auf den folgenden Algorithmus.

Algorithmus 3.8 (SQP mit BFGS-Update und Liniensuche)

Eingabe: $x^{(0)} \in \mathbb{R}^n$, $\varepsilon > 0$, $H_0 := B_0^{-1} \in \mathbb{R}^{n \times n}$

(0) Setze $k := 0$

(1) Berechne $y := -H_k x^{(k)}$
Bestimme durch Liniensuche $\alpha_k > 0$ und setze $x^{(k+1)} := x^{(k)} + \alpha_k y$
(berücksichtige dabei (3.5))
Setze $\Delta x^{(k)} := \alpha_k y$ und berechne H_{k+1} gemäß (3.7)

(2) Falls $\|x^{(k+1)} - x^{(k)}\| < \varepsilon$ beende den Algorithmus;
sonst setze $k := k + 1$ und gehe zu (1) □

Der Rechenaufwand jedes Iterationsschritts beträgt $O(n^2)$.

Für das BFGS-Verfahren gilt der folgende Konvergenzsatz. Der (recht lange) Beweis findet sich z.B. in Abschnitt 6.4 des Buchs von Nocedal und Wright [4].

Satz 3.9 Sei $f \in C^2(\mathbb{R}^n, \mathbb{R})$ konvex und $x \mapsto H_f(x)$ Lipschitz. Darüberhinaus nehmen wir an, dass $0 < \gamma \leq \Gamma$ existieren, so dass

$$\gamma \|v\|^2 \leq v^T H_f(x) v \leq \Gamma \|v\|^2$$

gilt für alle $x, v \in \mathbb{R}^n$ und dass B_0 spd ist. Dann findet die BFGS-Methode mit Liniensuche eine Folge $x^{(k)}$, die superlinear gegen x^* konvergiert.

Bemerkung 3.10 Es gibt verschiedene Varianten des Verfahrens, in denen $H_k x^{(k)}$ auf andere Weise berchnet wird. Zum Beispiel wird in einer Variante die explizite Speicherung von H_k vermieden und das Produkt $H_k x^{(k)}$ wird direkt mit einer rekursiven Formal aus B_0 bzw. B_0^{-1} und den $\Delta x^{(j)}$ und $\Delta f_j'$ berechnet. In einer anderen Variante wird statt B_k^{-1} die Choleski-Zerlegung $B_k = L_k L_k^T$ in jedem Schritt aktualisiert. □

Trust-Region Verfahren

Offensichtlich stimmen das quadratische Modell $q_{x^{(k)}}$ und die zu minimierende Funktion f nur in einer Umgebung von $x^{(k)}$ näherungsweise überein. Die Bedingung (3.3) in der Schrittweitenwahl sorgt dafür, dass dies nicht zu Fehlern führt, weil hier explizit der Abstieg von f sichergestellt wird. Es wird also zunächst eine Abstiegsrichtung auf Basis von $q_{x^{(k)}}$ gesucht und dann an Hand von f ermittelt, wie weit entlang dieser Richtung gegangen wird.

Eine alternative Methode ist es, diese Reihenfolge umzudrehen. In der Trust-Region Methode wird zunächst festgelegt, in welcher Region von $x^{(k)}$ die Näherung genau genug ist, um ihr zu "vertrauen" (Trust-Region = Vertrauensregion). Dadurch wird die maximale Schrittweite festgelegt. Innerhalb der Region wird dann $q_{x^{(k)}}$ optimiert. Die Größe der Trust-Region wird dabei stets überprüft. Stellt sich heraus, dass $q_{x^{(k)}}$ und f im ermittelten Minimierer nicht gut genug übereinstimmen, wird ihr Radius verkleinert, stimmen die Werte hingegen gut überein, wird der Radius vergrößert. Dies führt auf den folgenden Algorithmus.

Algorithmus 3.11 (Trust-Region-SQP Verfahren)

Eingabe: $x^{(0)} \in \mathbb{R}^n$, $\varepsilon > 0$, $\Delta_0 > 0$, $\eta \in (0, 1/4)$

(0) Setze $k := 0$

(1) Berechne $v_k := \operatorname{argmin}_{\|v\| \leq \Delta_k} q_{x^{(k)}}(v)$
Berechne $\rho_k := \frac{f(x^{(k)}+v_k)-f(x^{(k)})}{q_{x^{(k)}}(v_k)-q_{x^{(k)}}(0)}$
Falls $\rho_k > \eta$, setze $x^{(k+1)} := x^{(k)} + v_k$; sonst setze $x^{(k+1)} := x^{(k)}$

(2) Falls $\rho_k < 1/4$, setze $\Delta_{k+1} := \Delta_k/2$
Falls $\rho_k > 3/4$, setze $\Delta_{k+1} := 2\Delta_k$
sonst setze $\Delta_{k+1} := \Delta_k$

(3) Falls $\|x^{(k+1)} - x^{(k)}\| < \varepsilon$ beende den Algorithmus;
sonst setze $k := k + 1$ und gehe zu (1) □

Die Minimierung von $q_{x^{(k)}}$ kann z.B. mit dem CG-Verfahren durchgeführt werden, allerdings ist dabei die Bedingung $\|v\| \leq \Delta_k$ nicht unbedingt erfüllt. Es gibt aber Varianten des CG-Verfahrens, z.B. die sogenannte Steihaug-Methode, vgl. [4, Algorithm 7.2], welche die Beschränkung berücksichtigen.

3.2 Globalisierung des Newton-Verfahrens

Die meisten der bisher betrachteten Verfahren lassen sich als eine Abwandlung des Newton-Verfahrens zur Berechnung einer Nullstelle von ∇f auffassen. Sie heißen deshalb auch Quasi-Newton-Verfahren. Alle Newton-artigen Verfahren haben das Problem, dass sie nur lokal konvergieren, falls nicht spezielle Bedingungen (wie z.B. die Konvexität von f in Satz 3.9) vorliegen. Es stellt sich daher die Frage, ob es nicht Möglichkeiten gibt, das Newton-Verfahren so abzuwandeln, dass wir Konvergenz für beliebige Startwerte erhalten. Solche Modifikationen nennt man "Globalisierung". In diesem Abschnitt wollen wir einige solcher Ideen vorstellen. Wir machen dies der Einfachheit halber für das Standard-Newton-Verfahren zum Finden einer Nullstelle einer Abbildung $f : \mathbb{R}^n \to \mathbb{R}^n$.

Algorithmus 3.12 (Newton-Verfahren im $\mathbb{R}^n$)

Eingabe: $x^{(0)} \in \mathbb{R}^n$, $\varepsilon > 0$

(0) Setze $k := 0$

(1) Löse das lineare Gleichungssystem $Df(x^{(k)})\Delta x^{(k)} = f(x^{(k)})$
und berechne $x^{(k+1)} = x^{(k)} - \Delta x^{(k)}$

(2) Falls $\|\Delta x^{(k)}\| < \varepsilon$, beende den Algorithmus,
ansonsten setze $k = k + 1$ und gehe zu (1)

□

Im ersten Ansatz werden wir sehen, wie man das Standard-Newton-Verfahren als SQP-Algorithmus umschreiben kann. Dies gibt uns die Möglichkeit, das Newton-Verfahren um eine Schrittweitensuche zu erweitern, für die man unter geeigneten Voraussetzungen globale Konvergenz erhält. Die zweite Methode ist die sogenannte Pfadverfolgung. Diese Methode kann relativ leicht auf jedes Quasi-Newton-Verfahren zur Optimierung übertragen werden.

Das Newton-Verfahren als SQP-Verfahren

Um das Newton-Verfahren als SQP-Verfahren zu interpretieren, verwenden wir das im letzten Semester bereits betrachtete Gauß-Newton-Verfahren.

Algorithmus 3.13 (Gauß-Newton-Verfahren)

Eingabe: $x^{(0)} \in \mathbb{R}^n$, $\varepsilon > 0$

(0) Setze $k := 0$

(1) Löse das lineare Ausgleichsproblem $\|Df(x^{(k)})\Delta x^{(k)} - f(x^{(k)})\|_2^2 = \min$
und berechne $x^{(k+1)} = x^{(k)} - \Delta x^{(k)}$

(2) Falls $\|\Delta x^{(k)}\| < \varepsilon$, beende den Algorithmus,
ansonsten setze $k := k + 1$ und gehe zu (1)

□

Tatsächlich kann man das Gauß-Newton-Verfahren als SQP-Verfahren interpretieren. Im Gauß-Newton-Verfahren wollen wir das Problem

$$\text{minimiere } g(x) := \frac{1}{2}\|f(x)\|_2^2 \text{ über } x \in D \tag{3.8}$$

lösen. Im Verfahren lösen wir in jedem Schritt das lineare Ausgleichsproblem

$$\text{minimiere } \|Df(x^{(k)})\Delta x^{(k)} - f(x^{(k)})\|_2^2 \text{ über } \Delta x^{(k)}.$$

Wir minimieren also die Funktion

$$\begin{aligned}\|Df(x^{(k)})\Delta x^{(k)} - f(x^{(k)})\|_2^2 &= \langle Df(x^{(k)})\Delta x^{(k)} - f(x^{(k)}), Df(x^{(k)})\Delta x^{(k)} - f(x^{(k)})\rangle \\ &= \langle Df(x^{(k)})\Delta x^{(k)}, Df(x^{(k)})\Delta x^{(k)}\rangle \\ &\quad - 2\langle Df(x^{(k)})\Delta x^{(k)}, f(x^{(k)})\rangle \\ &\quad + \langle f(x^{(k)}), f(x^{(k)})\rangle.\end{aligned}$$

Teilen wir diese Funktion durch 2 und schreiben sie mit $x = x^{(k)} = x$ und $v = -\Delta x^{(k)}$ als

$$\begin{aligned}q_x^{GN}(v) &= \frac{1}{2}\langle Df(x)v, Df(x)v\rangle + \langle Df(x)v, f(x)\rangle + \frac{1}{2}\langle f(x), f(x)\rangle \\ &= \frac{1}{2}a_x(v,v) + Dg(x)v + g(x)\end{aligned}$$

mit $a_x(v,w) = v^T Df(x)^T Df(x)w$, so ist dies genau von der Form von q_x in (3.1) mit $A_x = Df(x)^T Df(x)$. Das Gauß-Newton-Verfahren ist also eine spezielle Variante des SQP-Verfahrens mit Schrittweite $\alpha_k \equiv 1$.

Die Idee liegt nun darin, die Schrittweitensuche aus den Quasi-Newton-Verfahren auf das Newton-Verfahren zu übertragen, indem wir die Nullstellenberechnung nicht mit dem Newton-Verfahren selbst sondern mit dem Gauß-Newton-Verfahren durchzuführen. Dazu müssen wir eine sogenannte Meritfunktion $m : \mathbb{R}^n \to \mathbb{R}$ definieren, so dass $f(x) = 0$ genau dann gilt, wenn $m(x)$ minimal wird. Eine einfache Wahl eines solchen m ist

$$m(x) := \frac{1}{2}\|f(x)\|^2 = \frac{1}{2}\langle f(x), f(x)\rangle.$$

Für diese Wahl von m können wir dieses Problem nun mit dem Gauß-Newton-Verfahren lösen, was auf das quadratische Modell

$$q_x^{GN}(v) = \frac{1}{2}a_x(v,v) + Dm(x)v + m(x)$$

mit $a_x(v,w) = v^T Df(x)^T Df(x)w$ führt. Dies kann nun in Algorithmus 3.1 mit variabler Schrittweite verwendet werden, was letztendlich auf ein Gauß-Newton-Verfahren mit variabler Schrittweite führt.

Man kann unter geeigneten Annahmen beweisen (siehe [4, Theorem 10.1]), dass dieses Verfahren für alle Startwerte — also global — gegen einen Punkt x^* mit $Dm(x^*) = 0$ konvergiert. Für diesen gilt entweder $f(x^*) = 0$ oder es gilt, dass $Df(x^*)$ nicht invertierbar ist und $Df(x^*)f(x^*) = 0$ ist.

Das quadratische Modell q_x^{GN} aus dem Gauß-Newton Verfahren kann auch in einem Trust-Region Verfahren verwendet werden. Der daraus resultierende Algorithmus ist als Levenberg-Marquardt Methode bekannt, für Details siehe [4, Section 10].

Neben der oben angegebenen Merit-Funktion sind natürlich andere Wahlen möglich, insbesondere wenn Nebenbedingungen berücksichtigt werden müssen, die wir im folgenden Abschnitt 3.3 betrachten. In diesem Fall erhalten wir keine Variante des Gauß-Newton-Verfahrens sondern gehen wieder wie beim allgemeinen SQP-Verfahren in Algorithmus 3.1ff. vor.

Pfadverfolgung

Die Pfadverfolgung ist eine weitere Strategie, das Newton-Verfahren zu globalisieren. Sie beruht darauf, das eigentlich zu lösende Problem $f(x) = 0$ in eine Familie von Problemen $F(x,\lambda) = 0$ mit $\lambda \in [0,1]$ — eine sogenannte Homotopie — einzubetten. Das bedeutet, dass $f(x) = F(x,1)$ gilt. Zudem nehmen wir an, dass wir $y^{(0)} \in \mathbb{R}^n$ mit $F(y^{(0)},0) = 0$ kennen. Das Verfahren läuft dann wie folgt ab.

Algorithmus 3.14 (Newton-Verfahren mit Pfadverfolgung)

Eingabe: $y^{(0)} \in \mathbb{R}^n$ mit $F(y^{(0)},0) = 0$, $\sigma \in (0,1)$

(0) Setze $\lambda := \sigma$, $k := 0$

(1) Versuche, $F(x, \lambda) = 0$ mit dem Newton-Verfahren mit dem Startwert $y^{(k)}$ zu lösen

(2) Falls das Verfahren erfolgreich war und ein Ergebnis x geliefert hat
Falls $\lambda = 1$, beende den Algorithmus, sonst:
Setze $\lambda := \min\{\lambda + \sigma, 1\}$
Setze $y^{(k+1)} := x$
Falls das Newton-Verfahren sehr schnell konvergiert ist, erhöhe σ

(3) Sonst: wähle $\rho \in (0, \sigma)$ und setze $\sigma := \sigma - \rho$, $\lambda := \lambda - \rho$

(4) Setze $k := k + 1$ und gehe zu (1)

□

Wichtig ist in diesem Verfahren, Kriterien dafür zu haben, wann das Newton-Verfahren schnell konvergiert (um σ in Schritt (2) zu erhöhen) und wann das Newton-Verfahren voraussichtlich nicht konvergiert (um es in Schritt (1) abbrechen zu können, falls keine Konvergenz zu erwarten ist).

Hierzu erinnern wir an die Konvergenzanalyse des Newton-Verfahrens in Satz 5.15 der Einführung in die Numerische Mathematik. Dort haben wir die Kontraktionskonstante ω des Algorithmus durch die Ungleichung

$$\|Df(x)^{-1}(f(y) - f(x) - Df(x)(y - x))\| \leq \frac{\omega}{2}\|y - x\|^2 \tag{3.9}$$

abgeschätzt. Aus dieser haben wir in dem Beweis die Ungleichung

$$\|x^{(k+1)} - x^*\| \leq \frac{\omega}{2}\|x^{(k)} - x^*\|^2 \tag{3.10}$$

hergeleitet.

Setzen wir in (3.9) $x = x^{(k)}$ und $y = x^{(k+1)}$, womit $y - x = \Delta x^{(k)}$ gilt, so erhalten wir

$$\begin{aligned} & Df(x^{(k)})^{-1}(f(x^{(k+1)}) - f(x^{(k)}) - Df(x^{(k)})(\Delta x^{(k)})) \\ = \; & Df(x^{(k)})^{-1} f(x^{(k+1)}) - \underbrace{Df(x^{(k)})^{-1} f(x^{(k)})}_{=-\Delta x^{(k)}} - \underbrace{Df(x^{(k)})^{-1} Df(x^{(k)})(\Delta x^{(k)})}_{=\Delta x^{(k)}} \\ = \; & Df(x^{(k)})^{-1} f(x^{(k+1)}) \quad =: \quad \widetilde{\Delta x}^{(k+1)}. \end{aligned}$$

Die Größe $\widetilde{\Delta x}^{(k+1)}$ wird auch "vereinfachter Newton-Schritt" genannt. Damit können wir ω schätzen durch

$$\tilde{\omega} = 2\frac{\|\widetilde{\Delta x}^{(k+1)}\|}{\|\Delta x^{(k)}\|^2}.$$

Beachte, dass dies nur eine Schätzung ist, weil die Ungleichung (3.9) eigentlich für alle $x, y \in \mathbb{R}^n$ gelten muss; hier testen wir sie nur für eine bestimmte Wahl. Aus (3.10) erhalten wir dann die Schätzung

$$\|x^{(k+1)} - x^*\| \leq \frac{\tilde{\omega}}{2}\|x^{(k)} - x^*\|^2 = \left(\frac{\tilde{\omega}}{2}\|x^{(k)} - x^*\|\right)\|x^{(k)} - x^*\|.$$

Das Verfahren konvergiert also gerade dann schnell, wenn $\frac{\tilde{\omega}}{2}\|x^{(k)} - x^*\|$ klein ist. Da wir $\|x^{(k)} - x^*\|$ allerdings nicht kennen, verwenden wir $\|x^{(k+1)} - x^{(k)}\|$ als Näherung. Dies ist durch die Dreiecksungleichung bzw. die umgekehrte Dreiecksungleichung gerechtfertigt, wenn $\frac{\tilde{\omega}}{2}\|x^k - x^*\|^2$ klein ist. Wir erhalten so den Schätzer für die Kontraktionsgeschwindigkeit

$$\Theta := \frac{\tilde{\omega}}{2}\|x^{(k+1)} - x^{(k)}\| = \frac{\|\widetilde{\Delta x}^{(k+1)}\|}{\|\Delta x^{(k)}\|}.$$

Damit können wir in Algorithmus 3.14 durch Berechnen von $\widetilde{\Delta x}^{(k+1)}$ in der Newton-Iteration in Schritt (1) und Festlegen von Konstanten $1 > \mu > \nu > 0$ (typische Konstanten sind z.B. $\mu = 0.9$ und $\nu = 0.2$ die folgenden Regeln definieren:

Falls $\Theta > \mu$: Verfahren voraussichtlich nicht konvergent, Abbruch der Iteration in (1)

Falls $\Theta < \nu$: sehr schnelle Konvergenz liegt vor, Erhöhung von σ in (2)

Eine mögliche Homotopie F für dieses Verfahren ist

$$F(x, \lambda) = f(x) - (1 - \lambda)f(y^{(0)})$$

für ein gegebenes $y^{(0)} \in \mathbb{R}^n$. Dann sind die Voraussetzungen des Algorithmus für dieses $y^{(0)}$ erfüllt. Das zu lösende Gleichungssystem in jedem Newton-Schritt ergibt sich dann als

$$Df(x)\Delta x + (f(x) - (1 - \lambda)f(y^{(0)})) = 0.$$

3.3 Beschränkte Optimierung

In den meisten Optimierungsaufgaben soll die Funktion f nicht einfach über ganz $\mathbb{R}^n$, sondern über eine Teilmenge $X \subset \mathbb{R}^n$ optimiert werden. Um die Menge X mathematisch zu beschreiben, nehmen wir an, dass diese durch zwei Funktionen

$$c : \mathbb{R}^n \to \mathbb{R}^{m_c}, \quad g : \mathbb{R}^n \to \mathbb{R}^{m_g}$$

als

$$X := \{x \in \mathbb{R}^n \,|\, c(x) = 0 \text{ und } g(x) \leq 0\}$$

definiert ist. Wie üblich schreiben wir $c = (c_1, \ldots, c_{m_c})^T$ mit $c_i : \mathbb{R}^n \to \mathbb{R}$ und $g = (g_1, \ldots, g_{m_g})^T$ mit $g_j : \mathbb{R}^n \to \mathbb{R}$ und die Funktionen müssen mindestens C^2 sein. Die durch c definierten Bedingungen an x werden als *Gleichungsbedingungen* bezeichnet, die durch g definierten als *Ungleichungsnebenbedingungen*. Die Optimierungsaufgabe

$$\min_{x \in \mathbb{R}^n} f(x) \quad \text{s.t.} \quad c(x) = 0,\, g(x) \leq 0 \tag{3.11}$$

wird als *beschränkte Optimierungsaufgabe* bezeichnet. Das Kürzel "s.t." steht dabei für "subject to", was man auf deutsch als "unter den Nebenbedingungen" übersetzen kann und kurz als "so dass" sprechen kann.

Optimierung mit Gleichungsnebenbedingungen

Analog zur Bedingung $\nabla f(x) = 0$ im unbeschränkten Fall wollen wir jetzt eine notwendige Optimalitätsbedingung herleiten. Falls nur Gleichungsnebenbedingungen existieren, also keine Funktion g existiert, können wir die bereits in der Analysis betrachtete Methode der Lagrange-Multiplikatoren anwenden.

Satz 3.15 Sei $f \in C^1(\mathbb{R}^n, \mathbb{R})$ mit $c : \mathbb{R}^n \to \mathbb{R}^{m_c}$, $m_c < n$ und $X := \{x \in \mathbb{R}^n \mid c(x) = 0\}$. Falls f in $x^* \in X$ ein lokales Minimum oder Maximum besitzt und $Dc(x^*)$ vollen Rang besitzt, so existiert ein Vektor $\lambda \in \mathbb{R}^{m_c}$ mit

$$Df(x^*) + \lambda^T Dc(x^*) = 0. \tag{3.12}$$

Der Vektor λ wird *Lagrange-Multiplikator* genannt.

Beweis: Die Matrix $Dc(x)$ besitzt vollen Rang und ist eine Matrix mit n Spalten und m_c Zeilen. Wegen $m_c < n$ existieren also m_c linear unabhängige Spalten. Durch Umnummerierung der x_i verschieben sich diese Spalten gerade; wir können daher so umnummerieren, dass gerade die m_c letzten Spalten linear unabhängig sind. Beachte, dass dies nur die Spalten von Df und Dc vertauscht, Gleichung (3.12) bleibt bei Umnummerierung der x_i also gültig. Teilen wir nun den Vektor x in der Form

$$x = \begin{pmatrix} z \\ y \end{pmatrix}$$

mit $z \in \mathbb{R}^{n-m_c}$ und $y \in \mathbb{R}^{m_c}$ auf und schreiben c in der Form $c(z, y)$, so erfüllt c die Bedingung des Satzes über implizite Funktionen. Teilen wir das lokale Minumum analog auf mittels $x^* = (z^*, y^*)$, so können wir Umgebungen V_1 von z^* und V_2 von y^* sowie eine stetig differenzierbare Funktion $b : V_1 \to V_2$ finden mit $c(z, b(z)) = 0$. Aus der Definition des lokalen Minimums unter Nebenbedingungen folgt dann, dass z^* ein lokales Minimum der stetig differenzierbaren Funktion

$$h(x) = f(z, b(z))$$

ist. Aus dem Satz über implizite Funktionen folgt

$$\begin{aligned} 0 \;=\; Dh(z^*) \;&=\; D_z f(z^*, b(z^*)) + D_y f(z^*, b(z^*)) Db(z^*) \\ &=\; D_z f(z^*, h(z^*)) - D_y f(z^*, b(z^*)) (D_y c(x^*))^{-1} D_z c(x^*). \end{aligned}$$

Setzen wir nun $\lambda = -\left[D_y f(z^*, b(z^*))(D_y c(x^*))^{-1}\right]^T$, so folgt

$$\begin{aligned} D_z f(z^*, b(z^*)) + \lambda^T D_z c(x^*) \;&=\; D_z f(z^*, b(z^*)) - D_y f(z^*, b(z^*)) (D_y c(x^*))^{-1} D_z c(x^*) \\ &=\; Dh(x^*) \;=\; 0 \end{aligned}$$

und

$$\begin{aligned} D_y f(z^*, b(z^*)) + \lambda^T D_y g(x^*) \;&=\; D_y f(z^*, h(z^*)) - D_y f(z^*, b(z^*)) (D_y c(x^*))^{-1} D_y c(x^*) \\ &=\; D_y f(z^*, b(z^*)) - D_y f(z^*, b(z^*)) \;=\; 0. \end{aligned}$$

Zusammen gilt also

$$Df(x^*) = (D_y f(z^*, y^*), D_z f(z^*, y^*)) = -(\lambda^T D_z c(x^*), \lambda^T D_y c(x^*)) = -\lambda^T Dc(x*).$$

□

Bemerkung 3.16 Beachte, dass Gleichung (3.12) nicht sicher stellt, dass $c(x^*) = 0$ gilt. Will man also Kandidaten für Extremwerte unter Nebenbedingungen ausrechnen, muss man $c(x^*) = 0$ als weitere Gleichung(en) zur Bestimmung von x^* und λ hinzufügen. Die notwendigen Optimalitätsbedingungen lauten also vollständig

$$\begin{aligned} Df(x^*) + \lambda^T Dc(x^*) &= 0 \\ c(x^*) &= 0 \end{aligned} \tag{3.13}$$

und bilden ein Gleichungssystem mit n linearen und m_c nichtlinearen Gleichungen, sowie $n + m_c$ Unbekannten $x_1, \ldots, x_n, \lambda_1, \ldots, \lambda_{m_c}$. Diese Gleichungen werden als KKT-Bedingungen (nach Karush, Kuhn und Tucker) bezeichnet.

□

Beispiel 3.17 Wir betrachten die Funktion $f(x) = x^T Ax + b^T x + c$ mit

$$A = \begin{pmatrix} -3 & 1 \\ 1 & 2 \end{pmatrix}, \quad b = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \quad c = 5$$

(siehe Abb. 3.1) und der Gleichungsnebenbedingung $g(x) = 0$ mit $c(x) = x_1^2 + x_2^2 - 1$.

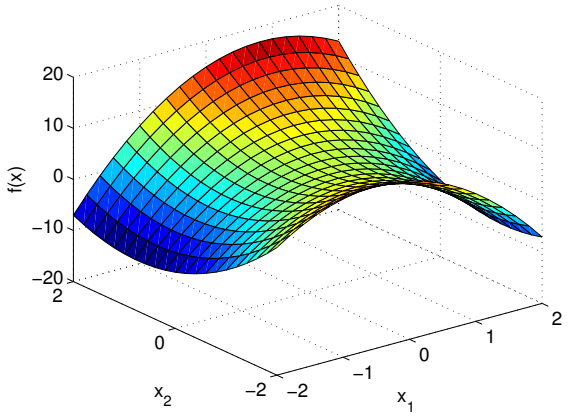

Abbildung 3.1: Graph der Funktion $f(x) = x^T Ax + b^T x + c$ mit A, b, c aus Beispiel 3.17

Wir suchen also die Extremwerte von f auf dem Einheitskreis, also der Menge aller x mit $\|x\|_2 = 1$. Die Dimensionen lauten hier $n = 2$ und $m = 1$ und es gilt

$$Df(x) = (2Ax + b)^T \quad \text{sowie} \quad Dc(x) = (2x_1, 2x_2).$$

Die Bedingung, dass Dc vollen Rang hat, ist hier also für $x \neq 0$ erfüllt. Damit ergibt sich (3.13) zu

$$\begin{aligned} -6x_1 + 2x_2 + 1 &= -\lambda 2x_1 \\ 2x_1 + 4x_2 + 1 &= -\lambda 2x_2 \\ x_1^2 + x_2^2 - 1 &= 0 \end{aligned}$$

Die Lösung des linearen Gleichungssystems aus den ersten beiden Gleichungen in Abhängigkeit von λ ergibt

$$x_1 = \frac{1}{2}\frac{-\lambda - 1}{\lambda^2 - \lambda - 7}, x_2 = \frac{1}{2}\frac{-\lambda + 4}{\lambda^2 - \lambda - 7}.$$

Eingesetzt in die weitere Gleichung $x_1^2 + x_2^2 - 1 = 0$ erhalten wir

$$\frac{1}{4}\frac{2\lambda^2 - 6\lambda + 17}{(\lambda^2 - \lambda - 7)^2} - 1 = 0,$$

was gerade die Nullstellen der Polynoms

$$P(\lambda) = 4\lambda^4 - 8\lambda^3 - 54\lambda^2 + 62\lambda + 179$$

als Lösungen besitzt. Die Berechnung der Nullstellen (mit MAPLE) liefert die näherungsweisen Werte

$$\lambda_1 = -2.779408674,\ \lambda_2 = -1.605041266,\ \lambda_3 = 2.793395794,\ \lambda_4 = 3.591054146$$

mit den zugehörigen Extremalstellen

$$\begin{pmatrix} 0.2538732892 \\ 0.9672374900 \end{pmatrix}, \begin{pmatrix} -0.1073224411 \\ -0.9942242700 \end{pmatrix}, \begin{pmatrix} 0.9529537485 \\ -0.3031157474 \end{pmatrix}, \begin{pmatrix} -0.9960563225 \\ 0.08872321925 \end{pmatrix}.$$

Abbildung 3.1 legt nahe, warum es gerade vier Extremalstellen mit dieser Vorzeichenstruktur gibt. □

In der Praxis wird man das Gleichungssystem (3.12) natürlich nicht per Hand sondern mit einem Algorithmus lösen, beispielsweise mit dem sogenannten Lagrange-Newton-Verfahren. Dazu führt man die Lagrange-Funktion

$$L(x, \lambda) = f(x) + \lambda^T c(x)$$

ein, mit $L : \mathbb{R}^n \times \mathbb{R}^{m_c} \to \mathbb{R}$. Wegen

$$DL(x, \lambda) = \left(\frac{\partial}{\partial x}L(x, \lambda), \frac{\partial}{\partial \lambda}L(x, \lambda)\right) = \left(Df(x) + \lambda^T Dc(x), c(x)^T\right)$$

sind die KKT-Bedingungen äquivalent zu der Gleichung

$$DL(x, \lambda) = 0.$$

Der Newton-Schritt für diese Gleichung (angewendet in der üblichen Zeilenvektorform, also auf $DL(x, \lambda)^T = 0$) ergibt sich aus der Lösung des linearen Gleichungssystems

$$H_L(x, \lambda)(\Delta x, \Delta\lambda) = -DL(x, \lambda)^T$$

($H_L(x,\lambda)$ ist die Hesse-Matrix von L bezüglich der Variablen (x,λ)). Mit $L_x = \frac{\partial}{\partial x}L$ und $L_{xx} = \frac{\partial^2}{\partial x^2}L(x,\lambda)$ kann man dies schreiben als

$$\begin{pmatrix} L_{xx}(x,\lambda) & Dc(x)^T \\ Dc(x) & 0 \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta\lambda \end{pmatrix} = \begin{pmatrix} L_x(x,\lambda)^T \\ c(x) \end{pmatrix}.$$

Hierbei ist

$$L_{xx}(x,\lambda) = \frac{\partial}{\partial x}L_x(x,\lambda) = \frac{\partial}{\partial x}\left[Df(x) + \lambda^T Dc(x)\right] = H_f(x) + \sum_{i=1}^{m_c} \lambda_i H_{c_i}(x).$$

Das Gleichungssystem enthält also gerade die KKT-Bedingungen des Optimierungsproblems

$$\min_{v\in\mathbb{R}^n} f(x) + L_x(x,\lambda)v + \frac{1}{2}L_{xx}(x,\lambda)(v,v) \quad \text{s.t. } Dc(x)v + c(x) = 0,$$

das eine eindeutige Lösung besitzt, falls $Dc(x)$ vollen Rang hat und $L_{xx}(x,\lambda)$ auf dem Kern von $Dc(x)$ positiv definit ist.

Um das Verfahren zu globalisieren, wird eine Merit-Funktion definiert, so dass wir die Schrittweitensuche des SQP-Verfahrens auf dieses Problem übertragen können. Eine übliche Wahl dafür ist

$$m(x) = f(x) + \gamma\|c(x)\|.$$

Optimierung mit Gleichungs- und Ungleichungsnebenbedingungen

Falls zusätzlich Ungleichungsnebenbedingungen vorliegen, müssen die KKT-Bedingungen angepasst werden. Dazu nennen wir eine Ungleichungsbedingung g_j *aktiv* in einem Punkt $x \in X$, falls $g_j(x) = 0$ gilt und *inaktiv*, falls $g_j(x) < 0$ gilt. Für die KKT-Bedingungen ist nun wichtig, welche Bedingungen im Minimum x^* aktiv sind. Wir definieren die *aktive* und die *inaktive Menge* $\mathcal{A},\mathcal{I} \subset \{1,\dots,m_g\}$ in einem lokalen Minimum als

$$\mathcal{A} := \{j \in \{1,\dots,m_g\} \,|\, g_j(x^*) = 0\}, \qquad \mathcal{I} := \{j \in \{1,\dots,m_g\} \,|\, g_j(x^*) < 0\}.$$

Offenbar bleibt x^* ein lokales Minimum, wenn wir alle g_j mit $j \in \mathcal{A}$ in Gleichungsnebenbedingungen umwandeln und alle g_j mit $j \in \mathcal{I}$ weglassen. Es sei $\tilde{c}$ die Funktion der neuen Gleichungsnebenbedingungen c_i und g_j mit $i = 1,\dots,m_c$ und $j \in \mathcal{A}$ und $m_a = m_c + |\mathcal{A}|$ die Anzahl der Komponentenfunktionen in $\tilde{c}$. Falls dann $D\tilde{c}(x^*)$ vollen Rang besitzt, folgt aus Satz 3.15 die Existenz von $\tilde{\lambda} \in \mathbb{R}^{m_a}$ mit

$$Df(x^*) + \tilde{\lambda}^T D\tilde{c}(x^*) = 0. \tag{3.14}$$

Zusätzlich gilt natürlich $c_i(x^*) = 0$ und $g_j(x^*) \le 0$ für $i = 1,\dots,m_c$, $j = 1,\dots,m_g$.

Wir machen nun die folgende weitere Annahme an $D\tilde{c}$, die sogenannte *Mangasarian-Fromowitz Constraint Qualification*: Es existiert ein $v \in \mathbb{R}^n$ mit

$$Dc(x^*)v = 0 \quad \text{und} \quad Dg_j(x^*)v < 0 \text{ für alle } j \in \mathcal{A}. \tag{3.15}$$

Für jedes solche v muss f in Richtung v zunehmen, denn ansonsten könnte x^* kein Minimum sein (das ist anschaulich einsichtig, der formale Beweis ist aber technisch aufwändig), also ist $Df(x^*)v \geq 0$. Wegen (3.14) gilt für dieses v nun aber

$$Df(x^*)v + \sum_{i=m_c+1}^{m_a} \tilde{\lambda}_i D\tilde{c}_i(x^*)v = 0.$$

Damit und mit (3.15) kann man mit einem wiederum etwas aufwändigen Beweis zeigen, dass die Komponenten von $\tilde{\lambda}$ für $i = m_c + 1, \ldots, m_a$ die Ungleichung $\tilde{\lambda}_i \geq 0$ erfüllen.

Setzen wir nun $\lambda_i := \tilde{\lambda}_i$ für $i = 1, \ldots, m_c$, $\nu_j := \tilde{\lambda}_k$ mit $\tilde{c}_k = g_j$ für alle $j \in \mathcal{A}$ und $\nu_j := 0$ für $j \in \mathcal{I}$, so folgt

$$\begin{aligned} Df(x^*) + \lambda^T Dc(x^*) + \nu^T Dg(x^*) &= 0 \\ c(x^*) &= 0 \\ g(x^*) &\leq 0 \\ \nu &\geq 0 \\ \nu^T g(x^*) &= 0 \end{aligned} \tag{3.16}$$

Dies sind die KKT-Bedingungen für das volle Problem. Ein Punkt x^*, der diese Bedingungen erfüllt, wird als KKT-Punkt bezeichnet.

Tatsächlich reicht es aus, die Existenz von v mit (3.15) und vollen Rang von $Dc(x^*)$ anzunehmen, um die KKT-Bedingungen zu folgern; unter diesen Bedingungen kann man aber nicht auf Satz 3.15 zurückgreifen, deswegen haben wir die stärkere Bedingung verwendet, dass $D\tilde{c}(x^*)$ vollen Rang besitzt.

Beispiel 3.18 Wir betrachten noch einmal das Problem aus Beispiel 3.17, jetzt aber mit der Nebenbedingung $\|x\|^2 \leq 1$, die wir mit

$$g(x) = x_1^2 + x_2^2 - 1$$

als $g(x) \leq 0$ schreiben können. Wir könnten jetzt versuchen, direkt die (Un-)Gleichungen (3.16) zu lösen. Einfacher ist es aber, wenn wir uns eine Kandidatin für die aktive Menge vorgeben. Da wir die aktive Menge nicht kennen, probieren wir alle Möglichkeiten durch. Da g nur eine Komponente hat, gibt es nur die beiden Möglichkeiten $\mathcal{A} = \{1\}$ und $\mathcal{A} = \emptyset$.

Im Fall $\mathcal{A} = \{1\}$ erhalten wir genau die Bedingungen aus Beispiel 3.17, nur dass λ jetzt ν heißt. Die Zusatzbedingung $\nu \geq 0$ ist für die dritte und vierte Extremalstelle in Beispiel 3.17 erfüllt, die zugehörigen x^* sind also Kandidaten für Minima, was gemäß Abb. 3.1 auch stimmt.

Im Fall $\mathcal{A} = \{0\}$ haben wir außer Df keine weiteren Terme in der ersten Gleichung mehr, wir suchen also nach Nullstellen x^* der Ableitung Df mit $g(x^*) \leq 0$. Die einzige solche Stelle ist der Punkt $x^* = (0, 0)$. Dieser ist auch tatsächlich eine Extremalstelle, der die KKT-Bedingungen erfüllt. Es ist aber weder ein Minimum noch ein Maximum sondern ein Sattelpunkt, was man durch Analyse der Hesse-Matrix H_f von f ermitteln kann.

Das Beispiel zeigt, dass die KKT-Bedingungen genau wie die bereits aus der Schule bekannte notwendigen Bedingung "$f'(x) = 0$" nur kritische Punkte charakterisieren. Diese müssen

nicht unbedingt Minima oder Maxima sein. Allerdings bieten die KKT-Bedingungen in der Form (3.16) wegen der Vorzeichenbedingung $\nu \geq 0$ bei kritischen Punkten am Rand die Möglichkeit, Minima und Maxima zu unterscheiden, wie der Fall $\mathcal{A} = \{1\}$ zeigt. □

Zur Lösung von (3.16) gibt es unterschiedliche algorithmische Strategien. Zu deren Darstellung ist es hilfreich, das Problem erst einmal in eine einfache Standardform zu bringen. Dazu führen wir sogenannte *Schlupfvariablen* s ein. Wir schreiben die Ungleichungebedingung um als

$$g(x) \leq 0 \quad \Leftrightarrow \quad g(x) + s = 0,\ s \geq 0,$$

wobei $s \in \mathbb{R}^{m_g}$ und "$s \geq 0$" komponentenweise zu verstehen ist. Schreiben wir dann die Funktionen c und $g + s$ in eine gemeinsame Funktion $d(x, s)$ mit $d : \mathbb{R}^n \times \mathbb{R}^{m_g} \to \mathbb{R}^{m_c+m_g}$, so können wir das Problem 3.11 in der Form

$$\min_{\substack{x\in\mathbb{R}^n \\ s\in\mathbb{R}^{m_g}}} f(x) \quad \text{s.t.} \quad d(x, s) = 0,\ s \geq 0$$

schreiben. Die KKT-Bedingungen (3.16) vereinfachen sich dann zu

$$\begin{aligned} Df(x^*) + \lambda^T Dd(x^*, s^*) - \nu &= 0 \\ d(x^*, s^*) &= 0 \\ s^* &\geq 0 \\ \nu &\geq 0 \\ \nu^T s^* &= 0 \end{aligned} \tag{3.17}$$

Auf Basis dieser vereinfachten KKT-Bedingungen können wir nun die Algorithmen erläutern.

Aktive-Mengen Strategien

Die aktive-Mengen Strategie nutzt aus, dass das Problem bei gegebener aktiver Menge ein Problem mit reinen Gleichungs-Nebenbedingungen ist, das wir z.B. mit Lagrange-Newton lösen können. Ausgehend von einer Startschätzung $\mathcal{A}_0$ wird die aktive Menge nach jedem Lösen des gleichungsbeschränkten Problems aktualisiert.

Algorithmus 3.19 (Aktive-Mengen Strategie) Eingabe: $\mathcal{A}_0 \subset \{1, \ldots, m_g\}$

(0) Setze $k := 0$

(1) Löse das Problem

$$\min_{\substack{x\in\mathbb{R}^n \\ s\in\mathbb{R}^{m_g}}} f(x) \quad \text{s.t.} \quad d(x, s) = 0,\ s_{\mathcal{A}_k} = 0,$$

wobei $s_{\mathcal{A}_k}$ der Teilvektor von s mit den Komponenten s_j, $j \in \mathcal{A}_k$ ist.

(2) Falls die KKT-Bedingungen (3.17) (bis auf eine vorgegebene Toleranz) erfüllt sind:
Ende des Algorithmus, KKT-Punkt gefunden

Sonst:

(3) Setze $\mathcal{A}_{k+1} := \mathcal{A}_k$

(4) Für $j = 1, \ldots, m_c$

(5) Falls $j \notin \mathcal{A}_{k+1}$ und $s_j < 0$ (s_j unzulässig):
nehme j in $\mathcal{A}_{k+1}$ auf

(6) Falls $j \in \mathcal{A}_{k+1}$ und $\nu_j < 0$ ($s_j = 0$ verhindert Optimalität):
entferne j aus $\mathcal{A}_{k+1}$

(7) Ende der j-Schleife

(8) Setze $k := k + 1$ und gehe zu (1)

□

Innere-Punkte-Verfahren

Im Innere-Punkte-Verfahren ist die Idee, die Ungleichungsbeschränkungen $s \leq 0$ komplett wegzulassen und stattdessen die Kostenfunktion f so zu verändern, dass die Kosten für $s > 0$ unendlich groß werden. Dabei soll f auf dem Inneren von X aber differenzierbar bleiben, damit man das dann unbeschränkte Optimierungsproblem effizient lösen kann.

Der übliche Ansatz für solch ein Optimierungsproblem ist

$$\min_{x \in \mathbb{R}^n} f(x) - \mu \sum_{j=1}^{m_c} \log(s_i) \quad \text{s.t.} \quad d(x, s) = 0$$

mit einem Parameter $\mu > 0$. Für festes $\mu > 0$ ergibt dieses Problem stets nur eine Näherungslösung für das Originalproblem, aber unter geeigneten Bedingungen kann man beweisen, dass diese Näherungslösung für $\mu \to 0$ gegen die Originallösung konvergiert. Allerdings ist das Problem für $\mu \approx 0$ numerisch sehr schwer zu lösen. Deswegen verwendet man üblicherweise ein Lagrange-Newton-Verfahren mit Pfadverfolgung für $\mu \to 0$.

Die KKT-Bedingungen für das Innere-Punkte Problem lauten (mit $\lambda = (\hat{\lambda}, \tilde{\lambda})$

$$\begin{aligned} Df(x) + \hat{\lambda}^T d_x(x, s) &= 0 \\ -\mu \left(\tfrac{1}{s_1}, \ldots, \tfrac{1}{s_{m_g}} \right)^T + \tilde{\lambda} &= 0 \\ d(x, s) &= 0. \end{aligned} \tag{3.18}$$

Hierbei entsteht die erste Zeile aus der Ableitung der Lagrangefunktion nach x und die zweite aus der Ableitung nach s. In der Implementierung multipliziert man üblicherweise die zweite Zeile mit $S = \operatorname{diag}(s_1, \ldots, s_{m_c})$ und erhält so die Bedingungen

$$\begin{aligned} Df(x) + \hat{\lambda}^T d_x(x, s) &= 0 \\ -\mu \mathbf{1} + \left(\tilde{\lambda}_1 s_1, \ldots, \tilde{\lambda}_{m_g} s_{m_g} \right) &= 0 \\ d(x, s) &= 0 \end{aligned} \tag{3.19}$$

mit $\mathbf{1} = (1, \ldots, 1)^T \in \mathbb{R}^n$. Diese Formulierung vermeidet die für kleine s_j numerisch ungünstige Verwendung von $1/s_j$.

Kapitel 4

Approximation von Funktionen

In diesem Kapitel werden wir uns damit befassen, wie man Funktionen numerisch darstellen kann. Da dies in den allermeisten Fällen nicht exakt möglich ist, spricht man von der Approximation von Funktionen. Wir haben bereits in der Einführung in die Numerik gesehen, dass z.B. Polynome und Splines hierfür geeignet sind. Hier wollen wir zunächst die theoretischen Grundlagen in deutlich allgemeinerer Form als in der Einführung betrachten und dann ausgewählte weitere Approximationsverfahren behandeln.

4.1 Bestapproximation

Aus mathematischer Sicht kann man die Funktionen, die man approximieren möchte, als Funktionenraum, also als einen unendlich dimensionalen Vektorraum X auffassen, den wir in diesem Kapitel immer als vollständig und normiert voraussetzen. Beispiele sind z.B. die stetigen Funktionen $X = C(\Omega, \mathbb{R}^n)$, $\Omega \subset \mathbb{R}^m$, mit Norm $\|f\|_X = \|f\|_\infty = \sup_{x\in\Omega} \|f(x)\|$ oder die p-mal stetig differenzierbaren Funktionen $X = C^p(\Omega, \mathbb{R}^n)$ mit Norm $\|f\|_X = \sup_{k=0,\dots,p} \|f^{(k)}\|_\infty$.

Die im Computer numerisch darstellbaren Funktionen sind hingegen Elemente eines endlich dimensionalen Unterraums $X_k \subset X$. Ein Beispiel ist der $n+1$-dimensionale Raum der Polynome vom Grad $\leq n$, der einen Unterraum von $X = C^p(\Omega, \mathbb{R})$ für alle $p \in \mathbb{N}$ und $\Omega = [a, b] \subset \mathbb{R}$ darstellt. Dieser Raum wird bei uns oft einen Index k haben, weil man typischerweise ganze Folgen solcher Räume betrachtet, z.B. die Räume $\mathcal{P}_k$ der Polynome von Grad $\leq k$.

Die Frage nach der Bestapproximation ist nun, wie gut ein Element $f \in X$ durch ein Element aus X_k approximiert werden kann. Man betrachtet also die Größe

$$\min_{f_k \in X_k} \|f - f_k\|_X,$$

den sogenannten Fehler der Bestapproximation. Wie der folgende Satz zeigt, existiert dieses Minimum immer.

Satz 4.1 Falls X_k ein endlichdimensionaler Teilraum von X ist, existiert ein $\hat{f}_k \in X_k$ mit

$$\|f - \hat{f}_k\|_X \leq \|f - f_k\|_X \quad \text{für alle } f_k \in X_k.$$

Beweis: Die Funktion $g(f_k) := \|f - f_k\|_X$ ist stetig auf X_k, weil die Norm eine stetige Funktion auf X und $X_k \subset X$ ist. Daher sind die Niveaumengen

$$L_\alpha := \{f_k \in X_k \,|\, g(f_k) \leq \alpha\}$$

abgeschlossen. Wegen $0 \in X_k$ ist $0 \in L_\alpha$ für $\alpha \geq \|f\|_X$, damit sind die L_α nicht leer für alle hinreichend großen α. Sie sind zudem beschränkt, denn für $\|f_k\|_X > \|f\|_X + \alpha$ gilt $g(f_k) = \|f - f_k\|_X \geq \|f_k\|_X - \|f\| > \alpha$ und damit $f_k \notin L_\alpha$. Weil X_k endlich dimensional ist, sind die L_k damit kompakt. Folglich existiert in jedem solchen L_α ein Minimierer, der gerade das gesuchte $\hat{f}_k$ ist. □

Die Fragen, die man sich nun stellt, sind z.B.

- Konvergiert der Fehler für eine Folge von Räumen X_k gegen 0, gilt also

$$\lim_{k\to\infty} \min_{f_k \in X_k} \|f - f_k\|_X = 0?$$

- Wie groß ist der Fehler der Bestapproximation für festes k und wie hängt er von der Wahl von f und X_k ab?

- Wie viel schlechter ist ein Algorithmus zur Berechnung einer Approximation $f_k \in X_k$ im Vergleich zur Bestapproximation?

Für die Polynominterpolation haben wir in der Einführung in die Numerik bereits eine Fehlerabschätzung hergeleitet. Für Polynome P vom Grad $\leq k$ gilt gerade

$$\|f - P\|_\infty \leq \frac{\|f^{(k+1)}\|_\infty}{(k+1)!}(b-a)^{k+1}. \tag{4.1}$$

Mit $X = C([a,b], R)$ und $Y = C^{k+1}([a,b], \mathbb{R})$ folgt daraus[1]

$$\|f - P\|_X \leq \frac{(b-a)^{k+1}}{(k+1)!}\|f\|_Y.$$

Dies ist die typische Form einer Ungleichung für den Fehler der Bestapproximation. Typisch dabei ist, dass links und rechts unterschiedliche Normen stehen, wobei die Norm rechts stärker ist (also für gleiche Argumente größere Werte liefert) und höhere Regularität von f verlangt als die Norm links (rechts $k+1$ mal stetige Differenzierbarkeit, links nur Stetigkeit). Diesen Unterschied nennt man *Regularitätslücke*.

Im Allgemeinen sucht man nach einer Konstanten $A(k, X, Y) > 0$, so dass die Ungleichung

$$\|f - \hat{f}_k\|_X \leq A(k, X, Y)\|f\|_Y,$$

die sogenannte Jackson-Ungleichung gilt.

[1] Beachte, dass wir hier den Term auf der rechten Seite durch einen u.U. deutlich größeren Ausdruck ersetzen. Das ist hier für die abstrakte Formulierung nützlich, bedeutet aber natürlich, dass wir die Abnschätzung schlechter machen. Wenn wir in Abschnitt 4.5 mehrdimensionale Polynome genauer betrachten, werden wir die bessere Abschätzung verwenden.

Hierbei möchte man $A(k, X, Y) \to 0$ für $k \to \infty$ oder noch besser asymptotische Abschätzungen der Form

$$A(k, X, Y) = O(k^{-\alpha}) \quad \text{oder} \quad A(k, X, Y) = O(\rho^k)$$

für $\alpha > 0$ oder $\rho \in (0, 1)$ haben. In jedem Fall bedeutet dies, dass die Abschätzung unabhängig von x ist, was in manchen Fällen nicht möglich ist. In diesen Fällen kann man manchmal immer noch

$$\lim_{k\to\infty} \|f - \hat{f}_k\|_X = 0 \quad \text{für alle } x \in X$$

erreichen. Wenn dies gilt, sagt man, dass $\bigcup_{k\in\mathbb{N}} X_k$ dicht in X liegt.

Beispiel 4.2 (i) Falls $f \in Y = C^\infty([a, b], \mathbb{R})$, so folgt für die Polynominterpolation mit $X_k = \mathcal{P}_k$, $X = C([a, b], \mathbb{R})$ und $Y = C^\infty$

$$A(k, X, Y) = O(\rho^k)$$

für jedes $\rho \in (0, 1)$. Dies gilt, weil für jedes solche ρ die Ungleichung $(b - a)/k < \rho$ gilt für hinreichend großes k, woraus die Existenz von $C > 0$ mit $(b - a)^{k+1}/(k + 1)! < C\rho^k$ folgt. Beachte aber, dass C unbeschränkt wächst, wenn ρ gegen Null geht.

(ii) Aus der Fehlerabschätzung für die kubischen Splineinterpolation in Satz 3.32 der Einführung in die Numerik folgt, dass für Splines mit k äquidistanten Stützstellen Δ_k auf $[a, b]$ die Abschätzung

$$A(k, X, Y) = O(k^{-4})$$

gilt, für $X = C([a, b], \mathbb{R})$, $X_k = S_{\Delta_k,3}$ und $Y = C^4([a, b], \mathbb{R})$. □

Beispiele für Dichtheitsresultate werden wir im übernächsten Abschnitt kennen lernen.

Für viele Probleme beobachtet man die folgenden beiden Tatsachen:

- Je reichhaltiger X_k und je größer die Regularitätslücke von Y nach X ist, desto kleiner wird $A(k, X, Y)$.
- Für $Y = X$ ist in der Regel nur ein Dichtheitsresultat aber keine quantitative Abschätzung möglich.

4.2 Projektionen

Eine Approximation einer Funktion $f \in X$ in X_k kann oft durch eine Projektion berechnet werden. Eine Abbildung $P : X \to X_k$ heißt *Projektion*, wenn $Pf_k = f_k$ gilt für alle $f_k \in X_k$. Eine Projektion heißt linear, wenn P zudem eine lineare Abbildung ist.

Beispiel 4.3 Ein Beispiel für eine lineare Projektion ist die Polynominterpolation auf $[a, b]$ mit einer festen Stützstellenmenge $a \le x_0 < x_1 < \ldots < x_k \le b$. Für die Lagrange-Polynome

$$L_i(x) = \prod_{\substack{j=0 \\ j\neq i}}^{k} \frac{x - x_j}{x_i - x_j}$$

ist die durch die Interpolation definierte Abbildung von X nach X_k dann gegeben durch

$$Pf = \sum_{i=0}^{k} L_i(x) f(x_i).$$

Die Abbildung ist offensichtlich linear und sie ist eine Projektion, weil das Interpolationspolynom eindeutig ist, die Interpolation eines Polynoms $P_k \in \mathcal{P}_k$ also gerade P_k selbst als Interpolationspolynom liefert. Das Gleiche gilt analog für die Splineinterpolation. □

Der folgende Satz gibt eine Abschätzung, wie weit die Approximation mit einer Projektion von der Bestapproximation entfernt ist.

Satz 4.4 Sei X_k ein endlichdimensionaler Unterraum von X und $P : X \to X_k$ eine lineare Projektion, für die eine Konstante $\Lambda_k > 0$ existiert, so dass die sogenannte *Stabilitätsabschätzung*

$$\|Pf\|_X \leq \Lambda_k \|f\|_X$$

für alle $f \in X$ gilt. Dann gilt für alle $f \in X$ für die Bestapproximation $\hat{f}_k$ von f die Abschätzung

$$\|f - Pf\|_X \leq (1 + \Lambda_k)\|f - \hat{f}_k\|_X.$$

Man spricht in diesem Fall von *Quasi-Optimalität* der Approximation.

Beweis: Es gilt

$$\begin{aligned} \|f - Pf\|_X &= \|f - \hat{f}_k + \hat{f}_k - Pf\|_X = \|f - \hat{f}_k + P\hat{f}_k - Pf\|_X \\ &= \|f - \hat{f}_k + P(\hat{f}_k - f)\|_X \leq \|f - \hat{f}_k\|_X + \Lambda_k \|f - \hat{f}_k\|_X. \end{aligned}$$

Daraus folgt direkt die Behauptung. □

Beispiel 4.5 Wir betrachten noch einmal die Polynominterpolation auf $[a, b]$, mit der Norm $\|\cdot\|_X = \|\cdot\|_\infty$. Aus der obigen Darstellung des Interpolationspolynoms mit Lagrange-Polynomen folgt

$$\|Pf\|_X = \left\| \sum_{i=0}^{k} L_i(x) f(x_i) \right\| \leq \sum_{i=0}^{k} |L_i(x)|\,|f(x_i)| \leq \sum_{i=0}^{k} |L_i(x)| \|f\|_X.$$

Die Konstante Λ_k ergibt sich also als

$$\Lambda_k := \left\| \sum_{i=0}^{k} |L_i| \right\|_\infty.$$

Diese Konstante — die sogenannte Lebesgue-Konstante — ist uns bereits bei der Untersuchung der Kondition der Polynominterpolation begegnet. Tabelle 4.1 zeigt ausgewählte Werte für Λ_k für verschiedene Stützstellen im Intervall $[-1, 1]$.

Für die Λ_k kann man das Folgende beweisen, siehe Theorem 15.2 in Trefethen [6]:

- Für alle Arten von Stützstellen gilt $\Lambda_k \geq O(\ln k)$
- Für Tschebyscheff-Stützstellen gilt $\Lambda_k = O(\ln k)$
- Für äquidistante Stützstellen gilt $\Lambda_k = O(2^k)$

□

k	Λ_k für äquidistante Stützstellen	Λ_k für Tschebyscheff-Stützstellen
5	3.11	2.10
10	29.89	2.49
15	512.05	2.73
20	10986.53	2.90
60	$2.97 \cdot 10^{15}$	3.58
100	$1.76 \cdot 10^{27}$	3.90

Tabelle 4.1: Λ_k für verschiedene Stützstellen

4.3 Dichtheitsresultate

Aussagen über die Dichtheit von Funktionenräumen der Form $\bigcup_{k\in\mathbb{N}} X_k$ in X für endliche Funktionenräume $X_k \subset X$ bilden die Grundlage für die Approximationstheorie. Wir werden in diesem Abschnitt einige solche Resultate vorstellen und beginnen mit dem Raum $C^\infty(\Omega, \mathbb{R})$. Dafür benötigen wir zunächst die Definitionen einiger Normen und Funktionenräume.

Sei $\Omega \subset \mathbb{R}^n$ offen und sei $f : \Omega \to \mathbb{R}$ eine i-mal differenzierbare Funktion mit Ableitungen $f^{(i)}$. Beachte, dass die Ableitung im Punkt x eine Multilinearform $f^{(i)}(x) : (\mathbb{R}^n)^i \to \mathbb{R}$ darstellt. Für diese definieren wir die Norm

$$\|f^{(i)}(x)\| := \sup_{\|v\|=1} |f^{(i)}(x)(v, v, \ldots, v)|.$$

Damit kann man für k-mal differenzierbare Funktionen die sogenannten *Sobolev-Normen*

$$\|f\|_{W^{k,p}} = \sqrt[p]{\sum_{i=0}^{k} \int_\Omega \|f^{(i)}(x)\|^p dx}$$

definieren.

Wir definieren zudem für $p \in \mathbb{N}$ bzw. $k \in \mathbb{N}$ die Funktionenräume

$$\begin{aligned}
L_p(\Omega) &:= \left\{ [f]_\mathcal{N} : \Omega \to \mathbb{R} \,;\, \int_\Omega |f(x)|^p dx \text{ existiert und ist endlich} \right\} \\
C^k(\overline{\Omega}) &:= \{ f : \overline{\Omega} \to \mathbb{R} \,:\, f^{(i)} \text{ existiert und ist stetig für alle } i = 0, \ldots, k \} \\
C^\infty(\overline{\Omega}) &:= \{ f : \overline{\Omega} \to \mathbb{R} \,:\, f^{(i)} \text{ existiert und ist stetig für alle } i \in \mathbb{N} \}
\end{aligned}$$

Hierbei bezeichnet $[f]_\mathcal{N}$ die Äquivalenzklasse von Funktionen, die sich nur auf einer Lebesgue-Nullmenge von f unterscheiden.

Wir erinnern daran, dass ein linearer Unterraum $Y \subset X$ dicht in X liegt, wenn für alle $x \in X$ eine Folge (y_k) in Y existiert mit

$$\lim_{k\to\infty} \|y_k - x\|_X = 0,$$

was äquivalent zu $\overline{Y} = X$ ist.

Dichtheit von C^∞

Der Raum $C^\infty(\Omega)$ liegt dicht in den anderen oben definierten Funktionenräumen, wofür wir hier einen Beweis skizzieren.

Dazu definieren wir für $\varepsilon > 0$ die C^∞-Funktion

$$g_\varepsilon(x) := \begin{cases} c(\varepsilon) \exp\left(-\frac{1}{1-|x/\varepsilon|^2}\right), & \text{falls } |x| \leq \varepsilon \\ 0, & \text{sonst} \end{cases},$$

wobei die Konstante $c(\varepsilon) > 0$ so gewählt ist, dass $\int_\Omega g_\varepsilon(x)dx = 1$ gilt. Für $f \in L_p(\Omega)$ bzw. $f \in C^k(\overline{\Omega})$ definiert man dann über die sogenannte Faltung die Funktion

$$f_\varepsilon(x) := \int_\Omega g_\varepsilon(x-y)f(y)dy.$$

Dann beweist man zum einen, dass $f_\varepsilon \in C^\infty(\overline{\Omega})$ gilt und zum anderen, dass die Konvergenzen

$$\lim_{\varepsilon \to 0} \|f_\varepsilon - f\|_{L_p} \text{ für alle } f \in L_p(\Omega)$$

und

$$\lim_{\varepsilon \to 0} \|f_\varepsilon - f\|_{C^k} \text{ für alle } f \in C^k(\overline{\Omega})$$

gelten. Damit ist die Dichtheit bewiesen.

Hierbei ist zu beachten, dass die Aussage für $C^k(\overline{\Omega})$ nur für kompaktes $\overline{\Omega}$ gilt. Für $L_p(\Omega)$ ist die Aussage auch dann richtig, wenn man $C^\infty(\Omega)$ auf $C_0^\infty(\Omega)$ einschränkt, d.h. auf die C^∞-Funktionen mit kompaktem Träger.

Als Beispiel betrachte die Funktion $f(x) = 1/\sqrt{|x|}$ für $x \neq 0$ mit $f(0) = 0$. Offenbar ist diese Funktion in $L_1((-1,1))$ (denn die Stammfunktion ist $\pm 2\sqrt{|x|}$), aber nicht in $C([-1,1])$, da sie in $x = 0$ offenbar nicht stetig ist und nahe $x = 0$ zudem unbeschränkt. Die obige Konstruktion schneidet den unbeschränkten Bereich nun nahe $x = 0$ differenzierbar ab, der Fehler in der L_1-Norm ist dann gerade die Größe der abgeschnittenen Fläche, die gegen Null konvergiert, wenn man immer näher bei $x = 0$ abschneidet.

Dichtheit von Polynomen

Wir skizzieren nun den Beweis dafür, dass jede C^∞-Funktion durch Polynome beliebig gut approximiert werden kann. Dies ist gerade die Aussage des Satzes von Weierstraß.

Satz 4.6 (Weierstraß) Der Raum der Polynome $\mathcal{P}$ auf $[a,b]$ ist dicht in $C([a,b])$.

Beweisskizze: Sei $f \in C([a,b])$. Wir setzen f stetig und mit kompektem Träger auf $\mathbb{R}$ fort und betrachten die Faltung

$$f_t(x) := \int_\Omega g_t(x-y)f(y)dy$$

mit

$$g_t(x) = \frac{1}{\sqrt{4\pi t}} \exp\left(-\frac{x^2}{4t}\right).$$

Dann kann man beweisen, dass f_t für $t \to 0$ gleichmäßig gegen f konvergiert. Es gibt also für beliebiges $\varepsilon > 0$ ein $t > 0$ mit

$$\|f_t - f\|_\infty < \varepsilon/2.$$

Zudem kann man beweisen, dass f_t für alle $t > 0$ als Funktion auf $\mathbb{C}$ eine holomorphe Funktion ist. Daher konvergiert ihre Taylor-Approximation P_k vom Grad k gleichmäßig auf $[a, b]$ für $k \to \infty$. Es existiert also ein Grad $k \in \mathbb{N}$ mit

$$\|P_k - f_t\|_\infty < \varepsilon/2.$$

Mit der Dreiecksungleichung erhält man dann

$$\|P_k - f\|_\infty \leq \|P_k - f_t\|_\infty + \|f_t - f\|_\infty < \varepsilon,$$

woraus die Behauptung folgt. □

Tatsächlich kann man die Polynome angeben, welche eine beliebige stetige Funktionen approximieren: Auf dem Intervall $[0, 1]$ leisten dies gerade die Bernstein-Polynome

$$(B_k f)(x) := \sum_{i=0}^{k} \binom{k}{i} f(i/n) x^i (1-x)^{k-i}.$$

Man kann beweisen, dass für $f \in C^k([0, 1])$

$$\lim_{k\to\infty} \|B_k f - f\|_{C^k} = 0$$

gilt, woraus

$$\lim_{k\to\infty} \|B_k f - f\|_{L_p} = 0$$

folgt. Für andere Intervalle als $[0, 1]$ lassen sich die Bernstein-Polynome entsprechend skalieren.

Beachte dass die Bernstein-Polynome die Funktion f nicht interpolieren; tatsächlich sind Interpolationspolynome ungeeignet, um beliebige stetige Funktionen zu approximieren, wie der folgende Satz zeigt.

Satz 4.7 (Faber) Sei $\Delta_k = \{x_{k,0}, \ldots, x_{k,k}\}$, $k \in \mathbb{N}$ eine beliebige Folge von Stützstellenmengen auf $[a, b]$. Dann gibt es ein $f \in C([a, b])$, so dass

$$\|P_k - f\|_\infty \not\to 0$$

für $k \to \infty$, wobei $P_k \in \mathcal{P}_k$ das Interpolationspolynom von f zur Stützstellenmenge Δ_k ist.

Beweisskizze: Wir haben am Ende von Abschnitt 4.2 gesehen, dass die Konstante Λ_k mindestens logarithmisch wächst. Man kann nun zeigen, dass es zu jeder Folge von Stützstellenmengen ein f gibt, so dass die Bestapproximation an f langsamer als logarithmisch konvergiert. Damit folgt die Aussage. □

Ebenso kann man zeigen, dass zu jedem $f \in C([a, b])$ eine Folge von Stützstellenmengen existiert, so dass P_k gleichmäßig auf $[a, b]$ gegen f konvergiert. Für allgemeine Gebiete $\Omega \subset \mathbb{R}^n$ und n-dimensionale Polynome gelten analoge Resultate.

Dichtheit von stückweisen Polynomen

Man kann das Intervall $[0,1]$ in 2^k Teilintervalle aufteilen und die Funktion f auf jedem Teilintervall durch ein Polynom vom Grad $\leq l$ interpolieren. Dabei können wir verlangen, dass die Polynome an den Nahtstellen mindestens p mal stetig differenzierbar sind (für $p = l-1$ würden wir so gerade die Splines der Ordnung l erhalten). Den dadurch entstehenden Raum der Polynome nennen wir $V_{p,l,k}$. Indem wir die Fehlerabschätzung für die Hermite-Interpolation aus der Einführung in die Numerik auf jedem Teilintervall anwenden, sehen wir, dass dadurch jede Funktion beliebig genau interpoliert werden kann, wenn wir nur k groß genug machen.

Also ist der Raum $\bigcup_{k\in\mathbb{N}} V_{p,l,k}$ dicht in $L_p([0,1])$ und $C^k([0,1])$.

Dichtheit von trigonometrischen Polynomen

Ebenso kann man zeigen, dass der Raum der reellen trigonometrischen Polynome

$$Q(x) = \frac{a_0}{2} + \sum_{j=1}^{k} (a_j \cos jt + b_j \sin jt)$$

dicht in der Menge der periodischen C- und L_p-Funktionen liegt.

4.4 Bestapproximationen in Hilberträumen

Hilberträume sind Vektorräume X, in denen ein Skalarprodukt $\langle\cdot,\cdot\rangle_X$ existiert, welches die Norm auf X mittels

$$\|f\|_X = \sqrt{\langle f, f\rangle_X}$$

definiert. Unter den genannten Funktionenräumen ist $L_2(\Omega)$ mit dem Skalarprodukt

$$\langle f, g\rangle_X = \int_\Omega f(x)g(x)dx$$

wegen

$$\langle f, f\rangle_X = \int_\Omega f(x)f(x)dx = \int_\Omega |f(x)|^2 dx = \|f\|_{L_2}^2$$

ein Hilbertraum. In einem Hilbertraum gilt

$$\left.\frac{d}{dt}\right|_{t=0} \|f - (f_k + tv_k)\|_X^2 = \left.\frac{d}{dt}\right|_{t=0} \langle f - (f_k + tv_k), f - (f_k + tv_k)\rangle_X = 2\langle f - f_k, v_k\rangle_X$$

Wenn $f_k \in X_k$ nun $\|f - f_k\|_X^2$ minimiert, so mimimiert $t = 0$ den Term $\|f - (f_k + tv_k)\|_X^2$ für alle $v_k \in X_k$. Folglich gilt

$$\langle f - f_k, v_k\rangle_X = 0$$

für alle $v_k \in X_k$. Für eine Basis $(s_1, \ldots, s_k)$ von X_k können wir den Minimierer f_k schreiben als $f_k = \sum_{j=1}^k \sigma_j s_j$ mit unbekannten $\sigma_j \in \mathbb{R}$. Dann gilt

$$0 = \left\langle f - \sum_{j=1}^{k} \sigma_j s_j, s_i \right\rangle_X = \langle f, s_i\rangle_X - \sum_{j=1}^{k} \sigma_j \langle s_j, s_i\rangle_X$$

für $i = 1, \ldots, k$. Definieren wir

$$\sigma = \begin{pmatrix} \sigma_1 \\ \vdots \\ \sigma_k \end{pmatrix}, \quad b = \begin{pmatrix} \langle f, s_1 \rangle_X \\ \vdots \\ \langle f, s_k \rangle_X \end{pmatrix} \quad \text{und} \quad M = \begin{pmatrix} \langle s_1, s_1 \rangle_X & \cdots & \langle s_k, s_1 \rangle_X \\ \vdots & & \vdots \\ \langle s_1, s_k \rangle_X & \cdots & \langle s_k, s_k \rangle_X \end{pmatrix}.$$

so können wir dies kurz schreiben als

$$M\sigma = b. \tag{4.2}$$

Die Koeffizienten σ_j der Bestapproximation f_k können durch die Lösung eines linearen Gleichungssystems berechnet werden. Beachte, dass M invertierbar ist, weil die s_j eine Basis bilden. Der folgende Satz folgt sofort aus dieser Tatsache.

Satz 4.8 Wenn $\|\cdot\|_X$ durch ein Skalarprodukt gegeben ist, so ist die Bestapproximation eindeutig und hängt linear von der zu approximierenden Funktion ab.

Wir betrachten nun den Fall, dass die s_k aus einer Orthonormalbasis von X stammen. Dazu benötigen wir zunächst die folgenden Definitionen.

Definition 4.9 Sei X ein Hilbertraum.

(i) Ein Orthonormalsystem in X ist eine abzählbare Menge $\{e_k \,|\, k \in \mathbb{N}\} \subset X$, so dass

$$\langle e_i, e_j \rangle_X = \delta_{ij} = \begin{cases} 1, & \text{falls } i = j \\ 0, & \text{sonst} \end{cases}$$

gilt für alle $i, j \in \mathbb{N}$.

(ii) Eine Orthonormalbasis in X ist ein Orthonormalsystem, dessen Aufspann dicht in X liegt, d.h. für alle $f \in X$ existieren $\lambda_i \in \mathbb{R}$, $i \in \mathbb{N}$ mit

$$\lim_{k\to\infty} \sum_{i=1}^{k} \lambda_i e_i = f.$$

□

Für eine Folge (y_i) in X mit paarweise unabhängigen y_i kann man mit dem Gram-Schmidt-Verfahren ein Orthonormalsystem konstruieren.

Beispiel 4.10 (i) Sei $X = L_2([-1, 1])$ mit Skalarprodukt

$$\langle f, g \rangle = \int_{-1}^{1} f(x)g(x)dx.$$

Wie wir in der Einführung in die Numerik bei der Gauß-Quadratur gesehen haben, bilden die Legendre-Polynome dann eine Orthonormalbasis des Raums der Polynome. Da dieser dicht in X liegt, bilden die Legendre-Polynome auch eine Orthonormalbasis von X.

(ii) Die trigonometrischen Funktionen

$$\{(2\pi)^{1/2}\}, \pi^{1/2} \cos kt, \pi^{1/2} \sin kt\}_{k\in\mathbb{N}}$$

bilden eine Orthonormalbasis von $X = L_2([0, \pi])$ □

Wenn die s_i in der Konstruktion der Bestapproximation die ersten k Elemente eines Orthonormalensystems (e_i) sind, folgt $M = \mathrm{Id}$ in (4.2). Damit gilt $\sigma = b$ und es folgt für die Bestapproximation f_k, dass

$$f_k = \sum_{i=1}^{k} \sigma_i e_i = \sum_{i=1}^{k} \langle f, e_i \rangle_X e_i. \tag{4.3}$$

Der folgende Satz gibt Eigenschaften von Orthonormalbasen (e_i) an sowie einen Fehler für die Bestapproximation $y_k \in X_k$ für $X_k = \mathrm{span}\{e_1, \ldots, e_k\}$.

Satz 4.11 Sei (e_i) eine Orthonormalbasis von X, und $f \in X$ und f_k die Bestapproximation in $X_k = \mathrm{span}\{e_1, \ldots, e_k\}$. Dann gilt

(i) $f = \lim_{k\to\infty} f_k = \lim_{k\to\infty} \sum_{i=1}^{k} \langle f, e_i \rangle_X e_i = \sum_{i=1}^{\infty} \langle f, e_i \rangle_X e_i$

(ii) $\langle f, w \rangle_X = \sum_{i=1}^{\infty} \langle f, e_i \rangle_X \langle w, e_i \rangle_X$

(iii) $\|f\|_X^2 = \sum_{i=1}^{\infty} \langle f, e_i \rangle_X^2$

(iv) $\|f - f_k\|_X^2 = \sum_{i=k+1}^{\infty} \langle f, e_i \rangle_X^2$

Beweis: (i) Die Gleichung $f_k = \sum_{i=1}^{k} \langle f, e_i \rangle_X e_i$ wurde bereits in (4.3) gezeigt. Weil der Unterraum $\mathrm{span}\{e_1, e_2, \ldots\}$ dicht in X liegt, gibt es Elemente $\tilde{f}_k \in \mathrm{span}\{e_1, \ldots, e_k\}$, $k \in \mathbb{N}$ mit $\|\tilde{f}_k - f\|_X \to 0$ für $k \to \infty$. Da die Bestapproximation per Definition $\|f_k - f\|_X \leq \|\tilde{f}_k - f\|_X$ erfüllt, folgt (i).

(ii) Mit (i) und der Stetigkeit des Skalarprodukts gilt

$$\langle f, w \rangle_X = \left\langle \lim_{k\to\infty} \sum_{i=1}^{k} \langle f, e_i \rangle_X e_i, w \right\rangle_X = \lim_{k\to\infty} \left\langle \sum_{i=1}^{k} \langle f, e_i \rangle_X e_i, w \right\rangle_X = \sum_{i=1}^{\infty} \langle f, e_i \rangle_X \langle w, e_i \rangle_X.$$

(iii) Folgt aus (ii) mit $w = f$.

(iv) Es gilt

$$\|f - f_k\|_X^2 = \langle f - f_k, f - f_k \rangle_X = \langle f, f \rangle_X - 2\langle f, f_k \rangle_X + \langle f_k, f_k \rangle_X$$

und mit (ii), (iii) und (4.3) und $\langle e_i, e_j \rangle_X = 0$ für $i \neq j$ können wir fortfahren

$$= \sum_{i=1}^{\infty} \langle f, e_i \rangle_X^2 - 2\sum_{i=1}^{k} \langle f, e_i \rangle_X^2 + \sum_{i=1}^{k} \langle f, e_i \rangle_X^2 = \sum_{i=k+1}^{\infty} \langle f, e_i \rangle_X^2.$$

□

Aus dem Abklingverhalten der Koeffizienten $c_i = \langle f, e_i \rangle_X$ aus Satz 4.11(i) kann man wegen $\|e_i\|_X = 1$ mit Satz 4.11(iv) auf die Güte der Bestapproximation schließen. Wenn z.B. $c_i \leq \theta^i$ für ein $\theta < 1$ und alle $i \in \mathbb{N}$ gilt, so folgt

$$\|f - f_k\|_X^2 \leq \sum_{i=k+1}^{\infty} \theta^{2i} = O(\theta^{2(k+1)}), \qquad \text{also } \|f - f_k\|_X = O(\theta^{k+1}).$$

Im Fall $c_i \leq k^{-(s+1/2)}$ für ein $s > 0$ folgt

$$\|f - f_k\|_X^2 \leq \sum_{i=k+1}^{\infty} k^{-2s-1} = O(k^{-2s}), \qquad \text{also } \|f - f_k\|_X = O(k^{-s}).$$

Oftmals (z.B. bei Polynombasen) ist es so, dass die Basiselemente e_i immer stärker oszillieren, also immer hochfrequenter werden, je größer i wird. Schnell abfallende Koeffizienten sind daher ein Indiz dafür, dass die Funktion f keine oder nur schwache hochfrequente Anteile besitzt.

4.5 Mehrdimensionale stückweise polynomiale Interpolation

In höheren Raumdimensionen $n \geq 2$ liegen Interpolationsdaten in der Regel auf räumlich verteilten Stützstellenmengen vor. Sofern diese nicht in alle Richtungen äquidistant verteilt sind, ist es i.A. schwierig, diese mit Polynomen zu interpolieren. Zudem ergibt sich die aus dem Eindimensionalen bekannte Schwierigkeit der starken Oszillationen bei einer großen Anzahl von Stützstellen. Auch die Splineinterpolation ist technisch sehr aufwändig. In vielen Anwendungen ist es aber auch gar nicht nötig, Daten mit glatten Funktionen zu interpolieren. In dem wichtigen Anwendungsgebiet der Numerik partieller Differentialgleichungen ist es z.B. fast immer ausreichend, mit stückweise glatten Funktionen zu interpolieren, da die Gleichungen zumeist in einer "schwachen" Integralform vorliegen, so dass Nichtdifferenzierbarkeiten kein Problem darstellen.

Für die stückweise polynomiale Interpolation teilt man die betrachtete Menge $\Omega \subset \mathbb{R}^n$ in N einfach geformte Gebiete T_i, $i = 1, \ldots, N$, wie z.B. Simplizes (also Dreiecke in $\mathbb{R}^2$, Tetraeder in $\mathbb{R}^3$), Quadern etc. auf. Diese sind paarweise disjunkt und ihre Abschlüsse überdecken den ganzen Raum, also

$$T_i \cup T_j = \emptyset \text{ für } i \neq j \quad \text{ und } \quad \bigcup_{i=1}^{N} \overline{T}_i = \overline{\Omega}.$$

Die Gesamtmenge der T_i nennen wir ein *Gitter* auf Ω. Die einzelnen T_j sind dabei affine Bilder eines *Referenzgebiets* $\widehat{T}$, d.h. es existieren invertierbare Matrizen $B_i \in \mathbb{R}^{n \times n}$ und reelle Zahlen $w_i \in \mathbb{R}$ mit

$$T_i = B_i \widehat{T} + w_i.$$

Für die folgenden Überlegungen sind die Konditionen $\kappa_i = \|B_i\| \, \|B_i^{-1}\|$ der Matrizen B_i wichtig sowie ihre Matrixnormen $h_i = \|B_i\|$. Beachte, dass die Größe der T_i proportional zu h_i ist, während die κ_i angeben, wie "verzerrt" die Elemente sind.

Der Vorteil dieser Konstruktion der T_i ist, dass wir die Abschätzungen für die Polynominterpolation nun nur auf $\widehat{T}$ machen müssen und dann auf die einzelnen T_i übertragen können. Wir nehmen daher an, dass für ein gegebenes Basiselement eine Projektion $\widehat{P} : C^\infty(\widehat{T}) \to \mathcal{P}$ existiert, die jeder Funktion v ein Polynom $\widehat{P}v \in \mathcal{P}$ zuordnet, das auf gegebenen Stützstellen in $\widehat{T}$ mit v übereinstimmt. Dies funktioniert beispielsweise, wenn $\widehat{T}$ ein Simplex im $\mathbb{R}^n$ ist, die Stützstellen gerade die Ecken des Simplex sind und $\mathcal{P}$ der Raum der linearen Polynome ist, oder wenn $\widehat{T}$ ein Quader im $\mathbb{R}^n$ ist, die Stützstellen wieder an den Ecken von $\widehat{T}$ liegen und $\mathcal{P}$ der Raum der quadratischen Polynome ist (im ersten Fall ist P eindeutig, im zweiten nicht). Die Funktionen, mit denen wir auf $\overline{\Omega}$ approximieren, enstehen nun dadurch, dass wir eine Projektion $p_i = P_i v$ auf jedem Element T_i definieren, indem wir die Funktion v erst mittels

$$\hat{v}_i(\hat{x}) = v(B_i\hat{x} + w_i)$$

transformieren, dann

$$\hat{p}_i = \widehat{P}\hat{v}_i$$

berechnen und diese Funktion dann mittels

$$p_i(x) = \hat{p}_i(B_i^{-1}(x - w_i))$$

zurücktransformieren. Die Funktion $p(x) = p_i(x)$, $x \in \overline{T}_i$ ist dann das gesuchte stückweise definierte Polynom[2]. Die Notation folgt hier der Konvention, dass Punkte in $\widehat{T}$ mit $\hat{x}$ bezeichnet werden und Punkte in den T_i mit x.

Als Fehlermaß auf Ω verwenden wir die Sobolev-Norm

$$\|f\|_{W^{k,p}} = \sqrt[p]{\sum_{i=0}^{k} \int_\Omega \|f^{(i)}(x)\|^p dx}$$

und für die Abschätzungen verwenden wir zusätzlich die Sobolev-Halbnorm

$$|f|_{W^{k,p}} = \sqrt[p]{\int_\Omega \|f^{(k)}(x)\|^p dx}.$$

Für die stückweise definierten Polynome sind diese (Halb)Normen nicht geeignet, da die Ableitungen an den Rändern der Elemente T_i i.A. nicht existieren. Wir können sie aber mittels

$$\|f\|_{W^{k,p}} = \sqrt[p]{\sum_{j=1}^{N}\sum_{i=0}^{k} \int_{T_j} \|f^{(i)}(x)\|^p dx} \quad \text{und} \quad |f|_{W^{k,p}} = \sqrt[p]{\sum_{j=1}^{N} \int_{T_j} \|f^{(k)}(x)\|^p dx}$$

entsprechend anpassen. Ebenso können wir die Norm auf dem Referenzgebiet definieren durch

$$\|f\|_{W^{k,p}(\widehat{T})} = \sqrt[p]{\sum_{i=0}^{k} \int_{\widehat{T}} \|f^{(i)}(\hat{x})\|^p d\hat{x}} \quad \text{und} \quad |f|_{W^{k,p}(\widehat{T})} = \sqrt[p]{\int_{\widehat{T}} \|f^{(k)}(\hat{x})\|^p d\hat{x}}.$$

[2] Gegebenenfalls können hierbei Unstetigkeiten und damit Uneindeutigkeiten an den Rändern der Elemente T_i entstehen. In diesem Fall muss geklärt werden, wie der Wert von p am Rand der Elemente genau definiert ist.

Für $k \geq 1$ und $p \geq n$ garantiert der Sobolev-Einbettungssatz die Existenz einer Konstanten $\tilde{c} > 0$, so dass

$$\|f\|_{W^{k,p}(\widehat{T})} \geq \tilde{c}\|f\|_\infty$$

für alle $f \in C^\infty$. Damit kann man auf dem Referenzgebiet für Polynome vom Grad $k-1$ die Fehlerabschätzung

$$\|v - Pv\|_{W^{k,p}(\widehat{T})} \leq c|v|_{W^{k,p}(\widehat{T})} \tag{4.4}$$

für eine Konstante $c > 0$ zeigen (dies entspricht der Abschätzung (4.1) im eindimensionalen Fall mit $k-1$ an Stelle von k für festes a und b, also z.B. $a = 0$ und $b = 1$, d.h. $\widehat{T} = (0, 1)$). Beachte, dass die Abschätzung in dieser Form nur von eingeschränktem Nutzen ist, da wir nur dann einen kleinen Fehler erhalten, wenn die k-te Ableitung von v klein ist. Wir werden aber gleich sehen, wie man aus dieser Ungleichung eine sinnvolle Fehlerabschätzung erhält.

Ziel der folgenden Rechnungen ist nun, aus der Abschätzung (4.4) auf $\widehat{T}$ Fehlerabschätzungen für die Interpolation auf den T_i zu erhalten. Dazu benötigen wir das folgende vorbereitende Lemma.

Lemma 4.12 Gegeben seien $B \in \mathbb{R}^{n\times n}$ invertierbar und $w \in \mathbb{R}^n$. Sei $f \in \mathbb{C}^\infty(T)$ für $T = B\widehat{T} + w$ und definiere $\hat{f} \in \mathbb{C}^\infty(\widehat{T})$ mittels $\hat{f}(\hat{x}) := f(B\hat{x} + w)$. Dann gilt für beliebige $k, p \in \mathbb{N}$

$$|\hat{f}|^p_{W^{k,p}(\widehat{T})} \leq \|B\|^{pk} |\det B|^{-1} |f|^p_{W^{k,p}(T)}$$

und

$$|f|^p_{W^{k,p}(\widehat{T})} \leq \|B^{-1}\|^{pk} |\det B| |\hat{f}|^p_{W^{k,p}(T)}.$$

Beweis: Nach Kettenregel gilt mit $A(\hat{x}) = B\hat{x} + w$

$$\hat{f}^{(k)}(\hat{x})(v_1, \ldots, v_k) = (f \circ A)^{(k)}(\hat{x})(v_1, \ldots, v_k) = f^{(k)} \circ A(\hat{x})(Bv_1, \ldots, Bv_k).$$

Daraus folgt

$$\|\hat{f}^{(k)}(\hat{x})\| \leq \|B\|^k \|f^{(k)}(A(\hat{x}))\|$$

und damit mit dem Transformationssatz und $DA(x) \equiv B$

$$\int_{\widehat{T}} \|\hat{f}^{(k)}(\hat{x})\|^p d\hat{x} \leq \|B\|^{kp} \int_{\widehat{T}} \|f^{(k)} \circ A(\hat{x})\|^p d\hat{x} = \|B\|^{kp} \int_T \|f^{(k)}(x)\|^p |\det B|^{-1} dx.$$

Dies zeigt die erste Behauptung. Die zweite folgt aus der ersten mit B^{-1} und $-B^{-1}w$ an Stelle von B und w, weil $f(x) = \hat{f}(B^{-1}x - B^{-1}w)$. □

Mit diesem Lemma können wir nun eine Approximationsaussage treffen. Hierbei nehmen wir an, dass alle Elemente klein sind in dem Sinne, dass $\|B_i\| \leq h$ gilt für eine kleine positive Zahl $h > 0$ und alle $i = 1, \ldots, N$.

Satz 4.13 Sei $m \in \mathbb{N}$ und $k \geq m$. Dann gilt für die stückweise Interpolation mit Polynomen vom Grad $k-1$ auf einem Gitter mit $h_i = \|B_i\| \leq h$, $h > 0$ und $\kappa_i \leq \kappa$, $\kappa > 0$ für die Konstante $c > 0$ aus (4.4) die Ungleichung

$$\|v - Pv\|_{W^{m,p}(\Omega)} \leq c\kappa^m h^{k-m} |v|_{W^{k,p}(\Omega)}.$$

Beweis: Es genügt,

$$\|v - Pv\|^p_{W^{m,p}(T_i)} \leq c^p \kappa_i^{mp} h_i^{(k-m)p} |v|^p_{W^{k,p}(T_i)} \tag{4.5}$$

für alle T_i zu zeigen. Die Aussage folgt dann durch Summieren über die T_i, Maximieren über κ_i und h_i und Ziehen der p-ten Wurzel. Mit Lemma 4.12 (einmal mit $k = m$ und einmal mit dem ursprünglichen k angewendet) und (4.4) folgt

$$\begin{aligned}
\|v - Pv\|^p_{W^{m,p}(T_i)} &\leq \|B_i^{-1}\|^{mp} |\det B_i| \, |\hat{v} - P\hat{v}|^p_{W^{m,p}(\widehat{T})} \\
&\leq \|B_i^{-1}\|^{mp} |\det B_i| \, \|\hat{v} - P\hat{v}\|^p_{W^{k,p}(\widehat{T})} \\
&\leq c^p \|B_i^{-1}\|^{mp} |\det B_i| \, |\hat{v}|^p_{W^{k,p}(\widehat{T})} \\
&\leq c^p \|B_i^{-1}\|^{mp} |\det B_i| \, \|B_i\|^{kp} |\det B_i|^{-1} \, |v|^p_{W^{k,p}(T_i)} \\
&= c^p \kappa_i^{mp} \|B_i\|^{(k-m)p} \, |v|^p_{W^{k,p}(T_i)} \\
&\leq c^p \kappa_i^{mp} h_i^{(k-m)p} \, |v|^p_{W^{k,p}(T_i)},
\end{aligned}$$

also (4.5). □

Wenn wir also z.B. lineare Polynome betrachten, ist der Grad $k-1 = 1$. Also ist $k = 2$ und wir können die Abschätzung für $m = 0$ oder $m = 1$ anwenden. Wir erhalten so einen Fehler der Ordnung $O(h^2)$ für die Funktion selbst und von der Ordnung $O(h)$ für ihre Ableitung.

Die Größe $h > 0$ gibt den Durchmesser der Elemente T_i (relativ zum Durchmesser von $\widehat{T}$) an. Wenn alle Elemente in etwa gleich groß sind, ist die Zahl N der benötigten Elemente proportional zu $1/h^n$. In Abhängigkeit von der Anzahl der Elemente gilt für den Fehler also

$$\|v - Pv\|_{W^{m,p}(\Omega)} \sim N^{(m-k)/n} |v|_{W^{k,p}(\Omega)}.$$

Die Anzahl der benötigten Elemente nimmt also nicht nur zu, wenn wir einen kleineren Fehler garantieren wollen, sondern auch — und zwar sehr schnell — wenn die Dimension n zunimmt. Genauer Dieser Effekt ist als “Fluch der Dimension” bekannt.

Wie bei der eindimensionalen Polynominterpolation wird die Fehlerabschätzung schlecht, wenn die höheren Ableitungen der Funktion v groß werden.

Literaturverzeichnis

[1] Bornemann, F.: *Numerische lineare Algebra: Eine konzise Einführung mit MATLAB und Julia*. 2. Auflage. Springer, 2019

[2] Bryan, K. ; Leise, T.: The $\$25,000,000,000$ eigenvector: the linear algebra behind Google. In: *SIAM Rev.* 48 (2006), Nr. 3, S. 569–581. – ISSN 0036–1445

[3] Deuflhard, P. ; Hohmann, A.: *Numerische Mathematik. I: Eine algorithmisch orientierte Einführung*. 3. Auflage. Berlin : de Gruyter, 2002

[4] Nocedal, J. ; Wright, S. J.: *Numerical optimization*. Second. New York : Springer-Verlag, 2006 (Springer Series in Operations Research and Financial Engineering). – xxii+664 S.

[5] Schwarz, H. R. ; Köckler, N.: *Numerische Mathematik*. 5. Auflage. Stuttgart : B. G. Teubner, 2004

[6] Trefethen, L. N.: *Approximation theory and approximation practice*. SIAM, 2013

Index