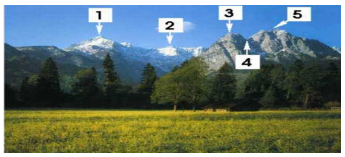
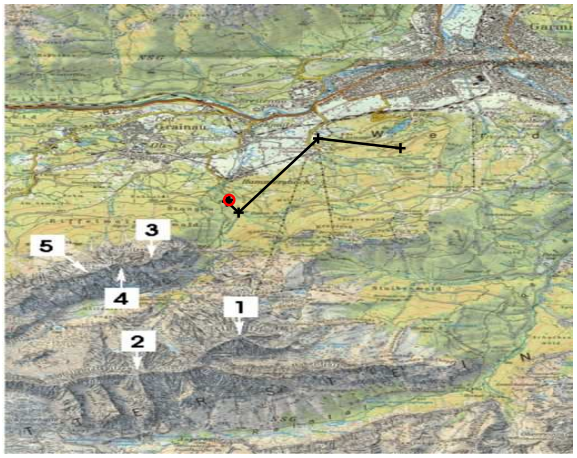
Stefan Funken

Numerik III

(Numerische Lineare Algebra und Optimierung)





SKRIPT, UNIVERSITÄT ULM, WINTERSEMESTER 2012/2013

Die aufmerksamen Leserinnen oder Leser seien ermuntert, mir Hinweise auf Druckfehler, sprachliche Unzulänglichkeiten oder inhaltliche Flüchtigkeiten per Email zukommen zu lassen (`stefan.funken@uni-ulm.de`).

Stand. Ulm, August 2009.

Inhaltsverzeichnis

1 Nichtlineare Ausgleichsprobleme 1

1.1 Gauß-Newton-Verfahren 4
1.2 Konvergenz des Gauß-Newton-Verfahrens 6
1.3 Levenberg-Marquardt-Verfahren 8

2 Eigenwertprobleme 11

2.1 Grundlagen aus der linearen Algebra 13
2.2 Abschätzungen und geometrische Lage der Eigenwerte 16
2.3 Potenzmethode 20
2.4 Inverse Iteration nach Wielandt 23
2.5 QR-Verfahren 26
2.6 Verfahren für symmetrische Matrizen 44
2.7 Singulärwertzerlegung 49
2.8 Verallgemeinerte Eigenwertprobleme 59
2.9 Nichtlineare Eigenwertprobleme 68

3 Nichtrestringierte Optimierung 71

3.1 Direkte Suchverfahren 71
3.2 Abstiegsverfahren 76
3.3 Schrittweitensteuerung 77
3.4 Algorithmen zur Auswahl der Schrittweite 80
3.5 Nichtlineares cg-Verfahren 86

4 Optimierung unter Nebenbedingungen 89

4.1 Straftermmethode 90
4.2 Lagrange-Multiplikatoren 91

Literaturverzeichnis 91

Stichwortverzeichnis 93

1 NICHTLINEARE AUSGLEICHSPROBLEME

In Numerik I haben wir uns bereits mit linearen Ausgleichsproblemen befasst. Wir erinnern uns daran, dass diese Probleme von der Form

$$\|Ax - d\|_2^2 \to \min \qquad \text{mit } A \in \mathbb{R}^{m\times n},\ d \in \mathbb{R}^m,\ m > n \tag{1.1}$$

sind. Es besteht also ein linearer Zusammenhang zwischen den „Messungen“ d und den Parametern x. Sie können mittels der Gaußschen Normalengleichung

$$A^T Ax = A^T d$$

bzw. numerisch mit der QR-Zerlegung gelöst werden. Solche Probleme entstehen beispielsweise bei einer linearen Regression durch mehrere Messpunkte. Betrachten wir nun anstelle einer linearen Regression folgendes Beispiel.

Beispiel 1.0.1 (Kreisinterpolation) Gegeben sei die Punktewolke $(s_i, t_i), i = 1, \ldots, m$, mit dem Hinweis, dass alle diese Punkte näherungsweise auf einem Kreis liegen. Geht man von einem Kreis mit Mittelpunkt $M = (\mu_s, \mu_t)$ und Radius $r > 0$ aus, so müssten die Punkte

$$\sqrt{(s_i - \mu_s)^2 + (t_i - \mu_t)^2} \approx r \qquad i = 1, \ldots, m$$

erfüllen. Die Aufgabe besteht also darin, den Mittelpunkt M und den Radius r so zu bestimmen, dass die Summe über die Quadrate der einzelnen Abstände

$$e_i = \sqrt{(s_i - \mu_s)^2 + (t_i - \mu_t)^2} - r \qquad i = 1, \ldots, m$$

der Punkte (s_i, t_i) vom Kreis mit Mittelpunkt M und Radius r minimiert wird.

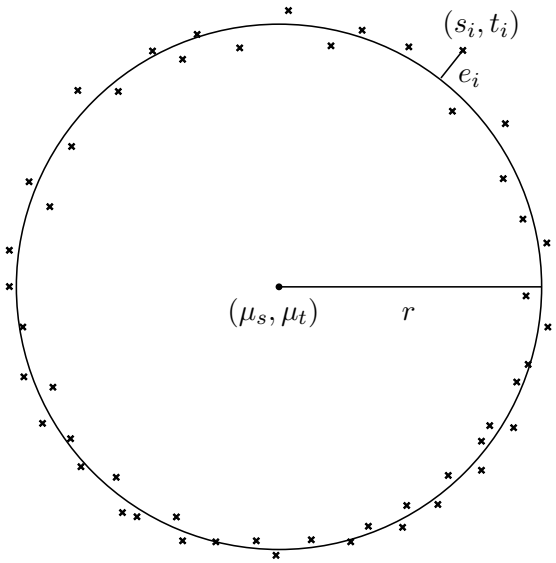


Abb. 1.1: Kreisinterpolation

Bezeichnet $e \in \mathbb{R}^m$ den Vektor $(e_1, \ldots, e_m)$, so lässt sich das Problem wie folgt formulieren:

$$\|e\|_2^2 \to \min \quad \Leftrightarrow \quad g(\mu_s, \mu_t, r) := \sum_{i=1}^{m} \left(\sqrt{(s_i - \mu_s)^2 + (t_i - \mu_t)^2} - r \right)^2 \to \min .$$

Es besteht also in diesem Fall kein linearer Zusammenhang zwischen den Parametern (μ_s, μ_t, r) und den Punkten (s_i, t_i).

Beispiel 1.0.2 (Wo war ich?) Haben Sie sich beim Anschauen Ihrer Urlaubsfotos[1] nicht schon einmal gefragt: Wo war ich da eigentlich? Wenn Sie sich nicht einmal daran erinnern, in welchem Land Sie waren, kann Ihnen die Mathematik auch nicht weiterhelfen. Aber wie sieht es aus, wenn Sie einige fotografierte Punkte so identifizieren können, dass Sie sie auf einer Landkarte wiederfinden? Die Problemstellung lautet nun: Man bestimme die Position des Fotografen auf der Karte und die Höhe über dem Meeresspiegel, die Brennweite und die Ausrichtung der Kamera zu den folgenden Daten.

Nr.	Berg	Höhe	Position in Metern in der Karte	
			x	y
1	Alpspitze	2627,6 m	3700	4125
2	Äußere Hölltalspitze	2720,0 m	2200	3175
3	Vorderer Waxenstein	2136,0 m	2350	6000
4	Mittagsscharte	2072,0 m	2175	5885
5	Großer Waxenstein	2276,5 m	1850	5950

Tab. 1.1: Angabe zur Lage der Gipfel

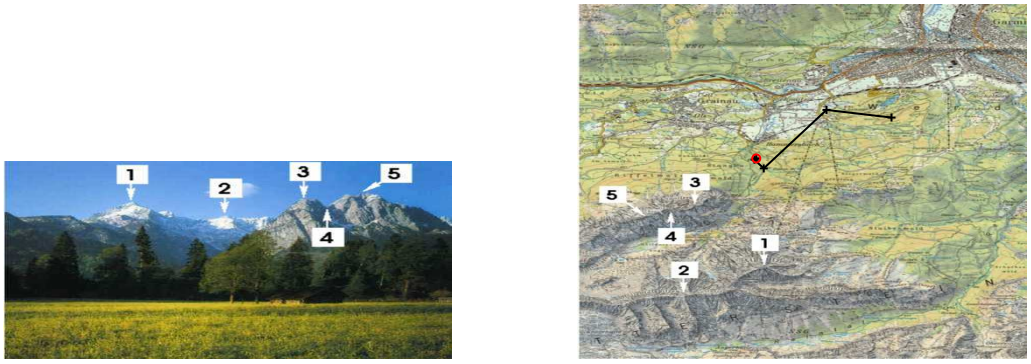


Abb. 1.2: Fotografie und Karte mit gekennzeichneten Gipfeln

Mathematisch formuliert führt dieses Problem auf ein nichtlineares Ausgleichsproblem: Der Vektor $w_i \in \mathbb{R}^3$ von der Linse im Objektiv zum Punkt u_i des Gipfels i auf dem Foto muss für alle $i = 1, \ldots, 5$ mit dem Vektor $x - x_i$ der Differenz aus der Position des Fotografen $x \in \mathbb{R}^3$ und der Position des Berges $x_i \in \mathbb{R}^3$ auf der Karte korrespondieren.

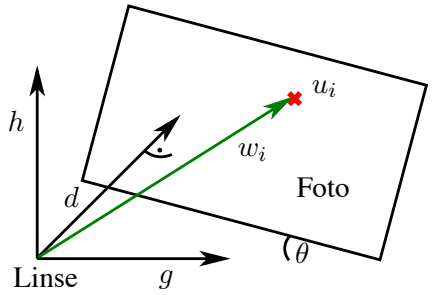


Abb. 1.3: Orthogonalsystem d, g, h, Drehwinkel θ und Vektor w_i von der Linse zum Gipfel i

[1] Die Idee dieser Aufgabenstellung geht auf Ernst Hairer, Gerhard Wanner und Stephanie Cirilli von der Universität Genf zurück.

D.h. man möchte

$$\sum_{i=1}^{5} \|w_i \times (x - x_i)\|_2^2 \to \min,$$

wobei $\times$ das Kreuzprodukt bezeichnet.

Denkbar wären auch folgende Beispiele aus der Stochastik bzw. der Physik.

Beispiel 1.0.3 (Dichte der Gamma-Verteilung) Aus der Vorlesung Stochastik I kennen wir die Gamma-Verteilung $\Gamma(\lambda, p)$, $\lambda > 0, p > 0$, mit der Dichte

$$f(\lambda, p; t) = \begin{cases} \frac{\lambda^p t^{p-1}}{\Gamma(p)}\, e^{-\lambda t} & , t \geq 0 \\ 0 & , t < 0\,, \end{cases}$$

wobei $\Gamma(p)$ die Gamma-Funktion

$$\Gamma(p) := \int_0^\infty x^{p-1} e^{-x}\, \mathrm{d}x\,, \quad p > 0$$

bezeichnet. Zu $t_i \geq 0$, $i = 1, \ldots, m$, seien die Werte f_i gegeben und man vermutet, dass $f_i \approx f(\lambda, p; t_i)$ gilt mit geeigneten $\lambda, p > 0$. Man sucht daher

$$\arg\min_{\lambda, p > 0} \|F(\lambda, p)\|_2^2 \quad \text{mit} \quad F(\lambda, p) := \Big(f(\lambda, p; t_i) - f_i\Big)_{i=1,\ldots,m}.$$

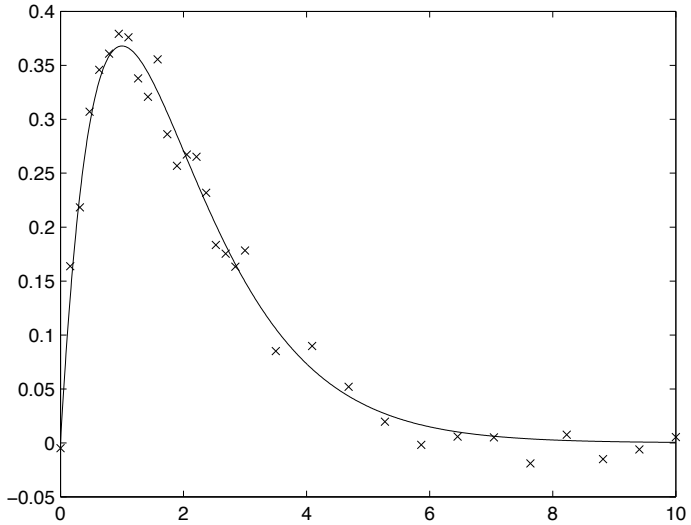


Abb. 1.4: Dichte der Gamma-Verteilung für $\lambda = 1, p = 2$ und Punktewolke

Beispiel 1.0.4 (Gedämpfte Schwingung) Die Auslenkung einer gedämpften Schwingung in Abhängigkeit von der Zeit lässt sich mit der Gleichung

$$u(t) = u_0\, e^{-\delta t} \sin(\omega t + \varphi_0)$$

beschreiben, wobei u_0 ein Anfangswert, φ_0 die Anfangsphase der Schwingung, δ die Abklingkonstante und ω die Frequenz sind. Diese Parameter kann man aus der Masse m, der Federkonstanten

D sowie der Dämpfungskonstanten b unter Berücksichtigung der Gültigkeit der Differenzialgleichung

$$\ddot{u} + \tfrac{b}{m}\,\dot{u} + \tfrac{D}{m}\,u = 0$$

bestimmen. Liegen diese Werte nicht vor, so ist man auf eine näherungsweise Bestimmung von u_0, δ, ω und φ_0 aus Messwerten $(t_i, u_i), i = 1, \ldots, m$, angewiesen.
Man betrachtet dazu die Fehlerfunktion

$$F : \mathbb{R}^4 \to \mathbb{R}^m, \quad (u_0, \delta, \omega, \varphi_0) \mapsto \Big(u_0\, e^{-\delta t_i} \sin(\omega t_i + \varphi_0) - u_i\Big)_{i=1,\ldots,m}$$

und möchte diese in der euklidischen Norm minimieren; man sucht also

$$\arg \min_{(u_0,\delta,\omega,\varphi_0)\in\mathbb{R}^4} \|F(u_0, \delta, \omega, \varphi_0)\|_2^2\,.$$

Alle diese Beispiele führen uns zu der Aufgabe, einen Parametervektor $x \in \mathbb{R}^n$ zu bestimmen, sodass

$$\|F(x)\|_2^2 \to \min \quad \Leftrightarrow \quad g(x) := \tfrac{1}{2}\, F(x)^T F(x) \to \min$$

mit einer (möglicherweise) nichtlinearen Funktion $F : \mathbb{R}^n \to \mathbb{R}^m$.

1.1 Gauss-Newton-Verfahren

Wir setzen nun voraus, dass $F : O \to \mathbb{R}^m$ zweimal stetig differenzierbar auf einer offenen Menge $O \subset \mathbb{R}^n$ ist und $m > n$ gilt, wir also nur den überbestimmten Fall betrachten. Unser Interesse gilt lokalen inneren Minima $x^* \in O$ von g, die den hinreichenden Bedingungen

$$\nabla g(x^*) = 0 \ \text{ und } \ g''(x^*) \ \text{ positiv definit} \tag{1.2}$$

genügen. Bezeichnet $F'(x) \in \mathbb{R}^{m\times n}$ die Jacobi-Matrix von F an der Stelle x sowie $F_i''(x)$ die Hesse-Matrix von F_i an der Stelle x, d.h. $F_i''(x) := \left(\frac{\partial^2 F_i(x)}{\partial x_j \partial x_k}\right)_{1\le j,k\le n} \in \mathbb{R}^{n\times n}$, dann gilt

$$\nabla g(x) = F'(x)^T F(x), \qquad g''(x) = F'(x)^T F'(x) + \sum_{i=1}^{m} F_i(x) F_i''(x).$$

Um $g(x)$ zu minimieren, betrachten wir also das nichtlineare Gleichungssystem

$$\nabla g(x) = F'(x)^T F(x) \stackrel{!}{=} 0\,.$$

Wendet man hierauf das Newton-Verfahren an, so lautet die Iterationsvorschrift

$$\begin{aligned} g''(x_k)\, s_k &= -\nabla g(x_k) \\ x_{k+1} &= x_k + s_k \\ &= x_k - [g''(x_k)]^{-1} \nabla g(x_k) \\ &= x_k - \Big[F'(x_k)^T F'(x_k) + \sum_{i=1}^{m} F_i(x_k) F_i''(x_k)\Big]^{-1} F'(x_k)^T F(x_k) \end{aligned} \tag{1.3}$$

Unter Annahme (1.2) ist g'' in einer Umgebung von x^* positiv definit und daher invertierbar. Geht man von zum Modell kompatiblen Daten aus, d.h. gilt $F(x^*) = 0$, so ist

$$g''(x^*) = F'(x^*)^T F'(x^*)\,.$$

Die Bedingung der positiven Definitheit der Hesse-Matrix $g''(x^*)$ fällt in diesem Fall zusammen mit der Bedingung, dass die Jacobi-Matrix $F'(x^*)$ vollen Rang n hat. Für (nahezu) kompatible Daten und für ein x aus einer Umgebung $U \subset O$ von x^* kann man also die Hesse-Matrix $g''(x)$ durch $F'(x)^T F'(x)$ approximieren (*Quasi-Newton-Verfahren*) und sich so die Berechnung von $F''(x)$ in (1.3) sparen.
Mit dieser Vorgehensweise erhalten wir den folgenden Algorithmus.

Algorithmus 1.1.1: Gauß-Newton-Verfahren

Input: x_0, F, F'
for $k = 0, 1, \ldots$

i) Berechne $F(x_k)$, $F'(x_k)$.

ii) Bestimme den Korrekturvektor s_k gemäß

$$F'(x_k)^T F'(x_k)\, s_k = -F'(x_k)^T\, F(x_k)\,. \tag{1.4}$$

iii) Setze $x_{k+1} = x_k + s_k$.

end

Bemerkungen 1.1.1 *i) Offensichtlich ist* (1.4) *die Normalengleichung zum Ausgangsproblem*

$$\|F'(x_k)\, s_k + F(x_k)\|_2 \to \min\,. \tag{1.5}$$

Somit wurde die numerische Lösung eines nichtlinearen Ausgleichsproblems auf die numerische Lösung einer Folge von linearen Ausgleichsproblemen zurückgeführt.

ii) Man hätte den Algorithmus 1.1.1 auch direkt – ähnlich wie beim Newton-Verfahren – durch Taylorentwicklung und Abbrechen nach dem linearen Term herleiten können. Daher heißt dieses Vorgehen auch Gauß-Newton-Verfahren.

iii) Wegen der Vernachlässigung des Terms aus (1.3), *der die zweiten Ableitungen von F enthält, geht die quadratische Konvergenz des Newton-Verfahrens verloren und das Gauß-Newton-Verfahren ist in der Regel nur linear konvergent.*

Eine einfache mögliche Realisierung des Gauß-Newton-Verfahrens in Matlab ist nachfolgend dargestellt.

MATLAB-Funktion: gauss_newton.m

```
1  function x = gauss_newton(F,DF,x0,maxit,tol)
2      k=0;
3      x=x0;
4      s=-(DF(x0)'*DF(x0))\(DF(x0)'*F(x0));
5      while norm(s)>tol && k<maxit
6          k=k+1;
7          x=x+s;
8          s=-(DF(x)'*DF(x))\(DF(x)'*F(x));
9      end
10 end
```

1.2 KONVERGENZ DES GAUSS-NEWTON-VERFAHRENS

Zur Analyse des Gauß-Newton-Verfahrens sei nun x^* ein kritischer Punkt von g, d.h. $\nabla g(x^*) = 0$, der in einer Umgebung U eindeutig ist. Außerdem gelte

$$\mathrm{Rang}(F'(x)) = n \quad \forall\, x \in U.$$

Dann hat die Normalengleichung (1.4) für $x_k \in U$ genau eine Lösung

$$s_k = -[F'(x_k)^T F'(x_k)]^{-1} F'(x_k)^T F(x_k)$$

und die Gauß-Newton-Iteration kann folgendermaßen formuliert werden:

$$\begin{aligned} x_{k+1} &= x_k - [F'(x_k)^T F'(x_k)]^{-1} F'(x_k)^T F(x_k) \\ &= \Phi(x_k) \end{aligned}$$

mit

$$\Phi(x) := x - [F'(x)^T F'(x)]^{-1} F'(x)^T F(x)\,. \tag{1.6}$$

Das Gauß-Newton-Verfahren ist demzufolge eine Fixpunktiteration mit Iterationsfunktion Φ. Hinreichend für die lokale Konvergenz der Fixpunktiteration ist die Beschränktheit der Norm der Jacobi-Matrix im Fixpunkt

$$\|\Phi'(x^*)\| < 1$$

für eine beliebige Operatornorm $\|\cdot\|$. Da $\mathrm{Rang}(F'(x^*)) = n$ gilt, ist die Matrix $F'(x^*)^T F'(x^*)$ positiv definit. Daher existiert eine (eindeutige) symmetrische, positiv definite Matrix $A \in \mathbb{R}^{n\times n}$, sodass

$$A^2 = F'(x^*)^T F'(x^*).$$

Wir definieren nun für die weitere Konvergenzuntersuchung die Matrix $K \in \mathbb{R}^{n\times n}$ durch

$$K := -A^{-1} \left(\sum_{i=1}^{m} \frac{F_i(x^*)}{\|F(x^*)\|_2} F_i''(x^*) \right) A^{-1}.$$

Da K ebenfalls eine symmetrische Matrix ist, sind alle Eigenwerte von K reell. Mit den Matrizen A und K kann man nun die Hesse-Matrix $g''(x^*)$ und die Jacobi-Matrix $\Phi'(x^*)$ folgendermaßen darstellen.

Lemma 1.2.1 *Es gilt*

$$g''(x^*) = A(I - \|F(x^*)\|_2 K)A, \tag{1.7}$$

$$\Phi'(x^*) = \|F(x^*)\|_2 A^{-1} K A. \tag{1.8}$$

Wenn x^* ein lokales Maximum oder ein Sattelpunkt von g ist, muss

$$\varrho(K)\|F(x^*)\|_2 \geq 1$$

gelten, wobei $\varrho(K)$ den Spektralradius von K, also den Betrag des betragsgrößten Eigenwertes von K, bezeichnet.

Beweis. Die Gleichungen (1.7) und (1.8) erhält man schnell durch Nachrechnen.
Damit $g''(x^*)$ nicht positiv definit ist, muss die Matrix $I - \|F(x^*)\|_2 K$ nichtpositive Eigenwerte besitzen, d.h. das Spektrum von $\|F(x^*)\|_2 K$ muss Werte ≥ 1 enthalten. Daher muss $\varrho(K)\|F(x^*)\|_2 \geq 1$ gelten. □

Als ein weiteres Hilfsmittel definieren wir die Vektornorm $\|x\|_A := \|Ax\|_2$ sowie für $B \in \mathbb{R}^{n\times n}$ die zugehörige Matrixnorm

$$\|B\|_A = \max_{\|x\|_A=1} \|Bx\|_A = \|ABA^{-1}\|_2 \,.$$

Satz 1.2.2 *Für die Gauß-Newton-Iterationsfunktion* Φ *aus* (1.6) *gilt*

$$\begin{aligned} &\|\Phi'(x^*)\|_A = \varrho(K)\|F(x^*)\|_2 \,, \\ &\|\Phi'(x^*)\| \geq \varrho(K)\|F(x^*)\|_2 \quad \textit{für jede Operatornorm} \quad \|\cdot\|. \end{aligned}$$

Beweis. Aus Lemma 1.2.1 erhält man

$$\|\Phi'(x^*)\|_A = \|A\Phi'(x^*)A^{-1}\|_2 = \|F(x^*)\|_2\|K\|_2 = \|F(x^*)\|_2\,\varrho(K)\,.$$

Für jede Operatornorm $\|\cdot\|$ und jede Matrix $B \in \mathbb{R}^{n\times n}$ gilt $\|B\| \geq \varrho(B)$. Damit ergibt sich

$$\|\Phi'(x^*)\| \geq \varrho(\Phi'(x^*)) = \|F(x^*)\|_2\,\varrho(A^{-1}KA) = \|F(x^*)\|_2\,\varrho(K)\,.$$

□

Mit Satz 1.2.2 können wir nun folgende Aussagen über das Verhalten der Gauß-Newton-Iteration treffen.

Korollar 1.2.3 *i) Im Normalfall ist* $F(x^*) \neq 0$, $K \neq 0$ *und deshalb* $\Phi'(x^*) \neq 0$.

Falls das Gauß-Newton-Verfahren konvergiert, ist die Konvergenz im Allgemeinen nicht schneller als linear.

Dies steht im Gegensatz zum Newton-Verfahren (vgl. Numerik II), das in der Regel quadratisch konvergent ist. ⚠

ii) Wenn der kritische Punkt x^* *ein lokales Maximum oder ein Sattelpunkt ist, gilt* $\varrho(K)\|F(x^*)\|_2 \geq 1$ *und* $\|\Phi'(x^*)\| \geq 1$ *für jede Operatornorm* $\|\cdot\|$.

Lokale Maxima und Sattelpunkte sind für das Gauß-Newton-Verfahren also abstoßend, was günstig ist, da ein lokales Minimum gesucht wird.

iii) Die Größe $\varrho(K)\|F(x^*)\|_2$ *ist entscheidend für die lokale Konvergenz des Gauß-Newton-Verfahrens.*

Für ein lokales Minimum x^* *der Funktion* g *ist die lokale Konvergenz des Gauß-Newton-Verfahrens gesichert, falls das Residuum* $\|F(x^*)\|_2$ *und die Größe* $\varrho(K)$ *hinreichend klein sind, sodass die Bedingung* $\varrho(K)\|F(x^*)\|_2 < 1$ *erfüllt ist.*

iv) Ist x^* *allerdings ein lokales Minimum von* g, *für das* $\varrho(K)\|F(x^*)\|_2 > 1$ *gilt, so ist* $\|\Phi'(x^*)\| > 1$ *für jede Operatornorm* $\|\cdot\|$.

Ein lokales Minimum von g *kann für das Gauß-Newton-Verfahren abstoßend sein.*

1.3 LEVENBERG-MARQUARDT-VERFAHREN

Das Problem des Gauß-Newton-Verfahrens ist, dass die Linearisierung nur für „kleine“ Schritte s_k zulässig ist, sofern die Iterierte x_k noch weit vom Minimum entfernt ist. Die folgende Methode versucht durch eine Steuerung der Länge des Korrekturvektors s_k eine Verbesserung zu erzielen.

Im Levenberg-Marquardt-Verfahren wird das Ausgleichsproblem (1.5) zur Bestimmung des Korrekturvektors s_k durch ein anderes leicht abgeändertes Minimierungsproblem

$$\|F'(x_k)\, s_k + F(x_k)\|_2^2 + \mu^2 \|s_k\|_2^2 \to \min \tag{1.9}$$

ersetzt, wobei $\mu > 0$ ein zu wählender Parameter ist. Als neue Annäherung wird dann wiederum

$$x_{k+1} = x_k + s_k$$

gesetzt. Aus der Gleichung

$$\left\| \begin{pmatrix} F'(x_k) \\ \mu I \end{pmatrix} s_k + \begin{pmatrix} F(x_k) \\ 0 \end{pmatrix} \right\|_2^2 = \|F'(x_k) s_k + F(x_k)\|_2^2 + \mu^2 \|s_k\|_2^2$$

folgt, dass die Minimierungsaufgabe (1.9) die äquivalente Formulierung

$$\left\| \begin{pmatrix} F'(x_k) \\ \mu I \end{pmatrix} s_k + \begin{pmatrix} F(x_k) \\ 0 \end{pmatrix} \right\|_2 \to \min \tag{1.10}$$

besitzt. Im Vergleich zum Ausgleichsproblem (1.5) im Gauß-Newton-Verfahren besitzt (1.10) immer eine eindeutige Lösung s_k, da die Matrix $\binom{F'(x_k)}{\mu I}$ vollen Rang hat.
Für den Korrekturvektor $s_k \neq 0$ gilt

$$\begin{aligned} \mu^2 \|s_k\|_2^2 &\leq \|F'(x_k)\, s_k + F(x_k)\|_2^2 + \mu^2 \|s_k\|_2^2 \\ &= \min_{s \in \mathbb{R}^n} \left\{ \|F'(x_k)\, s + F(x_k)\|_2^2 + \mu^2 \|s\|_2^2 \right\} \leq \|F(x_k)\|_2^2 \end{aligned}$$

und daher

$$\|s_k\|_2 \leq \frac{\|F(x_k)\|_2}{\mu}.$$

Der Parameter $\mu > 0$ kann folglich eine Dämpfung der Korrektur s_k bewirken und durch eine geeignete Wahl von μ kann man eine zu große Korrektur vermeiden. Man kann zeigen, dass unter bestimmten Voraussetzungen an F das Levenberg-Marquardt-Verfahren für „hinreichend großes“ μ konvergiert. Um Konvergenz zu gewährleisten darf man μ also nicht zu klein wählen; auf der anderen Seite führt ein großes μ aber nur zu einer kleinen Korrektur und somit erhält man nur sehr langsame Konvergenz.
Im Folgenden wird ein mögliches Verfahren zur Bestimmung des Parameters μ vorgestellt, in dem in jedem Schritt überprüft wird, ob μ zu klein oder zu groß gewählt ist, und μ gegebenenfalls angepasst wird.

Sei dazu $x_k \in \mathbb{R}^n$, $s_k = s_k(\mu)$ die Korrektur aus (1.9) sowie

$$\varepsilon_\mu := \frac{\|F(x_k)\|_2^2 - \|F(x_k + s_k)\|_2^2}{\|F(x_k)\|_2^2 - \|F(x_k) + F'(x_k)\, s_k\|_2^2} =: \frac{\Delta R(x_k, s_k)}{\Delta \widetilde{R}(x_k, s_k)},$$

wobei angenommen wird, dass $\Delta \widetilde{R}(x_k, s_k) \neq 0$ gilt. ε_μ beschreibt die Änderung des tatsächlichen Residuums $\Delta R(x_k, s_k)$ in Relation zu Änderung des Residuums im linearen Modell (Gauß-Newton-Modell) $\Delta \widetilde{R}(x_k, s_k)$. Aus (1.9) folgt, dass $\Delta \widetilde{R}(x_k, s_k) \geq 0$.

Für eine akzeptable Korrektur muss außerdem $\Delta R(x_k, s_k) > 0$ gelten, also $\varepsilon_\mu > 0$. Aufgrund der Taylorentwicklung konvergiert $\varepsilon_\mu \to 1$ für $\mu \to \infty$. Gibt man sich nun β_0, β_1 mit $0 < \beta_0 < \beta_1 < 1$ vor (z.B. $\beta_0 = 0.3, \beta_1 = 0.9$), so kann man zur Parametersteuerung folgendes Kriterium verwenden:

$\varepsilon_\mu \le \beta_0$:	s_k wird nicht akzeptiert; μ wird vergrößert, z.B. $\mu \mapsto 2\mu$, und die neue zugehörige Korrektur s_k wird berechnet (*Gewährleistung der Konvergenz*).
$\beta_0 < \varepsilon_\mu < \beta_1$:	s_k wird akzeptiert; bei der Berechnung von s_{k+1} wird als Anfangswert dasselbe μ gewählt.
$\varepsilon_\mu \ge \beta_1$:	s_k wird akzeptiert; bei der Berechnung von s_{k+1} wird als Anfangswert ein kleineres μ, z.B. $\frac{\mu}{2}$ genommen (*Effektivität*).

Insgesamt erhalten wir für das Levenberg-Marquardt-Verfahren folgenden Algorithmus.

Algorithmus 1.3.1: Levenberg-Marquardt-Verfahren

Input: x_0, F, F', μ, β_0, β_1
for $k = 0, 1, \ldots$

i) Berechne $F(x_k)$, $F'(x_k)$.

ii) Bestimme den Korrekturvektor s_k gemäß

$$\left[F'(x_k)^T F'(x_k) + \mu^2 I\right] s_k = -F'(x_k)^T F(x_k)\,.$$

iii) Teste, ob die Korrektur s_k akzeptabel ist:

- $\varepsilon_\mu \le \beta_0$: Setze $\mu = 2\mu$ und berechne s_k gemäß ii) neu.
- $\varepsilon_\mu \ge \beta_1$: Setze $\mu = \frac{\mu}{2}$ und behalte s_k.

iv) Setze $x_{k+1} = x_k + s_k$.

end

In Matlab sieht eine Realisierung dieses Verfahrens folgendermaßen aus.

MATLAB-Funktion: levenberg_marquardt.m

```
1  function x = levenberg_marquardt(F,DF,x0,mu0,beta0,beta1,maxit,tol)
2      n=length(x0);
3      k=0;
4      mu=mu0;
5      x=x0;
6      s=-(DF(x0)'*DF(x0)+mu^2*eye(n))\(DF(x0)'*F(x0));
7      while norm(s)>tol && k<maxit
8          [s,mu] = korrektur(F,DF,x,mu,beta0,beta1);
9          x=x+s;
10         k=k+1;
11     end
12 end
```

```
13
14 function [s,mu] = korrektur(F,DF,x,mu,beta0,beta1)
15     n=length(x);
16     s=-(DF(x)'*DF(x)+mu^2*eye(n))\(DF(x)'*F(x));
17     eps_mu=(F(x)'*F(x)-F(x+s)'*F(x+s))/...
18         (F(x)'*F(x)-(F(x)+DF(x)*s)'*(F(x)+DF(x)*s));
19     if eps_mu <= beta0
20         [s,mu]=korrektur(F,DF,x,2*mu,beta0,beta1);
21     elseif eps_mu >= beta1
22         mu=mu/2;
23     end
24 end
```

2 EIGENWERTPROBLEME

Viele numerische Lösungsansätze von z.B. physikalischen Problemen erfordern die Lösung eines sogenannten Eigenwertproblems. Zu einer Matrix $A \in \mathbb{R}^{n\times n}$ finde man eine Zahl $\lambda \in \mathbb{C}$ und einen Vektor $v \in \mathbb{C}^n, v \neq 0$, sodass die *Eigenwertgleichung*

$$Av = \lambda v$$

erfüllt ist. Die Zahl λ heißt *Eigenwert* und der Vektor v *Eigenvektor* zum Eigenwert λ.
Betrachten wir nun zunächst einige konkrete Beispiele, die auf ein Eigenwertproblem führen.

Beispiel 2.0.1 (Sturm-Liouville-Problem) Die mathematische Modellierung zur Beschreibung der Überlagerung von Schwingungsvorgängen – wie sie etwa beim Brückenbau durchgeführt wird, um Resonanzen zu vermeiden, die einen Brückeneinsturz verursachen könnten – führt auf das sogenannte Sturm-Liouville Problem: Zu einer bekannten stetigen Funktion $r(x) > 0, x \in [0, 1]$, finde man die Zahl λ und die Funktion $u(x)$, die die Differenzialgleichung

$$-u''(x) - \lambda\, r(x)\, u(x) = 0, \quad x \in (0, 1) \tag{2.1}$$

mit den Randbedingungen

$$u(0) = u(1) = 0$$

erfüllen. Um die Lösung über ein Diskretisierungsverfahren numerisch anzunähern, betrachten wir Gitterpunkte

$$x_j = jh, \quad j = 0, \ldots, n, \quad h = \frac{1}{n}$$

und ersetzen $u''(x_j)$ durch den zweiten zentrierten Differenzenquotienten

$$\frac{u(x_j + h) - 2u(x_j) + u(x_j - h)}{h^2}, \quad j = 1, \ldots, n - 1.$$

Es ergibt sich so ein System

$$Au - \lambda Ru = 0 \tag{2.2}$$

für die Unbekannten λ und $u_i \approx u(x_i), i = 1, \ldots, n - 1$, wobei

$$A = \frac{1}{h^2}\begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{pmatrix}, \quad R = \begin{pmatrix} r(x_1) & & & 0 \\ & r(x_2) & & \\ & & \ddots & \\ 0 & & & r(x_{n-1}) \end{pmatrix}. \tag{2.3}$$

Mit $R^{1/2} = \operatorname{diag}(\sqrt{r(x_1)}, \ldots, \sqrt{r(x_{n-1})})$, $R^{-1/2} = (R^{1/2})^{-1}$, $B := R^{-1/2}AR^{-1/2}$ sowie $v = R^{1/2}u$ erhält man aus (2.2) die transformierte Gleichung

$$Bv = \lambda v,$$

also ein Eigenwertproblem.

Bemerkung 2.0.2 *Da R regulär ist, hätte man die Gleichung (2.2) auch von links mit R^{-1} multiplizieren und so das Eigenwertproblem $R^{-1}Au = \lambda u$ erhalten können. Im Gegensatz zur Matrix B ist die Matrix $R^{-1}A$ jedoch nicht symmetrisch.*

Beispiel 2.0.3 (Wellengleichung auf $[0,1]$) Wir betrachten für $c > 0$ die Wellengleichung

$$\frac{1}{c^2}\, u_{tt}(t,x) - u_{xx}(t,x) = 0, \qquad t > 0,\ x \in (0,1) \tag{2.4}$$

mit den Anfangs- und Randbedingungen

$$u(t,0) = u(t,1) = 0, \qquad t > 0 \tag{2.5}$$
$$u(0,x) = u_0(x), \qquad x \in [0,1] \tag{2.6}$$
$$u_t(0,x) = u_1(x), \qquad x \in [0,1]\,. \tag{2.7}$$

Wendet man nun den Separationsansatz $u(t,x) = T(t) \cdot X(x)$ auf (2.4) an, so ergibt sich

$$\frac{1}{c^2}\, T''(t)X(x) - T(t)X''(x) = 0 \quad \Leftrightarrow \quad \frac{1}{c^2}\frac{T''(t)}{T(t)} = \frac{X''(x)}{X(x)} = const. = -\lambda.$$

Es folgt

$$-T''(t) - \lambda c^2\, T(t) = 0, \tag{2.8}$$
$$-X''(x) - \lambda X(x) = 0, \tag{2.9}$$
$$X(0) = X(1) = 0. \tag{2.10}$$

Es ist also als Teilproblem das Sturm-Liouville-Problem (2.9) – (2.10) bzgl. X zu lösen. Zu gegebenem $\lambda \in \mathbb{R}$ muss die Funktion T, welche (2.8) genügt, von der Form

$$T(t) = a\cos(\sqrt{\lambda}\, ct) + b\sin(\sqrt{\lambda}\, ct)$$

sein. Kann man nun m Eigenpaare (λ_k, X_k), $k = 1, \ldots, m$ des Sturm-Liouville-Problems näherungsweise numerisch bestimmen, so ist für beliebige $a_k, b_k \in \mathbb{R}$

$$\hat{u}(t,x) := \sum_{k=1}^{m} \Big(a_k \cos(\sqrt{\lambda_k}\, ct) + b_k \sin(\sqrt{\lambda_k}\, ct) \Big)\, X_k(x)$$

eine Näherungslösung von (2.4) – (2.5). Durch geeignete Wahl der Koeffizienten a_k und b_k kann man erreichen, dass auch die Anfangswerte (2.6) und (2.7) näherungsweise angenommen werden. Hierbei fordert man die Übereinstimmungen

$$\hat{u}(0,x_i) = \sum_{k=1}^{m} a_k\, X_k(x_i) \stackrel{!}{=} u_0(x_i),$$
$$\hat{u}_t(0,x_i) = \sum_{k=1}^{m} b_k \sqrt{\lambda_k}\, c\, X_k(x_i) \stackrel{!}{=} u_1(x_i), \qquad i = 0, \ldots, n,$$

wobei x_i die Gitterpunkte aus der Diskretisierung des Sturm-Liouville-Problems seien. Dies ist ein lineares Gleichungssystem, aus dem man die Koeffizienten bestimmen kann.

Bemerkung 2.0.4 (Fourierreihen) *Man kann zeigen, dass $\lambda_k = k^2\pi^2$ für $k \in \mathbb{N}$ die Eigenwerte des Sturm-Liouville-Problems* (2.9) – (2.10) *sind mit den zugehörigen Eigenfunktionen $X_k(x) = \sin(k\pi x)$. Beachtet man nun, dass für $N \in \mathbb{N}$ die endliche Linearkombination*

$$u_N(t,x) := \sum_{k=1}^{N} \Big(a_k \cos(k\pi ct) + b_k \sin(k\pi ct) \Big)\, \sin(k\pi x)$$

die Anfangswerte

$$u_N(0,x) = \sum_{k=1}^{N} a_k \sin(k\pi x)$$
$$\frac{\partial}{\partial t} u_N(0,x) = \sum_{k=1}^{N} b_k k\pi c \sin(k\pi x)$$

annimmt und wählt a_k als die Fourierkoeffizienten von u_0

$$a_k = 2\int_0^1 u_0(s)\sin(k\pi s)\,\mathrm{d}s$$

sowie b_k gemäß $b_k = \frac{\beta_k}{k\pi c}$ mit den Fourierkoeffizienten β_k von u_1

$$\beta_k = 2\int_0^1 u_1(s)\sin(k\pi s)\,\mathrm{d}s,$$

so löst unter bestimmten Differenzierbarkeitsanforderungen an u_0 und u_1 die Funktion

$$u(t,x) := \lim_{N\to\infty} u_N(t,x) = \sum_{k=1}^{\infty} \Big(a_k\cos(k\pi ct) + b_k \sin(k\pi ct)\Big)\sin(k\pi x)$$

die Wellengleichung auf $[0,1]$.

Eigenwerte beschreiben nicht nur physikalische Eigenschaften, sondern spielen auch eine wichtige Rolle in der Mathematik, beispielsweise in der Numerik, um die Norm oder Konditionszahl einer Matrix zu bestimmen.

Beispiel 2.0.5 (Norm und Konditionszahl) i) Für eine beliebige Matrix $A \in \mathbb{R}^{n\times n}$ gilt

$$\|A\|_2 = \sqrt{\varrho(A^T A)}\,,$$

wobei ϱ wiederum den Spektralradius einer Matrix, also den Betrag des betragsgrößten Eigenwertes, bezeichnet.

ii) Für eine symmetrische Matrix $A \in \mathbb{R}^{n\times n}$ gilt

$$\|A\|_2 = \varrho(A)\,.$$

iii) Für eine nichtsinguläre symmetrische Matrix $A \in \mathbb{R}^{n\times n}$ gilt

$$\kappa_2(A) = \varrho(A)\,\varrho(A^{-1})\,,$$

wobei $\kappa_2(A) = \|A\|_2\|A^{-1}\|_2$ die Konditionszahl von A bzgl. der 2-Norm bezeichnet.

2.1 Grundlagen aus der linearen Algebra

Bevor wir uns der Abschätzung sowie der numerischen Berechnung von Eigenwerten widmen, werden an dieser Stelle zunächst elementare Ergebnisse aus der linearen Algebra zusammengefasst, welche in Zusammenhang mit der Eigenwertberechnung stehen.
Sei $A \in \mathbb{C}^{n\times n}$, $\lambda \in \mathbb{C}$ ein Eigenwert der Matrix A und $v \in \mathbb{C}\setminus\{0\}$ ein zugehöriger Eigenvektor:

$$Av = \lambda v. \tag{2.11}$$

Zusätzlich zur Definition (2.11) eines (rechtsseitigen) Eigenvektors $v \in \mathbb{C}^n \setminus \{0\}$ kann man auch linksseitige Eigenvektoren betrachten. $x \in \mathbb{C}^n \setminus \{0\}$ heißt linksseitiger Eigenvektor von A, falls $x^H A = \lambda x^H$ gilt, wobei x^H denjenigen Vektor bezeichnet, der aus x durch Bildung der konjugiert komplexen Einträge und Transponieren hervorgeht.

Satz 2.1.1 *Die Eigenwerte einer Matrix $A \in \mathbb{C}^{n\times n}$ sind gerade die n Nullstellen des zugehörigen charakteristischen Polynoms*

$$P_A(\lambda) := \det(A - \lambda I)\,.$$

Beweis. Die Eigenwertgleichung (2.11) besagt offensichtlich, dass $(A - \lambda I)v = 0$ gilt, also die Matrix $A - \lambda I$ singulär ist. Letzteres ist zu $\det(A - \lambda I) = 0$ äquivalent. □

Definition 2.1.2 *Die Menge aller Eigenwerte einer Matrix $A \in \mathbb{C}^{n\times n}$ wird als Spektrum bezeichnet und man verwendet die Schreibweise*

$$\sigma(A) = \{\lambda \in \mathbb{C} \mid \lambda \text{ ist Eigenwert von } A\}\,.$$

Satz 2.1.3 *Sei $\sigma(A) = \{\lambda_1, \ldots, \lambda_n\}$. Es besteht folgender Zusammenhang zwischen der Determinante bzw. der Spur von A und den Eigenwerten λ_i:*

$$\det(A) = \prod_{i=1}^{n} \lambda_i\,,$$
$$\operatorname{trace}(A) = \sum_{i=1}^{n} a_{ii} = \sum_{i=1}^{n} \lambda_i\,.$$

Beweis. Die zweite Identität erhält man z.B. dadurch, dass man den Koeffizienten vor λ^{n-1} im charakteristischen Polynom untersucht.

$$\begin{aligned} P_A(\lambda) &= \det \begin{pmatrix} a_{11}-\lambda & a_{12} & \cdots & a_{1n} \\ a_{21} & \ddots & & \\ \vdots & & \ddots & \\ a_{n1} & & & a_{nn}-\lambda \end{pmatrix} \\ &= (a_{11}-\lambda)\cdot\ldots\cdot(a_{nn}-\lambda) + \text{Terme mit weniger als } (n-1)\ (a_{ii}-\lambda)\text{-Termen} \\ &= (-\lambda)^n + (-\lambda)^{n-1}(a_{11}+a_{22}+\ldots+a_{nn}) + \sum_{k=0}^{n-2} c_k\lambda^k \\ &\overset{!}{=} (\lambda_1-\lambda)(\lambda_2-\lambda)\cdot\ldots\cdot(\lambda_n-\lambda) \end{aligned}$$

□

Ein Unterraum $S \subseteq \mathbb{C}^n$ mit der Eigenschaft

$$x \in S \Rightarrow Ax \in S$$

wird als invariant bzgl. A bezeichnet. Der Aufspann eines (rechtsseitigen) Eigenvektors bildet somit einen eindimensionalen Unterraum, der invariant bleibt unter der Multiplikation mit A. Es sei

$$AX = XB, \quad B \in \mathbb{C}^{k\times k}, \quad X \in \mathbb{C}^{n\times k},$$

dann ist $\operatorname{Bild}(X)$ invariant bzgl. A und aus $By = \lambda y$, $y \in \mathbb{C}^k \setminus \{0\}$ folgt

$$A(Xy) = (AX)y = (XB)y = \lambda Xy\,.$$

Falls X vollen Spaltenrang hat, dann impliziert $AX = XB$ also $\sigma(B) \subseteq \sigma(A)$. Für den Fall einer regulären quadratischen Matrix X gilt für das Spektrum von A und $B = X^{-1}AX$ folgendes Ergebnis.

Lemma 2.1.4 *Sei $A \in \mathbb{C}^{n\times n}$ beliebig sowie $X \in \mathbb{C}^{n\times n}$ eine reguläre Matrix. Dann gilt*

$$\sigma(A) = \sigma(X^{-1}AX)\,.$$

Ähnliche Matrizen haben also das gleiche Spektrum.

Beweis. Die Aussage folgt mit direkt mit Satz 2.1.1 aus der Tatsache, dass ähnliche Matrizen das gleiche charakteristische Polynom haben:

$$\det(X^{-1}AX) = \det(X^{-1}(A-\lambda I)X) = \det(X^{-1})\det(A-\lambda I)\det(X) = \det(A-\lambda I)\,.$$

□

Multipliziert man eine hermitesche Matrix $A \in \mathbb{C}^{n\times n}$ mit einer nichtsingulären Matrix $X \in \mathbb{C}^{n\times n}$ wie folgt

$$X^H AX = B \in \mathbb{C}^{n\times n},$$

so hat B im Allgemeinen nicht die gleichen Eigenwerte wie A. Der Sylvester'sche Trägheitssatz sagt aus, dass wenigstens die Vorzeichen der Eigenwerte sich nicht ändern.

Satz 2.1.5 (Sylvester'scher Trägheitssatz) *Es sei $A \in C^{n\times n}$ hermitesch und $X \in C^{n\times n}$ regulär. Dann haben A und $X^H AX$ den gleichen Rang sowie die gleiche Anzahl positiver bzw. negativer Eigenwerte.*

Beweis. Siehe z.B. [Fischer]. □

Wir kommen nun zu einer wichtigen Matrixfaktorisierung, der Schur-Zerlegung. Im Gegensatz zu der ebenfalls aus der linearen Algebra bekannten Jordan'schen Normalform führt ihre Berechnung auf numerisch stabile Algorithmen, da man hierfür nur orthogonale bzw. unitäre Transformationen benötigt und diese normerhaltend sind.
Diese Faktorisierung spielt beim QR-Verfahren zur Berechnung von Eigenwerten eine zentrale Rolle.

Satz 2.1.6 (Satz von Schur) *Sei $A \in C^{n\times n}$. Dann gibt es eine unitäre Matrix $Q \in C^{n\times n}$ ($QQ^H = Q^HQ = I$), so dass*

$$Q^H AQ = R$$

ist, wobei $R \in C^{n\times n}$ eine obere Dreiecksmatrix ist und die Diagonaleinträge $r_{11}, \ldots, r_{nn}$ von R die Eigenwerte von A sind.

Ist A reell, so sollte man versuchen mit reeller Arithmetik auszukommen. Allerdings ist R dann nur eine obere Block-Dreiecksmatrix mit 1×1 oder 2×2 Blöcken.

Satz 2.1.7 (Reelle Schur-Zerlegung) *Für jedes $A \in \mathbb{R}^{n\times n}$, gibt es eine orthogonale Matrix $Q \in \mathbb{R}^{n\times n}$, so dass Q^TAQ in oberer Quasi-Dreiecksform*

$$Q^TAQ = \begin{pmatrix} R_{11} & \cdots & R_{1m} \\ & \ddots & \vdots \\ & & R_{mm} \end{pmatrix}$$

ist, wobei jedes R_{ii} entweder ein 1×1 oder 2×2 Block ist. Dabei sind die 1×1 Blöcke unter $R_{11}, \ldots, R_{mm}$ die reellen Eigenwerte von A und die 2×2 Blöcke enthalten die Paare von komplex-konjugierten Eigenwerten von A.

Zum Schluss dieses Abschnitts betrachten wir noch Matrixfaktorisierungen für eine besondere Klasse von Matrizen.

Definition 2.1.8 *$A \in \mathbb{C}^{n\times n}$ heißt normal, wenn gilt*

$$A^H A = AA^H.$$

Bemerkung 2.1.9 *Alle hermiteschen, schiefhermiteschen ($A^H = -A$), Diagonal- und unitären Matrizen sind Beispiele für normale Matrizen.*

Korollar 2.1.10 (Schur-Zerlegung von normalen Martizen) *Eine Matrix $A \in \mathbb{C}^{n\times n}$ ist genau dann normal, wenn eine unitäre Matrix $Q \in \mathbb{C}^{n\times n}$ existiert mit $Q^H AQ = D = \operatorname{diag}(\lambda_1, \ldots, \lambda_n)$.*

Beweis. Aus der unitären Ähnlichkeit von A zu einer Diagonalmatrix, folgt offensichtlich, dass A normal ist. Andererseits sei A normal und $Q^H AQ = R$ sei die zugehörige Schur'sche Normalform. Dann ist auch R normal.

$$R^H R = Q^H A^H QQ^H AQ = Q^H A^H AQ = Q^H AA^H Q = Q^H AQQ^H A^H Q = RR^H$$

Die Behauptung folgt nun aus der Tatsache, dass eine normale, obere Dreiecksmatrix eine Diagonalmatrix ist. □

Korollar 2.1.11 (Schur-Zerlegung von symmetrischen Martizen) *Jede reelle, symmetrische Matrix $A \in \mathbb{R}^{n\times n}$ lässt sich mittels einer orthogonalen Matrix $Q \in \mathbb{R}^{n\times n}$ auf Diagonalgestalt transformieren*

$$Q^T AQ = D = \operatorname{diag}(\lambda_1, \ldots, \lambda_n)\,,$$

wobei $\lambda_1, \ldots, \lambda_n$ die reellen Eigenwerte von A sind.

2.2 Abschätzungen und geometrische Lage der Eigenwerte

Im Folgenden werden einige Abschätzungen für das Spektrum einer Matrix $A \in \mathbb{C}^{n\times n}$ vorgestellt. Solche Abschätzungen können später bei der Wahl geeigneter Startwerte für lokal konvergente Iterationsverfahren zur Bestimmung der Eigenwerte nützlich sein. Eine erste Eingrenzung der Lage der Eigenwerte von A liefert der folgende Satz.

Satz 2.2.1 (Abschätzung mittels Matrixnorm) *Sei $\|\cdot\|$ eine konsistente Matrixnorm, d.h. es gebe eine Vektornorm $\|\cdot\|$ auf $\mathbb{C}^n$, sodass $\|Av\| \le \|A\|\,\|v\|$, dann gilt*

$$|\lambda| \le \|A\| \qquad \forall\, \lambda \in \sigma(A)\,.$$

Beweis. Seien λ ein Eigenwert von A und $v \neq 0$ ein dazugehöriger Eigenvektor. Da $\|\cdot\|$ konsistent ist, erhält man

$$|\lambda|\,\|v\| = \|\lambda v\| = \|Av\| \le \|A\|\,\|v\|,$$

sodass $|\lambda| \le \|A\|$ folgt. □

Definition 2.2.2 (Wertebereich einer Matrix) *Sei $A \in \mathbb{K}^{n\times n}$ ($\mathbb{K} = \mathbb{R}$ oder $\mathbb{K} = \mathbb{C}$). Die Menge aller Rayleigh-Quotienten $\frac{x^H Ax}{x^H x}$ mit $x \in \mathbb{C}^n \setminus \{0\}$, also*

$$W(A) := \left\{ \frac{x^H Ax}{x^H x} : x \in \mathbb{C}^n \setminus \{0\} \right\}$$

wird als Wertebereich der Matrix A bezeichnet.

Bemerkungen 2.2.3 *i) Selbst für eine reelle Matrix $A \in \mathbb{R}^{n\times n}$ werden beim Wertebereich alle komplexen Vektoren $x \in \mathbb{C}^n \setminus \{0\}$ durchlaufen.*

ii) Der Wertebereich $W(A)$ enthält alle Eigenwerte von A. Ist $x \in \mathbb{C}^n \setminus \{0\}$ ein Eigenvektor von A, dann ist der zugehörige Rayleigh-Quotient gerade der Eigenwert.

Lemma 2.2.4 (Eigenschaften des Wertebereichs) *Sei $A \in \mathbb{K}^{n\times n}$, $\mathbb{K} = \mathbb{R}$ oder $\mathbb{C}$.*

i) W(A) ist zusammenhängend.

ii) Ist A hermitesch, dann ist $W(A)$ das reelle Intervall $[\lambda_{\min}, \lambda_{\max}]$.

iii) Ist A schiefhermitesch ($A = -A^H$), dann ist $W(A)$ eine rein imaginäre Menge, nämlich die konvexe Hülle der Eigenwerte von A.

Satz 2.2.5 (Bendixson) *Das Spektrum von $A \in \mathbb{K}^{n\times n}$ ist in dem Rechteck*

$$R = W\left(\frac{A+A^H}{2}\right) + W\left(\frac{A-A^H}{2}\right)$$

enthalten, wobei die Summe zweier Menge folgendermaßen zu verstehen ist

$$A + B = \{a + b : a \in A, b \in B\}.$$

Beweis. Wir zeigen die stärkere Aussage, dass der Wertebereich von A in dem Rechteck R enthalten ist. Sei dazu $x \in \mathbb{C}^n \setminus \{0\}$. Dann gilt

$$\begin{aligned}\frac{x^HAx}{x^Hx} &= \frac{1}{x^Hx}\, x^H\left(\frac{A+A^H}{2} + \frac{A-A^H}{2}\right)x \\ &= \frac{1}{x^Hx}\, x^H\left(\frac{A+A^H}{2}\right)x + \frac{1}{x^Hx}\, x^H\left(\frac{A-A^H}{2}\right)x \;\in\; W\left(\frac{A+A^H}{2}\right) + W\left(\frac{A-A^H}{2}\right)\end{aligned}$$

□

Bemerkung 2.2.6 *Da $\frac{A+A^H}{2}$ hermitesch und $\frac{A-A^H}{2}$ schiefhermitesch sind, besagt der Satz von Bendixson, dass für jedes $\lambda \in \sigma(A)$ gilt*

$$\lambda_{\min}\left(\frac{A+A^H}{2}\right) \le \mathrm{Re}(\lambda) \le \lambda_{\max}\left(\frac{A+A^H}{2}\right)$$

$$\min\left\{\mathrm{Im}(\mu)\,|\,\mu \in \sigma\left(\frac{A-A^H}{2}\right)\right\} \le \mathrm{Im}(\lambda) \le \max\left\{\mathrm{Im}(\mu)\,|\,\mu \in \sigma\left(\frac{A-A^H}{2}\right)\right\}.$$

Eine weitere *a priori* Schranke wird durch das folgende Resultat gegeben.

Satz 2.2.7 (Gerschgorin Kreise) *Sei $A \in \mathbb{C}^{n\times n}$. Dann gilt*

$$\sigma(A) \subseteq \mathcal{S}_{\mathcal{R}} := \bigcup_{i=1}^{n} \mathcal{R}_i, \qquad \mathcal{R}_i = \left\{z \in \mathbb{C} : |z - a_{ii}| \le \sum_{\substack{j=1\\ j\neq i}}^{n} |a_{ij}|\right\}. \tag{2.12}$$

Die Mengen $\mathcal{R}_i$ werden Gerschgorin Kreise genannt.

Beweis. Sei $\lambda \in \sigma(A)$ und $Ax = \lambda x$ für ein $x \neq 0$. Dann existiert ein x_i mit $|x_j| \le |x_i|\ \forall\, i \neq j$. $(Ax)_i$ sei die i-te Komponente von Ax, dann ist

$$\begin{aligned} & \lambda x_i = (Ax)_i = \sum_{j=1}^{n} a_{ij}\, x_j \\ \Leftrightarrow \quad & \lambda - a_{ii} = \sum_{\substack{j=1\\ j\neq i}}^{n} a_{ij}\,\frac{x_j}{x_i} \\ \Rightarrow \quad & |\lambda - a_{ii}| = \left|\sum_{\substack{j=1\\ j\neq i}}^{n} a_{ij}\,\frac{x_j}{x_i}\right| \le \sum_{\substack{j=1\\ j\neq i}}^{n} |a_{ij}|\,\frac{|x_j|}{|x_i|} \le \sum_{\substack{j=1\\ j\neq i}}^{n} |a_{ij}|, \end{aligned}$$

also

$$\lambda \in \mathcal{R}_i \subset \bigcup_{i=1}^{n} \mathcal{R}_i \,.$$

□

Da A und A^T das gleiche Spektrum haben, gilt Satz 2.2.7 auch in der Form

$$\sigma(A) \subseteq \mathcal{S}_{\mathcal{C}} := \bigcup_{j=1}^{n} \mathcal{C}_j, \qquad \mathcal{C}_j = \left\{ z \in \mathbb{C} : |z - a_{jj}| \leq \sum_{\substack{i=1 \\ i \neq j}}^{n} |a_{ij}| \right\}. \tag{2.13}$$

Die Aussagen (2.12) und (2.13) liefern zusammen das erste Gerschgorin Theorem.

Satz 2.2.8 (Erstes Gerschgorin Theorem) *Für eine Matrix $A \in \mathbb{C}^{n\times n}$ erfüllen die Eigenwerte folgende Eigenschaft*

$$\forall\, \lambda \in \sigma(A): \quad \lambda \in \mathcal{S}_{\mathcal{R}} \cap \mathcal{S}_{\mathcal{C}} \,.$$

Das zweite Gerschgorin Theorem liefert in bestimmten Fällen eine Aussage über die Verteilung der Eigenwerte auf die verschiedenen Gerschgorin Kreise.

Satz 2.2.9 (Zweites Gerschgorin Theorem) *Seien*

$$\mathcal{M}_1 := \bigcup_{j=1}^{k} \mathcal{R}_{i_j} \,, \quad \mathcal{M}_2 := \bigcup_{j=k+1}^{n} \mathcal{R}_{i_j} \,.$$

Ist die Vereinigung $\mathcal{M}_1$ von k Kreisen disjunkt von der Vereinigung $\mathcal{M}_2$ der übrigen $n-k$ Kreise, also $\mathcal{M}_1 \cap \mathcal{M}_2 = \emptyset$, so enthält $\mathcal{M}_1$ genau k und $\mathcal{M}_2$ die übrigen $n-k$ Eigenwerte, jeder entsprechend seiner algebraischen Vielfachheit gezählt.

Beweis. Sei $D := \operatorname{diag}(a_{11}, \ldots, a_{nn})$ sowie für $t \in [0,1]$

$$A_t := D + t(A - D).$$

Dann ist $A_0 = D$ und $A_1 = A$. Die Eigenwerte von A_t sind stetige Funktionen in t. Wendet man Satz 2.2.7 auf A_t an, so erhält man für $t = 0$, dass genau k Eigenwerte von $A_0 = D$ in $\mathcal{M}_1$ liegen und die restlichen $n-k$ in $\mathcal{M}_2$, wobei mehrfache Eigenwerte entsprechend ihrer algebraischen Vielfachheit gezählt werden. Für $0 \leq t \leq 1$ müssen alle Eigenwerte von A_t ebenfalls in den Kreisen liegen. Daher und aufgrund der Stetigkeit der Eigenwerte folgt, dass auch k Eigenwerte von $A_1 = A$ in $\mathcal{M}_1$ und die übrigen $n-k$ in $\mathcal{M}_2$ liegen. □

Um das dritte Gerschgorin Theorem formulieren zu können, erinnern wir uns folgende Eigenschaft von Matrizen, die wir bereits aus Numerik I kennen.

Definition 2.2.10 *Eine Matrix $A \in \mathbb{C}^{n\times n}$ heißt reduzibel, wenn eine Permutationsmatrix P existiert, sodass*

$$PAP^T = \begin{pmatrix} B_{11} & B_{12} \\ 0 & B_{22} \end{pmatrix}$$

mit quadratischen Matrizen B_{11} und B_{22} gilt. A ist irreduzibel, wenn A nicht reduzibel ist.

Satz 2.2.11 (Drittes Gerschgorin Theorem) *Sei $A \in \mathbb{C}^{n\times n}$ eine irreduzible Matrix. Ein Eigenwert $\lambda \in \sigma(A)$ kann nicht auf dem Rand von $\mathcal{S}_{\mathcal{R}}$ liegen, es sei denn er liegt auf dem Rand eines jeden Kreises $\mathcal{R}_i$ für $i = 1, \ldots, n$.*

Beispiel 2.2.12 Wir betrachten die Matrix

$$A = \begin{pmatrix} 4 & 0 & -3 \\ 0 & -1 & 1 \\ -1 & 1 & -2 \end{pmatrix} \quad \text{mit} \quad A^H = A^T = \begin{pmatrix} 4 & 0 & -1 \\ 0 & -1 & 1 \\ -3 & 1 & -2 \end{pmatrix}$$

und werden zunächst den Satz von Bendixson, Satz 2.2.5, anwenden, um die Eigenwerte der Matrix A abzuschätzen. Wir erhalten

$$\frac{A + A^H}{2} = \begin{pmatrix} 4 & 0 & -2 \\ 0 & -1 & 1 \\ -2 & 1 & -2 \end{pmatrix}, \quad \frac{A - A^H}{2} = \begin{pmatrix} 0 & 0 & -1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}.$$

Wendet man auf diese beiden Matrizen Satz 2.2.7 an, so ergeben sich folgende erste Abschätzungen eines Eigenwertes λ von A

$$-5 \leq \mathrm{Re}(\lambda) \leq 6, \qquad |\mathrm{Im}(\lambda)| \leq 1\,.$$

Die Gerschgorin Kreise von A ergeben sich zu

$$\begin{aligned} \mathcal{R}_1 &= \{z \in \mathbb{C} : |z - 4| \leq 3\} \\ \mathcal{R}_2 &= \{z \in \mathbb{C} : |z + 1| \leq 1\} \\ \mathcal{R}_3 &= \{z \in \mathbb{C} : |z + 2| \leq 2\} \end{aligned}$$

sowie diejenigen von A^T zu

$$\begin{aligned} \mathcal{C}_1 &= \{z \in \mathbb{C} : |z - 4| \leq 1\} \\ \mathcal{C}_2 &= \{z \in \mathbb{C} : |z + 1| \leq 1\} \\ \mathcal{C}_3 &= \{z \in \mathbb{C} : |z + 2| \leq 4\}\,. \end{aligned}$$

Die Abschätzung nach Bendixson und die Gerschgorin Kreise sind in Abb. 2.1 dargestellt. Hierbei ist zu beachten, dass man die Menge $\mathcal{R}_1 \cap \mathcal{C}_3$ durch Anwenden des zweiten Gerschgorin Theorems auf A und A^T als in Frage kommenden Bereich für die Eigenwerte von A ausschließen kann.

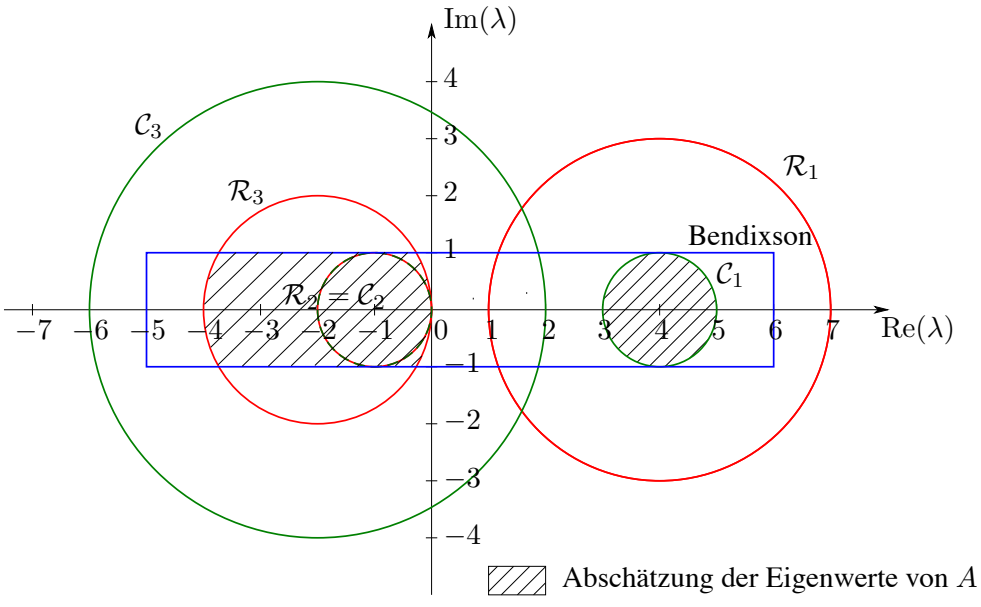


Abb. 2.1: Abschätzung nach Bendixson und Gerschgorin Kreise

2.3 POTENZMETHODE

Die Potenzmethode oder *Vektoriteration* ist eine sehr einfache, aber dennoch effektive Methode zur Bestimmung des betragsmäßig größten Eigenwertes.
Um die Grundidee des Verfahrens zu verstehen, nehmen wir zunächst an, dass die Matrix $A \in \mathbb{C}^{n\times n}$ diagonalisierbar ist. Dann gibt es eine Basis $v_1, \ldots, v_n$ von $\mathbb{C}^n$ aus Eigenvektoren v_i von A mit $\|v_i\| = 1$. Des Weiteren seien die Eigenwerte von A in der Form

$$|\lambda_1| > |\lambda_2| \geq |\lambda_3| \geq \ldots \geq |\lambda_n| \tag{2.14}$$

geordnet, wobei λ_1 die algebraische Vielfachheit 1 besäße. Gehen wir nun von einem Startvektor $x^{(0)}$ aus, so lässt sich dieser als Linearkombination der Eigenvektoren v_i schreiben:

$$x^{(0)} = \sum_{i=1}^{n} \alpha_i v_i \,. \tag{2.15}$$

Definiert man nun die Iterierten $a^{(k)}$, $k \in \mathbb{N}$, gemäß $a^{(k)} := A^k x^{(0)}$, so ergibt sich

$$a^{(k)} = A^k x^{(0)} = A^k \left(\sum_{i=1}^{n} \alpha_i v_i \right) = \sum_{i=1}^{n} \alpha_i \lambda_i^k \, v_i = \lambda_1^k \sum_{i=1}^{n} \alpha_i \left(\frac{\lambda_i}{\lambda_1} \right)^k v_i \,. \tag{2.16}$$

Falls der Koeffizient α_1 von Null verschieden ist, d.h. $x^{(0)}$ nicht auf v_1 senkrecht steht, wird sich dieser Ausdruck dem Summanden mit dem dominanten Eigenwert λ_1 annähern, also

$$a^{(k)} = A^k x^{(0)} \approx \lambda_1^k \, \alpha_1 \, v_1 \,.$$

Um in der Praxis einen Over- oder Underflow zu vermeiden, normiert man in jedem Iterationsschritt den Iterationsvektor. Insgesamt ergibt sich für die Potenzmethode folgender Algorithmus.

Algorithmus 2.3.1: Potenzmethode

`Input:` $x^{(0)}$ `mit` $v_1^T x^{(0)} \neq 0$ `und` $\|x^{(0)}\| = 1$
`for` $k = 0, 1, \ldots$

$$a^{(k)} = A x^{(k)}$$
$$\rho^{(k)} = x^{(k)\,T} a^{(k)} \qquad \% \text{ Rayleigh-Quotient}$$
$$x^{(k+1)} = \frac{a^{(k)}}{\|a^{(k)}\|}$$

`end`

Wir fassen die bisherigen Ergebnisse der Potenzmethode in einem Satz zusammen.

Satz 2.3.1 *Sei λ_1 ein einfacher Eigenwert der diagonalisierbaren Matrix $A \in \mathbb{C}^{n\times n}$ mit* (2.14) *und $x^{(0)} \in \mathbb{C}^n$ sei ein Vektor, der nicht senkrecht auf dem Eigenraum von λ_1 steht und $\|x^{(0)}\| = 1$ erfüllt. Dann konvergiert die Folge $x^{(k+1)} = a^{(k)}/\|a^{(k)}\|$ mit $a^{(k)} = Ax^{(k)}$ gegen einen normierten Eigenvektor von A zum Eigenwert λ_1.*

Beweis. Wir wissen aus (2.16), dass

$$a^{(k)} = A^k x^{(0)} = \sum_{i=1}^{n} \alpha_i \lambda_i^k \, v_i = \alpha_1 \lambda_1^k \underbrace{\left(v_1 + \sum_{i=2}^{n} \frac{\alpha_i}{\alpha_1} \left(\frac{\lambda_i}{\lambda_1} \right)^k v_i \right)}_{=:z_k}$$

für eine normierte Basis $v_1, \ldots, v_n$ des $\mathbb{C}^n$ aus Eigenvektoren von A gilt, wobei $\alpha_1 \neq 0$ ist, da $x^{(0)} \not\perp v_1$ vorausgesetzt wurde. Da $|\lambda_i| < |\lambda_1|$ für alle $i = 2, \ldots, n$ gilt, ist $\lim\limits_{k\to\infty} z_k = v_1$ und daher

$$x^{(k)} = \frac{a^{(k)}}{\|a^{(k)}\|} = \frac{z_k}{\|z_k\|} \to \pm v_1 \qquad \text{für } k \to \infty \,.$$

□

Bemerkungen 2.3.2 *i) Der Rayleigh-Quotient $\rho^{(k)}$ in Algorithmus 2.3.1 liefert im Grenzwert den betragsgrößten Eigenwert:*

$$\lim_{k\to\infty} \rho^{(k)} = \lambda_1 \,.$$

ii) Der Beweis des Satzes 2.3.1 lässt sich auch auf diagonalisierbare Matrizen mit eindeutig bestimmten betragsgrößtem Eigenwert λ_1, welcher aber nicht einfach zu sein braucht, d.h.

$$\lambda_1 = \lambda_2 = \ldots = \lambda_r$$
$$|\lambda_1| = \ldots = |\lambda_r| > |\lambda_{r+1}| \geq \ldots \geq |\lambda_n|$$

übertragen. Hierbei muss der Startvektor der Vektoriteration die Bedingung $\alpha_i \neq 0$ für ein $i \in \{1, \ldots, r\}$ erfüllen. Man beachte, dass für $r = 1$ (λ_1 ist also einfacher Eigenwert) der Grenzvektor v_1 ist und somit nicht von der Wahl von $x^{(0)}$ abhängt, sofern nur $\alpha_1 \neq 0$ ist. Ist λ_1 ein mehrfacher dominanter Eigenwert, $r > 1$, so hängt der gefundene Eigenvektor von den Verhältnissen $\alpha_1 : \alpha_2 : \ldots : \alpha_r$ und damit vom Startvektor $x^{(0)}$ ab.

iii) Man sieht in (2.16)*, dass für die Potenzmethode lineare Konvergenz mit Konvergenzfaktor $|\lambda_2/\lambda_1|$ bzw. $|\lambda_{r+1}/\lambda_1|$ vorliegt. Das Verfahren konvergiert also umso besser, je mehr die Beträge der Eigenwerte von A getrennt sind.*
Falls A symmetrisch ist, sind die Eigenvektoren v_i orthogonal. Mit Hilfe dieser Orthogonalitätseigenschaft kann man zeigen, dass in diesem Fall der Konvergenzfaktor sogar $|\lambda_2/\lambda_1|^2$ bzw. $|\lambda_{r+1}/\lambda_1|^2$ beträgt.

iv) Allgemein konvergiert das Verfahren nicht gegen λ_1 und einem zu λ_1 gehörigen Eigenvektor, sondern gegen λ_k und einem Eigenvektor zu λ_k, sofern in der Zerlegung von $x^{(0)}$ gilt

$$\alpha_1 = \alpha_2 = \ldots = \alpha_{k-1} = 0, \qquad \alpha_k \neq 0$$

und es keinen von λ_k verschiedenen Eigenwert gleichen Betrags gibt.
Praktisch konvergiert jedoch auch im Falle $\alpha_1 = 0$ das Verfahren gegen λ_1 und einem zugehörigen Eigenvektor, da infolge von Rundungsfehlern $\alpha_1^{(1)} \neq 0$ gilt ($\alpha_1^{(1)}$ der Koeffizient zu v_1 in $x^{(1)}$).

v) Da dieses Verfahren in jeder Iteration nur eine Matrix-Vektor-Multiplikation erfordert, ist der Aufwand der Potenzmethode nach k Iterationen $\mathcal{O}(kn^2)$.

Sei nun $A \in \mathbb{C}^{n\times n}$ eine *nicht diagonalisierbare* Matrix mit eindeutig bestimmtem betragsgrößtem Eigenwert λ_1, d.h. aus $|\lambda_1| = |\lambda_i|$ folgt $\lambda_1 = \lambda_i$. Ersetzt man die Darstellung (2.15) des Startvektors $x^{(0)}$ durch eine als Linearkombination von Eigen- und Hauptvektoren von A, so kann man auf dieselbe Weise wie im diagonalisierbaren Fall zeigen, dass unter analogen Voraussetzungen an $x^{(0)}$ in der Potenzmethode 2.3.1 der Rayleigh-Quotient $\rho^{(k)}$ gegen λ_1 und $x^{(k)}$ gegen einen zu λ_1 gehörigen Eigenvektor konvergieren.

Beispiel 2.3.3 Wir betrachten das Eigenwertproblem aus Beispiel 2.0.1 mit $R = I$, also $Ax = \lambda x$ mit $A \in \mathbb{R}^{(n-1)\times(n-1)}$ wie in (2.3). Die Eigenwerte der Matrix A lassen sich explizit angeben:

$$\lambda_{n-k} = \frac{4}{h^2}\sin^2\left(\frac{1}{2}k\pi h\right), \quad k = 1, 2, \ldots, n-1, \quad h := \frac{1}{n}.$$

Die Nummerierung ist hierbei so gewählt, dass $\lambda_1 > \lambda_2 > \ldots > \lambda_{n-1}$ gilt. Wegen

$$\begin{aligned}\left|\frac{\lambda_2}{\lambda_1}\right| = \frac{\lambda_2}{\lambda_1} &= \frac{\sin^2(\frac{1}{2}(n-2)\pi h)}{\sin^2(\frac{1}{2}(n-1)\pi h)} = \frac{\sin^2(\frac{1}{2}\pi - \pi h)}{\sin^2(\frac{1}{2}\pi - \frac{1}{2}\pi h)} \\ &= \frac{\cos^2(\pi h)}{\cos^2(\frac{1}{2}\pi h)} = \frac{(1-\frac{1}{2}(\pi h)^2)^2}{(1-\frac{1}{2}(\frac{1}{2}\pi h)^2)^2} + \mathcal{O}(h^4) = 1 - \frac{3}{4}\pi^2 h^2 + \mathcal{O}(h^4)\end{aligned}$$

ist für $h \ll 1$ eine sehr langsame Konvergenz mit dem Konvergenzfaktor $\left|\frac{\lambda_2}{\lambda_1}\right|^2 = 1 - \frac{3}{2}\pi^2 h^2$ (quadratisch, da die Matrix symmerisch ist) zu erwarten. Dies wird bestätigt von den Ergebnissen in Tabelle 2.1 und Abbildung 2.2, die aus der Anwendung der Vektoriteration auf die Matrix A mit $h = \frac{1}{30}$ und dem Startvektor $x^{(0)} = y^{(0)}/\|y^{(0)}\|_2, y^{(0)} = (1, 2, \ldots, 29)^T$ resultieren.

k	$\lvert\rho^{(k)} - \lambda_1\rvert$	$\frac{\lvert\rho^{(k)}-\lambda_1\rvert}{\lvert\rho^{(k-1)}-\lambda_1\rvert}$
1	1.79e+3	0.5117
5	4.81e+2	0.8151
15	1.64e+2	0.9319
50	43.60	0.9758
100	17.01	0.9844
150	8.12	0.9857
200	3.90	0.9852

Tab. 2.1: Konvergenz der Vektoriteration angewandt auf Beispiel 2.0.1

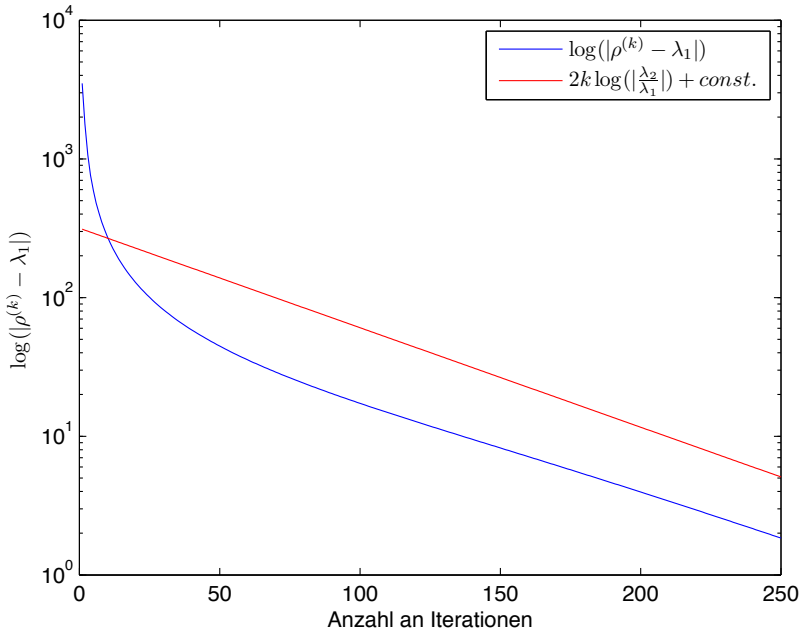


Abb. 2.2: Halblogarithmische Darstellung der Entwicklung des Fehlers

Beispiel 2.3.4 Es soll der betragsgrößte Eigenwert der Matrix

$$A = \begin{pmatrix} 7 & 4 & 9 & 3 \\ 2 & 2 & 4 & 9 \\ 3 & 1 & 0 & 4 \\ 2 & 7 & 5 & 1 \end{pmatrix}$$

bestimmt werden. A besitzt das Spektrum $\sigma(A) = \{15.2806, 4.4616, -2.7330, -7.0092\}$. Da A nicht symmetrisch ist, erwarten wir somit eine lineare Konvergenz der Potenzmethode mit Konvergenzfaktor

$$\left|\frac{\lambda_2}{\lambda_1}\right| = \frac{7.0092}{15.2806} \approx 0.4587\,.$$

Dieses theoretische Ergebnis wird von den konkreten Resultaten der Vektoriteration angewandt auf die Matrix A mit Startvektor $x^{(0)} = (0.5, 0.5, 0.5, 0.5)^T$ in Tabelle 2.2 bestätigt.

k	$\lvert\rho^{(k)} - \lambda_1\rvert$	$\frac{\lvert\rho^{(k)}-\lambda_1\rvert}{\lvert\rho^{(k-1)}-\lambda_1\rvert}$
0	0.4694	–
1	0.2625	0.5592
5	5.987e-4	0.1095
10	1.916e-5	0.5453
15	3.608e-7	0.4504
20	7.385e-9	0.4596
25	1.498e-10	0.4586
30	3.043e-12	0.4587

Tab. 2.2: Konvergenz der Vektoriteration angewandt auf Beispiel 2.3.4

2.4 Inverse Iteration nach Wielandt

Sei nun $A \in \mathbb{C}^{n\times n}$ regulär. Um den in vielen technischen Anwendungen gesuchten betragskleinsten Eigenwert λ_n

$$|\lambda_1| \geq |\lambda_2| \geq \ldots |\lambda_{n-1}| > |\lambda_n| > 0$$

zu finden, kann man die Tatsache ausnutzen, dass der betragskleinste Eigenwert von A der inverse betragsgrößte Eigenwert von A^{-1} ist, d.h. für die Eigenwerte ν_i von A^{-1} gilt

$$|\nu_n| > |\nu_{n-1}| \geq \ldots \geq |\nu_1|, \quad \text{wobei} \quad \nu_i = \frac{1}{\lambda_i}\,, i = 1, \ldots, n.$$

Die Anwendung der Potenzmethode auf A^{-1} liefert also eine Möglichkeit, $\nu_n = \frac{1}{\lambda_n}$, also auch den betragskleinsten Eigenwert von A zu bestimmen. In jeder Iteration ist dann der Vektor $a^{(k)} = A^{-1}x^{(k)}$ zu berechnen; dies entspricht dem Lösen des linearen Gleichungssystems $Aa^{(k)} = x^{(k)}$. Dies ist die sogenannte inverse Iteration nach Wielandt.

Mit Hilfe einer Verschiebung kann man auch die Berechnung der anderen Eigenwerte erreichen. Hierbei wird vorausgesetzt, dass man eine gute Näherung μ für einen Eigenwert λ_j von A kennt, sodass

$$|\mu - \lambda_j| < |\mu - \lambda_i| \qquad \forall\, i \neq j$$

gilt. Dann hat die Matrix $(A - \mu I)^{-1}$ den betragsgrößten Eigenwert $(\lambda_j - \mu)^{-1}$ und die gleichen Eigenvektoren wie A. Somit liefert die inverse Iteration angewandt auf $A - \mu I$ eine Approximation zu $\frac{1}{\lambda_j - \mu}$, woraus sich λ_j ergibt.

Algorithmus 2.4.1: Inverse Iteration nach Wielandt mit Spektralverschiebung

`Input:` $\mu \approx \lambda_j$, $x^{(0)}$ `mit` $v_j^T x^{(0)} \neq 0$ `und` $\|x^{(0)}\| = 1$
`for` $k = 0, 1, \ldots$

`Löse` $(A - \mu I)\, a^{(k)} = x^{(k)}$

$$\rho^{(k)} = (x^{(k)})^T a^{(k)} \qquad \% \text{ Rayleigh-Quotient}$$

$$x^{(k+1)} = \frac{a^{(k)}}{\|a^{(k)}\|}$$

`end`

Bemerkung 2.4.1 *Pro Iteration muss also ein lineares Gleichungssystem gelöst werden, dessen Koeffizientenmatrix aber konstant ist. Bestimmt man also einmal eine LR-Zerlegung von $A - \mu I$, so sind pro Iterationsschritt zwei Dreieckssysteme zu lösen, was einem Aufwand von $\mathcal{O}(n^2)$ entspricht.*

Mit Satz 2.3.1 können wir nun folgern, dass der Rayleigh-Quotient $\rho^{(k)}$

$$\rho^{(k)} \to \frac{1}{\lambda_j - \mu} \qquad \text{für } k \to \infty$$

erfüllt. Gemäß der Konvergenzanalyse der Potenzmethode in Abschnitt 2.3 ergibt sich die Konvergenzgeschwindigkeit aus dem Verhältnis zwischen $\frac{1}{\lambda_j - \mu}$ und dem betragsmäßig zweitgrößten Eigenwert von $(A - \mu I)^{-1}$, also durch den Faktor

$$\frac{\max\limits_{i \neq j} \frac{1}{|\lambda_i - \mu|}}{\frac{1}{|\lambda_j - \mu|}} = \frac{\frac{1}{\min\limits_{i \neq j} |\lambda_i - \mu|}}{\frac{1}{|\lambda_j - \mu|}} = \frac{|\lambda_j - \mu|}{\min\limits_{i \neq j} |\lambda_i - \mu|}$$

bestimmt wird.

Bemerkungen 2.4.2 *i) Ist μ eine gute Schätzung von λ_j, so gilt*

$$\frac{|\lambda_j - \mu|}{\min\limits_{i \neq j} |\lambda_i - \mu|} \ll 1$$

und das Verfahren konvergiert in diesem Fall sehr rasch.

ii) Die Kondition von $A - \mu I$ strebt für „immer besser" gewähltes μ gegen unendlich, die Matrix ist für $\mu \approx \lambda_j$ fast singulär. Daraus entstehen aber keine numerischen Schwierigkeiten, da nur die Richtung des Eigenvektors gesucht wird. Man ersetzt im Gauß-Verfahren ein auftretendes Pivotelement $\epsilon = 0$ durch die relative Maschinengenauigkeit eps.

Durch geeignete Wahl des Spektralverschiebungsparameters μ kann man also mit der inversen Vektoriteration 2.4.1 einzelne Eigenwerte und Eigenvektoren der Matrix A bestimmen. In der Praxis ist aber oft nicht klar, wie man für einen beliebigen Eigenwert λ_j diesen Parameter μ geeignet wählen kann.
Die Konvergenzgeschwindigkeit der Methode kann man noch erheblich verbessern, wenn man den Parameter μ nach jedem Schritt auf die aktuelle Annäherung $\lambda^{(k)} := \frac{1}{\rho^{(k)}} + \mu$ von λ_j setzt. Da die LR-Zerlegung dann aber in jedem Schritt neu berechnet werden muss, steigt damit der Rechenaufwand sehr stark an.

Beispiel 2.4.3 Wir betrachten die Matrix aus Beispiel 2.3.4 und wenden zur Berechnung des Eigenwerts $\lambda_3 = 4.4616$ dieser Matrix den Algorithmus 2.4.1 mit $\mu = 3.5$ und Startvektor $x^{(0)} = (0.5, 0.5, 0.5, 0.5)^T$ an. Die Resultate in Tabelle 2.3 bestätigen die obigen theoretischen Betrachtungen zur Konvergenz des Wielandt-Verfahrens, nach denen wir lineare Konvergenz mit einem Konvergenzfaktor in der Größenordnung

$$\frac{|\lambda_3 - 3.5|}{\min\limits_{i\neq 3} |\lambda_i - 3.5|} = \frac{|4.4616 - 3.5|}{|-2.7330 - 3.5|} = \frac{0.9616}{6.2330} \approx 0.1543\,.$$

erwarten.

k	$\lvert\lambda^{(k)} - \lambda_3\rvert$	$\frac{\lvert\lambda^{(k)}-\lambda_3\rvert}{\lvert\lambda^{(k-1)}-\lambda_3\rvert}$
0	0.8734	–
1	3.716e-3	0.0042
2	2.737e-3	0.7364
3	1.902e-4	0.0695
4	4.580e-5	0.2407
5	5.665e-6	0.1237
6	9.805e-7	0.1731
7	1.419e-7	0.1448
8	2.260e-8	0.1592
9	3.424e-9	0.1515
10	5.329e-10	0.1556

Tab. 2.3: Konvergenz des Wielandt-Verfahrens mit $\mu = 3.5$ angewandt auf Beispiel 2.3.4

Für die inverse Vektoriteration, wobei man den Parameter μ nach jedem Schritt auf die jeweils aktuelle Annäherung $\lambda^{(k)}$ vom λ_3 setzt,

$$\mu_0 = 3.5 \quad \mu_k = \lambda^{(k-1)} \quad \text{für } k \geq 1$$

sind einige Ergebnisse in Tabelle 2.4 dargestellt. Die Resultate zeigen, dass die Konvergenzgeschwindigkeit wesentlich schneller (genauer: quadratisch statt linear) ist.

k	$\lvert\lambda^{(k)} - \lambda_3\rvert$	$\lambda^{(k)}$
0	0.8735	5.3351
1	7.245e-3	4.4544
2	1.490e-5	4.4616
3	5.819e-12	4.4616

Tab. 2.4: Konvergenz des Wielandt-Verfahrens mit $\mu_k = \lambda^{(k-1)}$ angewandt auf Beispiel 2.3.4

Beispiel 2.4.4 Gegeben sei die Matrix

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

mit dem Spektrum $\sigma(A) = \{1, 2\}$. Gehen wir von einer Approximation $\mu = 1 - \epsilon$ von $\lambda_2 = 1$ mit $0 < |\epsilon| \ll 1$ aus, so ist die Matrix

$$(A - \mu I) = \begin{pmatrix} -2 + \epsilon & 3 \\ -2 & 3 + \epsilon \end{pmatrix}$$

fast singulär; ihre Inverse ist gegeben durch

$$(A-\mu I)^{-1} = \frac{1}{\epsilon^2+\epsilon}\begin{pmatrix} 3+\epsilon & -3 \\ 2 & -2+\epsilon \end{pmatrix}.$$

Da sich der Faktor $\frac{1}{\epsilon^2+\epsilon}$ bei der Normierung $x^{(k+1)} = \frac{a^{(k)}}{\|a^{(k)}\|}$ in der Wielandt-Iteration herauskürzt, ist die Berechnung der Richtung einer Lösung von $(A-\mu I)a^{(k)} = x^k$ gut konditioniert.

2.5 QR-VERFAHREN

Die in den vorherigen Abschnitten vorgestellten Methoden der Vektoriteration und der inversen Iteration nach Wielandt haben den schwerwiegenden Nachteil, dass man mit ihnen nur bestimmte Eigenwerte bestimmen kann. So liefert die Potenzmethode nur den betragsmäßig größten Eigenwert und bei der Wielandt-Iteration benötigt man zur Bestimmung eines Eigenwertes λ_i zunächst einen geeignet gewählten Parameter $\mu_i \approx \lambda_i$. In diesem Abschnitt werden wir uns nun einem effizienteren Verfahren zuwenden, mit dessen Hilfe man nicht nur einen, sondern *gleichzeitig alle* Eigenwerte einer Matrix $A \in \mathbb{R}^{n\times n}$ approximieren kann – dem QR-Verfahren.

2.5.1 Das Basisverfahren der QR-Iteration

Könnte man die Schur-Zerlegung $Q^H AQ = R$ aus Satz 2.1.6 einer Matrix $A \in \mathbb{R}^{n\times n}$ auf direkte Weise, also mit einer endlichen Anzahl an Operationen berechnen, so hätte man das Eigenwertproblem gelöst, da die Eigenwerte $\lambda_i(A)$ dann gerade durch die Diagonalelemente r_{ii} der oberen Dreiecksmatrix R gegeben wären. Leider ist die direkte Bestimmung der Matrix Q für $n \geq 5$ nicht möglich – dies folgt aus dem Abelschen Theorem (vgl. [Quarteroni et. al., Übung 8, Seite 258]). Daher kann das Eigenwertproblem nur durch den Übergang zu iterativen Methoden gelöst werden. Den Basisalgorithmus für das Verfahren, welches wir im Weiteren vorstellen und untersuchen möchten, liefert die *QR-Iteration*.

Algorithmus 2.5.1: QR-Basisiteration

`Input:` $A \in \mathbb{R}^{n\times n}$
`Setze` $A^{(0)} = A$`.`
`for` $k = 1, 2, \ldots$

$A^{(k-1)} = Q_k R_k$ `% Bestimme QR-Zerlegung`
$A^{(k)} = R_k Q_k$

`end`

Pro Iterationsschritt fällt also der Aufwand einer QR-Zerlegung $\mathcal{O}(n^3)$ und einer Matrixmultiplikation $\mathcal{O}(n^2)$ an. Insgesamt benötigt man also für dieses Verfahren $\mathcal{O}(k_{\max} n^3)$ Operationen, wobei $k_{\max}$ die Anzahl der Iterationen im Verfahren ist.
Hierbei ist zu beachten, dass für die Iterierten $A^{(k)}$ folgende Eigenschaften erfüllt sind.

Lemma 2.5.1 *Es sei $A \in \mathbb{R}^{n\times n}$, sowie $A^{(k)}$ für $k \in \mathbb{N}_0$ wie in Algorithmus 2.5.1 definiert. Dann gilt:*

i) Die Matrizen $A^{(k)}$ sind orthogonal ähnlich zu A.

ii) Ist A symmetrisch, so sind es auch die Matrizen $A^{(k)}$.

iii) Ist A symmetrisch und tridiagonal, so haben auch die Matrizen $A^{(k)}$ diese Gestalt.

Beweis. ad i) Sei $A \in \mathbb{R}^{n\times n}$. Mit den Bezeichnungen aus Algorithmus 2.5.1 gilt dann

$$\begin{aligned} A^{(k)} &= R_k Q_k = Q_k^T Q_k R_k Q_k = Q_k^T A^{(k-1)} Q_k = \ldots \\ &= Q_k^T Q_{k-1}^T \cdot \ldots \cdot Q_1^T A^{(0)} Q_1 \cdot \ldots \cdot Q_k = (Q_1 \cdot \ldots \cdot Q_k)^T A\,(Q_1 \cdot \ldots \cdot Q_k) \end{aligned}$$

ad ii) Sei $A \in \mathbb{R}^{n\times n}$ nun zusätzlich symmetrisch. Mit i) folgt, dass es für alle $k \in \mathbb{N}_0$ eine orthogonale Matrix $P_k \in \mathbb{R}^{n\times n}$ gibt, sodass

$$A^{(k)} = P_k^T A\, P_k$$

gilt. Daraus folgt

$$(A^{(k)})^T = \left(P_k^T A\, P_k\right)^T = P_k^T A^T P_k = P_k^T A\, P_k = A^{(k)}\,.$$

Für den Beweis von iii) sei auf [Deuflhard/Hohmann, Lemma 5.9] verwiesen. □

Für die QR-Basisiteration erhalten wir folgendes Konvergenzresultat, das Wilkinson[1] im Jahr 1965 in seiner Arbeit [Wilkinson65] beschrieben hat.

Satz 2.5.2 (Wilkinson) *Es sei $A \in \mathbb{R}^{n\times n}$ symmetrisch mit Eigenwerten*

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

und $A^{(k)}$, Q_k sowie R_k seien wie in Algorithmus 2.5.1 definiert. Dann gilt:

i) $\lim\limits_{k\to\infty} Q_k = I$,

ii) $\lim\limits_{k\to\infty} R_k = \operatorname{diag}(\lambda_1, \ldots, \lambda_n)$,

iii) $a_{ij}^{(k)} = \mathcal{O}\left(\left|\frac{\lambda_i}{\lambda_j}\right|^k\right)$ *für $i > j$ und $k \to \infty$, wobei $A^{(k)} = (a_{ij}^{(k)})_{i,j=1}^n$.*

Bemerkungen 2.5.3 *i) Sofern A keine symmetrische Matrix ist, aber die Eigenwerte von A immer noch getrennt sind, d.h.*

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

kann man zeigen, dass die Folge $A^{(k)}$ anstatt gegen eine Diagonal- gegen eine Dreiecksmatrix konvergiert:

$$\lim_{k\to\infty} A^{(k)} = \begin{pmatrix} \lambda_1 & * & * & \cdots & * \\ 0 & \lambda_2 & * & & \vdots \\ \vdots & 0 & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & * \\ 0 & \cdots & \cdots & 0 & \lambda_n \end{pmatrix}.$$

[1] James H. Wilkinson, * 27. September 1919 in Strood, Kent; † 5. Oktober 1986 in London, war ein britischer Mathematiker, der die numerische Mathematik vor allem durch Arbeiten zur Rückwärtsanalyse von Rundungsfehlern bereichert hat. 1970 wurde ihm der Turing-Preis verliehen.

ii) Haben wir nun eine Matrix $A \in \mathbb{R}^{n\times n}$ mit einem Paar konjugiert komplexer Eigenwerte, also $\lambda_{r+1} = \overline{\lambda}_r$ für ein $r \in \{1, \dots, n-1\}$ und somit

$$|\lambda_1| > |\lambda_2| > \ldots > |\lambda_r| = |\lambda_{r+1}| > \ldots > |\lambda_n|\,,$$

so konvergiert die Folge $A^{(k)}$ gegen eine Matrix der Form

$$\lim_{k\to\infty} A^{(k)} = \begin{pmatrix} \lambda_1 & * & \cdots & \cdots & \cdots & \cdots & \cdots & * \\ 0 & \ddots & \ddots & & & & & \vdots \\ \vdots & \ddots & \lambda_{r-1} & * & * & * & & \vdots \\ \vdots & & 0 & a_{rr}^{(k)} & a_{r,r+1}^{(k)} & * & & \vdots \\ \vdots & & 0 & a_{r+1,r}^{(k)} & a_{r+1,r+1}^{(k)} & * & & \vdots \\ \vdots & & 0 & 0 & 0 & \lambda_{r+2} & \ddots & * \\ \vdots & & & & & \ddots & \ddots & * \\ 0 & \cdots & \cdots & \cdots & \cdots & \cdots & 0 & \lambda_n \end{pmatrix},$$

wobei die Matrix $\begin{pmatrix} a_{rr}^{(k)} & a_{r,r+1}^{(k)} \\ a_{r+1,r} & a_{r+1,r+1} \end{pmatrix}$ für $k \to \infty$ im Allgemeinen divergiert, aber deren Eigenwerte konvergieren gegen λ_r und $\overline{\lambda}_r = \lambda_{r+1}$.

Beweis von Satz 2.5.2. Wir zeigen zunächst induktiv, dass für $k \in \mathbb{N}$ gilt:

$$A^k = \underbrace{Q_1 \ldots Q_k}_{=:P_k} \underbrace{R_1 \ldots R_k}_{=:U_k}\,.$$

Für $k = 1$ ist die Behauptung klar. Aus der Konstruktion der $A^{(k)}$ folgt wie im Beweis zu Lemma 2.5.1, dass

$$Q_{k+1}R_{k+1} = A^{(k)} = Q_k^T \ldots Q_1^T A Q_1 \ldots Q_k = P_k^T A P_k$$

gilt und somit auch der Induktionsschritt

$$A^{k+1} = AA^k \overset{IH}{=} AP_kU_k = P_kA^{(k)}U_k = P_kQ_{k+1}R_{k+1}U_k = P_{k+1}U_{k+1}\,.$$

Da $P_k \in \mathbb{R}^{n\times n}$ orthogonal ist und U_k eine obere Dreiecksmatrix, können wir die QR-Zerlegung $A^k = P_kU_k$ von A^k durch die QR-Zerlegung der $A^{(1)}, \ldots, A^{(k)}$ ausdrücken. Ferner ist A diagonalisierbar, da A symmetrisch ist, und daher folgt

$$A^k = Q^T\Lambda^k Q \qquad \text{mit } \Lambda = \operatorname{diag}(\lambda_1, \ldots, \lambda_n)\,.$$

Da man durch eine geeignete Permutation von A immer erreichen kann, dass Q eine LR-Zerlegung $Q = LR$ mit einer unipotenten unteren Dreiecksmatrix L und einer oberen Dreiecksmatrix R besitzt, nehmen dies wir im Folgenden o.B.d.A. an. Damit gilt

$$A^k = Q^T\Lambda^k Q = Q^T\Lambda^k LR = Q^T(\Lambda^k L\Lambda^{-k})(\Lambda^k R)\,.$$

Für die unipotente untere Dreiecksmatrix $(\Lambda^k L\Lambda^{-k})$ gilt

$$(\Lambda^k L\Lambda^{-k})_{ij} = l_{ij}\left(\frac{\lambda_i}{\lambda_j}\right)^k,$$

insbesondere verschwinden alle Nicht-Diagonalelemente für $k \to \infty$, d.h.

$$(\Lambda^k L\Lambda^{-k}) = I + E_k \quad \text{mit } E_k \to 0 \quad \text{für } k \to \infty\,.$$

Damit folgt mit einer QR-Zerlegung von $I + E_k = \widetilde{Q}_k \widetilde{R}_k$, dass

$$A^k = Q^T(I + E_k)\Lambda^k R = (Q^T \widetilde{Q}_k)(\widetilde{R}_k \Lambda^k R)\,, \tag{2.17}$$

wobei alle Diagonaleinträge von $\widetilde{R}_k$ positiv gewählt seien, was diese Zerlegung eindeutig macht. Aus $\lim\limits_{k\to\infty} E_k = 0$ folgt

$$\lim_{k\to\infty} \widetilde{Q}_k = I\,, \quad \lim_{k\to\infty} \widetilde{R}_k = I\,.$$

Wir haben in (2.17) eine weitere QR-Zerlegung von A^k gefunden, daher gilt bis auf Vorzeichen der Diagonale

$$P_k = Q^T \widetilde{Q}_k\,, \qquad U_k = \widetilde{R}_k \Lambda^k R\,.$$

Für den Grenzübergang $k \to \infty$ folgt dann schließlich

$$\begin{aligned} Q_k &= P_{k-1}^T P_k = \widetilde{Q}_{k-1}^T Q\, Q^T \widetilde{Q}_k = \widetilde{Q}_{k-1}^T \widetilde{Q}_k \to I \\ R_k &= U_k U_{k-1}^{-1} = \widetilde{R}_k \Lambda^k R\, R^{-1} \Lambda^{-(k-1)} \widetilde{R}_{k-1}^{-1} = \widetilde{R}_k \Lambda \widetilde{R}_{k-1}^{-1} \to \Lambda \end{aligned}$$

und

$$\lim_{k\to\infty} A^{(k)} = \lim_{k\to\infty} Q_k R_k = I\,\Lambda = \Lambda\,.$$

□

Der QR-Algorithmus 2.5.1 in seiner Grundform ist sehr aufwendig. Pro Iterationsschritt $A^{(k)} \to A^{(k+1)}$ benötigt man bei vollbesetzten Matrizen $\mathcal{O}(n^3)$ Operationen. Zudem besitzt das Verfahren in dieser Form auch den Nachteil, den schon die inverse Iteration nach Wielandt aufwies: Sind einige Eigenwerte von A betragsmäßig nur schlecht getrennt, d.h $|\lambda_j/\lambda_k| \approx 1$ für $j \neq k$, so ist die Konvergenz des Verfahrens nur sehr langsam, vgl. Satz 2.5.2 iii).
Im Folgenden werden für beide Probleme Lösungsansätze vorgestellt. Eine Möglichkeit, den Aufwand des Verfahrens zu verringern, besteht darin, die Matrix A auf *einfachere Gestalt* zu transformieren und auf diese transformierte Matrix die QR-Iteration mit einem geringeren Aufwand anzuwenden. Sogenannte *Shift-Techniken* oder *Verschiebungen* sorgen schließlich für eine Verbesserung des Konvergenzverhaltens des Verfahrens.

2.5.2 Transformation auf Hessenbergform

Um den Aufwand des Verfahrens zu reduzieren, wendet man den QR-Algorithmus 2.5.1 auf (orthogonal-) ähnliche Matrizen an, bei denen die Aufwandsabschätzung der QR-Zerlegung von $\mathcal{O}(n^3)$ auf $\mathcal{O}(n^2)$ Operationen verbessert werden kann. Wir betrachten dafür zunächst folgende Klasse von Matrizen, mit denen wir im Folgenden arbeiten werden.

Definition 2.5.4 (Hessenbergmatrix) *Eine Matrix $H \in \mathbb{K}^{n\times n}$ heißt obere Hessenbergmatrix, falls die Einträge unterhalb der ersten Nebendiagonalen verschwinden, d.h. $h_{ij} = 0$ für alle $i > j+1$. Obere Hessenbergmatrizen besitzen also die Gestalt*

$$H = \begin{pmatrix} & & & \\ & & & \\ & & & \\ 0 & & & \end{pmatrix}.$$

Eine obere Hessenbergmatrix H heißt unreduziert, falls $h_{j+1,j} \neq 0$ für alle $j = 1, \ldots, n-1$ gilt.

Bemerkung 2.5.5 *Ist $H \in \mathbb{R}^{n\times n}$ eine symmetrische Matrix in Hessenbergform, so muss H eine Tridiagonalmatrix sein.*

Eine gegebene Matrix $A \in \mathbb{R}^{n\times n}$ kann durch Ähnlichkeitstransformation mit dem Aufwand von $\mathcal{O}(n^3)$ Operationen auf obere Hessenbergform transformiert werden. Dazu sind $n-2$ Schritte erforderlich, in denen auf die Matrix A jeweils eine Ähnlichkeitstransformation mit einer Householdermatrix ausgeführt wird. Bezeichnen Q_j für $j = 1, \ldots, n-2$ diese Householdermatrizen, so kann die gesamte Ähnlichkeitstransformation Q als Produkt $Q = Q_1 \cdots Q_{n-2}$ aufgefasst werden.

Wie in [Numerik I, Kapitel 5] beschrieben kann zu dem Vektor $(a_{21}, \ldots, a_{n1})^T$ eine Householdermatrix $\widetilde{Q}_1 \in \mathbb{R}^{(n-1)\times(n-1)}$ gefunden werden, sodass

$$\widetilde{Q}_1^T \begin{pmatrix} a_{21} \\ \vdots \\ a_{n1} \end{pmatrix} = \begin{pmatrix} \boxplus \\ 0 \\ \vdots \\ 0 \end{pmatrix}.$$

Sei nun

$$Q_1 := \left(\begin{array}{c|c} 1 & 0 \\ \hline 0 & \widetilde{Q}_1 \end{array}\right),$$

dann gilt

$$Q_1^T A Q_1 = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ \boxplus & * & \cdots & * \\ 0 & * & \cdots & * \\ \vdots & \vdots & & \vdots \\ 0 & * & \cdots & * \end{pmatrix} \left(\begin{array}{c|c} 1 & 0 \\ \hline 0 & \widetilde{Q}_1 \end{array}\right) = \begin{pmatrix} a_{11} & * & \cdots & * \\ \boxplus & * & \cdots & * \\ 0 & * & \cdots & * \\ \vdots & \vdots & & \vdots \\ 0 & * & \cdots & * \end{pmatrix}.$$

Wählt man nun für $j = 2, \ldots, n-2$ die Matrizen $Q_j \in \mathbb{R}^{n\times n}$ analog zu Q_1 folgendermaßen

$$Q_j = \left(\begin{array}{c|c} I_j & 0 \\ \hline 0 & \widetilde{Q}_j \end{array}\right),$$

wobei $I_j \in \mathbb{R}^{j\times j}$ die j-dimensionale Einheitsmatrix bezeichnet und $\widetilde{Q}_j^T \in \mathbb{R}^{(n-j)\times(n-j)}$ eine Householdermatrix zu den unteren $n-j$ Einträgen der j-ten Spalte der Matrix

$$Q_{j-1}^T \ldots Q_1^T A\, Q_1 \ldots Q_{j-1}$$

ist, so entsteht durch sukzessive Ähnlichkeitstransformationen der Matrix A mit den Matrizen Q_j, $j = 1, \ldots, n-2$ eine Matrix in oberer Hessenbergform

$$Q_{n-2}^T \ldots Q_1^T A\, Q_1 \ldots Q_{n-2} = H = \begin{pmatrix} & & & \\ & & & \\ & & & \\ 0 & & & \end{pmatrix}.$$

Beispiel 2.5.6 (Reduktion auf Hessenbergform) Exemplarisch sei hier das oben beschriebene Schema für eine Matrix $A \in \mathbb{R}^{5\times 5}$ aufgeführt:

$$A = \begin{pmatrix} + & + & + & + & + \\ + & + & + & + & + \\ + & + & + & + & + \\ + & + & + & + & + \\ + & + & + & + & + \end{pmatrix} \xrightarrow{Q_1^T\cdot} \begin{pmatrix} + & + & + & + & + \\ * & * & * & * & * \\ & * & * & * & * \\ & * & * & * & * \\ & * & * & * & * \end{pmatrix} \xrightarrow{\cdot Q_1} \begin{pmatrix} + & * & * & * & * \\ + & * & * & * & * \\ & * & * & * & * \\ & * & * & * & * \\ & * & * & * & * \end{pmatrix}$$

$$\xrightarrow{Q_2^T\cdot} \begin{pmatrix} + & + & + & + & + \\ + & + & + & + & + \\ & * & * & * & * \\ & & * & * & * \\ & & * & * & * \end{pmatrix} \xrightarrow{\cdot Q_2} \begin{pmatrix} + & + & * & * & * \\ + & + & * & * & * \\ & + & * & * & * \\ & & * & * & * \\ & & * & * & * \end{pmatrix} \xrightarrow{Q_3^T\cdot} \begin{pmatrix} + & + & + & + & + \\ + & + & + & + & + \\ & + & + & + & + \\ & & * & * & * \\ & & & * & * \end{pmatrix} \xrightarrow{\cdot Q_3} \begin{pmatrix} + & + & + & * & * \\ + & + & + & * & * \\ & + & + & * & * \\ & & + & * & * \\ & & & * & * \end{pmatrix} = H.$$

Bemerkungen 2.5.7 *i) Der Aufwand des oben beschriebenen Verfahrens, eine Matrix $A \in \mathbb{R}^{n\times n}$ in obere Hessenbergform zu überführen, beträgt ungefähr $\frac{5}{3}n^3$ Operationen. Hierbei wird angenommen, dass die Householder-Matrizen Q_j nicht explizit berechnet werden, sondern nur implizit über den jeweiligen Householdervektor gegeben sind.*

ii) Eine symmetrische Matrix $A \in \mathbb{R}^{n\times n}$ wird mit obigem Algorithmus zu einer symmetrischen Tridiagonalmatrix transformiert, denn die Symmetrie bleibt erhalten.

Wir schließen diesen Abschnitt mit einer Matlab-Realisierung der Transformation einer beliebigen reellwertigen quadratischen Matrix auf obere Hessenbergform.

MATLAB-Funktion: HessRed.m

```
function [A] = HessRed(A)
% Reduction of the square matrix A to the upper
% Hessenberg form using Householder reflectors.
% Matrix A is overwritten with its upper Hessenberg form.
n = size(A,1);
for k=1:n-2
  [v,beta] = HouseholderVector(A(k+1:n,k));
  A(k+1:n,k:n) = A(k+1:n,k:n) - beta*v*(v'*A(k+1:n,k:n));
  A(:,k+1:n) = A(:,k+1:n) - beta*(A(:,k+1:n)*v)*v';
  Q = eye(numel(k+1:n)) - beta*(v*v');
end
```

MATLAB-Funktion: HouseholderVector.m

```
function [v,beta] = HouseholderVector(x)
n = length(x);
if n>1
  sigma = x(2:end)'*x(2:end);
  if sigma==0
    beta = 0;
  else
    mu = sqrt(x(1)^2+sigma);
    if x(1)<=0
      tmp = x(1) - mu;
    else
      tmp = -sigma / (x(1) + mu);
    end
    beta = 2*tmp^2/(sigma + tmp^2);
    x(2:end) = x(2:end)/tmp;
  end
  v = [1;x(2:end)];
else
  beta = 0;
  v = 1;
end
```

2.5.3 Das QR-Verfahren für Matrizen in Hessenbergform

Nachdem im letzten Abschnitt eine Möglichkeit zur Reduktion einer Matrix $A \in \mathbb{R}^{n\times n}$ in obere Hessenbergform gefunden worden ist, wird nun eine Variante der QR-Zerlegung für obere Hessenbergmatrizen vorgestellt, die nur einen Aufwand von $\mathcal{O}(n^2)$ Operationen benötigt.
Um die Eigenwerte der Matrix A zu bestimmen, kann man dann zunächst die Transformation auf Hessenbergform vornehmen und auf diese Matrix die hier beschriebene QR-Zerlegung ausführen. Der Rechenaufwand wird mit diesem Vorgehen von $\mathcal{O}(n^3)$ Operationen pro Iterationsschritt im Basisverfahren der QR-Iteration 2.5.1 auf **einmalig** $\mathcal{O}(n^3)$ für die Matrixreduktion sowie $\mathcal{O}(n^2)$ Operationen in jedem Iterationsschritt verringert.
Eine QR-Zerlegung einer Hessenbergmatrix $H \in \mathbb{R}^{n\times n}$ lässt sich mittels $n-1$ Givens-Rotationen effektiv berechnen. Bezeichnet $G_{i,j}$ für $i,j \in \{1,\dots,n\}$ diejenige Givens-Matrix, die bei Multiplikation von $G_{i,j}^T$ mit H von links bewirkt, dass der Eintrag h_{ij} verschwindet, so ist

$$G_{n,n-1}^T \dots G_{2,1}^T H = R$$

eine obere Dreiecksmatrix. Beispielsweise erhalten wir für $n = 5$:

$$H = \begin{pmatrix} + & + & + & + & + \\ + & + & + & + & + \\ & + & + & + & + \\ & & + & + & + \\ & & & + & + \end{pmatrix} \xrightarrow{G_{2,1}^T\cdot} \begin{pmatrix} * & * & * & * & * \\ & * & * & * & * \\ & + & + & + & + \\ & & + & + & + \\ & & & + & + \end{pmatrix} \xrightarrow{G_{3,2}^T\cdot} \begin{pmatrix} + & + & + & + & + \\ & * & * & * & * \\ & & * & * & * \\ & & + & + & + \\ & & & + & + \end{pmatrix}$$

$$\xrightarrow{G_{4,3}^T\cdot} \begin{pmatrix} + & + & + & + & + \\ & + & + & + & + \\ & & * & * & * \\ & & & * & * \\ & & & + & + \end{pmatrix} \xrightarrow{G_{5,4}^T\cdot} \begin{pmatrix} + & + & + & + & + \\ & + & + & + & + \\ & & + & + & + \\ & & & * & * \\ & & & & * \end{pmatrix} = R.$$

Eine QR-Zerlegung von H ist somit durch

$$H = QR \quad \text{mit} \quad Q = G_{2,1} \dots G_{n,n-1}$$

gegeben. Betrachtet man nun die Matrix RQ, so ist aufgrund der Struktur der Matrizen $G_{i+1,i}$, $i = 1,\dots,n-1$, klar, dass diese Matrix wieder obere Hessenbergform hat.
Für den oben aufgeführten Fall $n = 5$ sieht dies wie folgt aus:

$$R = \begin{pmatrix} + & + & + & + & + \\ & + & + & + & + \\ & & + & + & + \\ & & & + & + \\ & & & & + \end{pmatrix} \xrightarrow{\cdot G_{2,1}} \begin{pmatrix} * & * & + & + & + \\ * & * & + & + & + \\ & & + & + & + \\ & & & + & + \\ & & & & + \end{pmatrix} \xrightarrow{\cdot G_{3,2}} \begin{pmatrix} + & * & * & + & + \\ + & * & * & + & + \\ & & * & * & + \\ & & & + & + \\ & & & & + \end{pmatrix}$$

$$\xrightarrow{\cdot G_{4,3}} \begin{pmatrix} + & + & * & * & + \\ + & + & * & * & + \\ & + & * & * & + \\ & & * & * & + \\ & & & & + \end{pmatrix} \xrightarrow{\cdot G_{5,4}} \begin{pmatrix} + & + & + & * & * \\ + & + & + & * & * \\ & + & + & * & * \\ & & + & * & * \\ & & & * & * \end{pmatrix} = H^{(1)}.$$

Aufgrund der Eigenschaft, dass für die oben genannte QR-Zerlegung von H die Matrix RQ wiederum in oberer Hessenbergform ist, lässt sich die QR-Iteration 2.5.1 für Hessenbergmatrizen folgendermaßen formulieren.

Algorithmus 2.5.2: QR-Iteration für obere Hessenbergmatrizen

Input: $H \in \mathbb{R}^{n\times n}$ obere Hessenbergmatrix
Setze $H^{(0)} = H$.
for $k = 1, 2, \ldots$

$H^{(k-1)} = Q_k R_k$ % Bestimme QR-Zerlegung mit $n-1$ Givens-Rotationen
$H^{(k)} = R_k Q_k$ % wieder in Hessenbergform

end

Bemerkungen 2.5.8 *i) Man kann die Matrix $H^{(k)}$ aus $H^{(k-1)}$ auch direkt berechnen, indem man die einzelnen Matrixmultiplikationen gemäß der durch die folgende Klammerung angedeutete Reihenfolge durchführt*

$$H^{(k)} = \left(G_{n,n-1}^T \ldots \left(G_{3,2}^T \left((G_{2,1}^T H^{(k-1)}) G_{2,1}\right)\right) G_{3,2} \ldots\right) G_{n,n-1}. \tag{2.18}$$

Der Aufwand, auf diese Weise aus der Hessenbergmatrix $H^{(k-1)}$ die Hessenbergmatrix $H^{(k)}$ zu berechnen, liegt bei $\mathcal{O}(n^2)$ Operationen.

ii) Im symmetrischen Fall, d.h. $H^{(k-1)}$ ist eine symmetrische Tridiagonalmatrix, benötigt man für den QR-Schritt $H^{(k-1)} \to H^{(k)}$ sogar nur $\mathcal{O}(n)$ Operationen.

Aufgabe 2.5.9 Man zeige, dass die Berechnung der Matrix $H^{(k)}$ in (2.18) einen Aufwand von $\mathcal{O}(n^2)$ Operationen erfordert.

Eine Matlab-Realisierung der QR-Iteration für Matrizen in Hessenbergform ist nachfolgend dargestellt.

MATLAB-Funktion: qrMethod_hess.m

```
% QR Method for Hessenberg Matrices with Givens Rotations

function [H,Q,R] = qrMethod_hess(A,maxit)

n = max(size(A));
H = hess(A);
for iter = 1:maxit
  [Q,R,c,s] = qrGivens(H);
  H = R;
  for k = 1:n-1
    H = gaCol(H,c(k),s(k),1,k+1,k,k+1);
  end
end


```

```
% Determine QR-Factorization with Givens rotations
function [Q,R,c,s] = qrGivens(H)

n = size(H,2);

c = zeros(n-1,1);
s = zeros(n-1,1);
for k = 1:n-1
    [c(k),s(k)] = givCos(H(k,k),H(k+1,k));
    H = gaRow(H,c(k),s(k),k,k+1,k,n);
end
R = H;
Q = prodGiv(c,s,n);


% Calculate product of Givens-Rotation-Matrices
function Q = prodGiv(c,s,n)

n1 = n-1; n2 = n-2;
Q = eye(n);
%write last matrix
Q(n1,n1) =  c(n1);  Q(n, n)  =  c(n1);
Q(n1,n ) =  s(n1);  Q(n,n1)  = -s(n1);

for k = n2:-1:1
  Q(k  ,k) =  c(k);
  Q(k+1,k) = -s(k);
  q = Q(k+1, (k+1):n);
  Q(k  ,(k+1):n) = s(k)*q;
  Q(k+1,(k+1):n) = c(k)*q;
end

% Calculate c and s for one rotation
function [c,s] = givCos(xi,xk)

if xk==0
  c = 1; s=0;
else
  if abs(xk)>abs(xi)
    t = -xi/xk;
    s = 1/sqrt(1+t^2);
    c = s*t;
  else
    t = -xk/xi;
    c = 1/sqrt(1+t^2);
    s = c*t;
  end
end

%Calculate Q'*A whereQ is one rotation
function M = gaRow(M,c,s,i,k,j1,j2)

for j = j1:j2
  t1 = M(i,j);
  t2 = M(k,j);
  M(i,j) = c*t1 - s*t2;
  M(k,j) = s*t1 + c*t2;
end
```

```
74 %Calculate A*Q where Q is one rotation
75 function M = gaCol(M,c,s,j1,j2,i,k)
76
77 for j = j1:j2
78   t1 = M(j,i);
79   t2 = M(j,k);
80   M(j,i) = c*t1 - s*t2;
81   M(j,k) = s*t1 + c*t2;
82 end
```

MATLAB-Beispiel: QR-Iteration für Hessenbergmatrix

Man bestimme die Eigenwerte der Hessenbergmatrix

$$H = \begin{pmatrix} 5 & 2 & 1 \\ 2 & 3 & 1 \\ 0 & 8 & 5 \end{pmatrix}$$

mittels QR-Iteration.

```
>> qrMethod_hess(H,5)
ans =
    8.3275    2.0109    5.5293
   -0.2073    3.6750   -3.8067
         0    0.0019    0.9974
>> qrMethod_hess(H,10)
ans =
    8.2380    2.2182    5.6918
   -0.0040    3.7620   -3.5588
         0    0.0000    1.0000
>> qrMethod_hess(H,15)
ans =
    8.2361    2.2221    5.6949
   -0.0001    3.7639   -3.5538
         0    0.0000    1.0000
```

2.5.4 Das QR-Verfahren mit einfachem expliziten Shift

Analog zur Potenzmethode ist die Konvergenz der QR-Iteration abhängig von der Lage der Eigenwerte. Sind diese betragsmäßig schlecht voneinander getrennt, kann die Konvergenzgeschwindigkeit des Verfahrens sehr langsam sein, vgl. Satz 2.5.2. Wie bei der inversen Iteration nach Wielandt kann man mit Hilfe von *Shiftparametern* versuchen, das Spektrum der Matrix, deren Eigenwerte berechnet werden sollen, so zu verändern, dass die Konvergenz beschleunigt wird. Dies ist die Idee des *QR-Verfahrens mit explizitem Shift*.
Wir setzen im Folgenden stets voraus, dass eine Matrix $H \in \mathbb{R}^{n\times n}$ in oberer Hessenbergform vorliegt. Andernfalls kann die Matrix wie in Abschnitt 2.5.2 beschrieben auf diese Gestalt gebracht werden. Außerdem können wir o.B.d.A. annehmen, dass die Hessenbergmatrix H unreduziert ist. Ist sie dies nicht, so hat H die Form

$$H = \begin{pmatrix} H_{11} & H_{12} \\ 0 & H_{22} \end{pmatrix} \begin{matrix} p \\ n-p \end{matrix}$$
$$\begin{matrix} p & n-p \end{matrix}$$

wobei $1 \leq p < n$ und das Problem *zerfällt* in zwei kleinere Probleme, nämlich in die Bestimmung der Eigenwerte von H_{11} und H_{22}. Im Fall $p = n-1$ oder $p = n-2$ wird häufig auch von *Deflation* gesprochen.
Gegeben ein Shiftparameter $\mu_k \in \mathbb{R}$, wendet man bei QR-Verfahren mit einfachem expliziten

Shift im k-ten Schritt die QR-Zerlegung nicht auf $H^{(k-1)}$ an, sondern auf

$$\widetilde{H}^{(k-1)} = H^{(k-1)} - \mu_k I,$$

wobei μ_k so gewählt wird, dass

$$|\lambda_1 - \mu_k| \geq |\lambda_2 - \mu_k| \geq \ldots \geq |\lambda_{n-1} - \mu_k| \gg |\lambda_n - \mu_k|$$

nach eventueller Umnummerierung der Eigenwerte $\lambda_1, \ldots, \lambda_n$ von $H^{(k-1)}$ gilt. Es ergibt sich so der folgende Algorithmus.

Algorithmus 2.5.3: QR-Iteration mit einfachem expliziten Shift

`Input:` $H \in \mathbb{R}^{n\times n}$ `obere Hessenbergmatrix,`
$(\mu_k)_{k\in\mathbb{N}}$ `Folge von Shiftparametern`

`Setze` $H^{(0)} = H$`.`
`for` $k = 1, 2, \ldots$

$$H^{(k-1)} - \mu_k I = Q_k R_k \qquad \texttt{\% Bestimme QR-Zerlegung} \tag{2.19}$$
$$H^{(k)} = R_k Q_k + \mu_k I$$

`end`

Lemma 2.5.10 *Für die Folge $(H^{(k)})_{k\in\mathbb{N}}$ aus Algorithmus 2.5.3 gilt:*

i) *$H^{(k)}$ ist orthogonal ähnlich zu H für alle $k \in \mathbb{N}$, genauer gilt*

$$H^{(k)} = Q_k^T H^{(k-1)} Q_k = P_k^T H P_k,$$

mit $P_k := Q_1 Q_2 \ldots Q_k$.

ii) *Für $k \in \mathbb{N}$ gilt außerdem*

$$(H - \mu_k I) \cdot \ldots \cdot (H - \mu_1 I) = Q_1 \cdot \ldots \cdot Q_k R_k \cdot \ldots \cdot R_1.$$

iii) *Falls darüberhinaus R_k nichtsingulär ist, gilt auch*

$$H^{(k)} = R_k H^{(k-1)} R_k^{-1} = U_k H U_k^{-1},$$

mit $U_k := R_k R_{k-1} \ldots R_1$.

Beweis. Die Aussagen ergeben sich sofort aus Algorithmus 2.5.3 und seien daher dem Leser als Übung überlassen. □

Aufgabe 2.5.11 Man beweise Lemma 2.5.10.

Wir wissen bereits aus dem vorherigen Abschnitt, dass $H^{(k)}$ wiederum eine Matrix in oberer Hessenbergform ist, sofern $H^{(k-1)}$ diese Form besitzt. Man kann diese Aussage für die QR-Iteration mit explizitem Shift aber noch erweitern.

Satz 2.5.12 *Sei $H^{(k-1)} \in \mathbb{R}^{n\times n}$ eine unreduzierte obere Hessenbergmatrix. $H^{(k)}$ ist genau dann wieder eine unreduzierte Hessenbergmatrix, wenn der Shiftparameter μ_k kein Eigenwert von $H^{(k-1)}$ ist. Andernfalls besitzt $H^{(k)}$ die Form*

$$H^{(k)} = \begin{pmatrix} \widetilde{H}^{(k)} & * \\ 0 & \mu_k \end{pmatrix},$$

wobei $\widetilde{H}^{(k)} \in \mathbb{R}^{(n-1)\times(n-1)}$ eine unreduzierte Hessenbergmatrix ist.

Beweis. Aus (2.19) folgt

$$R_k = Q_k^T(H^{(k-1)} - \mu_k I). \tag{2.20}$$

Falls μ_k kein Eigenwert von $H^{(k-1)}$ ist, muss R_k regulär sein und es folgt aus

$$H^{(k)} = R_k H^{(k-1)} R_k^{-1},$$

für die Einträge von $H^{(k)}$, dass

$$h_{j+1,j}^{(k)} = e_{j+1}^T H^{(k)} e_j = e_{j+1}^T R_k H^{(k-1)} R_k^{-1} e_j = r_{j+1,j+1}^{(k)} h_{j+1,j}^{(k-1)} (r_{jj}^{(k)})^{-1} \neq 0.$$

Also zerfällt auch $H^{(k)}$ nicht.
Sei nun μ_k ein Eigenwert von $H^{(k-1)}$. Dann ist R_k singulär. Da $H^{(k-1)}$ eine unzerlegbare Hessenbergmatrix ist, sind die ersten $n-1$ Spalten von $H^{(k-1)} - \mu_k I$ linear unabhängig. Wegen (2.20) sind auch die ersten $n-1$ Spalten von R_k linear unabhängig. Aufgrund der Singularität von R_k muss die letzte Zeile von R_k verschwinden, also R_k die Gestalt

$$R_k = \begin{pmatrix} \widetilde{R}_k & * \\ 0 & 0 \end{pmatrix}$$

besitzen mit einer regulären oberen Dreiecksmatrix $\widetilde{R}_k \in \mathbb{R}^{(n-1)\times(n-1)}$. Wegen

$$H^{(k-1)} - \mu_k I = Q_k R_k = Q_k \begin{pmatrix} \widetilde{R}_k & * \\ 0 & 0 \end{pmatrix}$$

und aufgrund der Tatsache, dass $H^{(k-1)}$ unzerlegbar ist, ist auch Q_k eine unzerlegbare Hessenbergmatrix. Daher besitzt $H^{(k)}$ folgende Struktur

$$H^{(k)} = R_k Q_k + \mu_k I = \begin{pmatrix} \widetilde{R}_k & * \\ 0 & 0 \end{pmatrix} Q_k + \mu_k I = \begin{pmatrix} \widetilde{H}^{(k)} & * \\ 0 & \mu_k \end{pmatrix}$$

mit einer unzerlegbaren Hessenbergmatrix $\widetilde{H}^{(k)}$. □

Wenden wir uns nun der **Wahl der Shiftparameter** zu. Nach den obigen Überlegungen sollte man als Shiftparameter möglichst gute Näherungswerte für einen Eigenwert von H wählen. Gemäß Satz 2.5.2 gilt: Existiert nur ein betragsmäßig kleinster Eigenwert λ_n von H, so gilt unter bestimmten Voraussetzungen im QR-Verfahren $\lim\limits_{k\to\infty} h_{nn}^{(k)} = \lambda_n$. Das Element $h_{nn}^{(k)}$ wird also in diesem Fall für genügend großes k eine gute Näherung an λ_n sein. Es empfiehlt sich daher

$$\mu_k = h_{nn}^{(k-1)}$$

zu setzen, sobald die Konvergenz der Elemente $h_{nn}^{(k-1)}$ fortgeschritten ist. Ein möglicher Indikator dafür ist z.B., dass

$$\left| 1 - \frac{h_{nn}^{(k-2)}}{h_{nn}^{(k-1)}} \right| \leq \eta < 1$$

für ein kleines η gilt. Hierbei liefert bereits die Wahl $\eta = 1/3$ gute Ergebnisse.
Sofern die Einträge $h_{n,n-1}^{(k)}$ gegen Null konvergieren, geschieht dies mit quadratischer Konvergenzrate. Um dies zu sehen, betrachten wir ein Beispiel.

Beispiel 2.5.13 Nehmen wir an, H ist eine unreduzierte obere Hessenbergmatrix der Form

$$H = \begin{pmatrix} + & + & + & + & + \\ + & + & + & + & + \\ 0 & + & + & + & + \\ 0 & 0 & + & + & + \\ 0 & 0 & 0 & \epsilon & h_{nn} \end{pmatrix}$$

und wir führen einen Schritt der QR-Iteration mit einfachem expliziten Shift aus:

$$\begin{aligned} QR &= H - h_{nn}I \\ \bar{H} &= RQ + h_{nn}I\,. \end{aligned}$$

Nach $n-2$ Schritten in der Reduktion von $H - h_{nn}I$ auf obere Dreiecksform, erhält man eine Matrix mit der folgenden Struktur:

$$\widetilde{H} = \begin{pmatrix} + & + & + & + & + \\ 0 & + & + & + & + \\ 0 & 0 & + & + & + \\ 0 & 0 & 0 & a & b \\ 0 & 0 & 0 & \epsilon & 0 \end{pmatrix}.$$

Man kann zeigen, dass der $(n, n-1)$-te Eintrag von $\bar{H} = RQ + h_{nn}I$ durch

$$-\frac{\epsilon^2 b}{\epsilon^2 + a^2}$$

gegeben ist. Nimmt man an, dass $\epsilon \ll a$, so gilt $\bar{h}_{n,n-1} = \mathcal{O}(\epsilon^2)$, d.h. der Algorithmus konvergiert quadratisch.

2.5.5 Das QR-Verfahren mit doppeltem expliziten Shift

Leider ist der Algorithmus 2.5.3 in einigen Fällen nicht zielführend. Probleme können auftreten, wenn in einer Iteration die Eigenwerte a_1 und a_2 von

$$G = \begin{pmatrix} h_{mm} & h_{mn} \\ h_{nm} & h_{nn} \end{pmatrix} \qquad m = n-1$$

komplexwertig sind, da dann h_{nn} meistens eine schlechte Approximation für einen Eigenwert ist. Um dieses Problem zu umgehen, kann meine zwei einfache Shifts mit a_1 und a_2 als Shiftparameter ausführen.

$$\begin{aligned} H - a_1 I &= U_1 R_1 \\ H_1 &= R_1 U_1 + a_1 I \\ H_1 - a_2 I &= U_2 R_2 \\ H_2 &= R_2 U_2 + a_2 I \end{aligned}$$

Diese Vorgehensweise würde allerdings komplexwertige Arithmetik erfordern (a_1 und a_2 sind komplexwertig!) und ist daher noch nicht zufriedenstellend. Man kann aber analog zu Lemma 2.5.10 zeigen, dass

$$(U_1 U_2)(R_2 R_1) = (H - a_1 I)(H - a_2 I) =: M \tag{2.21}$$

gilt. Schaut man sich die Matrix M näher an

$$M = H^2 - (a_1 + a_2)H + a_1 a_2 I\,,$$

so stellt man fest, dass sie reellwertig ist, sofern H nur reelle Einträge besitzt, denn

$$a_1 + a_2 = h_{mm} + h_{nn} = \mathrm{trace}(G) \in \mathbb{R} \tag{2.22}$$
$$a_1 a_2 = h_{mm}h_{nn} - h_{mn}h_{nm} = \det(G) \in \mathbb{R}\,. \tag{2.23}$$

Daher ist (2.21) eine QR-Zerlegung einer reellwertigen Matrix und man kann die unitären Matrizen U_1 und U_2 so wählen, dass $Z := U_1U_2$ eine reellwertige orthogonale Matrix ist. Dann ist

$$H_2 = U_2^H H_1 U_2 = U_2^H (U_1^H H U_1) U_2 = (U_1U_2)^H H (U_1U_2) = Z^T H Z$$

ebenfalls reell.
Aufgrund von Rundungsfehlern ist es aber in der Regel nicht möglich von der komplexwertigen Arithmetik zur reellwertigen zurückzukehren. Um dies zu gewährleisten, könnte man

1. die reellwertige Matrix $M = H^2 - (a_1 + a_2)H + a_1a_2I$ explizit berechnen,
2. die reelle QR-Zerlegung $M = ZR$ berechnen und
3. $H_2 = Z^T H Z$ setzen.

Da allerdings der erste dieser Schritte $\mathcal{O}(n^3)$ Rechenoperationen erfordern würde, ist dieses Vorgehen zu aufwendig. Eine Lösung dieses Problems stellt das QR-Verfahren mit doppeltem impliziten Shift dar.

2.5.6 Das QR-Verfahren mit doppeltem impliziten Shift

Den Schlüssel zur Reduktion des Aufwandes von $\mathcal{O}(n^3)$ auf $\mathcal{O}(n^2)$ Rechenoperationen liefert der folgende Satz, das sog. *implizite Q-Theorem*.

Satz 2.5.14 (Implizites Q-Theorem, [Golub/Loan]) *$Q = [q_1, \ldots, q_n]$ und $V = [v_1, \ldots, v_n]$ seien orthogonale Matrizen mit der Eigenschaft, dass für eine Matrix $A \in \mathbb{R}^{n\times n}$ sowohl $Q^TAQ = H$ als auch $V^TAV = G$ in oberer Hessenbergform sind. Sei k der kleinste Index, für den $h_{k+1,k} = 0$ gilt, wobei wir definieren, dass $k = n$ gilt, wenn H unreduziert ist.*
Sofern $q_1 = v_1$ gilt, dann ist $q_i = \pm v_i$ für $i = 2, \ldots, k$ und $|h_{j+1,j}| = |g_{j+1,j}|$ für $j = 1, \ldots, k-1$. Ist $k < n$, so gilt $g_{k+1,k} = 0$.

Beweis. Wir definieren die orthogonale Matrix $W = [w_1, \ldots, w_n]$ via $W := V^TQ$ und bemerken, dass $GW = WH$ gilt. Daher erhalten wir für $i = 2, \ldots, k$

$$h_{i,i-1}w_i = Gw_{i-1} - \sum_{j=1}^{i-1} h_{j,i-1}w_j\,.$$

Die erste Spalte von W ist gleich dem ersten kanonischen Einheitsvektor, $w_1 = e_1$, und so folgt aus der letzten Gleichung, dass $[w_1, \ldots, w_k]$ obere Dreiecksform hat. Daher ist $w_i = \pm e_i$ für $i = 2, \ldots, k$. Da $w_i = V^Tq_i$ und $h_{i,i-1} = w_i^TGw_{i-1}$ gilt, erhalten wir schließlich, dass $v_i = \pm q_i$ und $|h_{i,i-1}| = |g_{i,i-1}|$ für $i = 2, \ldots, k$.
Ist $h_{k+1,k} = 0$, dann ergibt sich

$$\begin{aligned}|g_{k+1,k}| &= |e_{k+1}^TGe_k| = |e_{k+1}^TGWe_k| = |(e_{k+1}^TW)(He_k)| \\ &= |e_{k+1}^T\sum_{i=1}^{k} h_{ik}We_i| = |\sum_{i=1}^{k} h_{ik}e_{k+1}^Te_i| = 0.\end{aligned}$$

□

Bemerkung 2.5.15 *Man kann die Aussage des Satzes 2.5.14 auch auf den Fall unitärer Matrizen Q und V erweitern. Hierzu sei auf [Stoer, Satz 6.6.5.1] verwiesen.*

Wir sind nun in der Lage, den Übergang von H zu H_2 aus obigem Abschnitt 2.5.5 mit einem Aufwand von $\mathcal{O}(n^2)$ Rechenoperationen zu bewerkstelligen, indem wir folgendermaßen vorgehen:

1. berechne Me_1, also die erste Spalte von M,
2. bestimme eine Householdermatrix P_0, sodass $P_0(Me_1)$ ein Vielfaches von e_1 ist,
3. bestimme weitere Householdermatrizen $P_1, \ldots, P_{n-2}$, sodass für das Produkt $Z_1 = P_0 P_1 \ldots P_{n-2}$ die Matrix $Z_1^T H Z_1$ in oberer Hessenbergform ist und die erste Spalte von Z_1 mit derjenigen von Z übereinstimmt.

Unter diesen Umständen kann man mit dem impliziten Q-Theorem schließen, dass $Z^T H Z$ und $Z_1^T H Z_1$ bis auf Vorzeichen übereinstimmen, sofern sie beide unreduzierte Hessenbergmatrizen sind. Sind sie dies nicht, so zerfällt das Problem wie zu Beginn von Abschnitt 2.5.4 beschrieben in kleinere Probleme in unreduzierter Hessenberggestalt.
Die erste Spalte von M kann man leicht berechnen:

$$m_1 := Me_1 = (H - a_1 I)(H - a_2 I)e_1 = \begin{pmatrix} h_{11} - a_1 & h_{12} & \cdots & \cdots \\ h_{21} & h_{22} - a_1 & h_{23} & \cdots \\ & h_{32} & h_{33} - a_1 & \ddots \\ & & \ddots & \ddots \end{pmatrix} \begin{pmatrix} h_{11} - a_2 \\ h_{21} \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

$$= \begin{pmatrix} (h_{11} - a_1)(h_{11} - a_2) + h_{12} h_{21} \\ h_{21}(h_{11} - a_2) + h_{21}(h_{22} - a_1) \\ h_{32} h_{21} \\ 0 \\ \vdots \\ 0 \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

wobei nach Einsetzen von (2.22) und (2.23)

$$\begin{aligned} x &= h_{11}^2 + h_{12} h_{21} - h_{11}(h_{n-1,n-1} + h_{nn}) + h_{n-1,n-1} h_{nn} - h_{n-1,n} h_{n,n-1} \\ y &= h_{21}(h_{11} + h_{22} - h_{n-1,n-1} - h_{nn}) \\ z &= h_{21} h_{32} \,. \end{aligned}$$

Man bestimmt nun eine (reelle) Householdermatrix P_0, sodass der Vektor m_1 in ein Vielfaches des ersten kanonischen Basisvektors e_1 transformiert wird:

$$P_0 \, m_1 = P_0 \begin{pmatrix} x \\ y \\ z \\ 0 \\ \vdots \\ 0 \end{pmatrix} = r_{11} e_1 \,, \qquad r_{11} \in \mathbb{R}.$$

P_0 besitzt wegen

$$M = P_0^T P_0 M = P_0^2 M = P_0[r_{11} e_1, *, \ldots, *]$$

bis auf einen Faktor ± 1 die gleiche erste Spalte wie Z.
Da eine Ähnlichkeitstransformation mit P_0 nur eine Änderungen in den ersten drei Zeilen und

Spalten bewirkt, ergibt sich für die Matrix $P_0^T H P_0$ folgende Form (hier $n = 6$)

$$P_0^T H P_0 = \begin{pmatrix} * & * & * & * & * & * \\ * & * & * & * & * & * \\ * & * & * & * & * & * \\ & + & + & + & + & \\ & & + & + & + & \\ & & & + & + & \end{pmatrix} P_0 = \begin{pmatrix} * & * & * & + & + & + \\ * & * & * & + & + & + \\ * & * & * & + & + & + \\ * & * & * & + & + & + \\ & & & + & + & + \\ & & & & + & + \end{pmatrix}$$

Nun besteht die Aufgabe der Householdermatrizen $P_1, \ldots, P_{n-2}$ darin, die Hessenbergform der Matrix wiederherzustellen.

$$\begin{pmatrix} + & + & + & + & + & + \\ + & + & + & + & + & + \\ * & + & + & + & + & + \\ * & * & + & + & + & + \\ & & + & + & + & \\ & & & + & + & \end{pmatrix} \xrightarrow{P_1^T \cdot, \cdot P_1} \begin{pmatrix} + & + & + & + & + & + \\ + & + & + & + & + & + \\ & + & + & + & + & + \\ & * & + & + & + & + \\ & * & * & + & + & + \\ & & & & + & + \end{pmatrix} \xrightarrow{P_2^T \cdot, \cdot P_2} \begin{pmatrix} + & + & + & + & + & + \\ + & + & + & + & + & + \\ & + & + & + & + & + \\ & & + & + & + & + \\ & & * & + & + & + \\ & & * & * & + & + \end{pmatrix}$$

$$\xrightarrow{P_3^T \cdot, \cdot P_3} \begin{pmatrix} + & + & + & + & + & + \\ + & + & + & + & + & + \\ & + & + & + & + & + \\ & & + & + & + & + \\ & & & + & + & + \\ & & & * & + & + \end{pmatrix} \xrightarrow{P_4^T \cdot, \cdot P_4} \begin{pmatrix} + & + & + & + & + & + \\ + & + & + & + & + & + \\ & + & + & + & + & + \\ & & + & + & + & + \\ & & & + & + & + \\ & & & & + & + \end{pmatrix}$$

Hierbei haben die Householdermatrizen die Gestalt $P_k = \operatorname{diag}(I_k, \bar{P}_k, I_{n-k-3})$ für $k = 1, \ldots, n-3$, wobei $\bar{P}_k$ eine 3×3-Householdermatrix ist. Für $k = n-2$ ist $P_{n-2} = \operatorname{diag}(I_{n-2}, \bar{P}_{n-2})$ mit einer 2×2-Householdermatrix $\bar{P}_{n-2}$.

$$H \to P_0^T H P \to P_1^T P_0^T H P_0 P_1 \to \ldots \to P_{n-2}^T \cdots P_1^T P_0^T H P_0 P_1 \cdots P_{n-2} =: K$$

K ist wiederum in Hessenbergform. Da die erste Spalte von P_0 und von $P_0 \cdot \ldots \cdot P_{n-2}$ übereinstimmen, d.h.

$$P_0 \cdot \ldots \cdot P_{n-2} e_1 = P_0 e_1 \,,$$

und da P_0 und Z die gleiche erste Spalte haben, gilt nach dem impliziten Q-Theorem bis auf Vorzeichen $H_2 = K$.
Damit haben wir also eine Möglichkeit gefunden, H_2 aus H mit einem Aufwand von insgesamt $\mathcal{O}(n^2)$ Rechenoperationen zu erhalten ($n-1$ Householdertransformationen, die jeweils einen Aufwand von $\mathcal{O}(n)$ erfordern).

Diese implizite Bestimmung von H_2 aus H wurde zuerst von Francis im Jahr 1961 erwähnt und wird daher auch oft *Francis-QR-Step* genannt.

Algorithmus 2.5.4: Francis-QR-Step

`Gegeben sei eine unreduzierte obere Hessenbergmatrix` $H \in \mathbb{R}^{n \times n}$`,`
`deren Untermatrix` $\begin{pmatrix} h_{n-1,n-1} & h_{n-1,n} \\ h_{n,n-1} & h_{nn} \end{pmatrix}$ `die Eigenwerte` a_1 `und` a_2 `besitzt.`
`Dann überschreibt dieser Algorithmus` H `mit` $Z^T H Z$`, wobei`
$Z = P_1 \cdots P_{n-2}$ `ein Produkt von Householdermatrizen und`
$Z^T(H - a_1 I)(H - a_2 I)$ `eine obere Dreiecksmatrix ist.`

$m = n - 1$
% `Berechne die erste Spalte von`$(H - a_1 I)(H - a_2 I)$
$s = H(m,m) + H(n,n)$
$t = H(m,m)H(n,n) - H(m,n)H(n,m)$

$x = H(1,1)H(1,1) + H(1,2)H(2,1) - sH(1,1) + t$
$y = H(2,1)(H(1,1) + H(2,2) - s)$
$z = H(2,1)H(3,2)$
for $k = 0 : n-3$
  $P = \mathtt{house}([x\, y\, z]^T)$
  $q = \max\{1, k\}$
  $H(k+1:k+3, q:n) = P\, H(k+1:k+3, q:n)$
  $r = \min\{k+4, n\}$
  $H(1:r, k+1:k+3) = H(1:r, k+1:k+3)P$
  $x = H(k+2, k+1)$
  if $k < n-3$
    $z = H(k+4, k+1)$
  end
end
$P = \mathtt{house}([x\, y]^T)$
$H(n-1:n, n-2:n) = P\, H(n-1:n, n-2:n)$
$H(1:n, n-1:n) = H(1:n, n-1:n)P$

Bemerkung 2.5.16 *Der Algorithmus 2.5.4 benötigt $10n^2$ Rechenoperationen. Soll die orthogonale Matrix Z zusätzlich konkret berechnet und abgespeichert werden, so sind hierfür weitere $10n^2$ Rechenoperationen notwendig.*

2.5.7 Das gesamte QR-Verfahren

Für eine nichtsymmetrische Matrix $A \in \mathbb{R}^{n\times n}$ besteht die Standardmethode zur Lösung des Eigenwertproblems darin, A wie in 2.5.2 beschrieben auf obere Hessenbergform zu transformieren und dann den Algorithmus 2.5.4 iterativ anzuwenden, um die reelle Schurform der Matrix A zu bestimmen. Während der Iteration müssen die Elemente unterhalb der Diagonalen beobachtet werden, um eine mögliche Deflation auszumachen.
Der gesamte Prozess zur Bestimmung der Eigenwerte von A ist in dem folgenden Algorithmus umrissen.

Bemerkung 2.5.17 *Der folgende Algorithmus benötigt $25n^3$ Rechenoperationen, wenn Q und T berechnet werden. Sollen nur die Eigenwerte berechnet werden, dann sind $10n^3$ Rechenoperationen notwendig.*

Algorithmus 2.5.5: Gesamter QR-Algorithmus

Für eine Matrix $A \in \mathbb{R}^{n\times n}$ und einen Toleranzwert $\epsilon > 0$ berechnet dieser Algorithmus die reelle Schurform $Q^T AQ = T$. A wird mit der Hessenbergmatrix H überschrieben. Wenn Q und T bestimmt werden sollen, dann wird T in H gespeichert. Werden nur die Eigenwerte benötigt, dann werden die Diagonalblöcke von T in den entsprechenden Positionen von H abgespeichert.

1. Reduziere A mittels $n-2$ Householder-Transformationen auf obere Hessenbergform H;

2. **while** $q < n$
   Setze alle Elemente in der ersten unteren Nebendiagonalen $h_{i+1,i} = 0$, die

$$|h_{i+1,i}| \le \epsilon\,(|h_{ii}| + |h_{i+1,i+1}|) \quad i = 1, \ldots, n-1$$

   erfüllen.
   Finde größtes $q \ge 0$ und kleinstes $p \ge 0$, sodass

$$H = \begin{pmatrix} H_{11} & H_{12} & H_{13} \\ & H_{22} & H_{23} \\ & & H_{33} \end{pmatrix} \begin{matrix} p \\ n-p-q \\ q \end{matrix}$$
$$\begin{matrix} p & n-p-q & q \end{matrix}$$

   wobei

   - H_{11}: obere Hessenbergmatrix
   - H_{22}: unreduzierte obere Hessenbergmatrix und
   - H_{33}: obere Quasi-Dreiecksmatrix (d.h. eine Dreicksmatrix, die auch 2×2-Blöcke auf der Diagonalen enthalten darf)

   sind.
   if $q < n$
   Wende einen Francis QR-Schritt auf H_{22} an:

$$H_{22} = Z^T H_{22} Z$$

   if Q soll bestimmt werden

$$\begin{aligned} Q &= Q\,\mathrm{diag}(I_p, Z, I_q) \\ H_{12} &= H_{12} Z \\ H_{23} &= Z^T H_{23} \end{aligned}$$

   end
  end
 end

2.6 VERFAHREN FÜR SYMMETRISCHE MATRIZEN

In diesem Abschnitt beschäftigen wir uns mit der Eigenwertbestimmung von symmetrischen Matrizen $A \in \mathbb{R}^{n\times n}$. Hierbei wird zunächst der Fall symmetrischer Tridiagonalmatrizen untersucht, da mit dem Algorithmus aus Abschnitt 2.5.2 jede symmetrische Matrix auf diese Gestalt transformiert werden kann.
Anschließend wird das Lanczos-Verfahren zur Bestimmung der extremalen Eigenwerte symmetrischer Matrizen vorgestellt, das sich besonders für schwachbesetzte Matrizen eignet.

2.6.1 Methoden für symmetrische Tridiagonalmatrizen

Im Folgenden werden wir spezielle Verfahren für symmetrische Tridiagonalmatrizen vorstellen. Dies sind Matrizen $T \in \mathbb{R}^{n\times n}$ der Gestalt

$$T = \begin{pmatrix} \alpha_1 & \beta_1 & & \cdots & 0 \\ \beta_1 & \alpha_2 & \ddots & & \vdots \\ & \ddots & \ddots & \ddots & \\ \vdots & & \ddots & \ddots & \beta_{n-1} \\ 0 & \cdots & & \beta_{n-1} & \alpha_n \end{pmatrix} = T^T$$

mit $\alpha_i, \beta_j \in \mathbb{R}$ für $i \in \{1,\dots,n\}$ und $j \in \{1,\dots,n-1\}$. Es seien T_r für $r = 1,\dots,n$ die Hauptuntermatrizen und $p_r(x) := \det(T_r - xI)$ das charaterische Polynom zu T_r,

$$p_r(x) = \det \begin{pmatrix} \alpha_1 - x & \beta_1 & & 0 \\ \beta_1 & \ddots & \ddots & \\ & \ddots & \ddots & \beta_{r-1} \\ 0 & & \beta_{r-1} & \alpha_r - x \end{pmatrix} = (\alpha_r - x)\, p_{r-1}(x) + \beta_{r-1}^2\, p_{r-2}(x) \qquad (r = 2,\dots,n),$$

wobei $p_0(x) = 1$ gesetzt wird. Die Auswertung von $p_n(x)$ für ein $x \in \mathbb{R}$ erfordert daher nur $\mathcal{O}(n)$ Rechenoperationen und die Nullstellen des charakteristischen Polynoms können beispielsweise mit dem Bisektionsverfahren bestimmt werden.

Algorithmus 2.6.1: Bisektion zur Bestimmung von Eigenwerten

Input: $y, z \in \mathbb{R}$ **mit** $y < z$, **sodass** $p_n(y)p_n(z) < 0$, $\epsilon > 0$.

while $|y - z| > \epsilon\,(|y| + |z|)$
  $x = (y + z)/2$
  if $p_n(x)p_n(y) < 0$
    $z = x$
  else
    $y = x$
  end
end

In jedem Schritt wird der Fehler in etwa halbiert und daher konvergiert dieser Algorithmus linear gegen einen Eigenwert von T.
Leider liefert dieser Algorithmus aber nur einen Eigenwert und es ist a priori nicht klar, welcher Eigenwert dies ist. Manchmal ist es aber notwendig, den k-ten größten Eigenwert der Matrix T zu bestimmen. Hierfür stellt das Bisektionsverfahren eine effiziente Lösung dar, wenn man folgende Eigenschaft der Matrix T beachtet.

Satz 2.6.1 (Sturmsche Ketten) *Es sei $T \in \mathbb{R}^{n\times n}$ eine symmetrische tridiagonale Matrix mit Nebendiagonaleinträgen $b_j \neq 0$, $j = 1, \ldots, n-1$. Dann trennen die Eigenwerte von T_{r-1} strikt die Eigenwerte von T_r, d.h.*

$$\lambda_r(T_r) < \lambda_{r-1}(T_{r-1}) < \lambda_{r-1}(T_r) < \ldots < \lambda_2(T_r) < \lambda_1(T_{r-1}) < \lambda_1(T_r).$$

Bezeichnet $a(\lambda)$ die Anzahl an Vorzeichenwechsel in der Folge

$$\{p_0(\lambda), p_1(\lambda), \ldots, p_n(\lambda)\},$$

dann ist $a(\lambda)$ gleich der Anzahl an Eigenwerten von T, die kleiner sind als λ. Hierbei legen wir fest, dass $p_r(\lambda)$ ein anderes Vorzeichen als $p_{r-1}(\lambda)$ hat, falls $p_r(\lambda) = 0$ gilt.

Beweis. Siehe [Golub/Loan, Theorem 8.5.1]. □

Bemerkung 2.6.2 *Die erste Aussage von Satz 2.6.1 bleibt in ähnlicher Form für allgemeine symmetrische Matrizen gültig: Ist $A \in \mathbb{R}^{n\times n}$ symmetrisch und beschreibt A_r die $r\times r$ obere Teilmatrix von A, dann gilt für $r = 1, \ldots, n-1$*

$$\lambda_{r+1}(A_{r+1}) \leq \lambda_r(A_r) \leq \lambda_r(A_{r+1}) \leq \ldots \leq \lambda_2(A_{r+1}) \leq \lambda_1(A_r) \leq \lambda_1(A_{r+1}).$$

Beispiel 2.6.3 (Sturmsche Kette) Die Matrix

$$T = \begin{pmatrix} 1 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 3 & -1 \\ 0 & 0 & -1 & 4 \end{pmatrix}$$

besitzt das Spektrum $\sigma(T) \approx \{0.254, 1.82, 3.18, 4.74\}$. Die Folge

$$\{p_0(2), p_1(2), p_2(2), p_3(2), p_4(2)\} = \{1, -1, -1, 0, 1\}$$

bestätigt, dass T zwei Eigenwerte besitzt, die kleiner sind als $\lambda = 2$.

Wollen wir nun $\lambda_k(T)$, den k-ten größten Eigenwert von T, bestimmen, so liefert zunächst das erste Gerschgorin Theorem, dass $\lambda_k(T) \in [y, z]$, wobei

$$y = \min_{1\leq i\leq n} \{\alpha_i - |\beta_i| - |\beta_{i-1}|\}$$
$$z = \max_{1\leq i\leq n} \{\alpha_i + |\beta_i| + |\beta_{i-1}|\}$$

und $\beta_0 = \beta_n = 0$. Mit diesen Startwerten liefert der folgende Algorithmus eine Approximation zum Eigenwert $\lambda_k(T)$.

Algorithmus 2.6.2: Sturmsche Ketten

Input: $y, z \in \mathbb{R}$ **mit** $y < z$, **sodass** $\lambda_k(T) \in [y, z]$, $\epsilon > 0$.

while $|y - z| > \epsilon\,(|y| + |z|)$
    $x = (y + z)/2$
    if $a(x) \geq n - k$
        $z = x$
    else
        $y = x$
    end
end

Bemerkung 2.6.4 *Ist die Matrix $A \in \mathbb{R}^{n\times n}$ symmetrisch, aber nicht tridiagonal, kann sie wie in Abschnitt 2.5.2 beschrieben durch orthogonale Ähnlichkeitstransformationen auf Tridiagonalgestalt gebracht werden. Auf diese Tridiagonalmatrix können wir dann das Verfahren der Sturmschen Ketten anwenden.*

2.6.2 Das Lanczos-Verfahren

Sei $A \in \mathbb{R}^{n\times n}$ eine symmetrische, schwach besetzte Matrix, deren Eigenwerte in der Form

$$\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_{n-1} \geq \lambda_n$$

geordnet sind. Wenn n sehr groß ist, kann das Lanczos-Verfahren angewandt werden, um die extremalen Eigenwerte λ_1 und λ_n effektiv zu bestimmen. Hierbei wird eine Folge von „kleineren" Tridiagonalmatrizen $T_k \in \mathbb{R}^{k\times k}$ $(1 \leq k \leq n)$ erzeugt, deren extremalen Eigenwerte schnell gegen die extremalen Eigenwerte von A konvergieren.
Die Idee des Verfahrens besteht darin, die Bestimmung der Eigenwerte λ_1 und λ_n von A als ein Optimierungsproblem aufzufassen. Der Rayleigh-Quotient von A

$$r(x) = \frac{x^T A x}{x^T x}, \quad x \neq 0,$$

soll maximiert bzw. minimiert werden, da dieser für symmetrische Matrizen die Werte im Intervall $[\lambda_n, \lambda_1]$ annimmt; es gilt

$$\lambda_1 = \lambda_{\max}(A) = \max_{\substack{x\in\mathbb{R}^n \\ x\neq 0}} r(x) = \max_{\|x\|_2=1} r(x)$$
$$\lambda_n = \lambda_{\min}(A) = \min_{\substack{x\in\mathbb{R}^n \\ x\neq 0}} r(x) = \min_{\|x\|_2=1} r(x).$$

Nehmen wir an, dass $\{q_1, \ldots, q_n\} \subseteq \mathbb{R}^n$ eine Menge orthonormaler Vektoren ist und definieren die Skalare

$$M_j = \lambda_{\max}(Q_j^T A Q_j) = \max_{y \neq 0} \frac{y^T (Q_j^T A Q_j) y}{y^T y} = \max_{\|y\|_2 = 1} r(Q_j y) \leq \lambda_1(A) \tag{2.24}$$

$$m_j = \lambda_{\min}(Q_j^T A Q_j) = \min_{y \neq 0} \frac{y^T (Q_j^T A Q_j) y}{y^T y} = \min_{\|y\|_2 = 1} r(Q_j y) \geq \lambda_n(A), \tag{2.25}$$

wobei $Q_j = [q_1, \ldots, q_j]$ ist, so bestimmt der Lanczos Algorithmus die Vektoren q_j so, dass M_j und m_j immer bessere Approximationen an λ_1 bzw. λ_n sind.
Es sei $u_j \in \operatorname{span}\{q_1, \ldots, q_j\}$ so, dass $M_j = r(u_j)$. Da $r(x)$ am meisten in Richtung des Gradienten

$$\nabla r(x) = \frac{2}{x^T x}(Ax - r(x)x)$$

wächst, ist $M_{j+1} > M_j$ gesichert, wenn q_{j+1} so bestimmt wird, dass

$$\nabla r(u_j) \in \operatorname{span}\{q_1, \ldots, q_{j+1}\}$$

gilt. Falls $v_j \in \operatorname{span}\{q_1, \ldots, q_j\}$ die Gleichung $r(v_j) = m_j$ erfüllt, fordert man analog von q_{j+1}

$$\nabla r(v_j) \in \operatorname{span}\{q_1, \ldots, q_{j+1}\},$$

da $r(x)$ am schnellsten in Richtung $-\nabla r(x)$ fällt. Auch wenn es auf den ersten Blick als schwierig erscheint, q_{j+1} auf diese Weise zu bestimmen, kann man die Tatsache ausnutzen, dass $\nabla r(x) \in \operatorname{span}\{x, Ax\}$ und daher

$$\operatorname{span}\{q_1, \ldots, q_j\} = \operatorname{span}\{q_1, Aq_1, \ldots, A^{j-1}q_1\} = \mathcal{K}_j(A, q_1)$$

gilt, wobei $\mathcal{K}_j(A, q_1)$ den j-ten Krylov-Unterraum von A bzgl. q_1 bezeichnet. Die Krylovräume sind für eine Matrix $A \in \mathbb{R}^{n \times n}$ und einen Vektor $q_1 \in \mathbb{R}^n$ folgendermaßen definiert

$$\begin{aligned} &\mathcal{K}_j(A, q_1) := \operatorname{span}\{q_1, Aq_1, \ldots, A^{j-1}q_1\}, \quad j \geq 1 \\ &\mathcal{K}_0(A, q_1) := \{0\}. \end{aligned}$$

Mit $m \leq n$ sei der größte Index i bezeichnet, für den $\dim(\mathcal{K}_i(A, q_1)) = i$ gilt, also die Vektoren $q_1, Aq_1, \ldots, A^{i-1}q_1$ noch linear unabhängig sind. Der Vektor $A^m q_1$ ist dann linear abhängig von $q_1, Aq_1, \ldots, A^{m-1}q_1$.
Für das Lanczos-Verfahren wird nun eine spezielle orthonormale Basis $q_1, \ldots, q_m$ von $\mathcal{K}_m(A, q_1)$ benötigt. Gemäß obiger Motivation werden die Vektoren $q_1, \ldots, q_m$ dazu so definiert, dass für alle $j \in \{1, \ldots, m\}$ die Vektoren $q_1, \ldots, q_j$ gerade eine Orthonormalbasis von $\mathcal{K}_j(A, q_1)$ bilden.

Kommen wir nun zu der Konstruktion einer solchen Orthonormalbasis. Dafür schauen wir uns zunächst die Krylovmatrix

$$K(A, q_1, n) = [q_1, Aq_1, A^2q_1, \ldots, A^{n-1}q_1]$$

an. Angenommen, wir könnten eine orthonormale Matrix $Q \in \mathbb{R}^{n \times n}$ mit der ersten Spalte q_1 und mit der Eigenschaft, dass $Q^T A Q = T$ tridiagonal ist, finden, so gelte

$$K(A, q_1, n) = Q\,[e_1, Te_1, T^2e_1, \ldots, T^{n-1}e_1]$$

und wir hätten eine QR-Zerlegung der Krylovmatrix $K(A, q_1, n)$. Der Aufspann der ersten j Spalten der Matrix Q würde mit dem der ersten j Spalten von $K(A, q_1, n)$ übereinstimmen.

Daher können wir die Bestimmung der Vektoren $\{q_1, \ldots, q_m\}$ durch Tridiagonalisierung der Matrix A mit einer orthogonalen Matrix Q erreichen, deren erste Spalte q_1 ist.
Wir versuchen die Vektoren $q_1, \ldots, q_m$ direkt zu berechnen. Sei dazu $q \neq 0$ mit $\|q\|_2 = 1$ und setze $q_1 := q$. Für die Bestimmung einer Matrix $Q_k = [q_1, \ldots, q_k]$ mit $Q_k^T A Q_k = T_k$,

$$T_k = \begin{pmatrix} \alpha_1 & \beta_1 & & 0 \\ \beta_1 & \ddots & \ddots & \\ & \ddots & \ddots & \beta_{k-1} \\ 0 & & \beta_{k-1} & \alpha_k \end{pmatrix},$$

liefert ein Vergleich der Spalten in $AQ_k = Q_k T_k$

$$Aq_j = \beta_{j-1} q_{j-1} + \alpha_j q_j + \beta_j q_{j+1}, \quad 1 \leq j \leq k-1, \quad \beta_0 q_0 := 0.$$

Die Orthonormalität der Vektoren q_i liefert

$$\alpha_j = q_j^T A q_j$$

und falls

$$r_j = (A - \alpha_j I) q_j - \beta_{j-1} q_{j-1} \neq 0$$

gilt, setzen wir $q_{j+1} = r_j / \beta_j$ mit $\beta_j = \pm \|r_j\|_2$. Falls $\beta_m = \|r_m\|_2 = 0$ gilt, bricht das Verfahren ab, es gilt dann

$$m = \max_j \dim(\mathcal{K}_j(A, q)).$$

Hat man mit diesem Algorithmus die Vektoren $q_1, \ldots, q_k$ ($k \leq m \leq n$) gefunden (und somit auch die Matrizen $Q_k = [q_1, \ldots, q_k]$ und $T_k = Q_k^T A Q_k$), so kann man folgende Abschätzung der extremalen Eigenwerte vornehmen: Mit $T_k = Q_k^T A Q_k$ und den Gleichungen (2.24)–(2.25) ergibt sich

$$\begin{aligned} M_k &= \lambda_{\max}(T_k) = \lambda_{\max}(Q_k^T A Q_k) \leq \lambda_{\max}(A) \\ m_k &= \lambda_{\min}(T_k) = \lambda_{\min}(Q_k^T A Q_k) \geq \lambda_{\min}(A) \end{aligned}$$

Bei der Wahl der Vektoren q_k und der Matrizen T_k gemäß obigem Algorithmus gilt $M_{k+1} > M_k$ und $m_{k+1} < m_k$. Insgesamt konvergiert

$$\begin{aligned} M_k &\longrightarrow \lambda_{\max}(A) \quad \text{und} \\ m_k &\longrightarrow \lambda_{\min}(A). \end{aligned}$$

Für die Abschätzung werden die Eigenwerte $\lambda_{\max}(T_k)$ und $\lambda_{\min}(T_k)$ der kleineren Tridiagonalmatrix $T_k \in \mathbb{R}^{k \times k}$ benötigt. Hierfür bieten sich Verfahren für Tridiagonalmatrizen an wie etwa das Verfahren der Sturmschen Ketten.
Unter Ausnutzung der oben erwähnten Gleichungen ergibt sich die folgende Lanczos-Iteration.

Algorithmus 2.6.3: Lanczos-Verfahren

Input: $q_1 \in \mathbb{R}^n$ **mit** $\|q_1\|_2 = 1$**. Setze:** $r_0 = q_1$**,** $\beta_0 = 1$**,** $q_0 = 0$**,** $k = 0$**.**

while $\beta_k \neq 0$

$q_{k+1} = r_k/\beta_k$

$k = k + 1$

$\alpha_k = q_k^T A q_k$

$r_k = (A - \alpha_k I)q_k - \beta_{k-1} q_{k-1}$

$\beta_k = \|r_k\|_2$

end

2.7 SINGULÄRWERTZERLEGUNG

Analog zur Schur-Zerlegung für quadratische Matrizen 2.1.6 und 2.1.7 gibt es für nichtquadratische Matrizen $A \in \mathbb{R}^{m\times n}, m, n \in \mathbb{N}$, die sogenannte Singulärwertzerlegung.

Satz 2.7.1 (Singulärwertzerlegung) *Zu jeder Matrix $A \in \mathbb{R}^{m\times n}$ vom Rang r gibt es orthogonale Matrizen $U = [u_1, \ldots, u_m] \in \mathbb{R}^{m\times m}$ und $V = [v_1, \ldots, v_n] \in \mathbb{R}^{n\times n}$, sodass*

$$U^T A V = \Sigma \in \mathbb{R}^{m\times n}.$$

Die Matrix Σ hat Diagonalgestalt,

$$\Sigma = \begin{cases} \begin{pmatrix} \hat{\Sigma} \\ \hline 0 \end{pmatrix}, & \text{wenn } m \geq n, \\ \left(\begin{array}{c|c} \hat{\Sigma} & 0 \end{array} \right), & \text{wenn } m \leq n, \end{cases}$$

wobei $\hat{\Sigma} = \operatorname{diag}(\sigma_1, \ldots, \sigma_p) \in \mathbb{R}^{p\times p}$ mit $p = \min\{m, n\}$ und

$$\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_r > \sigma_{r+1} = \ldots = \sigma_p = 0.$$

*Die Zahlen $\sigma_1, \ldots, \sigma_r > 0$ heißen **Singulärwerte** von A. Weiter gilt:*

i) $\operatorname{Kern}(A) = \operatorname{span}\{v_{r+1}, \ldots, v_n\}$,

ii) $\operatorname{Bild}(A) = \operatorname{span}\{u_1, \ldots, u_r\}$,

iii) $\|A\|_F^2 = \sigma_1^2 + \ldots + \sigma_p^2$,

iv) $\|A\|_2 = \sigma_1$,

v) $u_i^T A = \sigma_i v_i^T, \quad Av_i = \sigma_i u_i, \quad i = 1, \ldots, p.$
Die Vektoren u_i und v_i werden linke bzw. rechte Singulärvektoren genannt.

Beweis. Siehe z.B. [Nipp/Stoffer, Satz 9.6]. □

Bemerkung 2.7.2 (Geometrische Interpretation der Singulärwerte) *Es sei*

$$\mathcal{S} = \{\alpha_1 v_1 + \ldots + \alpha_r v_r \,|\, \alpha_1^2 + \ldots + \alpha_r^2 = 1\}$$

die Einheitssphäre im Unterraum $\operatorname{span}\{v_1, \ldots, v_r\} \subseteq \mathbb{R}^n$. *Dann ist*

$$\mathcal{E} = \{Ax \,|\, x \in \mathcal{S}\}$$

ein Ellipsoid im Unterraum $\operatorname{span}\{u_1, \ldots, u_r\} \subseteq \mathbb{R}^m$ *mit den Halbachsen* $\{\sigma_i u_i \,|\, i = 1, \ldots, r\}$.

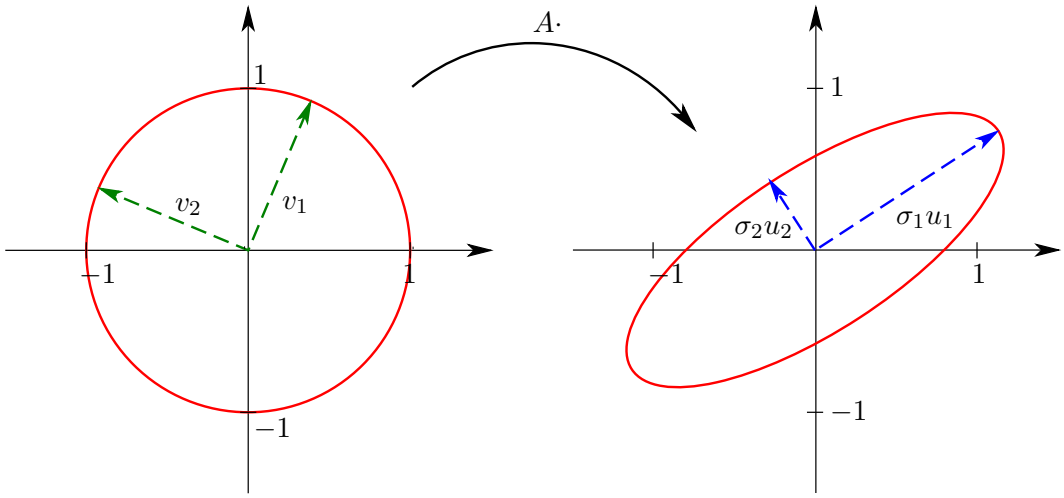


Abb. 2.3: Geometrische Interpretation der Singulärwerte

Mit der Singulärwertzerlegung einer Matrix $A \in \mathbb{R}^{m\times n}$ kann man z.B. das lineare Augleichsproblem $\|Ax - d\|_2 \to \min$ für $d \in \mathbb{R}^m$ mit $m > n$ lösen. Ist $\operatorname{Rang}(A) = r$ und

$$U^T A V = \Sigma = \begin{pmatrix} \Sigma_r & 0 \\ 0 & 0 \end{pmatrix}$$

die Singulärwertzerlegung von A mit einer regulären Diagonalmatrix $\Sigma_r \in \mathbb{R}^{r\times r}$, so gilt

$$\|Ax - d\|_2 = \|U \begin{pmatrix} \Sigma_r & 0 \\ 0 & 0 \end{pmatrix} V^T x - d\|_2 = \| \begin{pmatrix} \Sigma_r & 0 \\ 0 & 0 \end{pmatrix} V^T x - U^T d\|_2 = \left\| \begin{pmatrix} \Sigma_r y_1 - c_1 \\ -c_2 \end{pmatrix} \right\|_2 \to \min,$$

wobei

$$y = V^T x = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}, \quad c = U^T b = \begin{pmatrix} c_1 \\ c_2 \end{pmatrix}$$

und $y_1, c_1 \in \mathbb{R}^r, y_2 \in \mathbb{R}^{n-r}, c_2 \in \mathbb{R}^{m-r}$. Dann folgt

$$y_1 = \Sigma_r^{-1} c_1 \quad \text{und} \quad x = V \begin{pmatrix} \Sigma_r^{-1} c_1 \\ 0 \end{pmatrix}.$$

Ein weiteres Anwendungsgebiet der Singulärwertzerlegung findet sich in der Datenkompression in der Signal- und Bildverarbeitung. Dies wird durch folgenden Satz motiviert.

Satz 2.7.3 (Rang-k-Bestapproximation) *Es sei $A = U\Sigma V^T$ die Singulärwertzerlegung von $A \in \mathbb{R}^{m\times n}$, $k < r = \mathrm{Rang}(A)$ und*

$$A_k = \sum_{i=1}^{k} \sigma_i u_i v_i^T.$$

Dann gilt

$$\min_{\substack{B\in\mathbb{R}^{m\times n}\\ \mathrm{Rang}(B)=k}} \|A - B\|_2 = \|A - A_k\|_2 = \sigma_{k+1},$$

B ist also die beste Approximation an A mit Rang k.

Dieses Resultat lässt sich beispielsweise in der Bildverarbeitung folgendermaßen verwenden.

Beispiel 2.7.4 (Bildverarbeitung und Singulärwertzerlegung) Ein Bild der Größe $m \times n$ Pixel kann mit Hilfe von drei Matrizen $R, G, B \in \mathbb{R}^{m\times n}$ beschrieben werden, die im Eintrag (i, j) die Rot-, Grün- bzw. Blaustufe des Pixels an dieser Stelle enthalten. Sind

$$R = U_R\Sigma_R V_R^T \quad G = U_G\Sigma_G V_G^T \quad B = U_B\Sigma_B V_B^T$$

die Singulärwertzerlegungen von R, G und B, so sind

$$\begin{aligned} R_k &= U_R(:, 1:k)\,\Sigma_R(1:k, 1:k)\,V_R(:, 1:k)^T \\ G_k &= U_G(:, 1:k)\,\Sigma_G(1:k, 1:k)\,V_G(:, 1:k)^T \\ B_k &= U_B(:, 1:k)\,\Sigma_B(1:k, 1:k)\,V_B(:, 1:k)^T \end{aligned}$$

gemäß Satz 2.7.3 die besten Approximationen mit Rang k. Nachfolgend ist diese Datenkompression eines Bildes für $k = 5, 15$ und 100 durchgeführt worden.

MATLAB-Funktion: Bild_Kompression.m

```
function Bild_Kompression(k)
    close all
    % lade Daten
    X = imread('bild.jpeg');
    size(X)

    R = double(X(:,:,1));
    G = double(X(:,:,2));
    B = double(X(:,:,3));

    % Singulärwertzerlegung
    [Ur,Sr,Vr] = svd(R);
    [Ug,Sg,Vg] = svd(G);
    [Ub,Sb,Vb] = svd(B);

    % Darstellung der Singulärwerte
    sigmar = diag(Sr);
    sigmag = diag(Sg);
    sigmab = diag(Sb);

    figure(1)
```

```
22      semilogy(1:length(sigmar),sigmar,'r.', ...
23               1:length(sigmag),sigmag,'g.',...
24               1:length(sigmab),sigmab,'b.')
25      title('Singulärwerte von R,G,B');
26      legend('R','G','B');
27
28      % Rang-k-Bestapproximationen
29      k_length = length(k);
30      for i=1:k_length
31          figure(1)
32          hold on,semilogy(xlim,[sigmar(k(i)),sigmar(k(i))],'k-')
33          hold off
34          C(:,:,1) = Ur(:,1:k(i))*Sr(1:k(i),1:k(i))*Vr(:,1:k(i))';
35          C(:,:,2) = Ug(:,1:k(i))*Sg(1:k(i),1:k(i))*Vg(:,1:k(i))';
36          C(:,:,3) = Ub(:,1:k(i))*Sb(1:k(i),1:k(i))*Vb(:,1:k(i))';
```

Abb. 2.4: Bild-Datenkompression für $k = 5, 15$ und 100

Im Folgenden betrachten wir o.B.d.A. den Fall einer Matrix $A \in \mathbb{R}^{m\times n}$ mit $m \geq n$, da aus der Singulärwertzerlegung von A direkt diejenige von A^T folgt.
Zwischen der Singulärwertzerlegung von A und der Schur-Zerlegung der Matrizen A^TA bzw. AA^T besteht folgender Zusammenhang: Ist

$$U^TAV = \Sigma = \begin{pmatrix} \hat{\Sigma} \\ \hline 0 \end{pmatrix}$$

mit $\hat{\Sigma} = \mathrm{diag}(\sigma_1, \ldots, \sigma_n)$ die Singulärwertzerlegung von A, dann sind

$$V^T(A^TA)V = \mathrm{diag}(\sigma_1^2, \ldots, \sigma_n^2)$$

und

$$U^T(AA^T)U = \mathrm{diag}(\sigma_1^2, \ldots, \sigma_n^2, \underbrace{0, \ldots, 0}_{m-n})$$

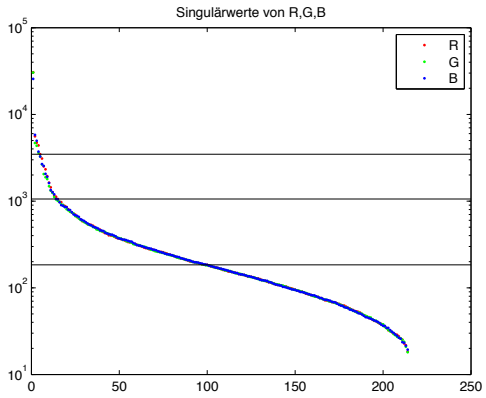


Abb. 2.5: Singulärwerte

die Schur-Zerlegungen von A^TA bzw. AA^T und die Singulärwerte von A sind die positiven Quadratwurzeln aus den Eigenwerten von A^TA bzw. AA^T. Das intuitive Vorgehen wäre also:

- berechne $C = A^TA$
- wende das QR-Verfahren auf C an, erhalte

  $$V^TCV = \mathrm{diag}(\sigma_1^2, \ldots, \sigma_n^2)$$

  und damit die Singulärwerte.

Dieses Vorgehen ist aber unvorteilhaft, denn sofern A schlecht konditioniert ist, würden sich die Rundungsfehler im QR-Verfahren angewandt auf A^TA quadrieren. Dieser Verlust an Genauigkeit soll folgendes Beispiel illustrieren.

Beispiel 2.7.5 Für die Matrix

$$A = \begin{pmatrix} 1 & 1 \\ \varepsilon & 0 \\ 0 & \varepsilon \end{pmatrix}, \quad |\varepsilon| < \sqrt{\mathrm{eps}}$$

mit eps der Maschinengenauigkeit des Rechners, ist die Matrix A^TA gegeben durch

$$A^TA = \begin{pmatrix} 1+\varepsilon^2 & 1 \\ 1 & 1+\varepsilon^2 \end{pmatrix}.$$

A besitzt die singulären Werte $\sigma_1 = \sqrt{2+\varepsilon^2}$ und $\sigma_2 = |\varepsilon|$. Bei Gleitpunktrechnung mit Genauigkeit eps erhält man statt A^TA die Matrix

$$\widetilde{A^TA} = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$

mit den Eigenwerte $\widetilde{\lambda_1} = 2$ und $\widetilde{\lambda_2} = 0$ und σ_2 stimmt nicht bis auf Maschinengenauigkeit mit $\sqrt{\widetilde{\lambda_2}} = 0$ überein.

Wir werden im Folgenden einen Algorithmus zur Bestimmung der Singulärwertzerlegung vorstellen, der diese Problematik berücksichtigt. Er wurde erstmals 1965 von Golub, Kahan und Reinsch beschrieben und besteht aus zwei Schritten:

1. Reduziere A auf eine Matrix B in Bidiagonalform,

$$U_B^T A V_B = \begin{pmatrix} B \\ 0 \end{pmatrix} = \begin{pmatrix} \diagdown & \\ & \diagdown \\ \hline & 0 \end{pmatrix} \in \mathbb{R}^{m \times n}.$$

2. Führe impliziten QR-Schritt auf $B^T B$ aus.

2.7.1 Transformation auf Bidiagonalform

Um den Aufwand des Verfahrens möglichst gering zu halten, reduziert man A auf eine Matrix in Bidiagonalform

$$U_B^T A V_B = \begin{pmatrix} B \\ 0 \end{pmatrix} \in \mathbb{R}^{m \times n}, \quad B = \begin{pmatrix} d_1 & f_1 & & 0 \\ & \ddots & \ddots & \\ & & \ddots & f_{n-1} \\ 0 & & & d_n \end{pmatrix} \in \mathbb{R}^{n \times n}.$$

Die Matrix B hat dieselben Singulärwerte wie A, denn es gilt

$$B^T B = \begin{pmatrix} B \\ 0 \end{pmatrix}^T \begin{pmatrix} B \\ 0 \end{pmatrix} = (U_B^T A V_B)^T (U_B^T A V_B) = V_B^T (A^T A) V_B.$$

Um A auf diese Form zu bringen, wendet man insgesamt n Householdertransformationen von links und $n-2$ Householdertransformationen von rechts auf A an. Dies geschieht folgendermaßen: Zunächst bestimmt man eine Householdermatrix $U_1 \in \mathbb{R}^{m \times m}$, sodass in $U_1^T A$ alle Elemente in der ersten Spalte bis auf a_{11} annuliert werden. Anschließend multipliziert man diese Matrix $U_1^T A$ mit einer Householderspiegelung der Form

$$V_1 = \left(\begin{array}{c|c} 1 & 0 \\ \hline 0 & \widetilde{V}_1 \end{array}\right)$$

von rechts, sodass die Elemente in der ersten Zeile auf den Positionen $(1,3), \ldots, (1,n)$ verschwinden.

$$A^{(2)} = U_1^T A V_1 = \begin{pmatrix} \boxplus & * & \cdots & \cdots & * \\ 0 & \vdots & & & \vdots \\ \vdots & \vdots & & & \vdots \\ \vdots & \vdots & & & \vdots \\ 0 & * & \cdots & \cdots & * \end{pmatrix} \left(\begin{array}{c|c} 1 & 0 \\ \hline 0 & \widetilde{V}_1 \end{array}\right) = \begin{pmatrix} \boxplus & \boxplus & 0 & \cdots & 0 \\ 0 & * & * & \cdots & * \\ \vdots & \vdots & \vdots & & \vdots \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & * & * & \cdots & * \end{pmatrix}.$$

Dasselbe Vorgehen wird dann auf die zweite Spalte und Zeile der so gewonnenen Matrix $A^{(2)}$ ausgeführt usw. Es ergibt sich so eine Folge von Matrizen $A^{(1)}, \ldots, A^{(n+1)}$, wobei $A^{(1)} = A$ gesetzt wird und

$$A^{(j+1)} = U_j^T A^{(j)} V_j \quad \text{für } j = 1, \ldots, n,$$

wobei

$$U_j^T = \left(\begin{array}{c|c} I_{j-1} & 0 \\ \hline 0 & \widetilde{U}_j^T \end{array}\right) \in \mathbb{R}^{m \times m} \quad j = 1, \ldots, n$$

und $\widetilde{U}_j^T$ die Householdermatrix zu $(a_{jj}^{(j)}, \ldots, a_{mj}^{(j)})^T$ ist sowie

$$V_j = \left(\begin{array}{c|c} I_j & 0 \\ \hline 0 & \widetilde{V}_j \end{array}\right) \in \mathbb{R}^{n\times n} \quad j = 1, \ldots, n-2,$$
$$V_{n-1} = V_n = I_n$$

und $\widetilde{V}_j$ ist die Householdermatrix zu $(a_{j,j+1}^{(j)}, \ldots, a_{jn}^{(j)})$. $A^{(n)}$ hat schließlich die gewünschte Bidiagonalgestalt,

$$A^{(n)} = U_n^T \ldots U_1^T A V_1 \ldots V_{n-2} = U_B^T A V_B = \begin{pmatrix} B \\ \hline 0 \end{pmatrix} = \left(\begin{array}{c} \diagdown \\ \hline 0 \end{array}\right).$$

Beispiel 2.7.6 Um das oben beschriebene Vorgehen zu illustrieren, wird das Schema hier beispielhaft für eine Matrix $A \in \mathbb{R}^{5\times 4}$ vorgestellt:

$$A^{(1)} = \begin{pmatrix} + & + & + & + \\ + & + & + & + \\ + & + & + & + \\ + & + & + & + \\ + & + & + & + \end{pmatrix} \xrightarrow{U_1^T\cdot} \begin{pmatrix} * & * & * & * \\ 0 & * & * & * \\ 0 & * & * & * \\ 0 & * & * & * \\ 0 & * & * & * \end{pmatrix} \xrightarrow{\cdot V_1} \begin{pmatrix} + & * & 0 & 0 \\ 0 & * & * & * \\ 0 & * & * & * \\ 0 & * & * & * \\ 0 & * & * & * \end{pmatrix}$$

$$\xrightarrow{U_2^T\cdot} \begin{pmatrix} + & + & 0 & 0 \\ 0 & * & * & * \\ 0 & 0 & * & * \\ 0 & 0 & * & * \\ 0 & 0 & * & * \end{pmatrix} \xrightarrow{\cdot V_2} \begin{pmatrix} + & + & 0 & 0 \\ 0 & + & * & 0 \\ 0 & 0 & * & * \\ 0 & 0 & * & * \\ 0 & 0 & * & * \end{pmatrix} \xrightarrow{U_3^T\cdot} \begin{pmatrix} + & + & 0 & 0 \\ 0 & + & + & 0 \\ 0 & 0 & * & * \\ 0 & 0 & 0 & * \\ 0 & 0 & 0 & * \end{pmatrix} \xrightarrow{U_4^T\cdot} \begin{pmatrix} + & + & 0 & 0 \\ 0 & + & + & 0 \\ 0 & 0 & + & + \\ 0 & 0 & 0 & * \\ 0 & 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} B \\ \hline 0 \end{pmatrix}.$$

Bemerkung 2.7.7 (Aufwand der Transformation) *Das Verfahren der Bidiagonalisierung erfordert $4mn^2 - \frac{4}{3}n^3$ FLOPs. Werden die Matrizen U_B und V_B explizit benötigt, so kann dies mit $4m^2n - \frac{4}{3}n^3$ bzw. $\frac{4}{3}n^3$ Rechenoperationen bewerkstelligt werden.*

2.7.2 Impliziter QR-Schritt auf B^TB

Wir setzen ab jetzt stets voraus, dass die Bidiagonalmatrix B unreduziert ist, d.h. in diesem Kontext $d_k, f_k \neq 0$ für alle $k = 1, \ldots, n-1$. Ist dies nicht der Fall, so zerfällt das Problem folgendermaßen in zwei kleinere Probleme. Sofern $f_k = 0$ für ein $k \in \{1, \ldots, n-1\}$ gilt, hat B die Struktur

$$B = \begin{array}{cc} \begin{pmatrix} B_1 & 0 \\ 0 & B_2 \end{pmatrix} & \begin{array}{c} k \\ n-k \end{array} \\ \begin{array}{cc} k & n-k \end{array} & \end{array}$$

und wir erhalten sofort die gewünschte Deflation. Im Fall $d_k = 0$ für ein $k \in \{1, \ldots, n-1\}$ ist dafür etwas mehr Arbeit nötig. Mit Hilfe von $n-k$ Givens-Rotationen ist es möglich die ganze k-te Zeile von B zu annullieren. Dafür werden Rotationen des Typs $(k, k+1), \ldots, (k, n)$ benötigt, wobei der Typ beschreibt, welche Zeilen bzw. Spalten durch die Multiplikation mit der jeweiligen Givens-Matrix von links verändert werden. Mit der der Annulierung der k-ten Zeile gilt insbesondere $f_k = 0$ und wir haben die Deflation der Matrix B bewirkt.
Für $B \in \mathbb{R}^{6\times 6}$ und $j = 3$ sieht dieses Vorgehen folgendermaßen aus:

$$B = \begin{pmatrix} + & + & & & \\ & + & + & & \\ & & 0 & * & \\ & & & + & + \\ & & & & + \end{pmatrix} \xrightarrow{G_{34}^T\cdot} \begin{pmatrix} + & + & & & \\ & + & + & & \\ & & 0 & * & \\ & & & + & + \\ & & & & + \end{pmatrix}$$

$$\xrightarrow{G_{35}^T\cdot} \begin{pmatrix} + & + & & & \\ & + & + & & \\ & & 0 & * & \\ & & & + & + \\ & & & & + \end{pmatrix} \xrightarrow{G_{36}^T\cdot} \begin{pmatrix} + & + & & & \\ & + & + & & \\ & & 0 & & \\ & & & + & + \\ & & & & + \end{pmatrix} = \left(\begin{array}{ccc|cc} + & + & & & \\ & + & + & & \\ & & 0 & & \\ \hline & & & + & + \\ & & & & + \end{array}\right) = \begin{pmatrix} B_1 & 0 \\ 0 & B_2 \end{pmatrix}.$$

Nachfolgend werden wir nun einen impliziten QR-Schritt auf die Tridiagonalmatrix $T = B^T B$ ausführen, ohne diese explizit zu berechnen, da dies wie oben im Beispiel 2.7.5 mit Rechenfehlern verbunden wäre. Dies geschieht in drei Schritten:

1. Als erstes bestimmt man den *Shiftparameter* μ als denjenigen Eigenwert von

$$T(m:n, m:n) = \begin{pmatrix} d_m^2 + f_m^2 & d_m f_n \\ d_m f_n & d_n^2 + f_n^2 \end{pmatrix}, \quad m = n-1,$$

der dichter an $d_n^2 + f_m^2$ liegt.

2. Nun wird auf die erste Spalte der geshifteten Matrix $T - \mu I$ eine Givens-Rotation angewendet: Man bestimmt $c_1 = \cos(\theta_1)$ und $s_1 = \sin(\theta_1)$, sodass

$$\begin{pmatrix} c_1 & -s_1 \\ s_1 & c_1 \end{pmatrix} \underbrace{\begin{pmatrix} d_1 - \mu \\ d_1 f_1 \end{pmatrix}}_{\text{1. Spalte von } T-\mu I} = \begin{pmatrix} * \\ 0 \end{pmatrix}$$

und somit die Givens-Matrix $G_1 = G(1, 2, \theta_1)$, welche in der Tridiagonalmatrix $T - \mu I$ das Element an Position $(2, 1)$ annulliert.
Nun könnte man Givens-Rotationen $G_2, \ldots, G_{n-1}$ bestimmen, sodass für

$$Q = G_1 \cdot \ldots \cdot G_{n-1} \tag{2.26}$$

die Matrix $Q^T T Q$ tridiagonal ist und $Qe_1 = G_1 e_1$ gilt. Nach dem impliziten Q-Theorem wäre $Q^T T Q$ das Ergebnis eines QR-Schrittes. Für dieses Vorgehen bräuchte man aber explizit die Matrix $T = B^T B$. Dies soll vermieden werden.

3. Um T nicht berechnen zu müssen, wendet man die Givens-Rotationen direkt auf B an. Nach der Multiplikation mit G_1 wird B in diese Form gebracht,

$$B = \begin{pmatrix} + & + & & & \\ & + & + & & \\ & & + & + & \\ & & & + & + \\ & & & & + \end{pmatrix} \xrightarrow{\cdot G_1} \begin{pmatrix} * & * & & & \\ * & * & + & & \\ & & + & + & \\ & & & + & + \\ & & & & + \end{pmatrix} = BG_1.$$

Man kann nun weitere Givens-Matrizen $U_1, \ldots, U_{n-1}$ und $V_2, \ldots, V_{n-1}$ bestimmen, sodass das Nichtnullelement an Position $(2, 1)$ unterhalb der Diagonalen nach unten verschoben wird und schließlich verschwindet:

$$BG_1 = \begin{pmatrix} + & + & & & \\ * & + & + & & \\ & & + & + & \\ & & & + & + \\ & & & & + \end{pmatrix} \xrightarrow{U_1^T\cdot} \begin{pmatrix} + & + & * & & \\ 0 & + & + & & \\ & & + & + & \\ & & & + & + \\ & & & & + \end{pmatrix} \xrightarrow{\cdot V_2} \begin{pmatrix} + & + & 0 & & \\ & + & + & & \\ & * & + & + & \\ & & & + & + \\ & & & & + \end{pmatrix}$$

$$\xrightarrow{U_2^T\cdot} \begin{pmatrix} + & + & & & \\ & + & + & * & \\ & 0 & + & + & \\ & & & + & + \\ & & & & + \end{pmatrix} \xrightarrow{\cdot V_3} \begin{pmatrix} + & + & & & \\ & + & + & 0 & \\ & & + & + & \\ & & * & + & + \\ & & & & + \end{pmatrix} \xrightarrow{U_3^T\cdot} \begin{pmatrix} + & + & & & \\ & + & + & & \\ & & + & + & \\ & & 0 & + & + \\ & & & & + \end{pmatrix}$$

Dieser Prozess führt BG_1 zurück in eine Bidiagonalmatrix

$$\begin{aligned}\bar{B} &= \underbrace{(U_1 \cdot \ldots \cdot U_{n-1})^T}_{=:U^T} B \underbrace{(G_1 V_2 \cdot \ldots \cdot V_{n-1})}_{=:V} \\ &= U^T B V.\end{aligned}$$

Damit ist

$$\bar{B}^T \bar{B} = V^T B^T B V$$

wiederum in Tridiagonalgestalt und die erste Spalte v_1 der orthogonalen Matrix V stimmt mit derjenigen von Q aus (2.26) überein,

$$v_1 = V e_1 = G_1 V_2 \ldots V_{n-1} = G_1 e_1 = Q e_1.$$

Mit dem impliziten Q-Theorem kann man schließlich folgern, dass V und Q im Wesentlichen gleich sind. Damit ist ein QR-Schritt auf $B^T B$ implizit durchgeführt worden.

Zum Schluss dieses Abschnittes fassen wir das Vorgehen in einem Algorithmus zusammen.

Algorithmus 2.7.1: Golub-Kahan SVD Step

Für eine Bidiagonalmatrix $B \in \mathbb{R}^{m \times n}$ mit $d_k, f_k \neq 0$ für $k = 1, \ldots, n-1$ überschreibt dieser Algorithmus B mit der Bidiagonalmatrix $\bar{B} = \bar{U}^T B \bar{V}$, wobei $\bar{U}$ und $\bar{V}$ orthogonal sind und $\bar{V}$ im Wesentlichen diejenige Matrix ist, die man mit einem impliziten QR-Schritt auf $T = B^T B$ erhalten würde.

Bestimme μ als den Eigenwert von

$$\begin{pmatrix} t_{n-1,n-1} & t_{n-1,n} \\ t_{n,n-1} & t_{nn} \end{pmatrix},$$

der näher an t_{nn} liegt.

$y = t_{11} - \mu$

$z = t_{12}$

for $k = 1, \ldots, n-1$

  Bestimme $c = \cos(\theta)$ und $s = \sin(\theta)$,sodass

  $\begin{pmatrix} y & z \end{pmatrix} \begin{pmatrix} c & s \\ -s & c \end{pmatrix} = \begin{pmatrix} * & 0 \end{pmatrix}$

  $B = B\, G(k, k+1, \theta)$

  $y = b_{kk}$

  $z = b_{k+1,k}$

  Bestimme $c = \cos(\theta)$ und $s = \sin(\theta)$,sodass

  $\begin{pmatrix} c & s \\ -s & c \end{pmatrix}^T \begin{pmatrix} y \\ z \end{pmatrix} = \begin{pmatrix} * \\ 0 \end{pmatrix}$

  $B = G(k, k+1, \theta)^T B$

```
    if k < n − 1
        y = b_{k,k+1}
        z = b_{k,k+2}
    end
end
```

2.7.3 Gesamter Singulärwert-Algorithmus

Für eine Matrix $A \in \mathbb{R}^{m\times n}$ wird die Singulärwertzerlegung bestimmt, indem A zunächst wie in Abschnitt 2.7.1 beschrieben auf Bidiagonalform transformiert wird und dann der Algorithmus 2.7.1 iterativ angewendet wird, um die Zerlegung $A = U\Sigma V^T$ zu erhalten. Wie beim QR-Verfahren müssen wiederum die Elemente unterhalb der Diagonalen beobachtet werden, um mögliche Deflationen zu erkennen. Der gesamte Prozess zur Bestimmung der Singulärwertzerlegung von A ist in dem folgenden Algorithmus festgehalten.

Algorithmus 2.7.2: Gesamte Singulärwertzerlegung

Gegeben: $A \in \mathbb{R}^{m\times n}$ $(m \geq n)$, $\epsilon > 0$ Toleranzwert.

1. Reduziere A mittels n Householdertransformationen von links und $n-2$ Householdertransformationen von rechts auf Bidiagonalform,
$$B = (U_1 \dots U_n)^T A V_1 \dots V_{n-2}$$

2. **while** $q < n$
   Setze alle Elemente in der oberen Nebendiagonalen $b_{i,i+1} = 0$, die
   $$|b_{i,i+1}| \leq \epsilon \left(|b_{ii}| + |b_{i+1,i+1}|\right) \quad i = 1, \dots, n-1$$
   erfüllen.
   Finde größtes $q \geq 0$ und kleinstes $p \geq 0$, sodass
   $$B = \begin{pmatrix} B_{11} & & \\ & B_{22} & \\ & & B_{33} \\ p & n-p-q & q \end{pmatrix} \begin{matrix} p \\ n-p-q \\ q \end{matrix}$$
   wobei
   - B_{11}: Bidiagonalmatrix
   - B_{22}: Bidiagonalmatrix ohne Nullen auf der oberen Nebendiagonalen
   - B_{33}: Diagonalmatrix

```
    sind.
    if q < n
       if ein Diagonaleintrag in B_22 ist null
          Transformiere die ganze Zeile in Nullen
       else
          Wende Algorithmus 2.7.1 auf B_22 an

                B = diag(I_p, U, I_{q+m-n})^T B diag(I_p, V, I_q)

       end
    end
end
```

2.8 VERALLGEMEINERTE EIGENWERTPROBLEME

Es seien $A, B \in \mathbb{C}^{n\times n}$. In diesem Teil untersuchen wir verallgemeinerte Matrixeigenwertprobleme, also Probleme der Form: Finde $\lambda \in \mathbb{C}$ und $x \in \mathbb{C}^n \setminus \{0\}$, sodass

$$Ax = \lambda Bx. \tag{2.27}$$

Bislang haben wir stets den Fall $B = I_n$ betrachtet.
Probleme dieser Art treten häufig in ingenieurwissenschaftlichen Anwendungen auf, z.B. bei Schwingungsproblemen von Gebäuden oder Brücken, vgl. (2.2) in Beispiel 2.0.1. Auch die Berechnung der extremalen Eigenwerte einer vorkonditionierten Matrix $B^{-1}A$ kann als verallgemeinertes Eigenwertproblem aufgefasst werden.
Um diese Probleme untersuchen zu können, benötigen wir zunächst einige Definitionen.

Definition 2.8.1 (Matrixbüschel) *Zu $A, B \in \mathbb{C}^{n\times n}$ bezeichnen wir die Menge*

$$(A, B) := \{A - \lambda B \,|\, \lambda \in \mathbb{C}\} \subset \mathbb{C}^{n\times n}$$

als (Matrix-)Büschel. Ein Büschel (A, B) heißt regulär, falls

$$\det(A - zB) \not\equiv 0.$$

Andernfalls ist das Büschel singulär.

Definition 2.8.2 (Charakteristisches Polynom) *Ist (A, B) ein reguläres Büschel, so heißt*

$$p(z) := \det(A - zB), \quad z \in \mathbb{C},$$

das charakteristische Polynom des Büschels (A, B). Die Nullstellen von p werden als endliche Eigenwerte von (A, B) bezeichnet. Falls $\operatorname{grad}(p(z)) = k$ $(k < n)$ gilt, dann sagen wir, dass $A-\lambda B$ $(n-k)$ unendliche Eigenwerte hat. Wir bezeichnen die Menge aller endlichen Eigenwerte des Büschels (A, B) mit

$$\sigma(A, B) := \{z \in \mathbb{C} \,|\, \det(A - zB) = 0\}$$

und ein Vektor $x \in \mathbb{C}^n \setminus \{0\}$, der die Gleichung (2.27) erfüllt, wird Eigenvektor von $A - \lambda B$ genannt.

Satz 2.8.3 *Das verallgemeinerte Eigenwertproblem* (2.27) *hat genau dann* n *endliche Eigenwerte, wenn* $\mathrm{Rang}(B) = n$ *gilt.*

Aufgabe 2.8.4 Man beweise Satz 2.8.3.

Das folgende Beispiel zeigt, dass die Menge der endlichen Eigenwerte $\sigma(A, B)$ sowohl leer als auch endlich als auch unendlich sein kann, wenn B singulär ist.

Beispiel 2.8.5

1. $A = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \quad B = \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} \quad \Rightarrow p(z) \equiv 1 \neq 0,\ \sigma(A, B) = \emptyset.$
2. $A = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} \quad B = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \quad \Rightarrow p(z) = 1 - z,\ \sigma(A, B) = \{1\}.$
3. $A = \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix} \quad B = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \quad \Rightarrow p(z) \equiv 0,\ \sigma(A, B) = \mathbb{C}.$
4. $A = \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix} \quad B = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \quad \Rightarrow p(z) = z^2 + 1,\ \sigma(A, B) = \{\pm i\}.$

Symmetrische Büschel können konjugiert komplexe Eigenwerte haben.

Bemerkung 2.8.6 *Ist* $0 \neq \lambda \in \sigma(A, B)$*, dann gilt* $\frac{1}{\lambda} \in \sigma(B, A)$*. Falls* B *regulär ist, gilt darüberhinaus*

$$\sigma(A, B) = \sigma(B^{-1}A, I) = \sigma(B^{-1}A).$$

Die letzte Eigenschaft legt in dem Fall, dass B invertierbar ist, das intuitive Vorgehen nahe, das verallgemeinerte Eigenwertproblem (2.27) zwei Schritten zu lösen:

1. Löse $BC = A$, z.B. mit Gauß-Elimination mit Pivotisierung,
2. Wende den QR-Algorithmus 2.5.5 auf C an.

Bei diesem Vorgehen kommt es im ersten Schritt zu Rundungsfehlern der Größenordnung $\mathrm{eps}\,\|A\|_2\|B^{-1}\|_2$. Ist B schlecht konditioniert, kann dieses Verfahren ungenaue Ergebnisse liefern. Dies soll vermieden werden.

Analog zum Eigenwertproblem $Ax = \lambda x$ existieren für verallgemeinerte Eigenwertprobleme verschiedene Normalformen. Das Pendant zur Jordan'schen Normalform wäre die Kronecker'sche Normalform, die jedoch aus numerischer Sicht ähnliche Nachteile hat wie Jordanform. Daher betrachtet man die folgende Zerlegung, das Pendant zur Schur-Zerlegung, Satz 2.1.6.

Satz 2.8.7 (Verallgemeinerte Schur-Form) *Es seien* $A, B \in \mathbb{C}^{n\times n}$*. Dann gibt es unitäre Matrizen* $Q, Z \in \mathbb{C}^{n\times n}$*, sodass*

$$Q^H AZ = T \quad \textit{und} \quad Q^H BZ = S$$

obere Dreiecksmatrizen sind. Falls für ein $k \in \{1, \ldots, n\}$ *sowohl* $t_{kk} = 0$ *als auch* $s_{kk} = 0$ *gilt, dann ist* $\sigma(A, B) = \mathbb{C}$*. Ansonsten gilt*

$$\sigma(A, B) = \{t_{ii}/s_{ii} \,|\, s_{ii} \neq 0\}. \tag{2.28}$$

Beweis. $\{B_k\}_{k\in\mathbb{N}}$ sei eine Folge regulärer Matrizen, die gegen B konvergieren. Für jedes $k \in \mathbb{N}$ sei $Q_k^H(AB_k^{-1})Q_k = R_k$ die Schur-Zerlegung von AB_k^{-1}. Außedem sei $Z_k \in \mathbb{C}^{n\times n}$ unitär und so, dass $Z_k^H(B_k^{-1}Q_k) = S_k^{-1}$ obere Dreiecksform hat. Dann folgt, dass auch

$$Q_k^H AZ_k = R_k S_k \quad \text{und} \quad Q_k^H B_k Z_k = S_k$$

obere Dreiecksmatrizen sind.
Nach dem Satz von Bolzano-Weierstraß besitzt die beschränkte Folge $\{[Q_k, Z_k]\}_{k\in\mathbb{N}}$ eine konvergente Teilfolge, $\lim_{i\to\infty}[Q_{k_i}, Z_{k_i}] = [Q, Z]$. Man kann leicht zeigen, dass Q und Z selbst unitär sind und dass $Q^H AZ$ und $Q^H BZ$ obere Dreiecksmatrizen sind.
Die Behauptung (2.28) folgt aus der Identität

$$\det(A - \lambda B) = \det(QZ^H)\prod_{i=1}^{n}(t_{ii} - \lambda s_{ii}).$$

□

Wenn A und B reellwertig sind, ist der folgende Satz interessant, der das Analogon zur reellen Schur-Zerlegung 2.1.7 darstellt.

Satz 2.8.8 (Verallgemeinerte reelle Schur-Form) *Zu $A, B \in \mathbb{R}^{n\times n}$ existieren orthogonale Matrizen $Q, Z \in \mathbb{R}^{n\times n}$, sodass $Q^T AZ$ eine obere Quasi-Dreiecksmatrix (obere Block-Dreiecksmatrix mit 1×1 oder 2×2-Blöcken auf der Diagonalen) und $Q^T BZ$ eine obere Dreiecksmatrix sind.*

Beweis. Siehe [Stewart]. □

Im Folgenden werden wir den *QZ-Algorithmus* zur Bestimmung der verallgemeinerten Eigenwerte vorstellen. Der gesamte QZ-Prozess besteht aus drei Teilen, die wir nacheinander gesondert behandeln werden:

1. Simultane Transformation von A auf obere Hessenberg- bzw. von B auf obere Dreiecksform,
2. Deflation,
3. QZ-Schritt.

2.8.1 Transformation auf Hessenberg- bzw. obere Dreiecksform

Um die verallgemeinerte reelle Schur-Form zu berechnen, besteht die erste Aufgabe darin, durch orthogonale Transformationen $A \in \mathbb{R}^{n\times n}$ in obere Hessenberg- und $B \in \mathbb{R}^{n\times n}$ in obere Dreiecksform zu bringen. Zunächst bestimmt man eine orthogonale Matrix U, sodass $U^T B$ obere Dreiecksform hat (QR-Zerlegung von B). Um die Eigenwerte zu erhalten, muss A in genau derselben Weise transformiert werden.
Wir werden das Vorgehen hier für den Fall $n = 4$ anschaulich darstellen, bevor wir den allgemeinen Algorithmus vorstellen.

$$A = U^T A = \begin{pmatrix} + & + & + & + \\ + & + & + & + \\ + & + & + & + \\ + & + & + & + \end{pmatrix}, \quad B = U^T B = \begin{pmatrix} + & + & + & + \\ & + & + & + \\ & & + & + \\ & & & + \end{pmatrix}$$

Um nun A in obere Hessnbergform zu überführen und die Struktur von B beizubehalten, multiplizieren wir zuerst beide mit einer Givens-Rotation Q_{34}^T von links

$$A = Q_{34}^T A = \begin{pmatrix} + & + & + & + \\ + & + & + & + \\ + & + & + & + \\ 0 & + & + & + \end{pmatrix}, \quad B = Q_{34}^T B = \begin{pmatrix} + & + & + & + \\ & + & + & + \\ & & + & + \\ & & * & + \end{pmatrix}$$

und danach mit einer Givens-Rotation Z_{34} von rechts.

$$A = AZ_{34} = \begin{pmatrix} + & + & + & + \\ + & + & + & + \\ + & + & + & + \\ 0 & + & + & + \end{pmatrix}, \quad B = BZ_{34} = \begin{pmatrix} + & + & + & + \\ & + & + & + \\ & & + & + \\ & & 0 & + \end{pmatrix}$$

Dieses Vorgehen führt man für die Einträge a_{31} und a_{42} fort:

$$A = Q_{23}^T A = \begin{pmatrix} + & + & + & + \\ + & + & + & + \\ 0 & + & + & + \\ 0 & + & + & + \end{pmatrix}, \quad B = Q_{23}^T B = \begin{pmatrix} + & + & + & + \\ & + & + & + \\ & * & + & + \\ & & & + \end{pmatrix}$$

$$A = AZ_{23} = \begin{pmatrix} + & + & + & + \\ + & + & + & + \\ 0 & + & + & + \\ 0 & + & + & + \end{pmatrix}, \quad B = BZ_{23} = \begin{pmatrix} + & + & + & + \\ & + & + & + \\ & 0 & + & + \\ & & & + \end{pmatrix}$$

$$A = \widetilde{Q}_{34}^T A = \begin{pmatrix} + & + & + & + \\ + & + & + & + \\ 0 & + & + & + \\ 0 & 0 & + & + \end{pmatrix}, \quad B = \widetilde{Q}_{34}^T B = \begin{pmatrix} + & + & + & + \\ & + & + & + \\ & & + & + \\ & & * & + \end{pmatrix}$$

$$A = A\widetilde{Z}_{34} = \underbrace{\begin{pmatrix} + & + & + & + \\ + & + & + & + \\ & + & + & + \\ & & + & + \end{pmatrix}}_{\text{obere Hessenbergmatrix}}, \quad B = B\widetilde{Z}_{34} = \underbrace{\begin{pmatrix} + & + & + & + \\ & + & + & + \\ & & + & + \\ & & 0 & + \end{pmatrix}}_{\text{obere Dreiecksmatrix}}$$

Im allgemeinen Fall $A, B \in \mathbb{R}^{n\times n}$ sieht der Algorithmus folgendermaßen aus.

Algorithmus 2.8.1: Reduktion auf Hessenberg-Dreicksform

Für $A, B \in \mathbb{R}^{n\times n}$ überschreibt dieser Algorithmus A mit der oberen Hessenbergmatrix $Q^T AZ$ und B mit der oberen Dreiecksmatrix $Q^T BZ$, wobei Q und Z orthogonal sind.

Verwende die QR-Zerlegung, um B mit $Q^T B = R$ zu überschreiben.
Hierbei ist Q orthogonal und R eine obere Dreiecksmatrix.

$A = Q^T A$
for $j = 1 : n-2$
  for $i = n : -1 : j+2$
    % Annulliere a_{ij}
    Bestimme $c = \cos(\theta)$ und $s = \sin(\theta)$,sodass
    $\begin{pmatrix} c & s \\ -s & c \end{pmatrix}^T \begin{pmatrix} a_{i-1,j} \\ a_{ij} \end{pmatrix} = \begin{pmatrix} * \\ 0 \end{pmatrix}$
    $A = G(j, i, \theta)^T A$
    $B = G(j, i, \theta)^T B$

```
        Bestimme c = cos(θ) und s = sin(θ) ,sodass
        (b_{i,i-1}  b_{ii}) ( c  s ) = (*  0)
                            (-s  c)
        A = A G(i-1, i, θ)
        B = B G(i-1, i, θ)
    end
end
```

2.8.2 Deflation

Bei der Beschreibung des QZ-Verfahrens können wir o.B.d.A. annehmen, dass A eine unreduzierte obere Hessenbergmatrix und B eine reguläre obere Dreiecksmatrix sind.
Der erste Fall ist klar, denn im Fall $a_{k+1,k} = 0$ für ein $k \in \{1, \dots, n-1\}$ gilt

$$A - \lambda B = \begin{pmatrix} A_{11} - \lambda B_{11} & A_{12} - \lambda B_{12} & k \\ 0 & A_{22} - \lambda B_{22} & n-k \\ k & n-k & \end{pmatrix}$$

und wir können uns auf die zwei kleineren Probleme $A_{11} - \lambda B_{11}$ und $A_{22} - \lambda B_{22}$ beschränken. Ist andererseits $b_{kk} = 0$, so ist es möglich, eine Null in dem Eintrag an Position $(n, n-1)$ in A zu erzeugen und so das Problem zu zerlegen. Wir zeigen dies an einem Beispiel für $n = 5, k = 3$.

$$A = \begin{pmatrix} + & + & + & + & + \\ + & + & + & + & + \\ & + & + & + & + \\ & & + & + & + \\ & & & + & + \end{pmatrix}, \quad B = \begin{pmatrix} + & + & + & + & + \\ & + & + & + & + \\ & & 0 & + & + \\ & & & + & + \\ & & & & + \end{pmatrix}$$

Die Null auf der Diagonalen von B kann an Position (n, n) verschoben werden

$$A = Q_{34}^T A = \begin{pmatrix} + & + & + & + & + \\ + & + & + & + & + \\ & + & + & + & + \\ & * & + & + & + \\ & & & + & + \end{pmatrix}, \quad B = Q_{34}^T B = \begin{pmatrix} + & + & + & + & + \\ & + & + & + & + \\ & & 0 & + & + \\ & & & 0 & + \\ & & & & + \end{pmatrix}$$

$$A = AZ_{23} = \begin{pmatrix} + & + & + & + & + \\ + & + & + & + & + \\ & + & + & + & + \\ & 0 & + & + & + \\ & & & + & + \end{pmatrix}, \quad B = BZ_{23} = \begin{pmatrix} + & + & + & + & + \\ & + & + & + & + \\ & & 0 & + & + \\ & & & 0 & + \\ & & & & + \end{pmatrix}$$

$$A = Q_{45}^T A = \begin{pmatrix} + & + & + & + & + \\ + & + & + & + & + \\ & + & + & + & + \\ & & + & + & + \\ & & * & + & + \end{pmatrix}, \quad B = Q_{45}^T B = \begin{pmatrix} + & + & + & + & + \\ & + & + & + & + \\ & & 0 & + & + \\ & & & 0 & + \\ & & & & 0 \end{pmatrix}$$

$$A = AZ_{34} = \begin{pmatrix} + & + & + & + & + & + \\ + & + & + & + & + & + \\ & & + & + & + & + \\ & & & + & + & + \\ & & & 0 & + & + \end{pmatrix}, \quad B = BZ_{34} = \begin{pmatrix} + & + & + & + & + & + \\ & + & + & + & + & \\ & & + & + & + & \\ & & & 0 & + & \\ & & & & 0 & \end{pmatrix}$$

$$A = AZ_{45} = \begin{pmatrix} + & + & + & + & + & + \\ + & + & + & + & + & + \\ & & + & + & + & + \\ & & & + & + & + \\ & & & & 0 & + \end{pmatrix}, \quad B = BZ_{45} = \begin{pmatrix} + & + & + & + & + & + \\ & + & + & + & + & \\ & & + & + & + & \\ & & & + & + & \\ & & & & 0 & \end{pmatrix}$$

Diese Technik ist allgemein und kann verwendet werden, um $a_{n-1,n} = 0$ zu erzielen, egal wo auf der Diagonalen von B eine Null steht.

2.8.3 QZ-Schritt

Die Idee des QZ-Schrittes liegt darin, auf das verallgemeinerte lineare Eigenwertproblem folgendes Update auszuführen:

$$(\bar{A} - \lambda\bar{B}) = \bar{Q}^T(A - \lambda B)\bar{Z},$$

wobei

- $\bar{A}$ wieder in oberer Hessenbergform,
- $\bar{B}$ wieder eine obere Dreiecksmatrix,
- $\bar{Q}$ und $\bar{Z}$ orthogonal sind.

$\bar{A}\bar{B}^{-1}$ soll im Wesentlichen die Matrix sein, die man nach einem Francis-QR-Schritt angewandt auf AB^{-1} erhalten würde.
Dafür definieren wir uns zunächst die (Hessenberg-) Matrix $M := AB^{-1}$ sowie die Skalare a und b, welche die Eigenwerte der 2×2 Matrix

$$\begin{pmatrix} m_{n-1,n-1} & m_{n-1,n} \\ m_{n,n-1} & m_{nn} \end{pmatrix}$$

sind. Außerdem ist

$$v := (M - aI)(M - bI)e_1 = \begin{pmatrix} * \\ * \\ * \\ 0 \\ \vdots \\ 0 \end{pmatrix} \in \mathbb{R}^n,$$

die erste Spalte von $(M - aI)(M - bI)$. Es sei $Q_1 \in \mathbb{R}^{n\times n}$ eine Householdermatrix, sodass

$$Q_1 v = \begin{pmatrix} * \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

ein Vielfaches von e_1 ist. Im Fall $n = 6$ folgt für A, B

$$A = Q_1 A = \begin{pmatrix} + & + & + & + & + & + \\ + & + & + & + & + & + \\ * & + & + & + & + & + \\ & & + & + & + & + \\ & & & + & + & + \\ & & & & + & + \end{pmatrix}, \quad B = Q_1 B = \begin{pmatrix} + & + & + & + & + & + \\ * & + & + & + & + & + \\ * & * & + & + & + & + \\ & & & + & + & + \\ & & & & + & + \\ & & & & & + \end{pmatrix}$$

Aufgabe ist es nun, A in obere Hessenbergform und B in obere Dreiecksform zurückzutransformieren. Dazu benötigt man zunächst ein Paar Householdermatrize Z_{11} und Z_{12}, sodass b_{31}, b_{32} und b_{21} annulliert werden.

$$A = AZ_{11} = \begin{pmatrix} + & + & + & + & + & + \\ + & + & + & + & + & + \\ * & + & + & + & + & + \\ * & * & + & + & + & + \\ & & & + & + & + \\ & & & & + & + \end{pmatrix}, \quad B = BZ_{11} = \begin{pmatrix} + & + & + & + & + & + \\ * & + & + & + & + & + \\ 0 & 0 & + & + & + & + \\ & & & + & + & + \\ & & & & + & + \\ & & & & & + \end{pmatrix}$$

$$A = AZ_{12} = \begin{pmatrix} + & + & + & + & + & + \\ + & + & + & + & + & + \\ * & + & + & + & + & + \\ * & * & + & + & + & + \\ & & & + & + & + \\ & & & & + & + \end{pmatrix}, \quad B = BZ_{12} = \begin{pmatrix} + & + & + & + & + & + \\ 0 & + & + & + & + & + \\ & & + & + & + & + \\ & & & + & + & + \\ & & & & + & + \\ & & & & & + \end{pmatrix}$$

Dann annulliere a_{31} und a_{41}.

$$A = Q_2A = \begin{pmatrix} + & + & + & + & + & + \\ + & + & + & + & + & + \\ 0 & + & + & + & + & + \\ 0 & * & + & + & + & + \\ & & & + & + & + \\ & & & & + & + \end{pmatrix}, \quad B = Q_2A = \begin{pmatrix} + & + & + & + & + & + \\ & + & + & + & + & + \\ & * & + & + & + & + \\ & * & * & + & + & + \\ & & & & + & + \\ & & & & & + \end{pmatrix}$$

Dies führt man so fort: Wiederherstellen der Dreiecksform von B, dann annullieren der Nicht-Hessenbergelemente in der nächsten Spalte von A.

$$A = AZ_{21}Z_{22} = \begin{pmatrix} + & + & + & + & + & + \\ + & + & + & + & + & + \\ & + & + & + & + & + \\ & * & + & + & + & + \\ & * & * & + & + & + \\ & & & & + & + \end{pmatrix}, \quad B = BZ_{21}Z_{22} = \begin{pmatrix} + & + & + & + & + & + \\ & + & + & + & + & + \\ & 0 & + & + & + & + \\ & 0 & 0 & + & + & + \\ & & & & + & + \\ & & & & & + \end{pmatrix}$$

$$A = Q_3A = \begin{pmatrix} + & + & + & + & + & + \\ + & + & + & + & + & + \\ & + & + & + & + & + \\ & 0 & + & + & + & + \\ & 0 & * & + & + & + \\ & & & & + & + \end{pmatrix}, \quad B = Q_3B = \begin{pmatrix} + & + & + & + & + & + \\ & + & + & + & + & + \\ & & + & + & + & + \\ & & * & + & + & + \\ & & * & * & + & + \\ & & & & & + \end{pmatrix}$$

$$A = AZ_{31}Z_{32} = \begin{pmatrix} + & + & + & + & + & + \\ + & + & + & + & + & + \\ & + & + & + & + & + \\ & & + & + & + & + \\ & & * & + & + & + \\ & & * & * & + & + \end{pmatrix}, \quad B = BZ_{31}Z_{32} = \begin{pmatrix} + & + & + & + & + & + \\ & + & + & + & + & + \\ & & + & + & + & + \\ & & 0 & + & + & + \\ & & 0 & 0 & + & + \\ & & & & & + \end{pmatrix}$$

$$A = Q_4A = \begin{pmatrix} + & + & + & + & + & + \\ + & + & + & + & + & + \\ & + & + & + & + & + \\ & & + & + & + & + \\ & & 0 & + & + & + \\ & & 0 & * & + & + \end{pmatrix}, \quad B = Q_4B = \begin{pmatrix} + & + & + & + & + & + \\ & + & + & + & + & + \\ & & + & + & + & + \\ & & & + & + & + \\ & & & * & + & + \\ & & & * & * & + \end{pmatrix}$$

$$A = AZ_{41}Z_{42} = \begin{pmatrix} + & + & + & + & + & + \\ + & + & + & + & + & + \\ & + & + & + & + & + \\ & & + & + & + & + \\ & & & + & + & + \\ & & & * & + & + \end{pmatrix}, \quad B = BZ_{41}Z_{42} = \begin{pmatrix} + & + & + & + & + & + \\ & + & + & + & + & + \\ & & + & + & + & + \\ & & & + & + & + \\ & & & 0 & + & + \\ & & & 0 & 0 & + \end{pmatrix}$$

$$A = Q_5 A = \begin{pmatrix} + & + & + & + & + & + & + \\ + & + & + & + & + & + & + \\ & + & + & + & + & + & + \\ & & + & + & + & + & \\ & & & + & + & + & \\ & & & 0 & + & + & \end{pmatrix}, \quad B = Q_5 B = \begin{pmatrix} + & + & + & + & + & + & + \\ & + & + & + & + & + & \\ & & + & + & + & + & \\ & & & + & + & + & \\ & & & & + & + & \\ & & & & * & + & \end{pmatrix}$$

$$A = A Z_5 = \begin{pmatrix} + & + & + & + & + & + & + \\ + & + & + & + & + & + & + \\ & + & + & + & + & + & + \\ & & + & + & + & + & \\ & & & + & + & + & \\ & & & & + & + & \end{pmatrix}, \quad B = B Z_5 = \begin{pmatrix} + & + & + & + & + & + & + \\ & + & + & + & + & + & \\ & & + & + & + & + & \\ & & & + & + & + & \\ & & & & + & + & \\ & & & & 0 & + & \end{pmatrix}$$

Man beachte, dass

$$Q = Q_1 \cdot \ldots \cdot Q_{n-1}$$

dieselbe erste Spalte wie Q_1 hat. Mit dem implizitem Q-Theorem 2.5.14 können wir schließen, dass $\bar{A}\bar{B}^{-1} = Q^T(AB^{-1})Q$ im Wesentlichen die gleiche Matrix ist, die man mit einem Francis-QR-Schritt erhalten würde.

Für allgemeines $n \in \mathbb{N}$ sieht der Algorithmus folgendermaßen aus.

Algorithmus 2.8.2: QZ-Schritt

Für eine unreduzierte obere Hessenbergmatrix $A \in \mathbb{R}^{n\times n}$ und eine reguläre obere Dreiecksmatrix $B \in \mathbb{R}^{n\times n}$ überschreibt dieser Algorithmus A mit einer oberen Hessenbergmatrix Q^TAZ und B mit der oberen Dreiecksmatrix Q^TBZ, wobei Q und Z orthogonal sind und Q dieselbe erste Spalte hat wie die orthogonale Ähnlichkeitstransformation, die man mit einem Francis QR-Schritt 2.5.4 angewandt auf AB^{-1} erhalten würde.

Setze $M = AB^{-1}$ und berechne $(M - aI)(M - bI)\, e_1 = (x, y, z, 0, \ldots, 0)^T$,
wobei a und b die Eigenwerte von $\left(\begin{smallmatrix} m_{n-1,n-1} & m_{n-1,n} \\ m_{n,n-1} & m_{nn} \end{smallmatrix}\right)$ sind.

for $k = 1 : n-2$

Bestimme eine Householdermatrix Q_k, sodass

$$Q_k \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} * \\ 0 \\ 0 \end{pmatrix}$$

$A = \operatorname{diag}(I_{k-1}, Q_k, I_{n-k-2})\, A$

$B = \operatorname{diag}(I_{k-1}, Q_k, I_{n-k-2})\, B$

Bestimme eine Householdermatrix Z_{k1}, sodass

$$\begin{pmatrix} b_{k+2,k} & b_{k+2,k+1} & b_{k+2,k+2} \end{pmatrix} Z_{k1} = \begin{pmatrix} 0 & 0 & * \end{pmatrix}$$

$A = A\, \operatorname{diag}(I_{k-1}, Z_{k1}, I_{n-k-2})$

$B = B\, \operatorname{diag}(I_{k-1}, Z_{k1}, I_{n-k-2})$

Bestimme eine Householdermatrix Z_{k2}, sodass

$$\begin{pmatrix} b_{k+1,k} & b_{k+1,k+1} \end{pmatrix} Z_{k2} = \begin{pmatrix} 0 & * \end{pmatrix}$$

$A = A\, \operatorname{diag}(I_{k-1}, Z_{k2}, I_{n-k-1})$

$B = B\, \operatorname{diag}(I_{k-1}, Z_{k2}, I_{n-k-1})$

$x = a_{k+1,k}$
$y = a_{k+2,k}$
if $k < n-2$
$z = a_{k+3,k}$
end
end
Bestimme eine Householdermatrix Q_{n-1}, sodass

$$Q_{n-1}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} * \\ 0 \end{pmatrix}$$

$A = \operatorname{diag}(I_{n-2}, Q_{n-1})\, A$
$B = \operatorname{diag}(I_{n-2}, Q_{n-1})\, B$
Bestimme eine Householdermatrix Z_{n-1}, sodass
$\begin{pmatrix} b_{n,n-1} & b_{nn} \end{pmatrix} Z_{n-1} = \begin{pmatrix} 0 & * \end{pmatrix}$
$A = A\ \operatorname{diag}(I_{n-2}, Z_{n-1})$
$B = B\ \operatorname{diag}(I_{n-2}, Z_{n-1})$

Bemerkung 2.8.9 *Algorithmus 2.8.2 benötigt $22n^2$ FLOPs. Q und Z können mit zusätzlich $8n^2$ bzw. $13n^2$ FLOPs bestimmt werden.*

2.8.4 Gesamtes QZ-Verfahren

Indem man mehrere QZ-Schritte nacheinander auf das „Hessenberg-Dreieck-Büschel" $A - \lambda B$ ausführt, ist es möglich, A in obere Quasi-Dreiecksform zu überführen. Dabei ist es wichtig, die Elemente in der unteren Nebendiagonalen von A und die Diagonalelemente von B zu beobachten, um eine mögliche Deflation auszumachen.
Das gesamte Verfahren ist in dem folgenden Algorithmus dargestellt.

Algorithmus 2.8.3: Gesamtes QZ-Verfahren

Für $A, B \in \mathbb{R}^{n\times n}$ bestimmt dieser Algorithmus orthogonale Matrizen Q und Z, sodass $Q^TAZ = T$ in oberer Quasi-Dreiecksform und $Q^TBZ = S$ in oberer Dreiecksform sind. A wird mit T und B mit S überschrieben.

1. Benutze Algorithmus 2.8.1, um A mit der oberen Hessenbergmatrix Q^TAZ und B mit der oberen Dreiecksmatrix Q^TBZ zu überschreiben.

2. **while** $q < n$
   Setze alle Elemente in der ersten unteren Nebendiagonalen $a_{i+1,i} = 0$, die

$$|a_{i+1,i}| \le \epsilon\,(|a_{ii}| + |a_{i+1,i+1}|) \quad i = 1, \ldots, n-1$$

   erfüllen.
   Finde größtes $q \ge 0$ und kleinstes $p \ge 0$, sodass

$$A = \begin{pmatrix} A_{11} & A_{12} & A_{13} \\ & A_{22} & A_{23} \\ & & A_{33} \end{pmatrix} \begin{matrix} p \\ n-p-q \\ q \end{matrix}$$
$$\begin{matrix} p & n-p-q & q \end{matrix}$$

wobei

- A_{11}: obere Hessenbergmatrix
- A_{22}: unreduzierte obere Hessenbergmatrix und
- A_{33}: obere Quasi-Dreiecksmatrix

sind. Teile B entsprechend auf:

$$B = \begin{pmatrix} B_{11} & B_{12} & B_{13} \\ & B_{22} & B_{23} \\ & & B_{33} \end{pmatrix} \begin{matrix} p \\ n-p-q \\ q \end{matrix}$$
$$\begin{matrix} p & n-p-q & q \end{matrix}$$

if $q < n$
  if B_{22} ist singulär
  Annulliere $a_{n-q,n-q-1}$
  else
  Wende einen QZ-Schritt 2.8.2 auf A_{22} und B_{22} an:

$$A = \operatorname{diag}(I_p, Q, I_q)^T A \operatorname{diag}(I_p, Q, I_q)$$
$$B = \operatorname{diag}(I_p, Q, I_q)^T B \operatorname{diag}(I_p, Q, I_q)$$

  end
 end
end

Bemerkung 2.8.10 *Algorithmus 2.8.3 benötigt $30n^3$ FLOPs. Für die Bestimmung von Q bzw. Z sind zusätzlich $16n^3$ bzw. $20n^3$ Rechenoperationen notwendig.*

2.9 Nichtlineare Eigenwertprobleme

Da die Eigenwertgleichung (2.11) linear im Eigenwert λ ist, spricht man auch von einem *linearen* Eigenwertproblem. In diesem Abschnitt werden wir nun allgemeinere *nichtlineare* Eigenwertgleichungen betrachten, die polynomiell in λ sind. Wir beginnen zunächst mit einem quadratischen Eigenwertproblem: Zu gegebenen Matrizen $M, C, K \in \mathbb{R}^{n\times n}$ bestimme man Eigenpaare $(\lambda, x) \in \mathbb{C} \times \mathbb{C}^n$, sodass

$$(\lambda^2 M + \lambda C + K)\, x = 0. \tag{2.29}$$

Bemerkung 2.9.1 (Vorsicht beim Rayleigh-Quotienten) *Ist (λ, x) ein Eigenpaar, welches das quadratische Eigenwertproblem (2.29) erfüllt, so gilt für den Rayleigh-Quotienten*

$$\lambda^2(\bar{x}^T M x) + \lambda(\bar{x}^T C x) + \bar{x}^T K x = 0.$$

Diese Formulierung liefert zu gegebenem Eigenvektor x einen „zweiten Eigenwert", der aber im Allgemeinen keiner ist!

Die Idee, um Problem (2.29) zu lösen, besteht darin, es auf ein linearen verallgemeinertes Eigenwertproblem zu transformieren. Setzt man

$$A = \begin{pmatrix} 0 & I_n \\ -K & -C \end{pmatrix}, \quad B = \begin{pmatrix} I_n & 0 \\ 0 & M \end{pmatrix}, \quad z = \begin{pmatrix} x \\ \lambda x \end{pmatrix},$$

so ist das Problem (2.29) äquivalent zur Bestimmung von $(\lambda, z) \in \mathbb{C} \times \mathbb{C}^{2n}$, sodass die allgemeine lineare Eigenwertgleichung

$$\begin{pmatrix} \lambda x \\ -Kx - C\lambda x \end{pmatrix} = Az = \lambda Bz = \lambda \begin{pmatrix} x \\ \lambda M x \end{pmatrix}$$

erfüllt ist.

Dieser „Trick“ funktioniert auch bei polynomialen Eigenwertproblemen.
Sei

$$\psi(\lambda) = \lambda^p C_p + \lambda^{p-1} C_{p-1} + \ldots + \lambda C_1 + C_0 \tag{2.30}$$

ein Polynom in λ vom Grad $p \in \mathbb{N}$ mit Matrix-Koeffizienten $C_j \in \mathbb{R}^{n \times n}$, wobei $C_p \neq 0$ vorausgesetzt wird.

Definition 2.9.2 *Das Polynom* (2.30) *heißt regulär, falls*

$$\det(\psi(\lambda)) \not\equiv 0.$$

Ansonsten ist ψ singulär.

Ist $\psi(\lambda)$ nun ein reguläres Polynom vom Grad p und sind $\lambda \in \mathbb{C}$ und $x \in \mathbb{C}^n$ mit

$$\psi(\lambda)\, x = 0$$

gesucht, so ist dieses Problem äquivalent zu dem allgemeinen linearen Eigenwertproblem

$$Az = \lambda Bz$$

der Größe pn, wobei

$$A = \begin{pmatrix} 0 & I_n & 0 & \cdots & \cdots & 0 \\ 0 & 0 & I_n & 0 & \cdots & 0 \\ \vdots & & & \ddots & & \vdots \\ \vdots & & & & \ddots & 0 \\ 0 & \cdots & \cdots & \cdots & 0 & I_n \\ -C_0 & -C_1 & -C_2 & \cdots & -C_{p-2} & -C_{p-1} \end{pmatrix} \in \mathbb{R}^{(np)\times(np)},$$

$$B = \begin{pmatrix} I_n & & & \\ & \ddots & & \\ & & I_n & \\ & & & C_p \end{pmatrix} \in \mathbb{R}^{(np)\times(np)} \quad \text{und}$$

$$z = \begin{pmatrix} x^T & \lambda x^T & \cdots & \lambda^{p-1} x^T \end{pmatrix}^T$$

gesetzt wird.

3 NICHTRESTRINGIERTE OPTIMIERUNG

Die Aufgabe, mit der wir uns im Folgenden beschäftigen werden, ist die Lösung von Minimierungsproblemen der Form

$$\text{minimiere } f(x) \text{ in } \mathbb{R}^n, \tag{3.1}$$

wobei $f : \mathbb{R}^n \to \mathbb{R}$ eine gegebene Zielfunktion ist. Dieses Problem heißt auch *nichtrestringiertes Optimierungsproblem*.
Es entspricht der Bestimmung der optimalen Verteilung von n konkurrierenden Ressourcen $x_1, \ldots, x_n$, die unbegrenzt verfügbar sind und einem speziellen Gesetz genügen. Sind die Ressourcen nicht unbegrenzt verfügbar, bedeutet dies für die mathematische Formulierung, dass das Minimum der Zielfunktion f innerhalb einer Teilmenge $\Omega \subset \mathbb{R}^n$ gesucht wird,

$$\text{minimiere } f(x) \text{ in } \Omega \subset \mathbb{R}^n. \tag{3.2}$$

In diesem Fall spricht man von einem *restringierten Optimierungsproblem*. Diese werden wir in Kapitel 4 untersuchen.

Kommen wir nun aber zunächst zu einem klassischen Beispiel für ein nichtrestringiertes Minimierungsproblem.

Beispiel 3.0.1 (Rosenbrock-Funktion) Bereits aus dem Kapitel zum Thema „Nichtlinearen Gleichungen“ in [Numerik II] kennen wir die Rosenbrock-Funktion

$$f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 \quad x = (x_1, x_2)^T \in \mathbb{R}^2.$$

Sie besitzt ein eindeutiges Minimum bei $x^* = (1, 1)^T$ und wird häufig als Maßstab für den Test numerischer Methoden für Minimierungsprobleme verwendet.

Bemerkung 3.0.2 *Es ist klar, dass sich Maximierungsaufgaben durch Vorzeichenwechsel in Minimierungsprobleme der From* (3.1) *umschreiben lassen.*

3.1 DIREKTE SUCHVERFAHREN

In diesem Abschnitt werden wir zwei direkte Verfahren zur Lösung von Problem vorstellen. Direkt bedeutet in diesem Kontext, dass nur die Stetigkeit, nicht aber die Differenzierbarkeit von f gefordert wird. Später werden wir mit den *Abstiegsverfahren* noch Methoden kennen lernen, die zudem die Berechnung von Werten der Ableitungen von f erfordern und im Allgemeinen ein besseres Konvergenzverhalten aufweisen.
Direkte Verfahren sind daher besonders für nicht differenzierbare Funktionen oder für Funktionen, deren Ableitungen unbekannt oder schwer auswertbar sind, geeignet. Außerdem bieten sich ableitungsfreie Methoden für stark oszillierende Funktionen f an.

3.1.1 Hooke-Jeeves-Verfahren (1960)

Das Verfahren von Hooke und Jeeves ist ein iteratives Verfahren zur Bestimmung einer Minimalstelle $x^* \in \mathbb{R}^n$ von f. Ausgehend von einer Näherung $x^{(k)} \in \mathbb{R}^n$ an x^* berechnet man eine bessere Näherung $x^{(k+1)}$ in zwei Schritten:

1. Erkundungsschritt
2. Fortschreitungsschritt.

Im **Erkundungsschritt** wird entlang der orthogonalen Koordinatenrichtungen um $x^{(k)}$ getestet, in welche Richtung die Werte von f kleiner werden. Hierzu seien $e_1, \ldots, e_n \in \mathbb{R}^n$ die kanonischen Basisvektoren des $\mathbb{R}^n$, $h = (h_1, \ldots, h_n)^T \in \mathbb{R}^n_+$ der Vektor der Schrittlängen $h_i > 0$ und $x^{(k)}$ eine Näherung an x^*.
Ausgehend von $x^{(k)}$ wird nun nacheinander in alle Richtungen getestet, ob

$$f(x^{(k)} + h_j e_j) < f(x^{(k)}), \quad j = 1, \ldots, n$$

erfüllt ist. Im Erfolgsfall wird die Näherung $x^{(k)}$ durch den Vektor $x^{(k)} + h_j e_j$ ersetzt. Andernfalls, d.h. wenn

$$f(x^{(k)} + h_j e_j) \geq f(x^{(k)})$$

gilt, überprüft man den Vektor $x^{(k)} - h_j e_j$. Falls

$$f(x^{(k)} - h_j e_j) < f(x^{(k)})$$

erfüllt ist, wird $x^{(k)}$ auf $x^{(k)} - h_j e_j$ gesetzt. Nach Test aller Richtungen, d.h. nach maximal $2n+1$ Funktionsauswertungen, erhalten wir so einen Vektor $y^{(k)}$ mit $f(y^{(k)}) \leq f(x^{(k)})$. Nun gibt es zwei Möglichkeiten:

i) $y^{(k)} = x^{(k)}$: In diesem Fall werden die Schrittlängen h_i halbiert,

$$h = \tfrac{1}{2} h$$

und ein weiterer Erkundungsschritt wird ausgehend von $x^{(k)}$ mit den neuen Schrittlängen durchgeführt.

ii) $y^{(k)} \neq x^{(k)}$: Mit dem Erkundungsschritt haben wir eine Richtung $y^{(k)} - x^{(k)}$ gefunden, in der der Funktionswert kleiner wird. Der **Fortschreitungsschritt** besteht nun darin, den Punkt $y^{(k)}$ weiter in Richtung $y^{(k)} - x^{(k)}$ fortzubewegen. Hierfür definieren wir den Vektor

$$w^{(k)} = 2y^{(k)} - x^{(k)} = x^{(k)} + 2\,(y^{(k)} - x^{(k)}).$$

Von $w^{(k)}$ ausgehend wird ein Erkundungsschritt in alle Richtungen ausgeführt. Ist dieser erfolgreich, d.h. führt er zu einem Vektor $z^{(k)}$ mit $f(z^{(k)}) < f(y^{(k)})$, so setzt man $x^{(k+1)} = z^{(k)}$. Andernfalls ist $x^{(k+1)} = y^{(k)}$.

Unter Berücksichtigung der Abbruchbedingung

$$\|x^{(k+1)} - x^{(k)}\|_\infty \leq \epsilon$$

für einen vorgegebenen Toleranzwert $\epsilon > 0$ sieht der gesamte Algorithmus des Hooke-Jeeves-Verfahrens folgendermaßen aus:

Algorithmus 3.1.1: Hooke-Jeeves-Verfahren

`Input:`
- $f \in \mathcal{C}(\mathbb{R}^n, \mathbb{R})$
- $x^{(0)} \in \mathbb{R}^n$ `Startvektor`
- $h = (h_1, \ldots, h_n)^T \in \mathbb{R}^n_+$ `Vektor der Schrittweiten`
- $\epsilon > 0$ `Toleranzwert`

$k = 0$
while $\|h\|_\infty > \epsilon$
    $y^{(k)} = \texttt{Erkunden}(f, x^{(k)}, h)$
    if $y^{(k)} = x^{(k)}$
        $h = \frac{1}{2}h$
    else
        $w^{(k)} = 2y^{(k)} - x^{(k)}$
        $z^{(k)} = \texttt{Erkunden}(f, w^{(k)}, h)$
        if $f(z^{(k)}) < f(y^{(k)})$
            $x^{(k+1)} = z^{(k)}$
        else
            $x^{(k+1)} = y^{(k)}$
        end
        $k = k + 1$
    end
end

Algorithmus 3.1.2: Erkunden

Input: $f \in \mathcal{C}(\mathbb{R}^n, \mathbb{R})$, $x \in \mathbb{R}^n$, $h \in \mathbb{R}^n_+$.

for $j = 1 : n$
    if $f(x + h_j e_j) < f(x)$
        $x = x + h_j e_j$
    else
        if $f(x - h_j e_j) < f(x)$
            $x = x - h_j e_j$
        end
    end
end

Bemerkung 3.1.1 *Die Schrittlängen $h_j > 0$ sollten so gewählt werden, dass die Größen*

$$|f(x^{(k)} \pm h_j e_j) - f(x^{(k)})|$$

für $j \in \{1, \ldots, n\}$ eine vergleichbare Größenordnung haben, damit eine bestimmte Genauigkeit erzielt werden kann.

3.1.2 Nelder-Mead-Verfahren (1965)

Die 1965 von Nelder und Mead vorgestellte Verfahren ist eine populäre, ebenfalls ableitungsfreie Methode zur Lösung von Minimierungsproblemen. Sie ist in Matlab in der Routine `fminsearch` realisiert und basiert auf folgender Idee: Es wird eine Folge von Simplexen erzeugt, die im Idealfall einen immer kleiner werdenden Durchmesser besitzen und sich immer mehr um das gesuchte Minimum herum anhäufen. Aus diesem Grund wird das Nelder-Mead-Verfahren häufig auch als Simplex-Verfahren bezeichnet. Es besteht aber kein Zusammenhang zu dem aus der Optimierung bekannten Simplex-Verfahren für lineare Programme.
In einem Iterationsschritt des Nelder-Mead-Verfahrens wird die Ecke des Simplex mit dem größten (schlechtesten) Funktionswert ersetzt durch einen Punkt mit einem besseren Wert. Dieser wird durch Spiegelung, durch Streckung oder durch Stauchung entlang einer Geraden durch die schlechteste Ecke gewonnen. Dies geschieht folgendermaßen: Die $n+1$ Ecken des aktuellen Simplex im $\mathbb{R}^n$ seien $\{x_1, \ldots, x_{n+1}\}$, wobei diese so sortiert seien, dass

$$f(x_1) \leq \ldots \leq f(x_{n+1})$$

gilt. Der Mittelwert der besten n Ecken wird mit

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

bezeichnet. Punkte entlang der Geraden durch $\bar{x}$ und x_{n+1} sind definiert durch

$$\bar{x}(t) = \bar{x} + t(x_{n+1} - \bar{x}), \quad t \in \mathbb{R}.$$

Ein Iterationsschritt des Nelder-Mead-Verfahrens ist dann durch den folgenden Algorithmus gegeben.

Algorithmus 3.1.3: Nelder-Mead-Iterationsschritt

```
Input: Spiegelungsfaktor α ≥ 0, Streckung β > 1, Stauchung γ ∈ (0,1).
Bestimme Spiegelungspunkt x̄(−α) und f_{−α} = f(x̄(−α))
if f(x_1) ≤ f_{−α} ≤ f(x_n)
    % Der neue Punkt liefert weder den besten
    % noch den schlechtesten Funktionswert
    Ersetze x_{n+1} durch x̄(−α)
elseif f_{−α} < f(x_1)
    % Der neue Punkt liefert den besten Funktionswert,
    % untersuche Punkte, die noch weiter in dieser Richtung liegen,
    % denn die Minimalstelle könnte außerhalb des Simplex liegen
    Bestimme x̄(−αβ) und f_{−αβ} = f(x̄(−αβ))
    if f_{−αβ} < f(x_1)
        Vergrößere das Simplex und ersetze x_{n+1} durch x̄(−αβ)
    else
        Verzichte auf die Expansion des Simplex und
        ersetze x_{n+1} durch x̄(−α)
    end
```

```
else    % f_{-α} > f(x_n)
    % Die Minimalstelle liegt voraussichtlich innerhalb des Simplex
    if f(x_n) < f_{-α} < f(x_{n+1})
        % äußere Kontraktion
        Bestimme x̄(-αγ) und f_{-αγ} = f(x̄(-αγ))
        if f_{-αγ} ≤ f_{-α}
            Ersetze x_{n+1} durch x̄(-αγ)
        else
            % Ziehe das Simplex um den besten Punkt x_1 zusammen
            Setze x_i = ½(x_1 + x_i)   i = 2,...,n+1
        end
    else   % f_{-α} ist der schlechteste Wert
        % innere Kontraktion
        Bestimme x̄(γ) und f_γ = f(x̄(γ))
        if f_γ < f(x_{n+1})
            Ersetze x_{n+1} durch x̄(γ)
        else
            % Ziehe das Simplex um den besten Punkt x_1 zusammen
            Setze x_i = ½(x_1 + x_i)   i = 2,...,n+1
        end
    end
end
```

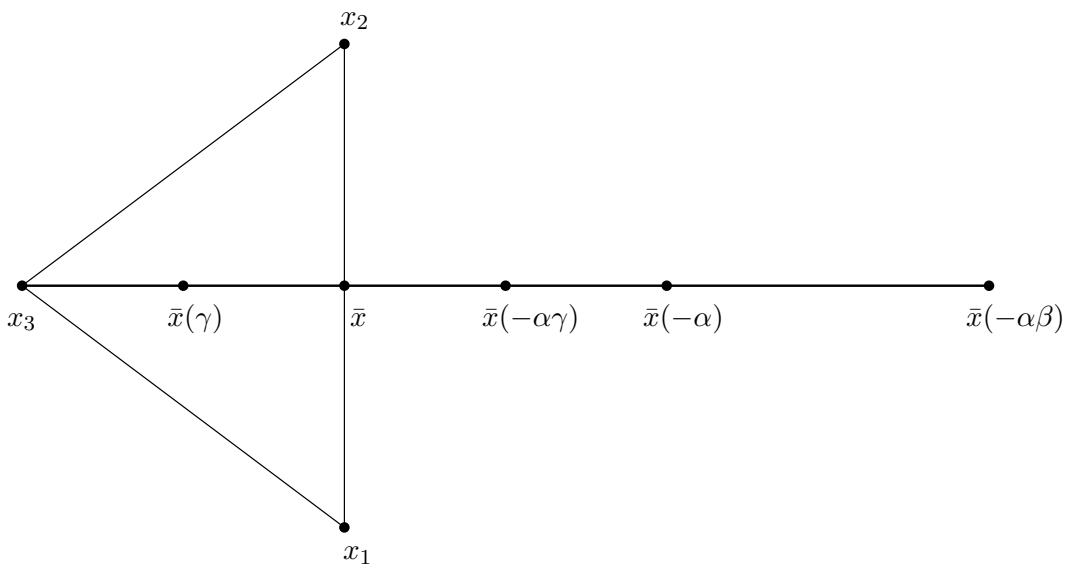


Abb. 3.1: Veranschaulichung der Nelder-Mead-Iteration

Bemerkungen 3.1.2 *i) Bei der Wahl der Parameter α, β und γ ist es in der Praxis üblich, die empirischen Werte $\alpha = 1, \beta = 2$ und $\gamma = 1/2$ zu verwenden.*

ii) Ein typisches Abbruchkriterium für das Nelder-Mead-Verfahren ist die Forderung, dass die Standardabweichung der Werte $f(x_1), \ldots, f(x_{n+1})$ von

$$\bar{f} = \frac{1}{n+1} \sum_{i=1}^{n+1} f(x_i)$$

kleiner als eine feste Toleranz $\epsilon > 0$ ist, d.h.

$$\frac{1}{n} \sum_{i=1}^{n+1} (f(x_i) - \bar{f})^2 < \epsilon \,.$$

3.2 ABSTIEGSVERFAHREN

Abstiegsverfahren bieten eine Möglichkeit, das Minimum des Optimierungsproblems (3.1) iterativ näherungsweise zu bestimmen. Jede Iteration eines Abstiegsverfahrens besteht aus zwei Schritten:

- bestimme Suchrichtung $p_k \in \mathbb{R}^n$,
- bestimme Schrittweite $\alpha_k > 0$.

Die Iterationsvorschrift ist dann gegeben durch

$$x_{k+1} = x_k + \alpha_k p_k, \quad k \in \mathbb{N}_0. \tag{3.3}$$

Die Güte eines Abstiegsverfahrens hängt von der Effektivität der Suche in Richtung p_k und der Schrittweite α_k ab. Unter der Annahme, dass f genügend glatt ist, liefert eine Taylor-Entwicklung von f

$$f(x + \alpha p) = f(x) + \alpha \nabla f(x)^T p + \varepsilon \,, \qquad \varepsilon \underset{\alpha \to 0}{\longrightarrow} 0.$$

Hieraus folgt die Bedingung an die Suchrichtung p_k

$$\begin{aligned} \nabla f(x_k)^T p_k < 0, &\quad \text{falls } \nabla f(x_k) \neq 0 \\ p_k = 0, &\quad \text{falls } \nabla f(x_k) = 0. \end{aligned}$$

Diese Eigenschaft gewährleistet, dass p_k eine Abstiegsrichtung ist, d.h. für hinreichend kleine $\alpha_k > 0$ gilt

$$f(x_k + \alpha_k p_k) < f(x_k).$$

Verschiedene Definitionen von p_k führen zu verschiedenen Abstiegsmethoden. Häufig wird die Suchrichtung

$$p_k = -B_k \nabla f(x_k)$$

gewählt, wobei B_k eine symmetrische und positiv definite Matrix ist. Typische Wahlmöglichkeiten für B_k bzw. p_k sind folgende:

- $B_k = I_n \Rightarrow p_k = -\nabla f(x_k)$: Gradientenverfahren
- $B_k^{-1} = D^2 f(x_k)$: Newton-Verfahren; $D^2 f(x)$ ist in einer hinreichend kleinen Umgebung von dem Minimum x^* positiv definit.
- $B_k^{-1} \approx D^2 f(x_k)$: Quasi-Newton Verfahren, bei denen die Hesse-Matrix $D^2 f(x_k)$ approximiert wird, z.B. das Broyden-Verfahren.

- $p_k = -\nabla f(x_k) + \beta_k p_{k-1}$: cg-Verfahren. Die Koeffizienten β_k werden dabei so gewählt, dass die Suchrichtungen $p_0, p_1, \ldots$ orthogonal zueinander sind.

Wie oben bereits erwähnt, liefert die Auswahl der Suchrichtung noch kein vollständiges Verfahren. Es muss auch die Schrittweite α_k in jeder Iteration definiert werden. Am besten wäre es, in jedem Schritt $k \in \mathbb{N}_0$ das globale Minimum von

$$\Phi_k(\alpha) := f(x_k + \alpha p_k) \tag{3.4}$$

zu bestimen. Dies ist allerdings meistens viel zu aufwändig und nur in wenigen Fällen kann das globale Minimum von Φ_k analytisch bestimmt werden. Ein Beispiel dafür sind quadratische Funktionen f.

Beispiel 3.2.1 $A \in \mathbb{R}^{n\times n}$ sei eine symmetrische, positiv definite Matrix, $b \in \mathbb{R}^n$ ein Vektor. Ist dann die Funktion $f : \mathbb{R}^n \to \mathbb{R}$ gegeben durch

$$f(x) = \tfrac{1}{2} x^T A x - b^T x$$

so minimiert

$$\alpha_k = -\frac{p_k^T (A x_k - b)}{p_k^T A p_k} = -\frac{p_k^T \nabla f(x_k)}{p_k^T A p_k}$$

die Funktion $\Phi_k(\alpha) = f(x_k + \alpha p_k)$.

Nachfolgend werden wir uns nun der Frage widmen, wie man α_k geeignet wählt, wenn f eine allgemeine nichtlineare Funktion ist.

3.3 SCHRITTWEITENSTEUERUNG

Bei der Steuerung der Schrittweite α ist zu berücksichtigen, dass sie einerseits nicht zu groß sein darf, damit ein Abstieg

$$f(x_k + \alpha_k p_k) < f(x_k) \tag{3.5}$$

gewährleistet ist, sie andererseits aber auch nicht zu klein sein darf, da sonst zu viele Iterationen nötig sind und das Verfahren ineffektiv wird. Wie Beispiel 3.3.1 zeigt, reicht darüberhinaus die Bedingung (3.5) im Allgemeinen nicht aus, um eine gegen das Minimum x^* konvergente Folge $(x_k)_{k\in\mathbb{N}_0}$ zu erzeugen. Bei der Schrittweitensteuerung werden hinreichende Bedingungen an die Schrittlänge gestellt und so ein Intervall der zulässigen Werte für α_k bestimmt.

Beispiel 3.3.1 Es sei $f : \mathbb{R} \to \mathbb{R}$ die konvexe Funktion

$$f(x) = (x-4)^2 - 1.$$

Dann besitzt f ein eindeutiges Minimum, $f(x^*) = -1$. Ist $(x_k)_{k\in\mathbb{N}_0}$ eine Folge, die

$$f(x_k) = \tfrac{1}{k}$$

erfüllt, so verringert sich zwar in jeder Iteration der Funktionswert, die Folge $(f(x_k))_{k\in\mathbb{N}_0}$ konvergiert aber gegen 0 anstatt gegen das Minimum -1.

Um einen hinreichenden Abstieg bzgl. der Funktion f zu gewährleisten, wird häufig folgende Anforderung an α_k gestellt, die sog. *Armijo-Bedingung*

$$f(x_k + \alpha_k p_k) \le f(x_k) + c_1 \alpha_k \nabla f(x_k)^T p_k \tag{3.6}$$

mit einer Konstanten $c_1 \in (0,1)$. Sie hat ihren Ursprung in der Taylor-Entwicklung

$$f(x_k + \alpha_k p_k) \approx f(x_k) + \underbrace{\alpha_k \nabla f(x_k)^T p_k}_{<0 \text{ gewählt}}$$

und kann mit der Bezeichnung aus Gleichung (3.4) auch folgendermaßen formuliert werden

$$\Phi_k(\alpha_k) \leq \Phi_k(0) + c_1 \alpha_k \Phi_k'(0).$$

Da $c_1 \in (0,1)$ und $\nabla f(x_k)^T p_k < 0$ gilt, ist Bedingung (3.6) für alle hinreichend kleinen Werte $\alpha_k > 0$ erfüllt. Daher ist die Armijo-Bedingung nicht ausreichend, um Effektivität des Verfahrens zu gewährleisten. Damit inakzeptabel kleine Schrittweiten vermieden werden, ist eine zweite Bedingung an α_k notwendig, z.B. die sog. *Wölbungsbedingung*

$$\nabla f(x_k + \alpha_k p_k)^T p_k \geq c_2 \nabla f(x_k)^T p_k \tag{3.7}$$

mit einer Konstanten $c_2 \in (c_1, 1)$, c_1 aus Bedingung (3.6). Wiederum kann diese Ungleichung auch als eine Anforderung an Φ_k formuliert werden,

$$\Phi_k'(\alpha_k) \geq c_2 \Phi_k'(0).$$

Die Wölbungsbedingung sichert also, dass die Steigung von Φ_k in α_k größer ist als der Faktor c_2 multipliziert mit der ursprünglichen Steigung $\Phi_k'(0)$. Dies entspricht der Tatsache, dass $f(x_{k+1})$ für größere Werte von α_k kleiner wird, falls die Steigung $\Phi_k'(\alpha_k)$ negativ ist.

Bemerkung 3.3.2 *In der Praxis wird c_1 meist relativ klein gewählt, etwa in der Größenordnung $c_1 = 10^{-4}$, für c_2 ist 0.9 ein üblicher Wert.*

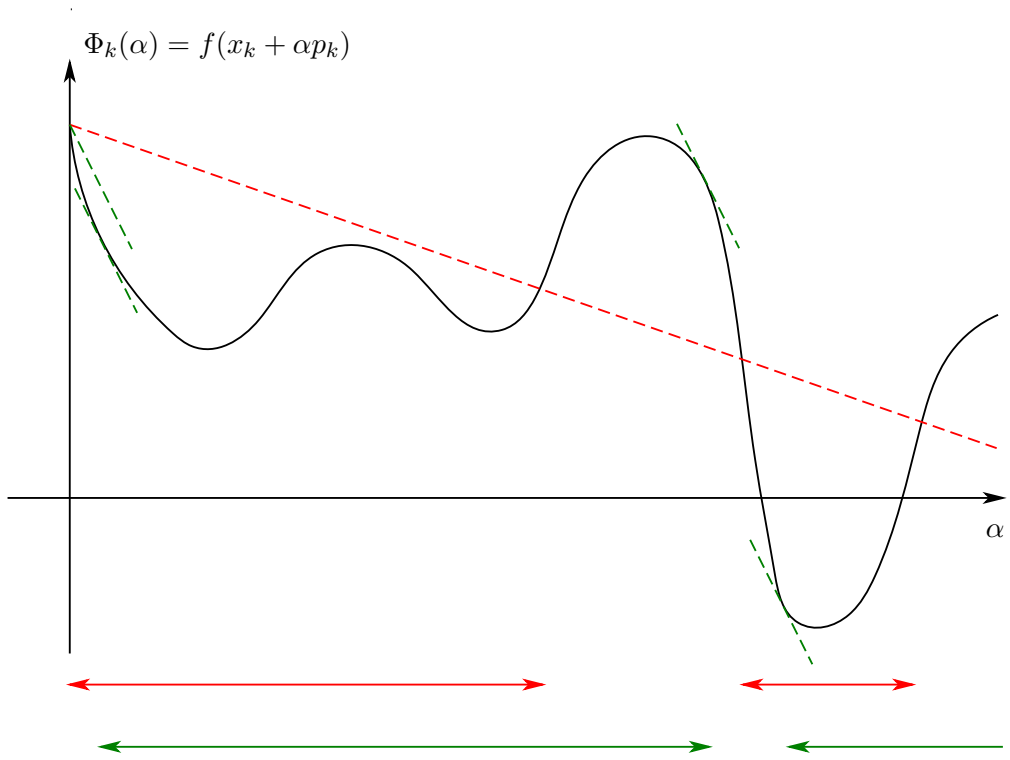


Abb. 3.2: Zulässiger Bereich gemäß der ersten (rot) und zweiten (grün) Wolfe-Bedingung

Die Bedingung von Armijo (3.6) und die Wölbungsbedingung (3.7) sind auch zusammen unter dem Namen *Wolfe-Bedingungen* bekannt und in Abbildung 3.2 anschaulich dargestellt. Diese Bedingungen sind allerdings immer noch relativ schwach und lassen auch Werte für α_k zu, sodass x_{k+1} noch sehr weit weg von der Minimalstelle x^* liegt. Um dies zu vermeiden, kann man diejenigen Werte für α_k ausschließen, deren Ableitung betragsmäßig zu groß ist, und die somit zu weit entfernt von lokalen und globalen Minima sind. Hierfür ersetzt man die Wölbungsbedingung (3.7) durch

$$|\nabla f(x_k + \alpha_k p_k)^T p_k| \leq c_2 |\nabla f(x_k)^T p_k|. \tag{3.8}$$

Die Bedingung von Armijo (3.6) zusammen mit der Bedingung (3.8) für $0 < c_1 < c_2 < 1$ werden auch *starke Wolfe-Bedingungen* genannt.
Es ist nicht schwer zu beweisen (siehe z.B. [Nocedal/Wright, Lemma 3.1]), dass es Schrittweiten gibt, die die Wolfe- bzw. die starken Wolfe-Bedingungen erfüllen.

Lemma 3.3.3 *$f : \mathbb{R}^n \to \mathbb{R}$ sei stetig differenzierbar und $p_k \in \mathbb{R}^n$ sei eine Abstiegsrichtung in dem Sinne, dass*

$$\nabla f(x_k)^T p_k < 0$$

gilt. Außerdem sei f auf der Menge $\{x_k + \alpha p_k \,|\, \alpha > 0\}$ nach unten beschränkt. Dann existieren für $0 < c_1 < c_2 < 1$ Intervalle für α_k, sodass die Wolfe-Bedingungen (3.6) *und* (3.7) *bzw. die starken Wolfe-Bedingungen* (3.6) *und* (3.8) *erfüllt sind.*

Neben den Wolfe-Bedingungen gibt es noch einige weitere Methoden, die Schrittlänge α_k zu bestimmen. Wir erwähnen an dieser Stelle noch zwei Bedingungen:

i) die *Goldstein-Bedingungen*

$$f(x_k) + (1-c)\alpha_k \nabla f(x_k)^T p_k \leq f(x_k + \alpha_k p_k) \leq f(x_k) + c\alpha_k \nabla f(x_k)^T p_k \tag{3.9}$$

mit einer Konstanten $c \in (0, 1/2)$. Die zweite Ungleichung ist dabei genau die Bedingung von Armijo (3.6); die erste soll sichern, dass die Schrittlänge nicht zu klein ist. Problematisch hierbei ist allerdings, dass dadurch Minimalstellen ausgeschlossen werden können.

ii) der Rücksetzungsalgorithmus. Dieser startet mit einer relativ großen Schrittweite $\alpha_k^{(0)}$ und verkleinert diese solange, bis die Armijo-Bedingung (3.6) erfüllt ist.

Algorithmus 3.3.1: Rücksetzungsalgorithmus zur Schrittweitensteuerung

Input: $\alpha_k^{(0)} > 0$, $\rho \in (0,1)$, $c \in (0,1)$.

Setze $\alpha_k = \alpha_k^{(0)}$
while $f(x_k + \alpha_k p_k) > f(x_k) + c\alpha_k \nabla f(x_k)^T p_k$
    Setze $\alpha_k = \rho \alpha_k$
end

In der Praxis ist der Rücksetzungsalgorithmus vor allem für das Newton-Verfahren geeignet, nicht aber für Quasi-Newton-Verfahren. Der Faktor $\rho \in (0,1)$ wird auch häufig pro Iteration $k \in \mathbb{N}_0$ angepasst.

Für Abstiegsverfahren mit Schrittweitensteuerung über die Wolfe-Bedingungen (3.6) und (3.7) kann man aus dem folgenden Satz von Zoutendijk eine Konvergenzaussage folgern.

Satz 3.3.4 (Zoutendijk) *Es sei $(x_k)_{k\in\mathbb{N}_0} \subset \mathbb{R}^n$ eine Folge der Form*

$$x_{k+1} = x_k + \alpha_k p_k, \qquad k \in \mathbb{N}_0,$$

wobei p_k Abstiegsrichtungen seien, d.h.

$$\nabla f(x_k)^T p_k < 0 \qquad \forall\, k \in \mathbb{N}_0,$$

und die Schrittweiten α_k die Wolfe-Bedingungen (3.6) *und* (3.7) *erfüllen. $f : \mathbb{R}^n \to \mathbb{R}$ sei nach unten beschränkt auf ganz $\mathbb{R}^n$ und stetig differenzierbar auf einer offenen Menge $\mathcal{N}$, welche die Levelset-Menge $\mathcal{L} := \{x \in \mathbb{R}^n | f(x) \le f(x_0)\}$ enthält, wobei x_0 der Startvektor sei. Außerdem sei der Gradient ∇f auf $\mathcal{N}$ Lipschitz-stetig, d.h. es existiert ein $L > 0$, sodass*

$$\|\nabla f(x) - \nabla f(y)\| \le L\|x - y\| \qquad \forall x, y \in \mathcal{N}.$$

Dann ist die Zoutendijk-Bedingung

$$\sum_{k=0}^{\infty} \cos^2\theta_k \|\nabla f(x_k)\|^2 < \infty \tag{3.10}$$

erfüllt, wobei

$$\cos\theta_k = \frac{-\nabla f(x_k)^T p_k}{\|\nabla f(x_k)\|\|p_k\|}.$$

Beweis. Siehe [Nocedal/Wright, Theorem 3.2] □

Die Zoutendijk-Bedingung (3.10) impliziert

$$\cos^2\theta_k \|\nabla f(x_k)\|^2 \longrightarrow 0 \qquad \text{für } k \to \infty.$$

Daraus ergibt sich ein Konvergenzresultat, d.h. $\|\nabla f(x_k)\| \to 0$ $(k \to \infty)$, falls der Winkel θ_k zwischen der Suchrichtung p_k und der Richtung des steilsten Abstiegs $-\nabla f(x_k)$ von 90° weg beschränkt ist, also ein $\delta > 0$ existiert mit

$$\cos\theta_k \ge \delta > 0 \qquad \forall k \in \mathbb{N}_0.$$

Bemerkung 3.3.5 *Ähnlich zu Satz 3.3.4 kann man zeigen, dass die Zoutendijk-Bedingung* (3.10) *ebenfalls erfüllt ist, wenn die Schrittweiten $(\alpha_k)_{k\in\mathbb{N}_0}$ nicht die Wolfe-Bedingungen* (3.6) *und* (3.7)*, sondern die starken Wolfe-Bedingungen* (3.6) *und* (3.8) *oder die Goldstein-Bedingungen* (3.9) *erfüllen.*

3.4 Algorithmen zur Auswahl der Schrittweite

Nachdem wir uns im letzten Abschnitt mit der Bestimmung von Intervallen zulässiger Schrittweiten beschäftigt haben, werden wir nun konkrete Verfahren zur Auswahl der Schrittweite α_k vorstellen. Man geht hierbei in zwei Stufen vor:

1. *Einschachtelungsphase*: Intervallbestimmung der zulässigen Werte,
2. *Bisektions-* oder *Interpolationsphase*.

Ausgehend von der Funktion Φ_k, difiniert wie in (3.4), und von einer gegebenen Anfangsschrittweite $\alpha_k^{(0)}$ überprüft man, ob diese die Armijo-Bedingung (3.6) erfüllt. Ist dies der Fall, so terminiert man die Suche mit $\alpha_k = \alpha_k^{(0)}$. Andernfalls liegt in $[0, \alpha_k^{(0)}]$ eine zulässige Schrittweite.
Die Idee des nachfolgenden Algorithmus besteht nun darin, anstatt die Minimalstelle von Φ_k exakt zu bestimmen, diese Funktion durch Polynome anzunähern und von diesen die Minimalstellen zu bestimmen. Dabei soll nicht in jeder Iteration der Wert der Ableitung Φ_k' an gewissen Stellen bestimmt werden, da dies häufig sehr aufwändig ist. Man verwendet lediglich $\Phi_k'(0) = \nabla f(x_k)^T p_k$ sowie Funktionswerte von Φ_k. Aus den Daten $\Phi_k(0)$, $\Phi_k'(0)$ und $\Phi_k(\alpha_k^{(0)})$ bestimmt man das quadratisches Polynom, welches die Interpolationsbedingungen

$$\begin{aligned}\Phi_{k,1}(0) &= \Phi_k(0),\\ \Phi_{k,1}'(0) &= \Phi_k'(0),\\ \Phi_{k,1}(\alpha_k^{(0)}) &= \Phi_k(\alpha_k^{(0)})\end{aligned}$$

erfüllt, sowie dessen Minimalstelle $\alpha_k^{(1)}$. Erfüllt $\alpha_k^{(1)}$ die Armijo-Bedingung (3.6), so beendet man die Suche und setzt $\alpha_k = \alpha_k^{(1)}$. Andernfalls bestimmt man aus den Daten $\Phi_k(0)$, $\Phi_k'(0)$, $\Phi_k(\alpha_k^{(0)})$ und $\Phi_k(\alpha_k^{(1)})$ ein kubisches Polynom $\Phi_{k,2}$, welches den Bedingungen

$$\begin{aligned}\Phi_{k,2}(0) &= \Phi_k(0),\\ \Phi_{k,2}'(0) &= \Phi_k'(0)\\ \Phi_{k,2}(\alpha_k^{(0)}) &= \Phi_k(\alpha_k^{(0)}),\\ \Phi_{k,2}(\alpha_k^{(1)}) &= \Phi_k(\alpha_k^{(1)})\end{aligned}$$

genügt. Erfüllt die Minimalstelle $\alpha_k^{(2)}$ von $\Phi_{k,2}$ die Armijo-Bedingung (3.6), so setzt man $\alpha_k = \alpha_k^{(2)}$. Ansonsten führt man die kubische Interpolation mit den Interpolationsdaten $\Phi_k(0)$, $\Phi_k'(0)$, $\Phi_k(\alpha_k^{(1)})$ und $\Phi_k(\alpha_k^{(2)})$ erneut durch usw. Sind bei dieser Iteration die Elemente $\alpha_k^{(j)}$ und $\alpha_k^{(j-1)}$ zu dicht beieinander oder ist die Differenz zwischen den beiden zu groß, so ersetzt man $\alpha_k^{(j)}$ durch $\alpha_k^{(j)} = \alpha_k^{(j-1)}/2$.

Der Algorithmus sieht dann folgendermaßen aus.

Algorithmus 3.4.1: Schrittweitenbestimmung nach Armijo

`Input:` $\alpha_k^{(0)} > 0$`,` Φ_k`,` $\delta_{\min}, \delta_{\max} > 0$`.`

`Berechne` $\Phi_k(0)$, $\Phi_k'(0)$, $\Phi_k(\alpha_k^{(0)})$
`if` $\Phi_k(\alpha_k^{(0)}) \le \Phi_k(0) + c_1\alpha_k^{(0)}\Phi_k'(0)$ `% Armijo-Bedingung`
    $\alpha_k = \alpha_k^{(0)}$
    `return`
`else`
    `% Bestimme quadratische Int.-Polynom und dessen Minimalstelle`
    $$\Phi_{k,1}(\alpha) = \left(\frac{\Phi_k(\alpha_k^{(0)}) - \Phi_k(0) - \alpha_k^{(0)}\Phi_k'(0)}{(\alpha_k^{(0)})^2}\right)\alpha^2 + \alpha\Phi_k'(0) + \Phi_k(0)$$

$$\alpha_k^{(1)} = -\frac{\Phi_k'(0)(\alpha_k^{(0)})^2}{2(\Phi_k(\alpha_k^{(0)}) - \Phi_k(0) - \Phi_k'(0)\alpha_k^{(0)})} \in [0, \alpha_k^{(0)}]$$

Berechne $\Phi_k(\alpha_k^{(1)})$
if $\alpha_k^{(0)} - \alpha_k^{(1)} < \delta_{\min}$ **or** $\alpha_k^{(0)} - \alpha_k^{(1)} > \delta_{\max}$
% Sind die Schrittweite oder der Unterschied zu klein?
$\alpha_k^{(1)} = \frac{\alpha_k^{(j)}}{2}$
end
if $\Phi_k(\alpha_k^{(1)}) \leq \Phi_k(0) + c_1\alpha_k^{(1)}\Phi_k'(0)$ % Armijo-Bedingung
$\alpha_k = \alpha_k^{(1)}$
return
else
for $j = 1, 2, \ldots$
% Bestimme kubisches Int.-Polynom und dessen Minimalstelle

$$\gamma_{j+1} = \frac{1}{(\alpha_k^{(j-1)}\alpha_k^{(j)})^2(\alpha_k^{(j)} - \alpha_k^{(j-1)})}$$

$$\begin{pmatrix} a_{j+1} \\ b_{j+1} \end{pmatrix} = \gamma_{j+1} \begin{pmatrix} (\alpha_k^{(j-1)})^2 & -(\alpha_k^{(j)})^2 \\ -(\alpha_k^{(j-1)})^3 & (\alpha_k^{(j)})^3 \end{pmatrix} \begin{pmatrix} \Phi_k(\alpha_k^{(j)}) - \Phi_k(0) - \Phi_k'(0)\alpha_k^{(j)} \\ \Phi_k(\alpha_k^{(j-1)}) - \Phi_k(0) - \Phi_k'(0)\alpha_k^{(j-1)} \end{pmatrix}$$

$$\Phi_{k,j+1}(\alpha) = a_{j+1}\alpha^3 + b_{j+1}\alpha^2 + \alpha\Phi_k'(0) + \Phi_k(0)$$

$$\alpha_k^{(j+1)} = \frac{-b_{j+1} + \sqrt{b_{j+1}^2 - 3a_{j+1}\Phi_k'(0)}}{3a_{j+1}}$$

Berechne $\Phi_k(\alpha_k^{(j+1)})$
if $\alpha_k^{(j)} - \alpha_k^{(j+1)} < \delta_{\min}$ **or** $\alpha_k^{(j)} - \alpha_k^{(j+1)} > \delta_{\max}$
$\alpha_k^{(j+1)} = \frac{\alpha_k^{(j)}}{2}$
end
if $\Phi_k(\alpha_k^{(j+1)}) \leq \Phi_k(0) + c_1\alpha_k^{(j+1)}\Phi_k'(0)$ % Armijo-Bedingung
$\alpha_k = \alpha_k^{(j+1)}$
return
end
end
end
end

Bemerkungen 3.4.1 *i) Algorithmus 3.4.1 testet in jeder Iteration nur die Armijo-Bedingung* (3.6)*. Dies hat den Vorteil, dass neben $\Phi_k'(0)$ keine weiteren Werte der Ableitungen berechnet werden müssen. Allerdings weiß man deshalb nicht, ob die Wolfe- oder gar die starken Wolfe-Bedingungen erfüllt sind.*

ii) Die Anfangsschrittweite wird beim Newton- oder bei Quasi-Newton-Verfahren normalerweise auf 1 gesetzt. Beim Gradienten- oder cg-Verfahren wird ein übliches Vorgehen zur Bestimmung der Anfangsschrittweite $\alpha_k^{(0)}$ aus die Annahme gewonnen, dass die Veränderung von dem Funktionswert $f(x_k)$ gegenüber $f(x_{k-1})$ bzgl. der ersten Ordnung von α_k bzw. α_{k-1} unegfähr gleich ist, d.h.

$$\alpha_k \nabla f(x_k)^T p_k \approx \alpha_{k-1} \nabla f(x_{k-1})^T p_{k-1}.$$

Daher setzt man

$$\alpha_k^{(0)} = \alpha_{k-1} \frac{\nabla f(x_{k-1})^T p_{k-1}}{\nabla f(x_k)^T p_k}.$$

Wie in der obigen Bemerkungen bereits erwähnt, gewährleistet Algorithmus 3.4.1 nicht, dass die (starken) Wolfe-Bedingungen erfüllt sind. Daher führen wir hier einen zweiten Algorithmus an, der Schrittweiten zurückgibt, welche den starken Wolfe-Bedingungen genügen. Dieser nutzt die Tatsache aus, dass in dem Intervall $[\alpha_k^{(j-1)}, \alpha_k^{(j)}]$ Schrittlängen liegen, die die starken Wolfe-Bedingungen erfüllen, falls einer der folgenden drei Fälle erfüllt ist:

1. $\alpha_k^{(j)}$ verletzt die Armijo-Bedingung (3.6),
2. $\Phi_k(\alpha_k^{(j)}) \geq \Phi_k(\alpha_k^{(j-1)})$,
3. $\Phi_k'(\alpha_k^{(j)}) \geq 0$.

Testet man diese neben den starken Wolfe-Bedingungen in einer geschickten Reihenfolge, sodass die Bedingungen, welche die Ableitung von Φ_k erfordern, zuletzt getestet werden, so ergibt sich der nachfolgende Algorithmus.

Algorithmus 3.4.2: Schrittweitenbestimmung über starke Wolfe-Bedingungen

`Input:` $\alpha_{\max} > 0$, $\alpha_k^{(1)} \in (0, \alpha_{\max})$.

`Setze` $\alpha_k^{(0)} = 0$
for $j = 1, 2, \ldots$
  `Berechne` $\Phi_k(\alpha_k^{(1)})$
  if $\Phi_k(\alpha_k^{(j)}) > \Phi_k(0) + c_1 \alpha_k^{(j)} \Phi_k'(0)$ **or** $[j > 1$ **and** $\Phi_k(\alpha_k^{(j)}) \geq \Phi_k(\alpha_k^{(j-1)})]$
    $\alpha_k =$ **Interpolation**$(\alpha_k^{(j-1)}, \alpha_k^{(j)})$
    return
  end
  `Berechne` $\Phi_k'(\alpha_k^{(j)})$
  if $|\Phi_k'(\alpha_k^{(j)})| \leq -c_2 \Phi_k'(0)$
    $\alpha_k = \alpha_k^{(j)}$
    return
  end
  if $\Phi_k'(\alpha_k^{(j)}) \geq 0$
    $\alpha_k =$ **Interpolation**$(\alpha_k^{(j)}, \alpha_k^{(j-1)})$
    return
  end
  `Wähle` $\alpha_k^{(j+1)} \in (\alpha_k^{(j)}, \alpha_{\max})$
end

Der hierbei aufgerufene Algorithmus `Interpolation` führt die Ermittlung einer Schrittweite $\alpha_k^{(i)}$ durch Interpolation und bzw. Bisektion so lange durch, bis eine Schrittweite gefunden ist, die die starken Wolfe-Bedingungen erfüllt. Hierbei ist zu beachten, dass in dem Funktionsaufruf `Interpolation`$(\alpha_k^{(\text{low})}, \alpha_k^{(\text{high})})$ im Allgemeinen nicht $\alpha_k^{(\text{low})} \leq \alpha_k^{(\text{high})}$ gilt, sondern $\alpha_k^{(\text{low})}$ und $\alpha_k^{(\text{high})}$ durch folgende Eigenschaften charakterisiert sind:

1. das Intervall, welches durch die Werte $\alpha_k^{(\text{low})}$ und $\alpha_k^{(\text{high})}$ begrenzt wid, enthält Schrittweiten, welche den starken Wolfe-Bedingungen genügen,
2. $\alpha_k^{(\text{low})}$ ist von allen bisher generierten Schrittlängen, die die Armijo-Bedingung (3.6) erfüllen, diejenige mit dem kleinsten Funktionswert bzgl. Φ_k.
3. $\alpha_k^{(\text{high})}$ wird so gewählt, dass

$$\Phi_k'(\alpha_k^{(\text{low})})(\alpha_k^{(\text{high})} - \alpha_k^{(\text{low})}) < 0$$

gilt.

Algorithmus 3.4.3: Interpolation

Input: $\alpha_k^{(\text{low})}, \alpha_k^{(\text{high})} > 0$.

for $i = 1, 2, \ldots$
  Bestimme mittels quadratischer oder kubischer Interpolation
  oder Bisektion ein $\alpha_k^{(i)}$ zwischen $\alpha_k^{(\text{low})}$ und $\alpha_k^{(\text{high})}$
  Berechne $\Phi_k(\alpha_k^{(i)})$
  if $\Phi_k(\alpha_k^{(i)}) > \Phi_k(0) + c_1\alpha_k^{(i)}\Phi_k'(0)$ **or** $\Phi_k(\alpha_k^{(i)}) \geq \Phi_k(\alpha_k^{(\text{low})})$
    $\alpha_k^{(\text{high})} = \alpha_k^{(i)}$
  else
    Berechne $\Phi_k'(\alpha_k^{(i)})$
    if $|\Phi_k'(\alpha_k^{(i)})| \leq -c_2\Phi_k'(0)$
      $\alpha_k = \alpha_k^{(i)}$
      return
    end
    if $\Phi_k'(\alpha_k^{(i)})(\alpha_k^{(\text{high})} - \alpha_k^{(\text{low})}) \geq 0$
      $\alpha_k^{(\text{high})} = \alpha_k^{(\text{low})}$
    end
    $\alpha_k^{(\text{low})} = \alpha_k^{(i)}$
  end
end

3.5 NICHTLINEARES CG-VERFAHREN

Nachdem wir uns nun ausführlich mit der Bestimmung der Schrittweite α_k beschäftigt haben, möchten wir uns in diesem Abschnitt der Ermittlung der Suchrichtung p_k nach dem cg-Verfahren widmen. Aus [Numerik I, Kapitel 4] ist bereits das (lineare) cg-Verfahren bekannt, mit dem iterativ die Lösung eines linearen Gleichungssystems $Ax = b$ für eine symmetrische, positiv definite Matrix A approximiert werden kann. Man fasst hierbei die Lösung x^* als Minimalstelle der quadratischen Funktion $\frac{1}{2}x^TAx - b^Tx$ auf. Um die Ähnlichkeit des nichtlinearen cg-Verfahrens mit dem linearen zu zeigen, führen wir - allerdings ohne weitere Erläuterungen - nochmals das cg-Verfahren für den linearen Fall an.

Algorithmus 3.5.1: Lineares cg-Verfahren

Input: $x_0 \in \mathbb{R}^n$, $A \in \mathbb{R}^{n\times n}$.

$r_0 = Ax_0 - b$
$p_0 = -r_0$
$k = 0$
while $\|r_k\| \neq 0$
  $\alpha_k = \dfrac{r_k^T r_k}{p_k^T A p_k}$
  $x_{k+1} = x_k + \alpha_k p_k$
  $r_{k+1} = Ax_{k+1} - b = r_k + \alpha_k A p_k$
  $\beta_{k+1} = \dfrac{r_{k+1}^T r_{k+1}}{r_k^T r_k}$
  $p_{k+1} = -r_{k+1} + \beta_{k+1} p_k$
  $k = k + 1$
end

Algorithmus 3.5.1 kann man nun leicht zur Minimierung einer Funktion f von dem Fall

$$f(x) = x^TAx - b^Tx$$

auf allgemeine glatte Funktionen $f : \mathbb{R}^n \to \mathbb{R}$ übertragen. Dabei müssen zwei Punkte beachtet werden:

1. Die Schrittweite α_k wird nun über einen der speziellen Algorithmen zur Schrittweitenbestimmung ermittelt, z.B. mittels Algorithmus 3.3.1, 3.4.1 oder 3.4.2.

2. Das Residuum r_k, welches nichts anderes als der Gradient von f ausgewertet an der Stelle x_k ist, wird nun durch den Wert des Gradienten der zu untersuchenden nichtlinearen Funktion f an der Stelle x_k ersetzt.

Dies führt zu folgendem Vorgehen, dem nichtlinearen cg-Verfahren nach Fletcher-Reeves.

Algorithmus 3.5.2: Nichtlineares cg-Verfahren nach Fletcher-Reeves

Input: $x_0 \in \mathbb{R}^n$, $f : \mathbb{R}^n \to \mathbb{R}$.

Berechne $\nabla f_0 = \nabla f(x_0)$
$p_0 = -\nabla f_0$
$k = 0$
while $\|\nabla f_k\| \neq 0$
  Bestimme Schrittweite α_k mit einem geeigneten Algorithmus
  $x_{k+1} = x_k + \alpha_k p_k$
  Berechne $\nabla f_{k+1} = \nabla f(x_{k+1})$
  $\beta_{k+1}^{\mathrm{FR}} = \dfrac{\nabla f_{k+1}^T \nabla f_{k+1}}{\nabla f_k^T \nabla f_k}$
  $p_{k+1} = -\nabla f_{k+1} + \beta_{k+1}^{\mathrm{FR}} p_k$
  $k = k + 1$
end

Bemerkungen 3.5.1 *i) Für $f(x) = \frac{1}{2}x^T Ax - b^T x$ stimmen die Algorithmen 3.5.2 und 3.5.1 überein.*

ii) Es sind weitere Varianten des nichtlinearen cg-Verfahrens bekannt. Diese unterscheiden sich im Wesentlichen in der Wahl des β_{k+1}. Wir erwähnen hier noch zwei andere Definitionen:

- *Variante von Polak-Rebière:*

$$\beta_{k+1}^{\mathrm{PR}} = \frac{\nabla f_{k+1}^T (\nabla f_{k+1} - \nabla f_k)}{\|\nabla f_k\|^2}$$

- *Variante von Hestenes-Stiefel:*

$$\beta_{k+1}^{\mathrm{HS}} = \frac{\nabla f_{k+1}^T (\nabla f_{k+1} - \nabla f_k)}{(\nabla f_{k+1} - \nabla f_k)^T p_k}.$$

4 Optimierung unter Nebenbedingungen

In diesem Kapitel werden wir uns nun der Frage widmen, wie man vorgehen kann, um eine Funktion $f : \mathbb{R}^n \to \mathbb{R}$ nicht mehr auf ganz $\mathbb{R}^n$, sondern auf einer Menge $\Omega \subset \mathbb{R}^n$ zu minimieren. Genauer werden wir den Fall betrachten, dass Ω eine Menge der Form

$$\Omega = \{x \in \mathbb{R}^n \mid h(x) = 0,\ g(x) \leq 0\}$$

ist. Dies entspricht dem Probem

$$\text{minimiere } f(x) \text{ in } \mathbb{R}^n$$

unter den Nebenbedingungen

$$\begin{aligned} h(x) &= 0 \qquad h : \mathbb{R}^n \to \mathbb{R}^m, \\ g(x) &\leq 0 \qquad g : \mathbb{R}^n \to \mathbb{R}^k \end{aligned}$$

mit $m + k \leq n$. Um unsere Notation zu verdeutlichen, betrachte man das nachfolgende Beispiel.

Beispiel 4.0.1 Es sei das folgende Optimierungsproblem gegeben:

$$\text{minimiere } (x_1 - 2)^2 + (x_2 - 2)^2 \text{ in } \mathbb{R}^2$$

unter den Nebenbedingungen

$$\begin{aligned} x_1^2 - x_2 &\leq 0 \\ x_1 + x_2 &\leq 2. \end{aligned}$$

Dann ist für $x = (x_1, x_2)^T \in \mathbb{R}^2$

$$\begin{aligned} f(x) &= (x_1 - 2)^2 + (x_2 - 2)^2, \\ g(x) &= \begin{pmatrix} g_1(x) \\ g_2(x) \end{pmatrix} = \begin{pmatrix} -x_1^2 - x_2 \\ x_1 + x_2 - 2 \end{pmatrix} \end{aligned}$$

und $m = 0, k = 2$.

Bevor wir nun zwei Verfahren zur Lösung von solchen restringierten Optimierungsproblemen vorstellen werden, bemerken wir an dieser Stelle noch, dass sich Restriktionen der Form $g(x) \leq 0$ in der Form $\widetilde{g}(x) = 0$ schreiben lassen, indem man

$$\widetilde{g}(x) := \max\{0, g(x)\}$$

setzt. Daher werden wir im Folgenden nur Probleme der Form

$$\begin{cases} \text{minimiere } f(x) \text{ in } \mathbb{R}^n \\ \text{unter der Nebenbedingung } h(x) = 0 \end{cases} \tag{4.1}$$

mit $f : \mathbb{R}^n \to \mathbb{R}$, $h : \mathbb{R}^n \to \mathbb{R}^m$ und $m \leq n$ betrachten.

4.1 STRAFTERMMETHODE

Die grundlegende Idee der Straftermmethode besteht darin, das restringierte Problem (4.1) in eine Folge nichtrestringierter Optimierungsprobleme zu überführen, deren Minimalstellen x_k^* gegen die Lösung x^* von (4.1) konvergieren.
Hierfür definieren wir die *Straf-Lagrangefunktion* $\mathcal{L}_\tau : \mathbb{R}^n \to \mathbb{R}$ über

$$\mathcal{L}_\tau(x) := f(x) + \frac{1}{2}\tau\|h(x)\|^2.$$

Der Parameter $\tau > 0$ heißt *Strafparameter*. Auf Ω, d.h. falls die Nebenbedingungen erfüllt sind, ist die Minimierung von f zur Minimierung von$\mathcal{L}_\tau$ äquivalent. Falls wir nun aber die Folge nichtrestringierter Optimierungsprobleme

$$\text{minimiere } \mathcal{L}_{\tau_k}(x) \text{ in } \mathbb{R}^n \tag{4.2}$$

betrachten, so stimmen deren Minimalstellen x_k^* im Allgemeinen nicht mit x^* überein. Wählt man aber $(\tau_k)_{k\in\mathbb{N}} \subset \mathbb{R}_+$ als eine monoton wachsende Folge von positiven Strafparametern, sodass

$$\tau_k \longrightarrow \infty \text{ für } k \to \infty$$

gilt, so strebt die Zielfunktion $\mathcal{L}_{\tau_k}(x)$ gegen Unendlich, es sei denn die Nebenbedingungen sind erfüllt, also $h(x) = 0$. Daher konvergiert unter bestimmten Voraussetzungen die Folge $(x_k^*)_{k\in\mathbb{N}}$ der Lösungen von (4.2) gegen die auf Ω gesuchte Minimalstelle x^*. Genauer gilt folgender Satz.

Satz 4.1.1 (Konvergenz der Straftermmethode) *Es seien $f : \mathbb{R}^n \to \mathbb{R}$ und $h : \mathbb{R}^n \to \mathbb{R}^m$ mit $m \leq n$ stetige Funktionen auf einer abgeschlossenen Menge $K \subset \mathbb{R}^n$ und die Folge der Strafparameter $\tau_k > 0$ divergiere monoton,*

$$\tau_k \longrightarrow \infty \textit{ für } k \to \infty.$$

x_k^* sei eine globale Minimalstelle des Problems

$$\textit{minimiere } \mathcal{L}_{\tau_k}(x) \textit{ in } K \subset \mathbb{R}^n$$

im Iterationsschritt $k \in \mathbb{N}$. Dann gilt

$$x_k^* \longrightarrow x^* \qquad \textit{für } k \to \infty,$$

wobei x^* eine globale Minimalstelle von f auf K ist und der Nebenbedingung $h(x^*) = 0$ genügt.

Bemerkungen 4.1.2 *i) Für $\tau_k \to \infty$ wird die Kondition der zugehörigen Optimierungsaufgaben* (4.2) *immer schlechter. Auf der anderen Seite darf die Folge der Strafparameter auch nicht zu langsam wachsen, da dies die Konvergenz des gesamten Verfahrens erheblich verlangsamen würde. Damit die numerische Lösung der Probleme* (4.2) *nicht zu aufwändig werden, hilft es, die Startvektoren für das Verfahren zur Bestimmung von x_k^* möglichst gut zu wählen. Geschickt ist zum Beispiel, die Minimalstelle x_k^* aus der k-ten Iteration in der $(k+1)$-ten Iteration als Starvektor einzusetzen.*

ii) Hinsichtlich der Wahl der Strafparameter τ_k ist es üblich, einen nicht zu großen Wert τ_1 auszuwählen und dann $\tau_{k+1} = \rho\tau_k$ für $k \in \mathbb{N}$ zu setzen, wobei ρ in der Praxis ein Faktor zwischen 4 und 10 ist.

4.2 LAGRANGE-MULTIPLIKATOREN

Eine Modifikation der Straftermmethode liefert der folgende Ansatz: Anstatt die Straf-Lagrangefunktion $\mathcal{L}_\tau$ zu minimieren, betrachtet man die Funktion $\mathcal{G}_\tau : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$, die durch

$$\mathcal{G}_\tau(x, \lambda) := f(x) + \lambda^T h(x) + \frac{1}{2}\tau \|h(x)\|^2 \tag{4.3}$$

definiert ist, wobei $\lambda \in \mathbb{R}^m$ ein Lagrange-Multiplikator ist. Wendet man nun das Vorgehen der Straftermmethode an, so erhält man für $k \in \mathbb{N}$ eine Folge von Problemen

$$\text{minimiere } \mathcal{G}_{\tau_k}(x, \lambda_k) \text{ für } x \in \mathbb{R}^n,$$

wobei $(\lambda_k)_{k\in\mathbb{N}}$ eine beschränkte Folge von Vektoren im $\mathbb{R}^m$ ist und $\tau_k > 0$ Strafparameter wie in Abschnitt 4.1 sind. Satz 4.1.1 lässt sich auch auf dieses Verfahren übertragen, sofern die Folge der Lagrange-Multiplikatoren als beschränkt angenommen werden darf. Üblicherweise wählt man die Multiplikation gemäß

$$\lambda_{k+1} = \lambda_k + \tau_k h(x_k^*),$$

wobei $\lambda_0 \in \mathbb{R}^m$ ein gegebener Startvektor ist.

Bemerkungen 4.2.1 *i) Ein Nachteil dieser Formulierung ist die Tatsache, dass die Existenz einer Minimalstelle von* (4.3) *nicht gewährleistet ist, nicht einmal in dem Fall, dass f ein eindeutiges globales Minimum besitzt.*

ii) Gegenüber der Straftermmethode vorteilhaft ist bei diesem Verfahren, dass die Folge $(\tau_k)_{k\in\mathbb{N}}$ nicht mehr gegen Unendlich zu divergieren braucht, sondern sondern konstant sein kann. Daher sind die im Verfahren auftretenden Optimierungsprobleme im Allgemeinen besser konditioniert. Darüberhinaus weist dieses Vorgehen deutlich bessere Konvergenzeigenschaften auf als die Straftermmethode. Die Konvergenz ist hier unter bestimmten Voraussetzungen superlinear, siehe [Quarteroni et. al., Band 1, Abschnitt 7.3.3].

Literaturverzeichnis

[Deuflhard/Hohmann] P. DEUFLHARD, A. HOHMANN, Numerische Mathematik 1, 4. Auflage, de Gruyter-Verlag, Berlin, 2008.

[Fischer] G. FISCHER, Linare Algebra, Vieweg-Verlag, Wiesbaden, 1989.

[Numerik I] S. FUNKEN, Skript zur Vorlesung „Numerik I“, gehalten im Wintersemester 10/11 an der Universität Ulm.

[Numerik II] S. FUNKEN, Skript zur Vorlesung „Numerik II“, gehalten im Sommersemester 11 an der Universität Ulm.

[Golub/Loan] G. H. GOLUB, C. F. VAN LOAN, Matrix Computations, 3. ed., Hopkins Univ. Press, 1996.

[Nipp/Stoffer] K. NIPP, D. STOFFER, Lineare Algebra: Eine Einführung für Ingenieure unter besonderer Berücksichtigung numerischer Aspekte, 5. Auflage, vdf Hochschulverlag AG an der ETH Zürich, Zürich, 2002.

[Nocedal/Wright] J. NOCEDAL, S. WRIGHT, Numerical Optimization, 2. ed., Springer Science and Business Media, 2006.

[Quarteroni et. al.] A. QUARTERONI, R. SACCO, F. SALERI, Numerische Mathematik, Band 1 & 2, Springer-Verlag, Berlin, 2002.

[Stewart] G. W. STEWART, On the Sensitivity of the Eigenvalue Problem $Ax = \lambda Bx$, SIAM Journal on Numerical Analysis 9, S. 669–686.

[Stoer] J. STOER, R. BULIRSCH, Numerische Mathematik, Band 2, 5. Auflage, Springer-Verlag, Berlin, 2005.

[Wilkinson65] J. H. WILKINSON, The Algebraic Eigenvalue Problem, Oxford University Press, 1965.