

# Einführung in die Numerische Mathematik

Lars Grüne  
Lehrstuhl für Angewandte Mathematik  
Mathematisches Institut  
Universität Bayreuth  
95440 Bayreuth  
[lars.gruene@uni-bayreuth.de](mailto:lars.gruene@uni-bayreuth.de)  
[www.math.uni-bayreuth.de/~lgruene/](http://www.math.uni-bayreuth.de/~lgruene/)

Vorlesungsskript  
Vierte Auflage  
Wintersemester 2009/2010



# Vorwort

Dieses Skript ist im Rahmen einer gleichnamigen Vorlesung entstanden, die ich im Wintersemester 2009/2010 an der Universität Bayreuth gehalten habe. Es ist die vierte Auflage einer Ausarbeitung aus dem Wintersemester 2002/2003 (mit dem Titel “Numerische Mathematik I”), die gegenüber der dritten Auflage korrigiert, aktualisiert und in wenigen Punkten erweitert wurde. Wegen der Umstellung auf die dreistündige Bachelor-Veranstaltung sind zudem im Vergleich zu den früheren Versionen einige Themen weggefallen.

Auf der WWW-Seite <http://www.math.uni-bayreuth.de/~lgruene/> ist eine elektronische Version dieses Skripts unter dem Link **Lehrveranstaltungen** erhältlich. Die zugehörigen Übungsaufgaben finden sich im E-Learning System der Universität Bayreuth, zu dem wir Ihnen auf Wunsch (E-Mail an den Autor genügt) gerne einen Gastzugang geben.

Dieser Text ist ein Vorlesungsskript und soll in erster Linie dazu dienen, den in der Vorlesung behandelten Stoff in kompakter schriftlicher Form für die Teilnehmerinnen und Teilnehmer der Vorlesung zusammen zu fassen. Insbesondere soll dieses Skript keinen Ersatz für ein Lehrbuch der Numerischen Mathematik darstellen. Beim Schreiben einzelner Passagen haben mir die Lehrbücher [2, 7, 8, 9] und die Skripten [4, 5, 6] sehr gute Dienste erwiesen.

Ich möchte mich an dieser Stelle bei meinen Mitarbeitern sowie bei allen Teilnehmerinnen und Teilnehmern dieser Vorlesung bedanken, die durch ihr aufmerksames Lesen während der Vorlesung oder bei der Prüfungsvorbereitung Fehler im Skript gefunden und mitgeteilt haben.

Bayreuth, Februar 2010

LARS GRÜNE



# Inhaltsverzeichnis

<b>Vorwort</b>	<b>i</b>
<b>1 Einführung</b>	<b>1</b>
1.1 Korrektheit . . . . .	2
1.2 Effizienz . . . . .	2
1.3 Robustheit und Kondition . . . . .	2
1.4 Mathematische Techniken . . . . .	3
<b>2 Lineare Gleichungssysteme</b>	<b>5</b>
2.1 Eine Anwendung linearer Gleichungssysteme: Ausgleichsrechnung . . . . .	6
2.2 Das Gauss'sche Eliminationsverfahren . . . . .	7
2.3 <i>LR</i> -Faktorisierung . . . . .	10
2.4 Fehlerabschätzungen und Kondition . . . . .	12
2.5 <i>QR</i> -Faktorisierung . . . . .	19
2.6 Aufwandsabschätzungen . . . . .	26
2.7 Iterative Verfahren . . . . .	29
2.8 Gauß-Seidel- und Jacobi-Verfahren . . . . .	31
2.9 Relaxation . . . . .	38
<b>3 Eigenwerte und Eigenvektoren</b>	<b>41</b>
3.1 Vektoriteration . . . . .	42
3.2 Der QR-Algorithmus . . . . .	45
<b>4 Interpolation</b>	<b>55</b>
4.1 Polynominterpolation . . . . .	56
4.1.1 Lagrange-Polynome und baryzentrische Koordinaten . . . . .	57
4.1.2 Kondition . . . . .	60
4.1.3 Fehlerabschätzungen . . . . .	62

4.2	Funktionsinterpolation und orthogonale Polynome . . . . .	64
4.2.1	Orthogonale Polynome . . . . .	64
4.3	Splineinterpolation . . . . .	70
<b>5</b>	<b>Integration</b>	<b>77</b>
5.1	Newton–Cotes–Formeln . . . . .	77
5.2	Zusammengesetzte Newton–Cotes–Formeln . . . . .	82
5.3	Gauss–Quadratur . . . . .	84
<b>6</b>	<b>Nichtlineare Gleichungssysteme</b>	<b>89</b>
6.1	Fixpunktiteration . . . . .	89
6.2	Das Bisektionsverfahren . . . . .	92
6.3	Konvergenzordnung . . . . .	94
6.4	Das Newton–Verfahren . . . . .	99
6.5	Das Sekanten–Verfahren . . . . .	105
6.6	Das Gauss–Newton–Verfahren für Ausgleichsprobleme . . . . .	108
	<b>Literaturverzeichnis</b>	<b>114</b>
	<b>Index</b>	<b>116</b>

# Kapitel 1

## Einführung

Die Numerische Mathematik — oder kurz Numerik — beschäftigt sich mit der Entwicklung und der Analyse von Algorithmen, mit denen mathematische Probleme am Computer gelöst werden können. Im Gegensatz zu symbolischen oder analytischen Rechnungen will man hier keine geschlossenen Formeln oder algebraischen Ausdrücke als Ergebnisse erhalten, sondern quantitative Zahlenwerte<sup>1</sup>, daher der Name „Numerische Mathematik“.

In der Grundvorlesung zur Numerik werden traditionell viele verschiedene mathematische Probleme behandelt; in dieser Vorlesung werden wir uns mit den folgenden Themen beschäftigen:

- Lösung linearer Gleichungssysteme
- Berechnung von Eigenwerten
- Interpolation
- Integration
- Nichtlineare Gleichungen und Gleichungssysteme

Ein wichtiger Bereich, der in dieser Aufzählung fehlt, sind die Differentialgleichungen; diese werden schwerpunktmäßig in der Vorlesung „Numerische Methoden für gewöhnliche Differentialgleichungen“ im kommenden Sommersemester behandelt.

Die Fülle unterschiedlicher Probleme aus der Analysis und der linearen Algebra bringt es mit sich, dass ganz verschiedene mathematische Techniken aus diesen Gebieten zur numerischen Lösung verwendet werden. Aus diesem Grund wird gerade die erste Numerik-Vorlesung oft als „Gemischtwarenladen“ aufgefasst, in dem verschiedene Themen scheinbar zusammenhanglos abgehandelt werden. Tatsächlich gibt es aber — trotz der unterschiedlichen Mathematik — eine Reihe von Grundprinzipien, die in der Numerik wichtig sind. Bevor wir im nächsten Kapitel mit der „harten“ Mathematik beginnen, wollen wir diese Prinzipien in dieser Einführung kurz und informell erläutern.

---

<sup>1</sup>die dann natürlich oft grafisch aufbereitet werden

## 1.1 Korrektheit

Eine der wesentlichen Aufgaben der numerischen Mathematik ist es, die Korrektheit von Algorithmen zu überprüfen, d.h. sicherzustellen, dass und unter welchen Voraussetzungen an die Problemdata tatsächlich das richtige Ergebnis berechnet wird. Diese Überprüfung soll natürlich mit mathematischen Methoden durchgeführt werden, d.h. am Ende steht ein formaler mathematischer Beweis, der die korrekte Funktion eines Algorithmus sicher stellt. In vielen Fällen wird ein Algorithmus kein exaktes Ergebnis in endlich vielen Schritten liefern, sondern eine Näherungslösung bzw. eine Folge von Näherungslösungen. In diesem Fall ist zusätzlich zu untersuchen, wie groß der Fehler der Näherungslösung in Abhängigkeit von den vorhandenen Parametern ist bzw. wie schnell die Folge von Näherungslösungen gegen den exakten Wert konvergiert.

## 1.2 Effizienz

Hat man sich von der Korrektheit eines Algorithmus' überzeugt, so stellt sich im nächsten Schritt die Frage nach der Effizienz eines Algorithmus. Liefert der Algorithmus ein exaktes Ergebnis in endlich vielen Schritten, so ist im Wesentlichen die Anzahl der Operationen „abzuzählen“, falls eine Folge von Näherungslösungen berechnet wird, so muss die Anzahl der Operationen pro Näherungslösung und die Konvergenzgeschwindigkeit gegen die exakte Lösung untersucht werden.

Oft gibt es viele verschiedene Algorithmen zur Lösung eines Problems, die je nach den weiteren Eigenschaften des Problems unterschiedlich effizient sind.

## 1.3 Robustheit und Kondition

Selbst wenn ein Algorithmus in der Theorie in endlich vielen Schritten ein exaktes Ergebnis liefert, wird dies in der numerischen Praxis nur selten der Fall sein. Der Grund hierfür liegt in den sogenannten *Rundungsfehlern*: Intern kann ein Computer nur endlich viele Zahlen darstellen, es ist also unmöglich, jede beliebige reelle (ja nicht einmal jede rationale) Zahl exakt darzustellen. Wir wollen diesen Punkt etwas formaler untersuchen. Für eine gegebene *Basis*  $B \in \mathbb{N}$  kann jede reelle Zahl  $x \in \mathbb{R}$  als

$$x = m \cdot B^e$$

dargestellt werden, wobei  $m \in \mathbb{R}$  die *Mantisse* und  $e \in \mathbb{Z}$  der *Exponent* genannt wird. Durch geeignete Wahl von  $e$  reicht es dabei, Zahlen der Form  $m = \pm 0.m_1m_2m_3 \dots$  mit den Ziffern  $m_1, m_2, \dots$  mit  $m_i \in \{0, 1, \dots, B-1\}$  zu benutzen. Computerintern wird üblicherweise die Basis  $B = 2$  verwendet, da die Zahlen als *Binärzahlen* dargestellt werden. Im Rechner stehen nun nur endlich viele Stellen für  $m$  und  $e$  zur Verfügung, z.B.  $l$  Stellen für  $m$  und  $n$  Stellen für  $e$ . Wir schreiben  $m = \pm 0.m_1m_2m_3 \dots m_l$  und  $e = \pm e_1e_2 \dots e_n$ . Unter der zusätzlichen *Normierungs-Bedingung*  $m_1 \neq 0$  ergibt sich eine eindeutige Darstellung der sogenannten *maschinendarstellbaren Zahlen*

$$\mathcal{M} = \{x \in \mathbb{R} \mid \pm 0.m_1m_2m_3 \dots m_l \cdot B^{\pm e_1e_2 \dots e_n}\} \cup \{0\}.$$



Zahlen, die nicht in dieser Menge  $\mathcal{M}$  liegen, müssen durch Rundung in eine maschinendarstellbare Zahl umgewandelt werden.

Die dadurch entstehenden Ungenauigkeiten beeinflussen offensichtlich das Ergebnis numerischer Algorithmen. Die Robustheit eines Algorithmus ist nun dadurch bestimmt, wie stark diese Rundungsfehler sich im Ergebnis auswirken. Tatsächlich betrachtet man die Robustheit mathematisch für allgemeine Fehler, so dass egal ist, ob diese durch Rundung oder weitere Fehlerquellen (Eingabe- bzw. Übertragungsfehler, Ungenauigkeiten in vorausgegangenen Berechnungen etc.) hervorgerufen worden sind.

Ein wichtiges Hilfsmittel zur Betrachtung dieser Problemstellung ist der Begriff der *Kondition* eines mathematischen Problems. Wenn wir das Problem abstrakt als Abbildung  $\mathcal{A} : D \rightarrow L$  der Problem Daten  $D$  (z.B. Matrizen, Messwerte...) auf die Lösung  $L$  des Problems betrachten, so gibt die Kondition an, wie stark sich kleine Änderungen in den Daten  $D$  in der Lösung  $L$  auswirken, was durch die Analyse der Ableitung von  $\mathcal{A}$  quantitativ bestimmt wird. Wenn kleine Änderungen in  $D$  große Änderungen in  $L$  verursachen können spricht man von einem *schlecht konditionierten* Problem. Beachte, dass die Kondition eine Eigenschaft des gestellten Problems und damit unabhängig von dem verwendeten Algorithmus ist. Allerdings ist die Robustheit eines Algorithmus besonders bei schlecht konditionierten Problemen wichtig, da hier kleine Fehler im Algorithmus große Fehler im Ergebnis verursachen können. Am schönsten entwickelt ist diese Theorie bei linearen Gleichungssystemen; in diesem Rahmen werden wir sie auch ausführlich untersuchen.

## 1.4 Mathematische Techniken

Wie bereits erwähnt, erfordern die unterschiedlichen Problemklassen ganz unterschiedliche mathematische Techniken. Trotzdem gibt es einige Gemeinsamkeiten, d.h. Techniken, die in der ein oder anderen Weise immer wieder auftauchen. Als Beispiel sei hier das Prinzip der Orthogonalität bzw. der orthogonalen Projektion genannt: Dies wird uns bei der effizienten Lösung linearer Gleichungssysteme (im Householder-Algorithmus) ebenso begegnen, wie bei der Polynom-Interpolation (bei den Tschebyscheff-Polynomen) und bei der numerischen Integration (bei der Gauß-Quadratur). All diese Verfahren haben gemeinsam, dass sie (bezüglich geeigneter Kriterien) zu den besten Verfahren ihrer Klasse gehören, wofür man allerdings den Preis zahlen muss, dass ihre Funktionsweise sich nicht auf den ersten Blick erschließt, da ihre Herleitung auf trickreichen mathematischen Ideen beruht. Weitere Beispiele sind der Banach'sche Fixpunktsatz, mit dem viele verschiedene iterative Verfahren analysiert werden können oder die Taylor-Entwicklung, die die Grundlage vieler Algorithmen der numerischen Analysis bildet.



## Kapitel 2

# Lineare Gleichungssysteme

Algorithmen zur Lösung linearer Gleichungssysteme bilden die Basis für viele Anwendungen der Numerik und stehen daher traditionell am Anfang vieler Numerik-Vorlesungen. Ausführlich aufgeschrieben besteht das Problem darin, Zahlen  $x_1, \dots, x_n \in \mathbb{R}$  zu bestimmen, für die das Gleichungssystem

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n &= b_1 \\ &\vdots \\ a_{n1}x_1 + a_{n2}x_2 + \dots + a_{nn}x_n &= b_n \end{aligned} \tag{2.1}$$

erfüllt ist. Die ausführliche Schreibweise in (2.1) ist etwas unhandlich, weswegen wir lineare Gleichungssysteme in der üblichen Matrix-Form schreiben werden, nämlich als

$$Ax = b, \tag{2.2}$$

mit

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix}, \quad x = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \quad \text{und} \quad b = \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix}. \tag{2.3}$$

Diese Schreibweise hat nicht nur den Vorteil, dass man ein Gleichungssystem viel kürzer aufschreiben kann, es wird sich auch zeigen, dass gewisse Eigenschaften der Matrix  $A$  entscheiden, was für ein Verfahren zur Lösung von (2.2) sinnvollerweise eingesetzt werden kann.

Einige kurze Bemerkungen zur Notation: Wir werden Matrizen typischerweise mit großen Buchstaben bezeichnen (z.B.  $A$ ) und Vektoren mit kleinen (z.B.  $b$ ). Ihre Einträge werden wir mit indizierten Kleinbuchstaben bezeichnen, wie in (2.3). Mit einem hochgestellten „T“ bezeichnen wir transponierte Matrizen und Vektoren, für  $A$  und  $x$  aus (2.3) also z.B.

$$x = (x_1, \dots, x_n)^T, \quad A^T = \begin{pmatrix} a_{11} & a_{21} & \dots & a_{n1} \\ \vdots & \vdots & & \vdots \\ a_{1n} & a_{2n} & \dots & a_{nn} \end{pmatrix}.$$

Da die Anzahl  $n$  der Gleichungen in (2.1) gleich der Anzahl der Unbekannten  $x_1, \dots, x_n$  ist, ist  $A$  in (2.2) eine quadratische Matrix der Dimension  $n \times n$ . Für quadratische Matrizen

ist aus der linearen Algebra bekannt, dass genau dann eine eindeutige Lösung von (2.2) existiert, wenn die Matrix invertierbar ist. Wir werden in diesem Kapitel immer annehmen, dass dies der Fall ist.

In den Übungen werden Beispiele von Problemen aus den Wirtschaftswissenschaften und der Physik behandelt, die auf die Lösung linearer Gleichungssysteme führen. Hier werden wir einige „innermathematische“ Probleme betrachten, deren numerische Behandlung ebenfalls auf die Lösung linearer Gleichungssysteme führt.

## 2.1 Eine Anwendung linearer Gleichungssysteme: Ausgleichsrechnung

Das erste unserer Anwendungsbeispiele ist für viele praktische Zwecke besonders wichtig, weswegen wir es etwas genauer untersuchen wollen.

Nehmen wir an, wir haben ein Experiment durchgeführt, bei dem wir für verschiedene Eingabewerte  $t_1, t_2, \dots, t_m$  Messwerte  $z_1, z_2, \dots, z_m$  erhalten. Auf Grund von theoretischen Überlegungen (z.B. auf Grund eines zugrundeliegenden physikalischen Gesetzes), kennt man eine Funktion  $f(t)$ , für die  $f(t_i) = z_i$  gelten sollte. Diese Funktion wiederum hängt aber nun von unbekanntem Parametern  $x_1, \dots, x_n$  ab; wir schreiben  $f(t; x)$  für  $x = (x_1, \dots, x_n)^T$ , um dies zu betonen. Z.B. könnte  $f(t; x)$  durch

$$f(t; x) = x_1 + x_2 t \quad \text{oder} \quad f(t; x) = x_1 + x_2 t + x_3 t^2$$

gegeben sein. Im ersten Fall beschreibt die gesuchte Funktion eine Gerade, im zweiten eine (verallgemeinerte) Parabel. Wenn wir annehmen, dass die Funktion  $f$  das Experiment wirklich exakt beschreibt und keine Messfehler vorliegen, so könnten wir die Parameter  $x_i$  durch Lösen des (im Allgemeinen nichtlinearen) Gleichungssystems

$$\begin{aligned} f(t_1; x) &= z_1 \\ &\vdots \\ f(t_k; x) &= z_k \end{aligned} \tag{2.4}$$

nach  $x$  bestimmen. In vielen praktischen Fällen ist dieses Gleichungssystem linear, so z.B. in den zwei obigen Beispielen, in denen sich (2.4) zu  $\tilde{A}x = z$  mit

$$\tilde{A} = \begin{pmatrix} 1 & t_1 \\ \vdots & \vdots \\ 1 & t_m \end{pmatrix} \quad \text{bzw.} \quad \tilde{A} = \begin{pmatrix} 1 & t_1 & t_1^2 \\ \vdots & \vdots & \vdots \\ 1 & t_m & t_m^2 \end{pmatrix} \quad \text{und} \quad z = \begin{pmatrix} z_1 \\ \vdots \\ z_m \end{pmatrix}$$

ergibt. Diese linearen Gleichungssysteme haben  $m$  Gleichungen (eine für jedes Wertepaar  $(t_i, z_i)$ ) und  $n$  Unbekannte (nämlich gerade die unbekanntem Parameter  $x_i$ ), wobei  $m$  üblicherweise sehr viel größer als  $n$  ist. Man sagt, dass das Gleichungssystem *überbestimmt* ist. Da Messwerte eines Versuchs praktisch immer mit Fehlern behaftet sind, ist es sicherlich zu optimistisch, anzunehmen, dass das Gleichungssystem  $\tilde{A}x = z$  lösbar ist (überbestimmte Gleichungssysteme haben oft keine Lösung!). Statt also den (vermutlich vergeblichen) Versuch zu machen, eine exakte Lösung  $x$  dieses Systems zu finden, wollen wir probieren,

eine möglichst gute Näherungslösung zu finden, d.h. wenn  $\tilde{A}x = z$  nicht lösbar ist, wollen wir zumindest ein  $x$  finden, so dass  $\tilde{A}x$  möglichst nahe bei  $z$  liegt. Dazu müssen wir ein Kriterium für „möglichst nahe“ wählen, das sowohl sinnvoll ist als auch eine einfache Lösung zulässt. Hier bietet sich das sogenannte *Ausgleichsproblem* (auch *Methode der kleinsten Quadrate* genannt) an:

Finde  $x = (x_1, \dots, x_n)^T$ , so dass  $\varphi(x) := \|z - \tilde{A}x\|^2$  minimal wird.

Hierbei bezeichnet  $\|y\|$  die euklidische Norm eines Vektors  $y \in \mathbb{R}^n$ , also

$$\|y\| = \sqrt{\sum_{i=1}^n y_i^2}.$$

Um die Funktion  $\varphi$  zu minimieren, setzen wir den Gradienten  $\nabla\varphi(x)$  gleich Null. Beachte dazu, dass der Gradient der Funktion  $g(x) := \|f(x)\|^2$  für beliebiges  $f : \mathbb{R}^n \rightarrow \mathbb{R}^n$  durch  $\nabla g(x) = 2Df(x)^T f(x)$  gegeben ist. Wir erhalten also

$$\nabla\varphi(x) = 2\tilde{A}^T \tilde{A}x - 2\tilde{A}^T z.$$

Falls  $\tilde{A}$  vollen Spaltenrang besitzt, ist die zweite Ableitung  $D^2\varphi(x) = 2\tilde{A}^T \tilde{A}$  positiv definit, womit jede Nullstelle des Gradienten  $\nabla\varphi$  ein Minimum von  $\varphi$  ist. Folglich minimiert ein Vektor  $x$  die Funktion  $\varphi$  genau dann, wenn die sogenannten *Normalengleichungen*  $\tilde{A}^T \tilde{A}x = \tilde{A}^T z$  erfüllt sind. Das Ausgleichsproblem wird also wie folgt gelöst:

$$\text{löse } Ax = b \text{ mit } A = \tilde{A}^T \tilde{A} \text{ und } b = \tilde{A}^T z.$$

Das zunächst recht kompliziert scheinende Minimierungsproblem für  $\varphi$  wird also auf die Lösung eines linearen Gleichungssystems zurückgeführt.

Neben der Ausgleichsrechnung gibt es viele weitere Anwendungen in der Numerik, die auf die Lösung eines linearen Gleichungssystems führen. Einige davon werden uns im weiteren Verlauf dieser Vorlesung noch begegnen, z.B. die Lösung *nichtlinearer* Gleichungssysteme mit dem *Newton-Verfahren*, bei dem eine Folge linearer Gleichungssysteme gelöst werden muss, oder die *Interpolation* von Punkten mittels *Splines*.

## 2.2 Das Gauß'sche Eliminationsverfahren

Wir werden nun ein erstes Verfahren zur Lösung linearer Gleichungssysteme kennen lernen. Das *Gauß'sche Eliminationsverfahren* ist ein sehr anschauliches Verfahren, das zudem recht leicht implementiert werden kann. Es beruht auf der einfachen Tatsache, dass ein lineares Gleichungssystem  $Ax = b$  leicht lösbar ist, falls die Matrix  $A$  in *oberer Dreiecksform* vorliegt, d.h.

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ 0 & a_{22} & \dots & a_{2n} \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & a_{nn} \end{pmatrix}.$$

In diesem Fall kann man  $Ax = b$  leicht mittels der rekursiven Vorschrift

$$x_n = \frac{b_n}{a_{nn}}, \quad x_{n-1} = \frac{b_{n-1} - a_{n-1n}x_n}{a_{n-1n-1}}, \dots, \quad x_1 = \frac{b_1 - a_{12}x_2 - \dots - a_{1n}x_n}{a_{11}}$$

oder, kompakt geschrieben,

$$x_i = \frac{b_i - \sum_{j=i+1}^n a_{ij}x_j}{a_{ii}}, \quad i = n, n-1, \dots, 1 \quad (2.5)$$

(mit der Konvention  $\sum_{j=n+1}^n a_{ij}x_j = 0$ ) lösen. Dieses Verfahren wird als *Rückwärtseinsetzen* bezeichnet.

Die Idee des Gauß'schen Eliminationsverfahrens liegt nun darin, das Gleichungssystem  $Ax = b$  in ein Gleichungssystem  $\tilde{A}x = \tilde{b}$  umzuformen, so dass die Matrix  $\tilde{A}$  in oberer Dreiecksform vorliegt. Wir wollen dieses Verfahren zunächst an einem Beispiel veranschaulichen.

**Beispiel 2.1** Gegeben sei das lineare Gleichungssystem (2.2) mit

$$A = \begin{pmatrix} 1 & 5 & 6 \\ 7 & 9 & 6 \\ 2 & 3 & 4 \end{pmatrix} \quad \text{und} \quad b = \begin{pmatrix} 29 \\ 43 \\ 20 \end{pmatrix}.$$

Um die Matrix  $A$  auf obere Dreiecksgestalt zu bringen, müssen wir die drei Einträge 7, 2 und 3 unterhalb der Diagonalen auf 0 bringen („eliminieren“). Wir beginnen mit der 2. Hierzu subtrahieren wir 2-mal die erste Zeile von der letzten und erhalten so

$$A_1 = \begin{pmatrix} 1 & 5 & 6 \\ 7 & 9 & 6 \\ 0 & -7 & -8 \end{pmatrix}$$

Das Gleiche machen wir mit dem Vektor  $b$ . Dies liefert

$$b_1 = \begin{pmatrix} 29 \\ 43 \\ -38 \end{pmatrix}.$$

Nun fahren wir fort mit der 7: Wir subtrahieren 7-mal die erste Zeile von der zweiten, sowohl in  $A_1$  als auch in  $b_1$ , und erhalten

$$A_2 = \begin{pmatrix} 1 & 5 & 6 \\ 0 & -26 & -36 \\ 0 & -7 & -8 \end{pmatrix} \quad \text{und} \quad b_2 = \begin{pmatrix} 29 \\ -160 \\ -38 \end{pmatrix}.$$

Im dritten Schritt „eliminieren“ wir die  $-7$ , die jetzt an der Stelle der 3 steht, indem wir  $7/26$ -mal die zweite Zeile von der dritten subtrahieren:

$$A_3 = \begin{pmatrix} 1 & 5 & 6 \\ 0 & -26 & -36 \\ 0 & 0 & \frac{22}{13} \end{pmatrix} \quad \text{und} \quad b_3 = \begin{pmatrix} 29 \\ -160 \\ \frac{66}{13} \end{pmatrix}.$$

Hiermit sind wir fertig und setzen  $\tilde{A} = A_3$  und  $\tilde{b} = b_3$ . Rückwärtseinsetzen gemäß (2.5) liefert dann

$$x_3 = \frac{\frac{66}{13}}{\frac{22}{13}} = 3, \quad x_2 = \frac{-160 - 3 \cdot (-36)}{-26} = 2 \quad \text{und} \quad x_1 = \frac{29 - 2 \cdot 5 - 3 \cdot 6}{1} = 1.$$

□

Wir formulieren den Algorithmus nun für allgemeine quadratische Matrizen  $A$ .

**Algorithmus 2.2 (Gauß–Elimination, Grundversion)**

Gegeben sei eine Matrix  $A \in \mathbb{R}^{n \times n}$  und ein Vektor  $b \in \mathbb{R}^n$ .

- (1) Für  $j$  von 1 bis  $n - 1$  ( $j =$  Spaltenindex des zu eliminierenden Eintrags)
  - (2) Für  $i$  von  $n$  bis  $j + 1$  (rückwärts zählend)
    - ( $i =$  Zeilenindex des zu eliminierenden Eintrags)
 

Subtrahiere  $a_{ij}/a_{jj}$ -mal die  $j$ -te Zeile von der  $i$ -ten Zeile:

Setze  $\alpha := a_{ij}/a_{jj}$  und berechne

$$a_{ik} := a_{ik} - \alpha a_{jk} \quad \text{für } k = j, \dots, n, \quad b_i := b_i - \alpha b_j$$

Ende der  $i$ -Schleife

- (3) Ende der  $j$ -Schleife

□

Es kann passieren, dass dieser Algorithmus zu keinem Ergebnis führt, obwohl das Gleichungssystem lösbar ist. Der Grund dafür liegt in Schritt (1), in dem durch das Diagonalelement  $a_{jj}$  geteilt wird. Hierbei wurde stillschweigend angenommen, dass dieses ungleich Null ist, was aber im Allgemeinen nicht der Fall sein muss. Glücklicherweise gibt es eine Möglichkeit, dieses zu beheben:

Nehmen wir an, dass wir Schritt (1) für gegebene Indizes  $i$  und  $j$  ausführen möchten und  $a_{jj} = 0$  ist. Nun gibt es zwei Möglichkeiten: Im ersten Fall ist  $a_{ij} = 0$ . In diesem Fall brauchen wir nichts zu tun, da das Element  $a_{ij}$ , das auf 0 gebracht werden soll, bereits gleich 0 ist. Im zweiten Fall gilt  $a_{ij} \neq 0$ . In diesem Fall können wir die  $i$ -te und  $j$ -te Zeile der Matrix  $A$  sowie die entsprechenden Einträge im Vektor  $b$  vertauschen, wodurch wir die gewünschte Eigenschaft  $a_{ij} = 0$  erreichen, nun allerdings nicht durch Elimination sondern durch Vertauschung. Dieses Verfahren nennt man *Pivotierung* und der folgende Algorithmus bringt nun tatsächlich jedes lineare Gleichungssystem in Dreiecksform.

**Algorithmus 2.3 (Gauß–Elimination mit Pivotierung)**

Gegeben sei eine Matrix  $A \in \mathbb{R}^{n \times n}$  und ein Vektor  $b \in \mathbb{R}^n$ .

- (1) Für  $j$  von 1 bis  $n - 1$  ( $j =$  Spaltenindex des zu eliminierenden Eintrags)

- (1a) Falls  $a_{jj} = 0$  wähle ein  $p \in \{j+1, \dots, n\}$  mit  $a_{pj} \neq 0$  und vertausche  $a_{jk}$  und  $a_{pk}$  für  $k = j, \dots, n$  sowie  $b_p$  und  $b_j$
- (2) Für  $i$  von  $n$  bis  $j+1$  (rückwärts zählend)  
 ( $i$  = Zeilenindex des zu eliminierenden Eintrags)  
 Subtrahiere  $a_{ij}/a_{jj}$ -mal die  $j$ -te Zeile von der  $i$ -ten Zeile:  
 Setze  $\alpha := a_{ij}/a_{jj}$  und berechne

$$a_{ik} := a_{ik} - \alpha a_{jk} \quad \text{für } k = j, \dots, n, \quad b_i := b_i - \alpha b_j$$

Ende der  $i$ -Schleife

- (3) Ende der  $j$ -Schleife

□

Das Element  $a_{ij}$ , das in Schritt (1a) mit  $a_{jj}$  getauscht wird, nennt man *Pivotelement*. Wir werden später im Abschnitt 2.4 eine Strategie kennen lernen, bei der — auch wenn  $a_{jj} \neq 0$  ist — gezielt ein Element  $a_{kj} \neq 0$  als Pivotelement ausgewählt wird, mit dem man ein besseres Robustheitsverhalten des Algorithmus' erhalten kann.

## 2.3 LR-Faktorisierung

In der obigen Version des Gauß-Verfahrens haben wir die Matrix  $A$  auf obere Dreiecksform gebracht und zugleich alle dafür notwendigen Operationen auch auf den Vektor  $b$  angewendet. Es gibt alternative Möglichkeiten, lineare Gleichungssysteme zu lösen, bei denen der Vektor  $b$  unverändert bleibt. Wir werden nun ein Verfahren kennen lernen, bei dem die Matrix  $A$  in ein Produkt von zwei Matrizen  $L$  und  $R$  zerlegt wird, also  $A = LR$ , wobei  $R$  in oberer Dreiecksform und  $L$  in *unterer Dreiecksform*

$$L = \begin{pmatrix} l_{11} & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ l_{n-11} & \dots & l_{n-1n-1} & 0 \\ l_{n1} & l_{n2} & \dots & l_{nn} \end{pmatrix}.$$

vorliegt. Die Zerlegung  $A = LR$  wird als *LR-Faktorisierung* oder *LR-Zerlegung* bezeichnet.

Um  $Ax = b$  zu lösen, kann man dann  $LRx = b$  wie folgt in zwei Schritten lösen: Zunächst löst man das Gleichungssystem  $Ly = b$ . Dies kann, ganz analog zum Rückwärtseinsetzen (2.5) durch *Vorwärtseinsetzen* geschehen:

$$y_i = \frac{b_i - \sum_{j=1}^{i-1} l_{ij}y_j}{l_{ii}}, \quad i = 1, 2, \dots, n \quad (2.6)$$



Im zweiten Schritt löst man dann durch Rückwärtseinsetzen das Gleichungssystem  $Rx = y$ . Dann gilt

$$Ax = LRx = Ly = b,$$

womit das gewünschte System  $Ax = b$  gelöst ist.

Das Gauß'sche Eliminationsverfahren lässt sich so erweitern, dass damit — bis auf die Zeilenvertauschungen in  $A$  — eine  $LR$ -Zerlegung einer invertierbaren Matrix  $A$  möglich ist:

Die Subtraktion „ $i$ -te Zeile  $- \alpha$ -mal  $j$ -te Zeile“ im  $m$ -ten Schritt des Algorithmus lässt sich durch Multiplikation von links mit der Matrix

$$F_m = \begin{pmatrix} 1 & & & & & \\ & \ddots & & & & \\ & & \ddots & & & \\ & & & \ddots & & \\ -\alpha & & & & \ddots & \\ & & & & & \ddots \\ & & & & & & 1 \end{pmatrix} \quad \text{mit} \quad F_m^{-1} = \begin{pmatrix} 1 & & & & & \\ & \ddots & & & & \\ & & \ddots & & & \\ & & & \ddots & & \\ \alpha & & & & \ddots & \\ & & & & & \ddots \\ & & & & & & 1 \end{pmatrix}$$

erzielen, hierbei steht  $\alpha$  in der  $i$ -ten Zeile und  $j$ -ten Spalte.

Wenn sich die Gauß-Elimination ohne Zeilenvertauschung durchführen lässt, so werden nur Operationen mit solchen Matrizen benötigt. Wenn man die dabei benötigten Matrizen  $F_1, F_2, \dots, F_k$  zu einer Matrix  $F = F_k \cdot F_{k-1} \cdot \dots \cdot F_1$  aufmultipliziert, so erhält man  $A = LR$  mit  $L = F^{-1} = F_1^{-1} \cdot F_2^{-1} \cdot \dots \cdot F_k^{-1}$ , also die  $LR$ -Faktorisierung (beachte, dass  $F^{-1}$ -Matrix als Produkt unterer Dreiecksmatrizen wieder eine untere Dreiecksmatrix ist).

Sind allerdings wegen der Pivotierung Zeilenvertauschungen nötig, so lässt sich i.A. keine  $LR$ -Zerlegung erhalten.<sup>1</sup> Die Vertauschung der  $i$ -ten und der  $j$ -ten Zeile in  $A$  im  $m$ -ten Schritt des Algorithmus lässt sich durch die Multiplikation von links mit Matrizen der Form

$$P_m = \begin{pmatrix} 1 & & & & & \\ & \ddots & & & & \\ & & 1 & & & \\ & & & 0 & & 1 \\ & & & & 1 & \\ & & & & & \ddots \\ & & & & & & 1 \\ & & & & & & & 0 \\ & & & & & & & & 1 \\ & & & & & & & & & \ddots \\ & & & & & & & & & & 1 \end{pmatrix}$$

erreichen, wobei die 0-Einträge gerade an der  $i$ -ten und  $j$ -ten Stelle stehen. Sind solche Zeilenvertauschungen notwendig, so kann man die benötigten Matrizen  $P_1, \dots, P_l$  ebenfalls zu einer Matrix  $P = P_l \cdot \dots \cdot P_1$  aufmultiplizieren ( $P$  nennt man *Permutationsmatrix*).

<sup>1</sup>tatsächlich existiert für allgemeine invertierbare Matrizen im Allgemeinen keine  $LR$ -Zerlegung

Diese Permutation muss dabei bei der Konstruktion der Matrix  $L$  berücksichtigt werden. Um dies zu erklären, verwenden wir die folgende Notation:

Es sei  $A_m$  die Matrix, die nach dem  $m$ -ten Schritt des Algorithmus erzeugt wurde (also  $A_0 = A$  und  $A_k = R$ , wenn  $k$  die Gesamtanzahl der Schritte im Algorithmus ist). Es sei  $P^{(m)} = P_m \cdots P_1$  die aufmultiplizierte Matrix der bis zum Schritt  $m$  angefallenen Permutationen, wobei wir zur einfacheren Zählung  $P_l = \text{Id}$  (Einheitsmatrix) setzen, falls keine Permutation durchgeführt wurde. Schließlich sei  $L^{(m)}$  die untere Dreiecksmatrix mit

$$P^{(m)}A = L^{(m)}A_m. \quad (2.7)$$

Falls bis zum  $m$ -ten Schritt keine Permutation durchgeführt wurde, gilt wie oben  $L^{(m)} = F_1^{-1} \cdot F_2^{-1} \cdots F_m^{-1}$ .

Nehmen wir nun an, dass vor der Durchführung des  $m + 1$ -ten Schritts eine Pivotierung durchgeführt wird, bei der die  $i$ -te mit der  $j$ -ten Zeile vertauscht wird. Bezeichnen wir die zugehörige Permutationsmatrix mit  $P_{m+1}$ , so wird die Matrix  $A_m$  in (2.7) also verändert zu  $\tilde{A}_m = P_{m+1}A_m$ . Aus der Konstruktion von  $P^{(m)}$  folgt zudem  $P^{(m+1)} = P_{m+1}P^{(m)}$ . Damit (2.7) nach der Pivotierung weiterhin gültig ist, muss die Matrix  $L^{(m)}$  also durch eine Matrix  $\tilde{L}^{(m)}$  ersetzt werden, für die die Gleichung

$$P^{(m+1)}A = \tilde{L}^{(m)}\tilde{A}_m$$

gilt. Aus (2.7) folgt nun

$$P^{(m+1)}A = P_{m+1}P^{(m)}A = P_{m+1}L^{(m)}A_m = P_{m+1}L^{(m)}P_{m+1}^{-1}\tilde{A}_m$$

und wir erhalten  $\tilde{L}^{(m)} = P_{m+1}L^{(m)}P_{m+1}^{-1}$ .

Aus der Form von  $P_{m+1}$  folgt  $P_{m+1}^{-1} = P_{m+1}$ . Die Multiplikation mit  $P_{m+1}$  von rechts bewirkt daher gerade die Vertauschung der  $i$ -ten und  $j$ -ten Spalte in  $L^{(m)}$ . Die Matrix  $\tilde{L}^{(m)}$  entsteht also, indem wir erst die  $i$ -te und  $j$ -te Zeile und dann die  $i$ -te und  $j$ -te Spalte in  $L^{(m)}$  vertauschen.

Beachte dabei, dass  $L^{(m)}$  auf Grund der Form der Matrizen  $F_l^{-1}$  stets 1-Einträge auf der Diagonalen besitzt und darüberhinaus Einträge ungleich Null nur unterhalb der Diagonalen und nur in den Spalten  $1, \dots, j - 1$  besitzt. Weil im Algorithmus zudem stets  $i > j$  gilt, werden durch die Zeilen- und Spaltenvertauschung daher gerade die Elemente  $l_{ik}$  und  $l_{jk}$  für  $k = 1, \dots, j - 1$  vertauscht. Insbesondere ist daher auch  $\tilde{L}^{(m)}$  wieder eine untere Dreiecksmatrix.

Details zur Implementierung dieses Algorithmus werden in den Übungen besprochen. Sie finden sich darüberhinaus z.B. in den Büchern von Deuffhard/Hohmann [2], Schwarz/Köckler [8] oder Stoer [9].

## 2.4 Fehlerabschätzungen und Kondition

Wie bereits im einführenden Kapitel erläutert, können Computer nicht alle reellen Zahlen darstellen, weswegen alle Zahlen intern gerundet werden, damit sie in die endliche Menge der *maschinendarstellbaren Zahlen* passen. Hierdurch entstehen *Rundungsfehler*. Selbst

wenn sowohl die Eingabewerte als auch das Ergebnis eines Algorithmus maschinen­darstellbare Zahlen sind, können solche Fehler auftreten, denn auch die (möglicherweise nicht darstellbaren) Zwischenergebnisse eines Algorithmus werden gerundet. Auf Grund dieser Fehler aber auch wegen Eingabe- bzw. Messfehlern in den vorliegenden Daten oder Fehlern aus vorhergehenden numerischen Rechnungen wird durch einen Algorithmus üblicherweise nicht die exakte Lösung  $x$  des linearen Gleichungssystems

$$Ax = b$$

berechnet, sondern eine Näherungslösung  $\tilde{x}$ . Um dies formal zu fassen, führt man ein „benachbartes“ oder „gestörtes“ Gleichungssystem

$$A\tilde{x} = b + \Delta b$$

ein, für das  $\tilde{x}$  gerade die exakte Lösung ist. Der Vektor  $\Delta b$  heißt hierbei das *Residuum* oder der *Defekt* der Näherungslösung  $\tilde{x}$ . Den Vektor  $\Delta x = \tilde{x} - x$  nennen wir den *Fehler* der Näherungslösung  $\tilde{x}$ . Da Rundung und andere Fehlerquellen i.A. nur kleine Fehler bewirken, ist es gerechtfertigt anzunehmen, dass  $\|\Delta b\|$  „klein“ ist. Das Ziel dieses Abschnittes ist es nun, aus der Größe des Residuums  $\|\Delta b\|$  auf die Größe des Fehlers  $\|\Delta x\|$  zu schließen. Insbesondere wollen wir untersuchen, wie *sensibel* die Größe  $\|\Delta x\|$  von  $\|\Delta b\|$  abhängt, d.h. ob kleine Residuen  $\|\Delta b\|$  große Fehler  $\|\Delta x\|$  hervorrufen können. Diese Analyse ist unabhängig von dem verwendeten Lösungsverfahren, da wir hier nur das Gleichungssystem selbst und kein explizites Verfahren betrachten.

Um diese Analyse durchzuführen, brauchen wir das Konzept der *Matrixnorm*. Man kann Matrixnormen ganz allgemein definieren; für unsere Zwecke ist aber der Begriff der *induzierten Matrixnorm* ausreichend. Wir erinnern zunächst an verschiedene Vektornormen für Vektoren  $x \in \mathbb{R}^n$ . In dieser Vorlesung verwenden wir üblicherweise die *euklidische Norm* oder *2-Norm*

$$\|x\|_2 = \sqrt{\sum_{i=1}^n x_i^2},$$

welche wir meistens einfach mit  $\|x\|$  bezeichnen. Weitere Normen sind die *1-Norm*

$$\|x\|_1 = \sum_{i=1}^n |x_i|$$

und die *Maximums-* oder  *$\infty$ -Norm*

$$\|x\|_\infty = \max_{i=1, \dots, n} |x_i|.$$

Wir werden gleich sehen, dass die zwei letzten Normen im Zusammenhang mit linearen Gleichungssystemen gewisse Vorteile haben.

Für alle Normen im  $\mathbb{R}^n$  kann man eine zugehörige *induzierte Matrixnorm* definieren.

**Definition 2.4** Sei  $\mathbb{R}^{n \times n}$  die Menge der  $n \times n$ -Matrizen und sei  $\|\cdot\|_p$  eine Vektornorm im  $\mathbb{R}^n$ . Dann definieren wir für  $A \in \mathbb{R}^{n \times n}$  die zu  $\|\cdot\|_p$  gehörige *induzierte Matrixnorm*  $\|A\|_p$  als

$$\|A\|_p := \max_{\substack{x \in \mathbb{R}^n \\ \|x\|_p = 1}} \|Ax\|_p = \sup_{x \in \mathbb{R}^n \setminus \{0\}} \frac{\|Ax\|_p}{\|x\|_p}.$$

□

Die Gleichheit der beiden Ausdrücke auf der rechten Seite folgt dabei aus der Beziehung  $\|\alpha x\|_p = |\alpha| \|x\|_p$  für  $\alpha \in \mathbb{R}$ . Dass es sich hierbei tatsächlich um Normen auf dem Vektorraum  $\mathbb{R}^{n \times n}$  handelt, wird in den Übungen bewiesen.

Da im Allgemeinen keine Verwechslungsgefahr besteht, bezeichnen wir die Vektornormen und die von ihnen induzierten Matrixnormen mit dem gleichen Symbol.

**Satz 2.5** Für die zu den obigen Vektornormen gehörigen induzierten Matrixnormen und  $A \in \mathbb{R}^{n \times n}$  gelten die folgenden Gleichungen

$$\|A\|_1 = \max_{j=1, \dots, n} \sum_{i=1}^n |a_{ij}| \quad (\text{Spaltensummennorm})$$

$$\|A\|_\infty = \max_{i=1, \dots, n} \sum_{j=1}^n |a_{ij}| \quad (\text{Zeilensummennorm})$$

$$\|A\|_2 = \sqrt{\rho(A^T A)} \quad (\text{Spektralnorm}),$$

wobei  $\rho(A^T A)$  den maximalen Eigenwert der symmetrischen und positiv definiten Matrix  $A^T A$  bezeichnet.

**Beweis:** Wir beweisen die Gleichungen, indem wir die entsprechenden Ungleichungen beweisen.

„ $\|A\|_1$ “: Für einen beliebigen Vektor  $x \in \mathbb{R}^n$  gilt

$$\|Ax\|_1 = \sum_{i=1}^n \left| \sum_{j=1}^n a_{ij} x_j \right| \leq \sum_{i=1}^n \sum_{j=1}^n |a_{ij}| |x_j| = \sum_{j=1}^n |x_j| \sum_{i=1}^n |a_{ij}|.$$

Sei nun  $j^*$  der Index, für den die innere Summe maximal wird, also

$$\sum_{i=1}^n |a_{ij^*}| = \max_{j=1, \dots, n} \sum_{i=1}^n |a_{ij}|.$$

Dann gilt für Vektoren mit  $\|x\|_1 = 1$

$$\sum_{j=1}^n |x_j| \sum_{i=1}^n |a_{ij}| \leq \underbrace{\sum_{j=1}^n |x_j|}_{=1} \sum_{i=1}^n |a_{ij^*}| = \sum_{i=1}^n |a_{ij^*}|,$$

woraus, da  $x$  beliebig war, die Ungleichung

$$\|A\|_1 \leq \sum_{i=1}^n |a_{ij^*}| = \max_{j=1, \dots, n} \sum_{i=1}^n |a_{ij}|$$

folgt. Andererseits gilt für den  $j^*$ -ten Einheitsvektor  $e_{j^*}$

$$\|A\|_1 \geq \|Ae_{j^*}\|_1 = \sum_{i=1}^n \underbrace{\left| \sum_{j=1}^n a_{ij} [e_{j^*}]_j \right|}_{=|a_{ij^*}|} = \sum_{i=1}^n |a_{ij^*}| = \max_{j=1, \dots, n} \sum_{i=1}^n |a_{ij}|$$

und damit die behauptete Gleichung.

„ $\|A\|_\infty$ “: Ähnlich wie für die 1-Norm mit  $x^* = (1, \dots, 1)^T$  an Stelle von  $e_{j^*}$ .

„ $\|A\|_2$ “: Da  $A^T A$  symmetrisch ist, können wir eine Orthonormalbasis von Eigenvektoren  $v_1, \dots, v_n$  wählen, also  $\|v_i\|_2 = 1$  und  $\langle v_i, v_j \rangle = 0$  für  $i \neq j$ . Sei nun  $x \in \mathbb{R}^n$  ein beliebiger Vektor mit Länge 1, dann lässt sich  $x$  als Linearkombination  $x = \sum_{i=1}^n \mu_i v_i$  mit  $\sum_{i=1}^n \mu_i^2 = 1$  schreiben. Seien  $\lambda_i$  die zu den  $v_i$  gehörigen Eigenwerte von  $A^T A$  und sei  $\lambda_{i^*}$  der maximale Eigenwert, also  $\lambda_{i^*} = \rho(A^T A)$ . Dann gilt

$$\begin{aligned} \|Ax\|_2^2 &= \langle Ax, Ax \rangle = \langle A^T Ax, x \rangle = \sum_{i,j=1}^n \mu_i \mu_j \underbrace{\langle A^T A v_i, v_j \rangle}_{=\lambda_i \delta_{ij}} \\ &= \sum_{\substack{i,j=1 \\ i \neq j}}^n \mu_i \mu_j \lambda_i \underbrace{\langle v_i, v_j \rangle}_{=0} + \sum_{i=1}^n \mu_i^2 \lambda_i \underbrace{\langle v_i, v_i \rangle}_{=1} = \sum_{i=1}^n \mu_i^2 \lambda_i. \end{aligned}$$

Damit folgt

$$\|Ax\|_2^2 = \sum_{i=1}^n \mu_i^2 \lambda_i \leq \underbrace{\sum_{i=1}^n \mu_i^2}_{=1} \lambda_{i^*} = \lambda_{i^*},$$

also, da  $x$  beliebig war, auch

$$\|A\|_2^2 \leq \lambda_{i^*} = \rho(A^T A).$$

Andererseits gilt die Ungleichung

$$\|A\|_2^2 \geq \|Av_{i^*}\|_2^2 = \langle A^T Av_{i^*}, v_{i^*} \rangle = \lambda_{i^*} \underbrace{\langle v_{i^*}, v_{i^*} \rangle}_{=1} = \lambda_{i^*} = \rho(A^T A).$$

□

Für jede Matrixnorm können wir die zugehörige *Kondition* einer invertierbaren Matrix definieren.

**Definition 2.6** Für eine gegebene Matrixnorm  $\|\cdot\|_p$  ist die *Kondition* einer invertierbaren Matrix  $A$  (bzgl.  $\|\cdot\|_p$ ) definiert durch

$$\text{cond}_p(A) := \|A\|_p \|A^{-1}\|_p.$$

□

Wenn wir das Verhältnis zwischen dem Fehler  $\Delta x$  und dem Residuum  $\Delta b$  betrachten, können wir entweder die *absoluten Größen* dieser Werte, also  $\|\Delta x\|_p$  und  $\|\Delta b\|_p$  betrachten, oder, was oft sinnvoller ist, die *relativen Größen*  $\|\Delta x\|_p / \|x\|_p$  und  $\|\Delta b\|_p / \|b\|_p$ . Der folgende Satz zeigt, wie man den Fehler durch das Residuum abschätzen kann.

**Satz 2.7** Sei  $\|\cdot\|_p$  eine Vektornorm mit zugehöriger (und gleich bezeichneter) induzierter Matrixnorm. Sei  $A \in \mathbb{R}^{n \times n}$  eine gegebene invertierbare Matrix und  $b, \Delta b \in \mathbb{R}^n$  gegebene Vektoren. Seien  $x, \tilde{x} \in \mathbb{R}^n$  die Lösungen der linearen Gleichungssysteme

$$Ax = b \quad \text{und} \quad A\tilde{x} = b + \Delta b.$$

Dann gelten für den Fehler  $\Delta x = \tilde{x} - x$  die Abschätzungen

$$\|\Delta x\|_p \leq \|A^{-1}\|_p \|\Delta b\|_p \quad (2.8)$$

und

$$\frac{\|\Delta x\|_p}{\|x\|_p} \leq \text{cond}_p(A) \frac{\|\Delta b\|_p}{\|b\|_p}. \quad (2.9)$$

**Beweis:** Seien  $C \in \mathbb{R}^{n \times n}$  und  $y \in \mathbb{R}^n$  eine beliebige Matrix und ein beliebiger Vektor. Dann gilt nach Übungsaufgabe 6(b) (Blatt 2)

$$\|Cy\|_p \leq \|C\|_p \|y\|_p. \quad (2.10)$$

Schreiben wir  $\tilde{x} = x + \Delta x$ , und ziehen die Gleichung

$$Ax = b$$

von der Gleichung

$$A(x + \Delta x) = b + \Delta b$$

ab, so erhalten wir

$$A\Delta x = \Delta b.$$

Weil  $A$  invertierbar ist, können wir die Gleichung auf beiden Seiten von links mit  $A^{-1}$  multiplizieren und erhalten so

$$\Delta x = A^{-1}\Delta b.$$

Daraus folgt

$$\|\Delta x\|_p = \|A^{-1}\Delta b\|_p \leq \|A^{-1}\|_p \|\Delta b\|_p,$$

wobei wir (2.10) mit  $C = A^{-1}$  und  $y = \Delta b$  benutzt haben. Dies zeigt (2.8). Aus (2.10) mit  $C = A$  und  $y = x$  folgt

$$\|b\|_p = \|Ax\|_p \leq \|A\|_p \|x\|_p,$$

und damit

$$\frac{1}{\|x\|_p} \leq \frac{\|A\|_p}{\|b\|_p}.$$

Wenn wir erst diese Ungleichung und dann (2.8) anwenden, erhalten wir

$$\frac{\|\Delta x\|_p}{\|x\|_p} \leq \frac{\|A\|_p \|\Delta x\|_p}{\|b\|_p} \leq \frac{\|A\|_p \|A^{-1}\|_p \|\Delta b\|_p}{\|b\|_p} = \text{cond}_p(A) \frac{\|\Delta b\|_p}{\|b\|_p},$$

also (2.9). □

Für Matrizen, deren Kondition  $\text{cond}_p(A)$  groß ist, können sich kleine Fehler im Vektor  $b$  (bzw. Rundungsfehler im Verfahren) zu großen Fehlern im Ergebnis  $x$  verstärken. Man spricht in diesem Fall von *schlecht konditionierten* Matrizen. Das folgende Beispiel illustriert diesen Sachverhalt.

**Beispiel 2.8** Betrachte das Gleichungssystem  $Ax = b$  mit

$$A = \begin{pmatrix} 1 & 0 \\ 1000 & 1 \end{pmatrix} \quad \text{und} \quad b = \begin{pmatrix} 0.001 \\ 1 \end{pmatrix}.$$

Durch Nachrechnen sieht man leicht, dass die Lösung gegeben ist durch  $x = (0.001, 0)^T$ . Ebenso überprüft man leicht, dass

$$A^{-1} = \begin{pmatrix} 1 & 0 \\ -1000 & 1 \end{pmatrix}$$

gilt. In der Zeilensummennorm gilt also  $\|A\|_\infty = 1001$  und  $\|A^{-1}\|_\infty = 1001$  und damit

$$\text{cond}_\infty(A) = 1001 \cdot 1001 \approx 1\,000\,000.$$

Für die gestörte rechte Seite  $\tilde{b} = (0.002, 1)^T$ , d.h. für  $\Delta b = (0.001, 0)^T$  erhält man die Lösung  $\tilde{x} = (0.002, -1)^T$ . Es gilt also  $\Delta x = (0.001, -1)^T$  und damit

$$\frac{\|\Delta x\|_\infty}{\|x\|_\infty} = \frac{1}{0.001} = 1000 \quad \text{und} \quad \frac{\|\Delta b\|_\infty}{\|b\|_\infty} = \frac{0.001}{1} = 0.001.$$

Der relative Fehler 0.001 in  $\tilde{b}$  wird also um den Faktor 1 000 000 zu dem relativen Fehler 1000 in  $\tilde{x}$  vergrößert, was fast exakt der Kondition  $\text{cond}_\infty(A)$  entspricht.  $\square$

Bei schlecht konditionierten Matrizen können sich Rundungsfehler im Algorithmus ähnlich stark wie Fehler in der rechten Seite  $b$  auswirken. Ein wichtiges Kriterium beim Entwurf eines Lösungsverfahrens ist es, dass das Verfahren auch für schlecht konditionierte Matrizen noch zuverlässig funktioniert. Beim Gauß-Verfahren kann man dazu die Auswahl der Pivotelemente so gestalten, dass möglichst wenig Rundungsfehler auftreten.

Hierzu muss man sich überlegen, welche Operationen in der Gauß-Elimination besonders fehleranfällig sind; dies kann man sehr ausführlich und formal durchführen, wir werden uns hier aber auf ein eher heuristisches Kriterium beschränken: Die Rechenoperationen in der Gauß-Elimination sind gegeben durch die Subtraktionen

$$a_{ik} - \frac{a_{ij}}{a_{jj}} a_{jk}, \quad b_i - \frac{a_{ij}}{a_{jj}} b_j$$

Im Prinzip können wir im Schritt (1a) des Algorithmus eine beliebige andere Zeile mit der  $j$ -ten Zeile vertauschen, solange diese die gleiche Anzahl von führenden Nulleinträgen besitzt, was gerade für die Zeilen  $j, \dots, n$  gilt. Dies gibt uns die Freiheit, den Zeilenindex „ $j$ “ durch einen beliebigen Index  $p \in \{j, \dots, n\}$  zu ersetzen, was im Algorithmus durch Tauschen der  $j$ -ten mit der  $p$ -ten Zeile realisiert wird. Beachte, dass das „ $j$ “ in „ $a_{ij}$ “ der Spaltenindex des zu eliminierenden Elementes ist, der sich durch die Vertauschung nicht ändert; wir können also durch Zeilenvertauschung nur die Elemente  $a_{jj}$ ,  $a_{jk}$  und  $b_j$  oder anders gesagt gerade die Brüche  $a_{jk}/a_{jj}$  und  $b_j/a_{jj}$  beeinflussen.

Die wesentliche Quelle für Rundungsfehler in einer Subtraktion „ $c - d$ “ im Computer entsteht, wenn die zu berechnende Differenz betragsmäßig klein ist und die einzelnen Terme  $c$  und  $d$  im Vergleich dazu betragsmäßig groß sind. Um dies zu veranschaulichen nehmen

wir an, dass wir im Dezimalsystem auf 5 Stellen genau rechnen. Wenn wir nun die Zahlen 1,234 von der Zahl 1,235 subtrahieren, so erhalten wir das richtige Ergebnis 0,001, wenn wir aber die Zahl 1000,234 von der Zahl 1000,235 subtrahieren, so wird nach interner Rundung auf 5 Stellen die Rechnung  $1000,2 - 1000,2 = 0$  ausgeführt, was zu einem deutlichen Fehler führt (dieser spezielle Fehler wird auch „Auslöschung“ genannt). Die Strategie, solche Fehler in der Gauß–Elimination so weit wie möglich zu vermeiden, besteht nun darin, den Zeilenindex  $p$  bei der Pivotierung so auszuwählen, dass die zu subtrahierenden Ausdrücke betragsmäßig klein sind, ein Verfahren, das man *Pivotsuche* nennt. Da wir nur die Brüche  $a_{jk}/a_{jj}$  ( $k = j, \dots, n$ ) und  $b_j/a_{jj}$  beeinflussen können, sollte man  $p$  dazu also so wählen, dass eben diese Brüche im Betrag möglichst klein werden. Eine einfache Variante, die sich oft in der Literatur findet, besteht darin, das *Pivotelement*  $a_{pj}$  (also den Nenner der Brüche) so zu wählen, dass  $|a_{pj}|$  maximal wird. Im folgenden Algorithmus 2.9 verwenden wir eine etwas aufwändigere Strategie, bei der auch die Zähler der auftauchenden Brüche berücksichtigt werden.

**Algorithmus 2.9 (Gauß–Elimination mit Pivotsuche)**

Gegeben sei eine Matrix  $A \in \mathbb{R}^{n \times n}$  und ein Vektor  $b \in \mathbb{R}^n$ .

(1) Für  $j$  von 1 bis  $n - 1$  ( $j =$  Spaltenindex des zu eliminierenden Eintrags)

(1a) Wähle aus den Zeilenindizes  $p \in \{j, \dots, n \mid a_{pj} \neq 0\}$  denjenigen aus, für den der Ausdruck

$$K(p) = \max \left\{ \max_{k=j, \dots, n} \frac{|a_{pk}|}{|a_{pj}|}, \frac{|b_p|}{|a_{pj}|} \right\}$$

minimal wird. Falls  $p \neq j$  vertausche  $a_{jk}$  und  $a_{pk}$  für  $k = j, \dots, n$  sowie  $b_p$  und  $b_j$

(2) Für  $i$  von  $n$  bis  $j + 1$  (rückwärts zählend)

( $i =$  Zeilenindex des zu eliminierenden Eintrags)

Subtrahiere  $a_{ij}/a_{jj}$ -mal die  $j$ -te Zeile von der  $i$ -ten Zeile:

Setze  $\alpha := a_{ij}/a_{jj}$  und berechne

$$a_{ik} := a_{ik} - \alpha a_{jk} \quad \text{für } k = j, \dots, n, \quad b_i := b_i - \alpha b_j$$

Ende der  $i$ -Schleife

(3) Ende der  $j$ -Schleife

□

Die hier verwendete Form der Pivotsuche wird *Spaltenpivotsuche* genannt, da in jedem Schritt innerhalb der  $j$ -ten Spalte nach dem besten Pivotelement  $a_{pj}$  gesucht wird. Eine erweiterte Form ist die *vollständige* oder *totale Pivotsuche* bei der auch in den Zeilen gesucht wird und dann gegebenenfalls auch Spaltenvertauschungen durchgeführt werden. Gute Implementierungen der Gauß–Elimination verwenden immer solche Pivotsuchmethoden. Diese bietet eine Verbesserung der Robustheit aber keinen vollständigen Schutz gegen große Fehler bei schlechter Kondition — aus prinzipiellen mathematischen Gründen, wie wir im nächsten Abschnitt näher erläutern werden.



Eine allgemeinere Strategie zur Vermeidung schlechter Kondition ist die sogenannte *Präkonditionierung*, bei der eine Matrix  $P \in \mathbb{R}^{n \times n}$  gesucht wird, für die die Kondition von  $PA$  kleiner als die von  $A$  ist, so dass dann das besser konditionierte Problem  $PAx = Pb$  gelöst werden kann. Wir kommen hierauf bei der Betrachtung iterativer Verfahren zurück.

Eine weitere Strategie zur Behandlung schlecht konditionierter Gleichungssysteme, die wir nun genauer untersuchen wollen, ist die *QR-Faktorisierung* (oder *QR-Zerlegung*) einer Matrix.

## 2.5 QR-Faktorisierung

Die *LR-Zerlegung*, die explizit oder implizit Grundlage der bisher betrachteten Lösungsverfahren war, hat unter Konditions-Gesichtspunkten einen wesentlichen Nachteil: Es kann nämlich passieren, dass die einzelnen Matrizen  $L$  und  $R$  der Zerlegung deutlich größere Kondition haben als die zerlegte Matrix  $A$ .

**Beispiel 2.10** Betrachte die Matrix

$$A = \begin{pmatrix} 0.001 & 0.001 \\ 1 & 2 \end{pmatrix} \quad \text{mit} \quad A^{-1} = \begin{pmatrix} 2000 & -1 \\ -1000 & 1 \end{pmatrix}$$

und *LR-Faktorisierung*

$$L = \begin{pmatrix} 1 & 0 \\ 1000 & 1 \end{pmatrix}, \quad R = \begin{pmatrix} 0.001 & 0.001 \\ 0 & 1 \end{pmatrix}.$$

Wegen

$$L^{-1} = \begin{pmatrix} 1 & 0 \\ -1000 & 1 \end{pmatrix} \quad \text{und} \quad R^{-1} = \begin{pmatrix} 1000 & -1 \\ 0 & 1 \end{pmatrix}$$

gilt

$$\text{cond}_\infty(A) \approx 6000, \quad \text{cond}_\infty(L) \approx 1000000 \quad \text{und} \quad \text{cond}_\infty(R) \approx 1000.$$

Die  $\infty$ -Kondition von  $L$  ist also etwa 166-mal so groß wie die von  $A$ . □

Selbst wenn wir eventuelle Fehler in der Berechnung von  $R$  und  $L$  vernachlässigen oder z.B. durch geschickte Pivotsuche vermindern, kann die schlechte Konditionierung von  $R$  und  $L$  durch die Verstärkung der beim Rückwärts- und Vorwärtseinsetzen auftretenden Rundungsfehler zu großen Fehlern  $\Delta x$  führen, besonders wenn die Matrix  $A$  selbst bereits schlecht konditioniert ist. Bei der *LR-Faktorisierung* kann es also passieren, dass die Kondition der Teilprobleme, die im Algorithmus auftreten, deutlich schlechter ist als die des Ausgangsproblems. Beachte, dass die Kondition des Ausgangsproblems nur von der Problemstellung abhängt, die der Teilprobleme aber von dem verwendeten Algorithmus, weswegen diese auch als *numerische Kondition* bezeichnet werden.

Um die numerische Kondition zu verringern, wollen wir nun eine andere Form der Zerlegung betrachten, bei der die Kondition der einzelnen Teilprobleme (bzw. der zugehörigen Matrizen) nicht größer ist als die des Ausgangsproblems (also der Ausgangsmatrix  $A$ ).

Hierzu verwenden wir die folgenden Matrizen.

**Definition 2.11** Eine Matrix  $Q \in \mathbb{R}^{n \times n}$  heißt *orthogonal*, falls  $QQ^T = 1$  ist<sup>2</sup>.  $\square$

Unser Ziel ist nun ein Algorithmus, mit dem eine Matrix  $A$  in ein Produkt  $QR$  zerlegt wird, wobei  $Q$  eine orthogonale Matrix ist und  $R$  eine obere Dreiecksmatrix.

Offenbar ist ein Gleichungssystem der Form  $Qy = b$  leicht zu lösen, indem man die Matrixmultiplikation  $y = Q^T b$  durchführt. Deswegen kann man das Gleichungssystem  $Ax = b$  wie bei der  $LR$ -Faktorisierung leicht durch Lösen der Teilsysteme  $Qy = b$  und  $Rx = y$  lösen.

Bevor wir den entsprechenden Algorithmus herleiten, wollen wir beweisen, dass bei dieser Form der Zerlegung die Kondition tatsächlich erhalten bleibt — zumindest für die euklidische Norm.

**Satz 2.12** Sei  $A \in \mathbb{R}^{n \times n}$  eine invertierbare Matrix mit  $QR$ -Zerlegung. Dann gilt

$$\text{cond}_2(Q) = 1 \quad \text{und} \quad \text{cond}_2(R) = \text{cond}_2(A).$$

**Beweis:** Da  $Q$  orthogonal ist, gilt  $Q^{-1} = Q^T$ . Daraus folgt für beliebige Vektoren  $x \in \mathbb{R}^n$

$$\|Qx\|_2^2 = \langle Qx, Qx \rangle_2 = \langle Q^T Qx, x \rangle_2 = \langle x, x \rangle_2 = \|x\|_2^2,$$

also auch  $\|Qx\|_2 = \|x\|_2$ . Damit folgt für invertierbare Matrizen  $B \in \mathbb{R}^{n \times n}$

$$\|QB\|_2 = \max_{\|x\|_2=1} \|QBx\|_2 = \max_{\|x\|_2=1} \|Q(Bx)\|_2 = \max_{\|x\|_2=1} \|Bx\|_2 = \|B\|_2$$

und mit  $Qx = y$  auch

$$\|BQ\|_2 = \max_{\|x\|_2=1} \|BQx\|_2 = \max_{\|Q^T y\|_2=1} \|By\|_2 = \max_{\|y\|_2=1} \|By\|_2 = \|B\|_2,$$

da mit  $Q$  auch  $Q^T = Q^{-1}$  orthogonal ist. Also folgt

$$\text{cond}_2(Q) = \|Q\|_2 \|Q^{-1}\|_2 = \|Q\|_2 \|Q^T\|_2 = \|Q\|_2 \|Q\|_2 = 1$$

und

$$\begin{aligned} \text{cond}_2(R) &= \text{cond}_2(Q^T A) = \|Q^T A\|_2 \|(Q^T A)^{-1}\|_2 = \|Q^T A\|_2 \|A^{-1} Q\|_2 \\ &= \|A\|_2 \|A^{-1}\|_2 = \text{cond}_2(A). \end{aligned}$$

$\square$

Zwar gilt dieser Satz für andere Matrixnormen nicht, da für je zwei Vektornormen aber Abschätzungen der Form  $\|x\|_p \leq C_{p,q} \|x\|_q$  gelten, ist zumindest eine extreme Verschlechterung der numerischen Kondition auch bezüglich anderer induzierter Matrixnormen ausgeschlossen.

<sup>2</sup>Das komplexe Gegenstück hierzu sind die unitären Matrizen, mit denen sich all das, was wir hier im Reellen machen, auch im Komplexen durchführen lässt

**Beispiel 2.13** Als Beispiel betrachten wir noch einmal das Gleichungssystem aus Beispiel 2.10. Eine  $QR$ -Zerlegung von  $A$  ist gegeben durch

$$Q = \begin{pmatrix} -0.001 & -1 \\ -1 & 0.001 \end{pmatrix} \quad \text{und} \quad R = \begin{pmatrix} -1 & -2 \\ 0 & 0.001 \end{pmatrix}$$

mit

$$Q^{-1} = \begin{pmatrix} -0.001 & -1 \\ -1 & 0.001 \end{pmatrix} \quad \text{und} \quad R^{-1} = \begin{pmatrix} -1 & -2000 \\ 0 & 1000 \end{pmatrix}.$$

Hier gilt also  $\text{cond}_\infty(Q) \approx 1$  und  $\text{cond}_\infty(R) \approx 6000$ , d.h. es entsteht auch bzgl. der  $\infty$ -Kondition keine wesentliche Verschlechterung der Kondition.  $\square$

Die Idee der Algorithmen zur  $QR$ -Zerlegung liegt nun darin, die Spalten der Matrix  $A$  als Vektoren aufzufassen und durch orthogonale Transformationen auf die gewünschte Form zu bringen. Orthogonale Transformationen sind dabei gerade die linearen Transformationen, die sich durch orthogonale Matrizen darstellen lassen. Geometrisch sind dies die Transformationen, die die (euklidische) Länge des transformierten Vektors sowie den Winkel zwischen zwei Vektoren unverändert lassen — nichts anderes haben wir im Beweis von Satz 2.12 ausgenutzt.

Zur Realisierung eines solchen Algorithmus bieten sich zwei mögliche Transformationen an: Drehungen und Spiegelungen. Wir wollen hier den nach seinem Erfinder benannten *Householder-Algorithmus* herleiten, der auf Basis von Spiegelungen funktioniert<sup>3</sup>. Wir veranschaulichen die Idee zunächst geometrisch: Sei  $a_j \in \mathbb{R}^n$  die  $j$ -te Spalte der Matrix  $A$ . Wir wollen eine Spiegelung  $H^{(j)}$  finden, die  $a_j$  auf einen Vektor der Form  $a_j^{(j)} = \underbrace{(*, *, \dots, *)}_j, 0)^T$  bringt. Der Vektor soll also in die Ebene  $E_j = \text{span}(e_1, \dots, e_j)$  gespiegelt werden.

Um diese Spiegelung zu konstruieren, betrachten wir allgemeine Spiegelmatrizen der Form

$$H = H(v) = \text{Id} - \frac{2vv^T}{v^T v}$$

wobei  $v \in \mathbb{R}^n$  ein beliebiger Vektor ist (beachte, dass dann  $vv^T \in \mathbb{R}^{n \times n}$  und  $v^T v \in \mathbb{R}$  ist). Diese Matrizen heißen nach dem Erfinder des zugehörigen Algorithmus *Householder-Matrizen*. Offenbar ist  $H$  symmetrisch und es gilt

$$HH^T = H^2 = \text{Id} - \frac{4vv^T}{v^T v} + \frac{2vv^T}{v^T v} \frac{2vv^T}{v^T v} = \text{Id},$$

also Orthogonalität. Geometrisch entspricht die Multiplikation mit  $H$  der Spiegelung an der Ebene mit Normalenvektor  $n = v/\|v\|$ .

Um nun die gewünschte Spiegelung in die Ebene  $E_j$  zu realisieren, muss man  $v$  geeignet wählen. Hierbei hilft das folgende Lemma.

<sup>3</sup>ein Algorithmus auf Basis von Drehungen ist der sogenannte Givens-Algorithmus

**Lemma 2.14** Betrachte einen Vektor  $w = (w_1, \dots, w_n)^T \in \mathbb{R}^n$ . Für einen gegebenen Index  $j \in \{1, \dots, n\}$  betrachte

$$\begin{aligned} c &= \operatorname{sgn}(w_j) \sqrt{w_j^2 + w_{j+1}^2 + \dots + w_n^2} \in \mathbb{R} \\ v &= (0, \dots, 0, c + w_j, w_{j+1}, \dots, w_n)^T \\ H &= \operatorname{Id} - \frac{2vv^T}{v^T v} \end{aligned}$$

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

$$Hw = (w_1, w_2, \dots, w_{j-1}, -c, 0, \dots, 0).$$

Darüberhinaus gilt für jeden Vektor  $z \in \mathbb{R}^n$  der Form  $z = (z_1, \dots, z_{j-1}, 0, \dots, 0)$  die Gleichung  $H z = z$ .

**Beweis:** Falls  $v \neq 0$  ist gilt

$$\begin{aligned} \frac{2v^T w}{v^T v} &= \frac{2((c + w_j)w_j + w_{j+1}^2 + \dots + w_n^2)}{c^2 + 2cw_j + w_j^2 + w_{j+1}^2 + \dots + w_n^2} \\ &= \frac{2(cw_j + w_j^2 + w_{j+1}^2 + \dots + w_n^2)}{2cw_j + 2w_j^2 + 2w_{j+1}^2 + \dots + 2w_n^2} = 1. \end{aligned}$$

Hieraus folgt

$$Hw = w - v \frac{2v^T w}{v^T v} = w - v = \begin{pmatrix} w_1 \\ \vdots \\ w_{j-1} \\ w_j \\ w_{j+1} \\ \vdots \\ w_n \end{pmatrix} - \begin{pmatrix} 0 \\ \vdots \\ 0 \\ c + w_j \\ w_{j+1} \\ \vdots \\ w_n \end{pmatrix} = \begin{pmatrix} w_1 \\ \vdots \\ w_{j-1} \\ -c \\ 0 \\ \vdots \\ 0 \end{pmatrix}.$$

Falls  $v = 0$  ist, sieht man sofort, dass  $w_{j+1} = \dots = w_n = 0$  gelten muss. Damit folgt  $c = w_j$ , also  $w_j + c = 2w_j$ , weswegen auch  $w_j = 0$  sein muss. In diesem Fall gilt also bereits  $w_j = w_{j+1} = \dots = w_n = 0$ , so dass die Behauptung mit  $H = \operatorname{Id}$  gilt.

Für die zweite Behauptung verwenden wir, dass für Vektoren  $z$  der angegebenen Form die Gleichung  $v^T z = 0$  gilt, woraus sofort

$$Hz = z - v \frac{2v^T z}{v^T v} = z,$$

also die Behauptung folgt. □

Die Idee des Algorithmus liegt nun nahe:

Wir setzen  $A^{(1)} = A$  und konstruieren im ersten Schritt  $H^{(1)}$  gemäß Lemma 2.14 mit  $j = 1$  und  $w = a_{\cdot 1}^{(1)}$ , der ersten Spalte der Matrix  $A^{(1)}$ . Damit ist dann  $A^{(2)} = H^{(1)}A^{(1)}$  von der Form

$$A^{(2)} = H^{(1)}A^{(1)} = \begin{pmatrix} a_{11}^{(2)} & a_{12}^{(2)} & \cdots & a_{1n}^{(2)} \\ 0 & a_{22}^{(2)} & \cdots & a_{2n}^{(2)} \\ \vdots & \vdots & & \vdots \\ 0 & a_{n2}^{(2)} & \cdots & a_{nn}^{(2)} \end{pmatrix}.$$

Im zweiten Schritt konstruieren wir  $H^{(2)}$  gemäß Lemma 2.14 mit  $j = 2$  und  $w = a_{\cdot 2}^{(2)}$ , der zweiten Spalte der Matrix  $A^{(2)}$ . Da die erste Spalte der Matrix  $A^{(2)}$  die Voraussetzungen an den Vektor  $z$  in Lemma 2.14 erfüllt, folgt die Form

$$A^{(3)} = H^{(2)}A^{(2)} = \begin{pmatrix} a_{11}^{(3)} & a_{12}^{(3)} & a_{13}^{(3)} & \cdots & a_{1n}^{(3)} \\ 0 & a_{22}^{(3)} & a_{23}^{(3)} & \cdots & a_{2n}^{(3)} \\ 0 & 0 & a_{33}^{(3)} & \cdots & a_{3n}^{(3)} \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & a_{n3}^{(3)} & \cdots & a_{nn}^{(3)} \end{pmatrix}.$$

Wenn wir sukzessive fortfahren, erhalten wir nach  $n - 1$  Schritten die gewünschte  $QR$ -Zerlegung mit

$$Q^T = H^{(n-1)} \cdots H^{(1)} \quad \text{und} \quad R = A^{(n)},$$

denn es gilt

$$\begin{aligned} QR &= H^{(1)T} \cdots H^{(n-1)T} A^{(n)} \\ &= H^{(1)T} \cdots H^{(n-2)T} A^{(n-1)} \\ &\quad \vdots \\ &= H^{(1)T} A^{(2)} \\ &= A^{(1)} = A \end{aligned}$$

Beachte, dass die  $QR$ -Faktorisierung *immer* funktioniert, auch wenn  $A$  nicht invertierbar ist, denn die obigen Überlegungen liefern einen konstruktiven Beweis für die Existenz. Die resultierende Matrix  $Q$  ist immer invertierbar, die Matrix  $R$  ist invertierbar genau dann, wenn  $A$  invertierbar ist.

In der folgenden Implementierung dieses Algorithmus berechnen wir zusätzlich zu den Matrizen  $Q^T$  und  $R$  auch den Vektor  $y = Q^T b$ . Die Matrix  $R$  wird hierbei im oberen Dreiecksteil der Matrix  $A$  gespeichert, die 0-Elemente werden also nicht explizit gespeichert. Die Multiplikation  $H^{(j)}w$  wird hier in der Form

$$d = \frac{2}{v^T v}, \quad e = dv^T w, \quad H^{(j)}w = w - ev$$

durchgeführt.

**Algorithmus 2.15** ( $QR$ -Zerlegung mittels Householder-Algorithmus)

**Eingabe:** Matrix  $A = (a_{ij})$ , Vektor  $b = (b_i)$

(0) für  $i$  von 1 bis  $n$   
 setze  $y_i := b_i$   
 für  $j$  von 1 bis  $n$   
 setze  $q_{ij} := 1$ , falls  $i = j$ ;  $q_{ij} := 0$ , falls  $i \neq j$   
 Ende der Schleifen

(1) für  $j$  von 1 bis  $n - 1$   
 setze  $c := \operatorname{sgn}(a_{jj}) \sqrt{\sum_{i=j}^n a_{i,j}^2}$   
 falls  $c = 0$ , fahre fort mit  $j + 1$ ; sonst  
 setze  $a_{jj} := c + a_{jj}$  (die Einträge  $v_i$ ,  $i \geq j$  stehen jetzt in  $a_{ij}$ ,  $i \geq j$ )  
 setze  $d := 1/(ca_{jj})$

*Berechnung von  $H^{(j)} \cdot (H^{(j-1)} \dots H^{(1)})$ :*  
 für  $k$  von 1 bis  $n$   
 setze  $e := d \left( \sum_{i=j}^n a_{ij} q_{ik} \right)$   
 für  $i$  von  $j$  bis  $n$  setze  $q_{ik} := q_{ik} - ea_{ij}$   
 Ende der  $i$  und  $k$  Schleifen

*Berechnung von  $H^{(j)} y^{(j)}$ :*  
 setze  $e := d \left( \sum_{i=j}^n a_{ij} y_i \right)$   
 für  $i$  von  $j$  bis  $n$  setze  $y_i := y_i - ea_{ij}$   
 Ende der  $i$  Schleife

*Berechnung von  $H^{(j)} A^{(j)}$  für die Spalten  $j + 1, \dots, n$ :  
 (die  $j$ -te Spalte wird noch zur Speicherung von  $v_j$  benötigt)*  
 für  $k$  von  $j + 1$  bis  $n$   
 setze  $e := d \left( \sum_{i=j}^n a_{ij} a_{ik} \right)$   
 für  $i$  von  $j$  bis  $n$  setze  $a_{ik} := a_{ik} - ea_{ij}$   
 Ende der  $i$  und  $k$  Schleifen

*Berechnung der  $j$ -ten Spalte von  $H^{(j)} A^{(j)}$ :  
 (hier ändert sich nur das Diagonalelement, vgl. Lemma 2.14)*  
 setze  $a_{jj} = -c$   
 Ende der  $j$  Schleife

**Ausgabe:**  $R = (a_{ij})_{i \leq j}$ ,  $Q^T = (q_{ij})$ ,  $Q^T b = y = (y_i)$ . □

Bei der Lösung eines linearen Gleichungssystems sollte die Invertierbarkeit von  $R$  vor dem Rückwärtseinsetzen getestet werden (eine obere Dreiecksmatrix ist genau dann invertierbar, wenn alle Diagonalelemente ungleich Null sind). Dies kann schon im obigen Algorithmus geschehen, indem überprüft wird, ob die  $c$ -Werte (die gerade die Diagonalelemente von  $R$  bilden) ungleich Null sind.

Die  $QR$ -Zerlegung kann tatsächlich mehr als nur lineare Gleichungssysteme lösen: Wir haben im Abschnitt 2.1 das lineare Ausgleichsproblem kennen gelernt, bei dem  $x \in \mathbb{R}^n$  gesucht ist, so dass der Vektor

$$r = z - \tilde{A}x$$

minimale 2-Norm  $\|r\|_2$  hat, und haben gesehen, dass dieses Problem durch Lösen der Normalgleichungen  $\tilde{A}^T \tilde{A}x = \tilde{A}^T z$  gelöst werden kann. Gerade diese Matrix  $\tilde{A}^T \tilde{A}$  kann aber (bedingt durch ihre Struktur, vgl. Aufgabe 10, Übungsblatt 4) sehr große Kondition haben, so dass es hier erstrebenswert ist, (a) ein robustes Verfahren zu verwenden und (b) die explizite Lösung der Normalgleichungen zu vermeiden. Mit dem QR-Algorithmus kann man beides erreichen, man kann nämlich das Ausgleichsproblem *direkt* lösen.

Die QR-Zerlegung (und auch der obige Algorithmus) kann auf die nichtquadratische Matrix  $\tilde{A}$  mit  $n$  Spalten und  $m > n$  Zeilen angewendet werden, indem man  $j$  in Schritt (1) bis  $n$  und alle anderen Indizes mit Ausnahme von  $k$  in der Berechnung von  $H^{(j)} A^{(j)}$  bis  $m$  laufen lässt. Das Resultat ist dann eine Faktorisierung  $\tilde{A} = QR$  mit

$$R = \begin{pmatrix} R_1 \\ 0 \end{pmatrix}$$

wobei  $R_1 \in \mathbb{R}^{n \times n}$  eine obere Dreiecksmatrix ist. Beachte, dass die Normalgleichungen genau dann lösbar sind, wenn  $\tilde{A}$  vollen Spaltenrang besitzt, was wir annehmen. In diesem Fall ist auch die Matrix  $R_1$  invertierbar.

Wenn man dann  $x$  so wählt, dass der Vektor

$$s = Q^T r = Q^T z - Q^T \tilde{A}x$$

minimale 2-Norm besitzt, dann hat auch  $r$  minimale 2-Norm, da aus der Orthogonalität von  $Q^T$  die Gleichung  $\|r\|_2 = \|s\|_2$  folgt. Wir zerlegen  $s$  in  $s^1 = (s_1, \dots, s_n)^T$  und  $s^2 = (s_{n+1}, \dots, s_m)^T$ . Wegen der Form von  $R = Q^T \tilde{A}$  ist der Vektor  $s^2$  unabhängig von  $x$  und wegen

$$\|s\|_2^2 = \sum_{i=1}^m s_i^2 = \sum_{i=1}^n s_i^2 + \sum_{i=n+1}^m s_i^2 = \|s^1\|_2^2 + \|s^2\|_2^2$$

wird diese Norm genau dann minimal, wenn die Norm  $\|s^1\|_2^2$  minimal wird. Wir suchen also ein  $x \in \mathbb{R}^n$ , so dass

$$\|s^1\|_2 = \|y^1 - R_1 x\|_2$$

minimal wird, wobei  $y^1$  die ersten  $n$  Komponenten des Vektors  $y = Q^T z$  bezeichnet. Da  $R_1$  eine invertierbare obere Dreiecksmatrix ist, kann man durch Rückwärtseinsetzen eine Lösung  $x$  des Gleichungssystems  $R_1 x = y^1$  finden, für die dann

$$\|s^1\|_2 = \|y^1 - R_1 x\|_2 = 0$$

gilt, womit offenbar das Minimum erreicht wird. Zusammenfassend kann man also das Ausgleichsproblem wie folgt mit der QR-Faktorisierung direkt lösen:

**Algorithmus 2.16 (Lösung des Ausgleichsproblems mit QR-Faktorisierung)**

**Eingabe:** Matrix  $\tilde{A} \in \mathbb{R}^{m \times n}$  mit  $m > n$  und (maximalem) Spaltenrang  $n$ , Vektor  $z \in \mathbb{R}^m$

- (1) Berechne Zerlegung  $\tilde{A} = QR$  mit  $R = \begin{pmatrix} R_1 \\ 0 \end{pmatrix}$  und oberer Dreiecksmatrix  $R_1 \in \mathbb{R}^{n \times n}$
- (2) Löse das Gleichungssystem  $R_1 x = y^1$  durch Rückwärtseinsetzen, wobei  $y^1$  die ersten  $n$  Komponenten des Vektors  $y = Q^T z \in \mathbb{R}^m$  bezeichnet

**Ausgabe:** Vektor  $x \in \mathbb{R}^n$ , für den  $\|\tilde{A}x - z\|_2$  minimal wird.  $\square$

Geometrisch passiert hier das Folgende: Das Bild von  $\tilde{A}$  wird durch die orthogonale Transformation  $Q^T$  längentreu auf den Unterraum  $\text{span}(e_1, \dots, e_n)$  abgebildet, in dem wir dann das entstehende Gleichungssystem lösen können, vgl. Abbildung 2.1.

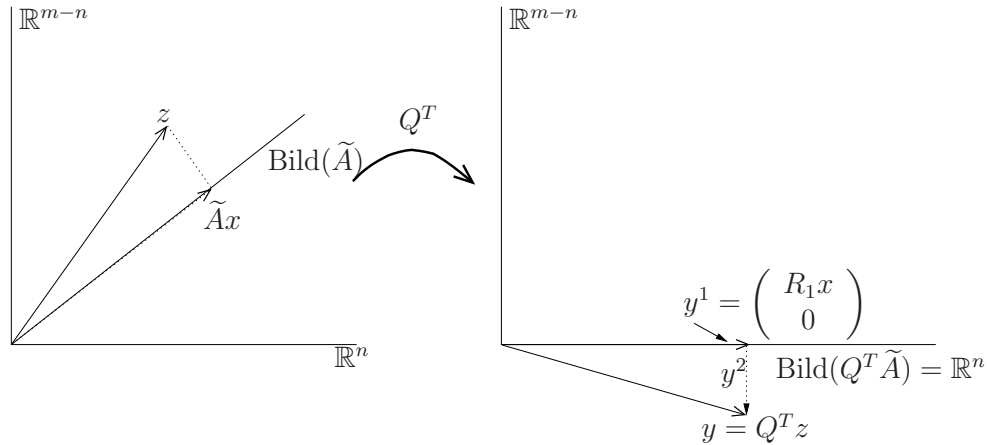


Abbildung 2.1: Veranschaulichung von Algorithmus 2.16

## 2.6 Aufwandsabschätzungen

Ein wichtiger Aspekt bei der Analyse numerischer Verfahren ist es zu untersuchen, wie lange diese Verfahren in der Regel benötigen, um zu dem gewünschten Ergebnis zu kommen. Da dies natürlich entscheidend von der Leistungsfähigkeit des verwendeten Computers abhängt, schätzt man nicht direkt die Zeit ab, sondern die Anzahl der Rechenoperationen, die ein Algorithmus benötigt. Da hierbei die *Gleitkommaoperationen*, also Addition, Multiplikation etc. von reellen Zahlen, die mit Abstand zeitintensivsten Operationen sind, beschränkt man sich in der Analyse üblicherweise auf diese<sup>4</sup>.

Die Verfahren, die wir bisher betrachtet haben, liefern nach endlich vielen Schritten ein Ergebnis (man spricht von *direkten Verfahren*), wobei die Anzahl der Operationen von der Dimension  $n$  der Matrix abhängt. Zur *Aufwandsabschätzung* genügt es also, die Anzahl der Gleitkommaoperationen (in Abhängigkeit von  $n$ ) „abzuzählen“. Wie man dies am geschicktesten macht, hängt dabei von der Struktur des Algorithmus ab. Zudem muss man einige Rechenregeln aus der elementaren Analysis ausnutzen, um die entstehenden Ausdrücke zu vereinfachen. Speziell benötigen wir hier die Gleichungen

$$\sum_{i=1}^n i = \frac{(n+1)n}{2} \quad \text{und} \quad \sum_{i=1}^n i^2 = \frac{1}{3}n^3 + \frac{1}{2}n^2 + \frac{1}{6}n.$$

<sup>4</sup>Tatsächlich sind Multiplikation, Division und die Berechnung von Wurzeln etwas aufwändiger als Addition und Subtraktion, was wir hier aber vernachlässigen werden.



Wir beginnen mit dem **Rückwärtseinsetzen**, und betrachten zunächst die Multiplikationen und Divisionen: Für  $i = n$  muss eine Division durchgeführt werden, für  $i = n - 1$  muss eine Multiplikation und eine Division durchgeführt werden, für  $i = n - 2$  müssen zwei Multiplikationen und eine Division durchgeführt werden, usw. So ergibt sich die Anzahl dieser Operationen als

$$1 + 2 + 3 + \cdots + n = \sum_{i=1}^n i = \frac{(n+1)n}{2} = \frac{n^2}{2} + \frac{n}{2}.$$

Für die Anzahl der Additionen und Subtraktionen zählt man ab

$$0 + 1 + 2 + \cdots + n - 1 = \sum_{i=1}^{n-1} i = \frac{n(n-1)}{2} = \frac{n^2}{2} - \frac{n}{2}.$$

Insgesamt kommt man also auf

$$\frac{n^2}{2} + \frac{n}{2} + \frac{n^2}{2} - \frac{n}{2} = n^2$$

Gleitkommaoperationen. Da das **Vorwärtseinsetzen** völlig analog funktioniert, gilt dafür die gleiche Abschätzung.

Bei der **Gauß-Elimination** betrachten wir hier die Version aus Algorithmus 2.3 ohne Pivotsuche und nehmen den schlechtesten Fall an, nämlich dass Schritt (1b) jedes Mal durchgeführt wird. Wir gehen spaltenweise vor und betrachten die Elemente, die für jedes  $j$  eliminiert werden müssen. Für jedes zu eliminierende Element in der  $j$ -ten Spalte benötigt man je  $n - (j - 1) + 1$  Additionen und Multiplikationen (die „+1“ ergibt sich aus der Operation für  $b$ ) sowie eine Division zur Berechnung von  $\alpha$ , d.h.,  $2(n+2-j)+1$  Operationen. In der  $j$ -ten Spalte müssen dabei  $n-j$  Einträge eliminiert werden, nämlich für  $i = n, \dots, j+1$ . Also ergeben sich für die  $j$ -te Spalte

$$(n-j)(2(n+2-j)+1) = 2n^2 + 4n - 2nj - 2jn - 4j + 2j^2 + n - j = 2j^2 - (4n+5)j + 5n + 2n^2$$

Operationen. Dies muss für die Spalten  $j = 1, \dots, n - 1$  durchgeführt werden, womit wir auf

$$\begin{aligned} & \sum_{j=1}^{n-1} (2j^2 - (4n+5)j + 5n + 2n^2) \\ &= 2 \sum_{j=1}^{n-1} j^2 - (4n+5) \sum_{j=1}^{n-1} j + \sum_{j=1}^{n-1} (5n + 2n^2) \\ &= \frac{2}{3}(n-1)^3 + (n-1)^2 + \frac{1}{3}(n-1) - (4n+5) \frac{(n-1)n}{2} + (n-1)(5n + 2n^2) \\ &= \frac{2}{3}n^3 + \frac{3}{2}n^2 - \frac{13}{6}n \end{aligned}$$

Operationen kommen.

Für die **QR-Faktorisierung mittels Householder-Spiegelungen** betrachten wir hier nur die Berechnung von  $R$  und  $y = Q^T b$ . Für jedes  $j = 1, \dots, n - 1$  muss  $c$  berechnet

werden  $(2(n-j+1))$  Operationen, wobei die Berechnung von  $\text{sgn}$  vernachlässigbar schnell ist), sowie  $a_{jj}$  und  $d$  (weitere 3 Operationen). Für die Berechnung von  $y$  muss zunächst  $e$  berechnet werden ( $2(n-j+1)$  Operationen) und dann  $y$  (nochmal  $2(n-j+1)$  Operationen), also insgesamt  $4(n-j+1)$  Operationen. Für die Berechnung von  $R$  schließlich müssen die gleichen Berechnungen  $(n-j)$ -mal durchgeführt werden, also  $(n-j)4(n-j+1) = 4n^2 + 4j^2 - 8nj + 4n - 4j$  Operationen. Insgesamt kommt man so für jedes  $j$  auf

$$\begin{aligned} & 2(n-j+1) + 3 + 4(n-j+1) + 4n^2 + 4j^2 - 8nj + 4n - 4j \\ &= 4j^2 - (8n+10)j + 4n^2 + 10n + 9 \end{aligned}$$

Operationen, insgesamt also

$$\begin{aligned} & \sum_{j=1}^{n-1} (4j^2 - (8n+10)j + 4n^2 + 10n + 9) \\ &= \frac{4}{3}(n-1)^3 + 2(n-1)^2 + \frac{2}{3}(n-1) \\ & \quad - (8n+10)\frac{n(n-1)}{2} + 4n^2(n-1) + 10n(n-1) + 9(n-1) \\ &= \frac{4}{3}n^3 + 3n^2 + \frac{14}{3}n - 9 \end{aligned}$$

Operationen.

Zur vollständigen **Lösung eines linearen Gleichungssystems** müssen wir nun einfach die Operationen der Teilalgorithmen aufaddieren.

Für den Gauß-Algorithmus kommt man so auf

$$\frac{2}{3}n^3 + \frac{3}{2}n^2 - \frac{13}{6}n + n^2 = \frac{2}{3}n^3 + \frac{5}{2}n^2 - \frac{13}{6}n$$

Operationen und für die  $QR$ -Zerlegung auf

$$\frac{4}{3}n^3 + 3n^2 + \frac{14}{3}n - 9 + n^2 = \frac{4}{3}n^3 + 4n^2 + \frac{14}{3}n - 9$$

Operationen. Berücksichtigt man, dass für große  $n$  die führenden " $n^3$ -Terme" dominant werden, so ergibt sich, dass die Gauß-Elimination etwa 2-mal so schnell ist wie die  $QR$ -Faktorisierung.

Um einen Eindruck von den tatsächlichen Rechenzeiten zu bekommen, nehmen wir an, dass wir einen PC verwenden, der mit einem guten Core i7 Prozessor etwa eine Leistung von 33 GFLOPS (FLOPS = floating point operations per second; GFLOPS = GigaFLOPS =  $10^9$  Flops) schafft. Nehmen wir weiterhin (sehr optimistisch) an, dass wir Implementierungen der obigen Algorithmen haben, die diese Leistung optimal ausnutzen. Dann ergeben sich für  $n \times n$  Gleichungssysteme die folgenden Rechenzeiten

n	Gauß		QR	
1000	20.25 ms		40.49 ms	
10000	20.21 s		40.41 s	
100000	5.61 h		11.22 h	
500000	29.23 d		58.46 d	

Spätestens im letzten Fall  $n = 500\,000$  sind die Zeiten kaum mehr akzeptabel: Wer will schon mehr als vier Wochen auf ein Ergebnis warten?

Zum Abschluss dieses Abschnitts wollen wir noch ein größeres Konzept der Aufwandsabschätzung einführen, das für praktische Zwecke oft ausreicht. Oft ist man nämlich nicht an der exakten Zahl der Operationen für ein gegebenes  $n$  interessiert, sondern nur an einer Abschätzung für große Dimensionen. Genauer möchte man wissen, wie schnell der Aufwand in Abhängigkeit von  $n$  wächst, d.h. wie er sich *asymptotisch* für  $n \rightarrow \infty$  verhält. Man spricht dann von der *Ordnung* eines Algorithmus.

**Definition 2.17** Ein Algorithmus hat die *Ordnung*  $O(n^q)$ , wenn  $q > 0$  die minimale Zahl ist, für die es eine Konstante  $C > 0$  gibt, so dass der Algorithmus für alle  $n \in \mathbb{N}$  weniger als  $Cn^q$  Operationen benötigt. □

Diese Zahl  $q$  ist aus den obigen Aufwandsberechnungen leicht abzulesen: Es ist gerade die höchste auftretende Potenz von  $n$ . Somit haben Vorwärts- und Rückwärtseinsetzen die Ordnung  $O(n^2)$ , während Gauß-, Choleski-Verfahren und QR-Verfahren die Ordnung  $O(n^3)$  besitzen.

## 2.7 Iterative Verfahren

Wir haben im letzten Abschnitt gesehen, dass die bisher betrachteten Verfahren — die sogenannten *direkten Verfahren* — die Ordnung  $O(n^3)$  besitzen: Wenn sich also  $n$  verzehnfacht, so vertausendfacht sich die Anzahl der Operationen und damit die Rechenzeit. Für große Gleichungssysteme mit mehreren 100 000 Unbekannten, die in der Praxis durchaus auftreten, führt dies wie oben gesehen zu unakzeptabel hohen Rechenzeiten.

Eine Klasse von Verfahren, die eine niedrigere Ordnung hat, sind die *iterativen Verfahren*. Allerdings zahlt man für den geringeren Aufwand einen Preis: Man kann bei diesen Verfahren nicht mehr erwarten, eine (bis auf Rundungsfehler) exakte Lösung zu erhalten, sondern muss von vornherein eine gewisse Ungenauigkeit im Ergebnis in Kauf nehmen.

Das Grundprinzip iterativer Verfahren funktioniert dabei wie folgt:

Ausgehend von einem Startvektor  $x^{(0)}$  berechnet man mittels einer Rechenvorschrift  $\Phi : \mathbb{R}^n \rightarrow \mathbb{R}^n$  iterativ eine Folge von Vektoren

$$x^{(i+1)} = \Phi(x^{(i)}), \quad i = 0, 1, 2, \dots,$$

die für  $i \rightarrow \infty$  gegen die Lösung  $x^*$  des Gleichungssystems  $Ax = b$  konvergieren, also  $\lim_{i \rightarrow \infty} \|x^{(i)} - x^*\|_p = 0$ . Wenn die gewünschte Genauigkeit erreicht ist, wird die Iteration abgebrochen und der letzte Wert  $x^{(i)}$  als Näherung des Ergebnisses verwendet.

Bevor wir explizite Beispiele solcher Verfahren betrachten, wollen wir zunächst einen Satz aus der Analysis wiederholen, der bei der Analyse iterativer Verfahren hilfreich ist.

**Satz 2.18 (Banach'scher Fixpunktsatz)** Sei  $A$  eine abgeschlossene Teilmenge eines vollständigen normierten Raumes mit Norm  $\|\cdot\|$  und sei  $\Phi : A \rightarrow A$  eine Kontraktion, d.h. es existiere eine Konstante  $k \in (0, 1)$ , so dass die Ungleichung

$$\|\Phi(x) - \Phi(y)\| \leq k\|x - y\|$$

für alle  $x, y \in A$  gilt. Dann existiert ein eindeutiger Fixpunkt  $x^* \in A$ , gegen den alle Folgen der Form  $x^{(i+1)} = \Phi(x^{(i)})$  mit beliebigen  $x^{(0)} \in A$  konvergieren. Darüberhinaus gelten die *a priori* und *a posteriori* Abschätzungen

$$\|x^{(i)} - x^*\| \leq \frac{k^i}{1-k} \|x^{(1)} - x^{(0)}\| \quad \text{und} \quad \|x^{(i)} - x^*\| \leq \frac{k}{1-k} \|x^{(i)} - x^{(i-1)}\|.$$

**Beweis:** Wir zeigen zunächst, dass jede Folge  $(x^{(i)})_{i \in \mathbb{N}_0}$  der Form  $x^{(i+1)} = \Phi(x^{(i)})$  mit beliebigem  $x^{(0)} \in A$  eine Cauchy-Folge ist: Aus der Kontraktionseigenschaft folgen mit Induktion für beliebige  $i, j \in \mathbb{N}_0$  mit  $j \geq i$  die Abschätzungen

$$\|x^{(j+1)} - x^{(j)}\| \leq k^{j-i} \|x^{(i+1)} - x^{(i)}\| \quad \text{und} \quad \|x^{(i+1)} - x^{(i)}\| \leq k^i \|x^{(1)} - x^{(0)}\| \quad (2.11)$$

Damit gilt

$$\begin{aligned} \|x^{(i+n)} - x^{(i)}\| &= \left\| \sum_{j=i}^{i+n-1} (x^{(j+1)} - x^{(j)}) \right\| \\ &\leq \sum_{j=i}^{i+n-1} \|x^{(j+1)} - x^{(j)}\| \leq \sum_{j=i}^{i+n-1} k^{j-i} \|x^{(i+1)} - x^{(i)}\| \\ &= \frac{1-k^n}{1-k} \|x^{(i+1)} - x^{(i)}\| \leq \frac{1}{1-k} \|x^{(i+1)} - x^{(i)}\| \\ &\leq \frac{k}{1-k} \|x^{(i)} - x^{(i-1)}\| \leq \frac{k^i}{1-k} \|x^{(1)} - x^{(0)}\|, \end{aligned} \quad (2.12)$$

weswegen diese Folge wegen  $k^i \rightarrow 0$  eine Cauchy-Folge ist.

Wir zeigen nun, dass  $x^{(i)}$  gegen einen Fixpunkt von  $\Phi$  konvergiert: Da  $A$  Teilmenge eines vollständigen Raumes ist, existiert ein Grenzwert  $x^*$  dieser Cauchy-Folge, der wegen der Abgeschlossenheit von  $A$  wieder in  $A$  liegt, also  $\lim_{i \rightarrow \infty} x^{(i)} = x^* \in A$ . Da  $\Phi$  eine Kontraktion, also insbesondere stetig ist, folgt

$$\Phi(x^*) = \Phi(\lim_{i \rightarrow \infty} x^{(i)}) = \lim_{i \rightarrow \infty} \Phi(x^{(i)}) = \lim_{i \rightarrow \infty} x^{(i+1)} = x^*,$$

also ist  $x^*$  ein Fixpunkt von  $\Phi$ . Es folgt also, dass jede Folge der angegebenen Form gegen einen Fixpunkt von  $\Phi$  konvergiert. Es bleibt die Eindeutigkeit des Fixpunktes zu zeigen: Wir nehmen dazu an, dass zwei Fixpunkte  $x^*, x^{**} \in A$  von  $\Phi$  mit  $x^* \neq x^{**}$ , also  $\|x^* - x^{**}\| > 0$ , existieren. Aus der Kontraktionseigenschaft folgt dann

$$\|x^* - x^{**}\| = \|\Phi(x^*) - \Phi(x^{**})\| \leq k\|x^* - x^{**}\| < \|x^* - x^{**}\|,$$

was ein Widerspruch zu  $\|x^* - x^{**}\| > 0$  ist und daher die Eindeutigkeit zeigt.

Zuletzt zeigen wir die zwei Abschätzungen. Beide folgen aus (2.12) mittels

$$\begin{aligned} \|x^{(i)} - x^*\| &= \lim_{n \rightarrow \infty} \|x^{(i)} - x^{(i+n)}\| \leq \lim_{n \rightarrow \infty} \frac{1}{1-k} \|x^{(i+1)} - x^{(i)}\| \\ &= \frac{k}{1-k} \|x^{(i)} - x^{(i-1)}\| \leq \frac{k^i}{1-k} \|x^{(1)} - x^{(0)}\|. \end{aligned}$$

□

## 2.8 Gauß–Seidel– und Jacobi–Verfahren

Wir wollen nun zwei klassische iterative Verfahren kennen lernen, die beide nach dem gleichen Prinzip funktionieren: Man zerlegt die Matrix  $A$  in eine Differenz zweier Matrizen

$$A = M - N,$$

wobei man annimmt, dass  $M$  leicht (d.h. mit sehr wenig Aufwand) zu invertieren ist. Dann wählt man einen Startvektor  $x^{(0)}$  (z.B. den Nullvektor) und berechnet iterativ

$$x^{(i+1)} = M^{-1}Nx^{(i)} + M^{-1}b, \quad i = 0, 1, 2, \dots \quad (2.13)$$

Wenn die Zerlegung (unter passenden Annahmen an  $A$ ) geeignet gewählt wurde, kann man erwarten, dass die Vektoren  $x_i$  gegen die gesuchte Lösung konvergieren. Präzise ist dies in dem folgenden Lemma beschrieben.

**Lemma 2.19** Gegeben sei das lineare Gleichungssystem  $Ax = b$  mit invertierbarer Matrix  $A$  sowie eine Zerlegung  $A = M - N$  mit invertierbarer Matrix  $M$ . Sei  $\|\cdot\|$  eine Vektornorm mit zugehöriger induzierter Matrixnorm, für die die Abschätzung  $k = \|M^{-1}N\| < 1$  gilt. Dann konvergiert das Verfahren (2.13) für beliebige Startwerte  $x^{(0)}$  gegen die Lösung  $x^*$  des Gleichungssystems und die Iterationsabbildung ist eine Kontraktion bzgl. der induzierten Matrixnorm mit Konstante  $k$ . Darüberhinaus gelten die Abschätzungen

$$\|x^{(i)} - x^*\| \leq \frac{k}{1-k} \|x^{(i)} - x^{(i-1)}\| \leq \frac{k^i}{1-k} \|x^{(1)} - x^{(0)}\|.$$

**Beweis:** Wir zeigen zunächst, dass die Abbildung  $\Phi : \mathbb{R}^n \rightarrow \mathbb{R}^n$  gegeben durch  $\Phi(x) = M^{-1}Nx + M^{-1}b$  eine Kontraktion bezüglich der induzierten Matrixnorm  $\|\cdot\|$  ist: Es gilt

$$\begin{aligned}\|\Phi(x) - \Phi(y)\| &= \|M^{-1}Nx + M^{-1}b - M^{-1}Ny - M^{-1}b\| \\ &= \|M^{-1}N(x - y)\| \leq \|M^{-1}N\| \|x - y\| = k\|x - y\|.\end{aligned}$$

Aus dem Banach'schen Fixpunktsatz 2.18 folgt also, dass die Iteration (2.13) gegen einen eindeutigen Fixpunkt  $x^*$  konvergiert und darüberhinaus die angegebenen Abschätzungen gelten.

Wegen

$$\begin{aligned}\Phi(x^*) = x^* &\Leftrightarrow M^{-1}Nx^* + M^{-1}b = x^* \\ &\Leftrightarrow Nx^* + b = Mx^* \\ &\Leftrightarrow b = (M - N)x^* = Ax^*\end{aligned}$$

ist dieser Fixpunkt tatsächlich die gesuchte Lösung des Gleichungssystems.  $\square$

Bei iterativen Algorithmen brauchen wir noch ein Abbruchkriterium, um zu entscheiden, wann wir die Iteration stoppen. Hier gibt es mehrere Möglichkeiten; ein einfaches aber trotzdem effizientes Kriterium ist es, sich ein  $\varepsilon > 0$  vorzugeben, und die Iteration dann abzubrechen, wenn die Bedingung

$$\|x^{(i+1)} - x^{(i)}\| < \varepsilon \quad (2.14)$$

für eine vorgegebene Vektornorm  $\|\cdot\|$  erfüllt ist. Wenn wir hier die Vektornorm nehmen, für die Lemma 2.19 gilt, so ist mit diesem Kriterium die Genauigkeit

$$\|x^{(i+1)} - x^*\| \leq \frac{k}{1-k}\varepsilon$$

gewährleistet. Auch hier kann man bei Bedarf den relativen Fehler

$$\frac{\|x^{(i+1)} - x^{(i)}\|}{\|x^{(i+1)}\|} < \varepsilon$$

verwenden. Will man bis zum Erreichen der maximal möglichen Rechengenauigkeit iterieren, so wählt man im relativen Abbruchkriterium  $\varepsilon$  als die Maschinengenauigkeit (typischerweise  $10^{-8}$  bei einfacher und  $10^{-16}$  bei doppelter Genauigkeit).

**Beispiel 2.20** Wir illustrieren ein solches Verfahren an dem dreidimensionalen linearen Gleichungssystem mit

$$A = \begin{pmatrix} 15 & 3 & 4 \\ 2 & 17 & 3 \\ 2 & 3 & 21 \end{pmatrix} \quad \text{und} \quad b = \begin{pmatrix} 33 \\ 45 \\ 71 \end{pmatrix}.$$

Als Zerlegung  $A = M - N$  wählen wir

$$M = \begin{pmatrix} 15 & 0 & 0 \\ 0 & 17 & 0 \\ 0 & 0 & 21 \end{pmatrix} \quad \text{und} \quad N = - \begin{pmatrix} 0 & 3 & 4 \\ 2 & 0 & 3 \\ 2 & 3 & 0 \end{pmatrix},$$

d.h. wir zerlegen  $A$  in ihren Diagonalanteil  $M$  und den Nicht–Diagonalanteil  $-N$ . Diagonalmatrizen sind sehr leicht zu invertieren: Man muss einfach jedes Element durch seinen Kehrwert ersetzen, also

$$M^{-1} = \begin{pmatrix} 1/15 & 0 & 0 \\ 0 & 1/17 & 0 \\ 0 & 0 & 1/21 \end{pmatrix}.$$

Damit erhalten wir

$$M^{-1}N = \begin{pmatrix} 0 & -1/5 & -4/15 \\ -2/17 & 0 & -3/17 \\ -2/21 & -1/7 & 0 \end{pmatrix} \quad \text{und} \quad M^{-1}b = \begin{pmatrix} 11/5 \\ 45/17 \\ 71/21 \end{pmatrix}.$$

Wir berechnen nun gemäß der Vorschrift (2.13) die Vektoren  $x^{(1)}, \dots, x^{(10)}$ , wobei wir  $x^{(0)} = (0\ 0\ 0)^T$  setzen. Es ergeben sich (jeweils auf vier Nachkommastellen gerundet)

$$\begin{pmatrix} 2.2000 \\ 2.6471 \\ 3.3810 \end{pmatrix}, \begin{pmatrix} 0.7690 \\ 1.7916 \\ 2.7933 \end{pmatrix}, \begin{pmatrix} 1.0968 \\ 2.0637 \\ 3.0518 \end{pmatrix}, \begin{pmatrix} 0.9735 \\ 1.9795 \\ 2.9817 \end{pmatrix}, \begin{pmatrix} 1.0090 \\ 2.0064 \\ 3.0055 \end{pmatrix}, \\ \begin{pmatrix} 0.9973 \\ 1.9980 \\ 2.9982 \end{pmatrix}, \begin{pmatrix} 1.0009 \\ 2.0006 \\ 3.0005 \end{pmatrix}, \begin{pmatrix} 0.9997 \\ 1.9998 \\ 2.9998 \end{pmatrix}, \begin{pmatrix} 1.0001 \\ 2.0001 \\ 3.0001 \end{pmatrix}, \begin{pmatrix} 1.0000 \\ 2.0000 \\ 3.0000 \end{pmatrix}.$$

□

Je nach Wahl von  $M$  und  $N$  erhält man verschiedene Verfahren. Hier wollen wir zwei Verfahren genauer beschreiben und die Iteration (2.13) nicht mit Matrix–Multiplikationen sondern ausführlich für die Einträge  $x_j^{(i+1)}$  der Vektoren  $x^{(i+1)}$  aufschreiben, so dass die Verfahren dann direkt implementierbar sind. Das erste Verfahren ist das, welches wir auch im Beispiel 2.20 verwendet haben.

### Algorithmus 2.21 (Jacobi–Verfahren oder Gesamtschrittverfahren)

Wir wählen  $M = M_J$  als Diagonalmatrix

$$M_J = \begin{pmatrix} a_{11} & 0 & \dots & 0 \\ 0 & a_{22} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & a_{nn} \end{pmatrix}$$

und  $N = N_J$  als  $N_J = M_J - A$ . Dann ergibt sich (2.13) zu

$$x_j^{(i+1)} = \frac{1}{a_{jj}} \left( b_j - \sum_{\substack{k=1 \\ k \neq j}}^n a_{jk} x_k^{(i)} \right), \quad \text{für } j = 1, \dots, n.$$

□

Eine etwas andere Zerlegung führt zu dem folgenden Verfahren.

**Algorithmus 2.22 (Gauß–Seidel–Verfahren oder Einzelschrittverfahren)**

Wir wählen  $M = M_{GS}$  als untere Dreiecksmatrix

$$M_{GS} = \begin{pmatrix} a_{11} & 0 & \dots & 0 \\ a_{21} & a_{22} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ a_{n1} & \dots & a_{nn-1} & a_{nn} \end{pmatrix}$$

und  $N = N_{GS}$  als  $N_{GS} = M_{GS} - A$ . Wenn wir (2.13) von links mit  $M_{GS}$  multiplizieren ergibt sich

$$M_{GS}x^{(i+1)} = N_{GS}x^{(i)} + b.$$

Nun können wir die Komponenten  $x_j^{(i+1)}$  mittels Vorwärtseinsetzen bestimmen und erhalten so

$$x_j^{(i+1)} = \frac{1}{a_{jj}} \left( b_j - \sum_{k=1}^{j-1} a_{jk}x_k^{(i+1)} - \sum_{k=j+1}^n a_{jk}x_k^{(i)} \right), \quad \text{für } j = 1, \dots, n.$$

□

Der folgende Satz gibt ein Kriterium, unter dem diese Verfahren konvergieren.

**Satz 2.23** Sei  $A$  eine (*strikt*) *diagonaldominante* Matrix, d.h. es sei die Ungleichung

$$|a_{ii}| > \sum_{\substack{j=1 \\ j \neq i}}^n |a_{ij}|$$

für alle  $i = 1, \dots, n$  erfüllt.

Dann ist die Voraussetzung von Lemma 2.19 für das Jacobi- und das Gauß–Seidel-Verfahren mit der Zeilensummennorm  $\|\cdot\|_\infty$  erfüllt und die Konstanten  $k = k_J$  und  $k = k_{GS}$  für die beiden Verfahren lassen sich durch

$$k_{GS} \leq k_J = \max_{i=1, \dots, n} \left( \sum_{\substack{j=1 \\ j \neq i}}^n \frac{|a_{ij}|}{|a_{ii}|} \right) < 1$$

abschätzen. Insbesondere konvergieren also beide Verfahren für alle Startwerte gegen die Lösung des Gleichungssystems und es gelten die Abschätzungen aus Lemma 2.19.

**Beweis:** Es genügt, die Abschätzungen  $k = \|M^{-1}N\|_\infty < 1$  für die beiden Verfahren zu zeigen.



Wir beginnen mit dem Jacobi–Verfahren. Hier gilt  $M = M_J = \text{diag}(a_{11}, \dots, a_{nn})$  und  $N = N_J = M_J - A$ . Wegen  $M_J^{-1} = \text{diag}(a_{11}^{-1}, \dots, a_{nn}^{-1})$  folgt

$$M_J^{-1}N_J = \begin{pmatrix} 0 & -\frac{a_{12}}{a_{11}} & \cdots & -\frac{a_{1n}}{a_{11}} \\ -\frac{a_{21}}{a_{22}} & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & -\frac{a_{n-1n}}{a_{n-1n-1}} \\ -\frac{a_{n1}}{a_{nn}} & \cdots & -\frac{a_{nn-1}}{a_{nn}} & 0 \end{pmatrix},$$

also

$$\|M_J^{-1}N_J\|_\infty = \max_{i=1, \dots, n} \sum_{\substack{j=1 \\ j \neq i}}^n \frac{|a_{ij}|}{|a_{ii}|}.$$

Wegen der strikten Diagonaldominanz ist diese Summe für alle  $i$  echt kleiner als 1, also folgt  $k_J := \|M_J^{-1}N_J\|_\infty < 1$ .

Für das Gauß–Seidel–Verfahren seien  $M = M_{GS}$  und  $N = N_{GS}$  aus Algorithmus 2.22 gegeben. Wir setzen  $k_{GS} := \|M_{GS}^{-1}N_{GS}\|_\infty$  und zeigen die gewünschte Abschätzung  $k_{GS} < 1$ , indem wir  $k_{GS} \leq k_J$  beweisen.

Zum Beweis dieser Ungleichung sei  $x \in \mathbb{R}^n$  ein beliebiger Vektor mit  $\|x\|_\infty = 1$  und  $y = M_{GS}^{-1}N_{GS}x \in \mathbb{R}^n$ , also  $M_{GS}y = N_{GS}x$ . Zu zeigen ist  $\|y\|_\infty \leq k_J$ . Wir zeigen die Abschätzung einzeln für die Einträge  $|y_i|$  von  $y$  per Induktion über  $i = 1, \dots, n$ :

Für  $i = 1$  folgt aus  $[N_{GS}x]_1 = [M_{GS}y]_1 = a_{11}y_1$  die Ungleichung

$$\begin{aligned} |y_1| &= \left| \frac{1}{a_{11}} [N_{GS}x]_1 \right| \leq \frac{1}{|a_{11}|} \sum_{j=2}^n |a_{1j}| |x_j| \leq \sum_{j=2}^n \frac{|a_{1j}|}{|a_{11}|} \underbrace{\|x\|_\infty}_{=1} \\ &= \sum_{j=2}^n \frac{|a_{1j}|}{|a_{11}|} \leq \max_{q=1, \dots, n} \sum_{\substack{j=1 \\ j \neq q}}^n \frac{|a_{qj}|}{|a_{qq}|} = \|M_J^{-1}N_J\|_\infty = k_J. \end{aligned}$$

Für den Induktionsschritt  $i - 1 \rightarrow i$  nehmen wir als Induktionsvoraussetzung an, dass  $|y_j| \leq k_J$  für  $j = 1, \dots, i - 1$  gilt. Wegen

$$[N_{GS}x]_i = [M_{GS}y]_i = \sum_{j=1}^i a_{ij}y_j = \sum_{j=1}^{i-1} a_{ij}y_j + a_{ii}y_i$$

folgt

$$\begin{aligned} |y_i| &\leq \frac{1}{|a_{ii}|} \left( \sum_{j=1}^{i-1} |a_{ij}| |y_j| + |[N_{GS}x]_i| \right) \leq \frac{1}{|a_{ii}|} \left( \sum_{j=1}^{i-1} |a_{ij}| |y_j| + \sum_{j=i+1}^n |a_{ij}| |x_j| \right) \\ &= \sum_{j=1}^{i-1} \frac{|a_{ij}|}{|a_{ii}|} |y_j| + \sum_{j=i+1}^n \frac{|a_{ij}|}{|a_{ii}|} |x_j| \leq \sum_{j=1}^{i-1} \frac{|a_{ij}|}{|a_{ii}|} \underbrace{k_J}_{<1} + \sum_{j=i+1}^n \frac{|a_{ij}|}{|a_{ii}|} \underbrace{\|x\|_\infty}_{=1} \\ &\leq \left( \sum_{j=1}^{i-1} \frac{|a_{ij}|}{|a_{ii}|} + \sum_{j=i+1}^n \frac{|a_{ij}|}{|a_{ii}|} \right) \leq \max_{q=1, \dots, n} \sum_{\substack{j=1 \\ j \neq q}}^n \frac{|a_{qj}|}{|a_{qq}|} \\ &= \|M_J^{-1}N_J\|_\infty = k_J. \end{aligned}$$

□

Der Beweis zeigt insbesondere, dass die Kontraktionskonstante  $k_{GS}$  des Gauß–Seidel–Verfahrens kleiner oder gleich der des Jacobi–Verfahrens  $k_J$  ist, und in der Tat ist das Gauß–Seidel–Verfahren in der Praxis oft deutlich schneller.

Die strikte Diagonaldominanz ist ein recht spezielles Kriterium, das in der Praxis aber bei Diskretisierungen von Differentialgleichungen durchaus erfüllt ist.

Für das Gauß–Seidel–Verfahren lässt sich eine weitere Bedingung angeben, unter der dieses Verfahren konvergiert. Hierzu benötigen wir zunächst ein weiteres vorbereitendes Lemma, das eine weitere Bedingung an  $M^{-1}N$  für die Konvergenz der Verfahren zeigt.

**Lemma 2.24** Gegeben sei das lineare Gleichungssystem  $Ax = b$  mit invertierbarer Matrix  $A$  sowie eine Zerlegung  $A = M - N$  mit invertierbarer Matrix  $M$ . Es gelte  $\rho(M^{-1}N) < 1$ , wobei  $\rho(E) := \max_i |\lambda_i(E)|$  den *Spektralradius*, also den maximalen Betrag der Eigenwerte  $\lambda_1(E), \dots, \lambda_d(E)$  einer Matrix  $E \in \mathbb{R}^{n \times n}$  bezeichnet. Dann konvergiert Verfahren (2.13) für beliebige Startwerte  $x^{(0)}$  gegen die Lösung  $x^*$  des Gleichungssystems.

**Beweis:** Wir beweisen zunächst die folgende Eigenschaft für beliebige Matrizen  $E \in \mathbb{R}^{n \times n}$ : Für jedes  $\varepsilon \in (0, 1)$  existiert eine Vektornorm  $\|\cdot\|_{E,\varepsilon}$ , so dass für die zugehörige induzierte Matrixnorm die Abschätzung

$$\|E\|_{E,\varepsilon} \leq \rho(E) + \varepsilon \quad (2.15)$$

gilt. Um dies zu beweisen, benötigen wir die aus der linearen Algebra bekannte Jordan'sche Normalform:

Zu jeder Matrix  $E \in \mathbb{R}^{n \times n}$  existiert eine invertierbare Matrix  $S \in \mathbb{C}^{n \times n}$ , so dass  $R = S^{-1}ES$  in Jordan'scher Normalform vorliegt. Die Matrix  $R$  besitzt also als Diagonalelemente  $r_{ii}$  gerade die Eigenwerte von  $E$ , für die Elemente oberhalb der Diagonalen gilt  $r_{i,i+1} \in \{0, 1\}$ , alle weiteren Einträge sind gleich Null.

Zur Konstruktion der Norm  $\|\cdot\|_{E,\varepsilon}$  beachte, dass für jede invertierbare Matrix  $C \in \mathbb{C}^{n \times n}$  die Norm  $\|x\|_C := \|C^{-1}x\|_\infty$  eine Vektornorm mit induzierter Matrixnorm

$$\|A\|_C = \sup_{x \in \mathbb{C}^n \setminus \{0\}} \frac{\|C^{-1}Ax\|_\infty}{\|C^{-1}x\|_\infty} = \sup_{y = C^{-1}x \in \mathbb{C}^n \setminus \{0\}} \frac{\|C^{-1}ACy\|_\infty}{\|y\|_\infty} = \|C^{-1}AC\|_\infty$$

ist. Für das gegebene  $\varepsilon \in (0, 1)$  setzen wir  $C_\varepsilon := SD_\varepsilon$  mit  $S$  von oben und  $D_\varepsilon = \text{diag}(1, \varepsilon, \varepsilon^2, \dots, \varepsilon^{n-1})$ . Wir schreiben  $R_\varepsilon = C_\varepsilon^{-1}EC_\varepsilon$ . Man rechnet leicht nach, dass für die Elemente  $r_{\varepsilon,ij}$  von  $R_\varepsilon$  die Gleichung  $r_{\varepsilon,ij} = \varepsilon^{j-i}r_{ij}$  gilt. Insbesondere gilt also  $r_{\varepsilon,ii} = r_{ii}$  und  $r_{\varepsilon,i,i+1} = \varepsilon r_{i,i+1}$  während alle anderen Elemente von  $R_\varepsilon$  gleich Null sind. Damit gilt

$$\begin{aligned} \|E\|_{C_\varepsilon} &= \|C_\varepsilon^{-1}EC_\varepsilon\|_\infty = \|R_\varepsilon\|_\infty = \max_{i=1,\dots,n} \sum_{j=1}^n |r_{\varepsilon,ij}| \\ &= \max_{i=1,\dots,n} \{|r_{ii}| + |\varepsilon r_{i,i+1}|\} \leq \max_{k=1,\dots,d} \{|\lambda_k| + \varepsilon\} = \rho(E) + \varepsilon, \end{aligned}$$

also die gewünschte Abschätzung.

Zum Beweis des Lemmas sei nun  $\rho(M^{-1}N) < 1$  und  $\varepsilon \in (0, 1 - \rho(M^{-1}N))$  beliebig. Dann finden wir nach (2.15) eine Norm  $\|\cdot\|_{M^{-1}N,\varepsilon}$  mit  $\|M^{-1}N\|_{M^{-1}N,\varepsilon} \leq \rho(M^{-1}N) + \varepsilon < 1$ . Also folgt die Behauptung mit Lemma 2.19. □

**Bemerkung 2.25** Beachte, dass der Spektralradius mittels  $k = \rho(M^{-1}N) + \varepsilon$  eine Kontraktionskonstante und damit über die Abschätzungen in Lemma 2.19 insbesondere ein Maß für die Geschwindigkeit der Konvergenz liefert — allerdings in der im Allgemeinen unbekannt Norm  $\|\cdot\|_{M^{-1}N, \varepsilon}$ .  $\square$

Tatsächlich ist die Bedingung in Lemma 2.24 nicht nur *hinreichend* sondern auch *notwendig* für die Konvergenz und stellt somit das schärfste mögliche Kriterium dar (auf den Beweis wollen wir hier nicht eingehen).

Das Lemma 2.24 ist für beliebige iterative Verfahren der Form (2.13) anwendbar, und kann verwendet werden, um die Konvergenz dieser Verfahren für bestimmte Matrizen oder bestimmte Klassen von Matrizen zu beweisen. Für das Gauß–Seidel–Verfahren liefert der folgende Satz ein entsprechendes Resultat.

**Satz 2.26** Sei  $A \in \mathbb{R}^{n \times n}$  eine symmetrische, positiv definite Matrix. Dann konvergiert das Gauß–Seidel–Verfahren aus Algorithmus 2.22 für alle Startwerte  $x^{(0)} \in \mathbb{R}^n$  gegen die Lösung  $x^*$  des Gleichungssystems  $Ax = b$ .

**Beweis:** Wir zeigen, dass das Gauß–Seidel–Verfahren für symmetrische und positiv definite Matrizen  $A$  die Voraussetzungen von Lemma 2.24 erfüllt. Wir zeigen also, dass  $\rho(M^{-1}N) < 1$  ist. Sei dazu  $\lambda \in \mathbb{C}$  ein Eigenwert von  $M^{-1}N$  mit Eigenvektor  $z \in \mathbb{C}^n$ , also

$$M^{-1}Nz = \lambda z,$$

oder, äquivalent,

$$2Nz = 2\lambda Mz. \quad (2.16)$$

Wir müssen zeigen, dass  $|\lambda| < 1$  ist. Wir setzen nun  $D := \text{diag}(a_{11}, \dots, a_{nn})$ . Für alle  $i = 1, \dots, n$  gilt wegen der positiven Definitheit von  $A$  die Ungleichung  $a_{ii} = e_i^T A e_i > 0$ , wobei  $e_i$  wie üblich den  $i$ -ten Standard-Basisvektor bezeichnet. Daher ist auch  $D$  eine symmetrische und positiv definite Matrix. Wegen der Symmetrie von  $A$  gelten nun die Gleichungen  $A = D - N^T - N$  und  $M = D - N^T$  und damit (unter Verwendung von  $M = A + N$ ) auch

$$2N = D - A + N - N^T \quad \text{und} \quad 2M = D + A + N - N^T.$$

Setzen wir diese Ausdrücke in (2.16) ein und multiplizieren die Gleichung dann von links mit  $\bar{z}^T$ , so erhalten wir

$$\bar{z}^T D z - \bar{z}^T A z + \bar{z}^T (N - N^T) z = \lambda (\bar{z}^T D z + \bar{z}^T A z + \bar{z}^T (N - N^T) z).$$

Da  $A$  und  $D$  symmetrisch und positiv definit sind, sind die Werte  $a = \bar{z}^T A z$  sowie  $d = \bar{z}^T D z$  reell und positiv. Da  $N - N^T$  schiefsymmetrisch ist, nimmt  $\bar{z}^T (N - N^T) z$  einen rein imaginären Wert  $ib$  an. Aus der obigen Gleichung folgt damit  $d - a + ib = \lambda(d + a + ib)$ , also

$$\lambda = \frac{d - a + ib}{d + a + ib}.$$

Dies ist der Quotient zweier komplexer Zahlen mit gleichem Imaginärteil, wobei der Zähler betragsmäßig kleineren Realteil besitzt, weswegen der Quotient einen Betrag kleiner als 1 besitzt, also  $|\lambda| < 1$ , was zu zeigen war.  $\square$

Wir wollen nun den Aufwand dieser Iterationsverfahren abschätzen. Um die Diskussion kurz zu halten, beschränken wir uns dabei auf einen einfachen Fall: Wir nehmen an, dass wir eine Familie von Matrizen betrachten, bei denen die Kontraktionskonstante  $k$  für eine gegebene Norm  $\|\cdot\|$  unabhängig von der Dimension  $n$  des Problems ist. Dann folgt, dass bei geeigneter Wahl der Startwerte  $x^{(0)}$  die Anzahl der Iterationen  $N_\varepsilon$  bis zum Erreichen einer vorgegebenen Genauigkeit  $\varepsilon$  unabhängig von  $n$  ist. Für einen Iterationsschritt und eine Komponente  $x_j^{(i+1)}$  benötigen wir in beiden Verfahren  $n - 1$  Multiplikationen und Additionen für die Summe und dazu eine Division, also  $2n - 1$  Operationen. Für die  $n$  Komponenten von  $x^{(i+1)}$  ergeben sich so  $n(2n - 1) = 2n^2 - n$  Operationen, und damit insgesamt

$$N_\varepsilon(2n^2 - n)$$

Operationen. Insbesondere haben diese Algorithmen unter den oben gemachten Annahmen an die Probleme die Ordnung  $O(n^2)$ , der Aufwand wächst also deutlich langsamer in  $n$  als bei der Gauß-Elimination oder beim Choleski-Verfahren.

Unter zusätzlichen Annahmen an  $A$  kann sich der Aufwand dieser Verfahren beträchtlich verringern: Viele sehr große Gleichungssysteme haben die Eigenschaft, dass in jeder Zeile der (sehr großen) Matrix  $A$  nur relativ wenige Einträge einen Wert ungleich Null besitzen, man sagt, die Matrix  $A$  ist *schwach besetzt*. Wenn wir annehmen, dass — unabhängig von  $n$  — in jeder Zeile von  $A$  höchstens  $m$  Einträge ungleich Null sind, ist die Anzahl der Operationen in der Berechnung von  $x_j^{(i+1)}$  höchstens gleich  $2m - 1$ , und die Gesamtzahl der Operationen ergibt sich zu  $N_\varepsilon 2(m - 1)n$ . Wir erhalten so die Ordnung  $O(n)$ , d.h. die Anzahl der Operationen wächst linear in  $n$ . Allerdings kann sich unter solchen Bedingungen auch die Anzahl der Operationen in der Gauß-Elimination oder im Choleski-Verfahren verringern, typischerweise wird die Ordnung dort i.A. nicht kleiner als  $O(n^2)$ . Eine Ausnahme bilden *Bandmatrizen* mit sehr einfacher Bandstruktur, für die man Algorithmen zur *LR-Zerlegung* mit dem Aufwand  $O(n)$  formulieren kann, vgl. den Algorithmus für Tridiagonalmatrizen aus Übungsaufgabe 9, Blatt 9.

Auch bei iterativen Verfahren kann schlechte Konditionierung von  $A$  Schwierigkeiten verursachen, die sich hier meist in der Form äußern, dass die Iteration auf Grund von Rundungsfehlern keinen Fortschritt mehr zeigt, noch bevor die gewünschte Genauigkeit erreicht ist. Üblicherweise macht sich schlechte Konditionierung in Kontraktionskonstanten  $k$ , die nahe an 1 liegen, bemerkbar. Eine mögliche Abhilfe bietet hier die früher schon erwähnte *Präkonditionierung*, bei der eine Matrix  $P$  so gewählt wird, dass  $PA$  besser konditioniert ist als  $A$  und dann  $PAx = Pb$  gelöst wird. Eine mögliche Strategie dafür wurde in Übungsaufgabe 5 auf Blatt 2 behandelt.

## 2.9 Relaxation

Wir wollen in diesem letzten Abschnitt eine Variante der beiden betrachteten Verfahren ohne Beweise kurz erläutern. Die Variante beruht auf der sogenannten *Relaxation*, die auf Basis entweder des Jacobi- oder des Gauß-Seidel-Verfahrens durchgeführt werden kann. Ziel dieser Relaxation ist es, die Konvergenz dieser Verfahren zu beschleunigen.

Die Idee ist wie folgt: Beim Jacobi-Verfahren wählt man einen reellen Parameter  $\omega > 0$

und ändert die Iteration

$$x_j^{(i+1)} = \frac{1}{a_{jj}} \left( b_j - \sum_{\substack{k=1 \\ k \neq j}}^n a_{jk} x_k^{(i)} \right) =: \Phi_j(x^{(i)})$$

aus Algorithmus 2.21 in

$$x_j^{(i+1)} = (1 - \omega)x_j^{(i)} + \omega\Phi_j(x^{(i)}),$$

d.h. man wählt den neuen Näherungswert als eine gewichtete Summe zwischen dem alten und dem von der Iterationsvorschrift  $\Phi$  gelieferten neuen Wert.

Genau so geht man beim Gauß–Seidel Verfahren vor. Hier gilt gemäß Algorithmus 2.22 für jede Komponente  $j$  die Vorschrift

$$x_j^{(i+1)} = \frac{1}{a_{jj}} \left( b_j - \sum_{k=1}^{j-1} a_{jk} x_k^{(i+1)} - \sum_{k=j+1}^n a_{jk} x_k^{(i)} \right) =: \Phi_j(x^{(i)}, x^{(i+1)}),$$

die in

$$x_j^{(i+1)} = (1 - \omega)x_j^{(i)} + \omega\Phi_j(x^{(i)}, x^{(i+1)})$$

geändert wird.

In beiden Verfahren spricht man für  $\omega < 1$  von *Unterrelaxation*, für  $\omega > 1$  von *Überrelaxation*. In der Praxis ist die Überrelaxation das gebräuchlichere Verfahren. Dieses wird auch als SOR-Verfahren (SOR = successive overrelaxation) bezeichnet.

Die Konvergenzbeweise für diese relaxierten Verfahren verlaufen ähnlich wie die entsprechenden Beweise für die zu Grunde liegenden Verfahren, für Details siehe z.B. Deuffhard/Hohmann [2, Kapitel 8].



## Kapitel 3

# Eigenwerte und Eigenvektoren

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

$$Av = \lambda v$$

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

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

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

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

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

mit den Nullstellen  $\lambda_i = i$  für  $i = 1, \dots, 20$ .<sup>1</sup> Wenn dieses Polynom als charakteristisches Polynom einer Matrix berechnet wird (z.B. ist es gerade das charakteristische Polynom  $\chi_A(\lambda)$

---

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

der Matrix  $A = \text{diag}(1, 2, \dots, 20)$ , liegt es üblicherweise nicht in der obigen “Nullstellen”-Darstellung sondern in anderer Form vor, z.B. ausmultipliziert. Wenn man das obige  $P(\lambda)$  ausmultipliziert, ergeben sich Koeffizienten zwischen 1 (für  $\lambda^{20}$ ) und  $20! \approx 10^{20}$  (der konstante Term). Stört man nun den Koeffizienten vor  $\lambda^{19}$  (der den Wert 210 hat) mit dem sehr kleinen Wert  $\varepsilon = 2^{-23} \approx 10^{-7}$ , so erhält man die folgenden Nullstellen für das gestörte Polynom  $\tilde{P}(\lambda) = P(\lambda) - \varepsilon\lambda^{19}$ :

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

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

### 3.1 Vektoriteration

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

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

$$x^{(i+1)} = Ax^{(i)} \tag{3.1}$$

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

**Satz 3.1** Sei  $A \in \mathbb{R}^{n \times n}$  eine symmetrische Matrix und  $\lambda_1 = \lambda_1(A)$  ein einfacher Eigenwert, für den die Ungleichung

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

für alle anderen Eigenwerte  $\lambda_j = \lambda_j(A)$  gilt. Sei weiterhin  $x^{(0)} \in \mathbb{R}^n$  ein Vektor, für den  $\langle x^{(0)}, v_1 \rangle \neq 0$  für den zu  $\lambda_1(A)$  gehörigen (normierten) Eigenvektor  $v_1$  gilt. Dann konvergiert die Folge  $y^{(i)} := x^{(i)} / \|x^{(i)}\|$  für  $x^{(i)}$  aus (3.1) gegen  $\pm v_1$ , also gegen einen normierten Eigenvektor zum Eigenwert  $\lambda_1$ . Insbesondere konvergiert damit der sogenannte *Rayleigh'sche Quotient*

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

gegen den Eigenwert  $\lambda_1$ .



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

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

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

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

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

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

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

$$\begin{aligned} \langle Ay^{(i)}, y^{(i)} \rangle &= \langle A(v_1 + r^{(i)}), v_1 + r^{(i)} \rangle \\ &= \langle Av_1, v_1 \rangle + \underbrace{\langle Ar^{(i)}, v_1 \rangle + \langle Av_1, r^{(i)} \rangle + \langle Ar^{(i)}, r^{(i)} \rangle}_{\rightarrow 0} \\ &\rightarrow \langle Av_1, v_1 \rangle = \langle \lambda_1 v_1, v_1 \rangle = \lambda_1 \|v_1\|^2 = \lambda_1. \end{aligned}$$

□

Beachte, dass die Symmetrie der Matrix  $A$  hier nur eine hinreichende aber keine notwendige Bedingung für die Konvergenz des Verfahrens ist. So ist z.B. in [1] bewiesen, dass das Verfahren auch für die (nicht symmetrische) Matrix aus dem Seitenranking funktioniert, vgl. auch das 6. Übungsblatt.

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

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

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

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

Die inverse Vektoriteration ist nun gegeben durch

$$x^{(i+1)} = (A - \tilde{\lambda} \text{Id})^{-1} x^{(i)}. \quad (3.2)$$

Aus Satz 3.1 (angewendet auf  $(A - \tilde{\lambda}\text{Id})^{-1}$  an Stelle von  $A$ ) folgt, dass diese Iteration gegen einen normierten Eigenvektor  $v_j$  von  $(A - \tilde{\lambda}\text{Id})^{-1}$  zum Eigenwert  $1/(\lambda_j - \tilde{\lambda})$  konvergiert. Wegen

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

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

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

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

Die tatsächliche Implementierung der Iteration (3.2) ist hier etwas komplizierter als bei der direkten Iteration (3.1). Während dort in jedem Schritt eine Matrix-Vektor Multiplikation mit Aufwand  $O(n^2)$  durchgeführt werden muss, geht hier die Inverse  $(A - \tilde{\lambda}\text{Id})^{-1}$  ein. In der Praxis berechnet man nicht die Inverse (weil dies numerisch sehr aufwändig ist), sondern löst das lineare Gleichungssystem

$$(A - \tilde{\lambda}\text{Id})x^{(i+1)} = x^{(i)}, \quad (3.3)$$

wobei bei der Verwendung eines direkten Verfahrens die Matrix  $(A - \tilde{\lambda}\text{Id})$  nur einmal am Anfang der Iteration faktorisiert werden muss und dann in jedem Iterationsschritt einmal Vorwärts- bzw. Rückwärtseinsetzen durchgeführt werden muss. Der Aufwand  $O(n^3)$  der Zerlegung kommt also hier zum Aufwand des Verfahrens dazu, die einzelnen Iterationsschritte haben bei diesem Vorgehen allerdings keinen höheren Rechenaufwand als bei der direkten Iteration, da das Vorwärts- bzw. Rückwärtseinsetzen wie die Matrix-Vektor Multiplikation den Aufwand  $O(n^2)$  besitzen.

Für sehr gute Schätzwerte  $\tilde{\lambda} \approx \lambda_j$  wird die Matrix  $(A - \tilde{\lambda}\text{Id})$  "fast" singular (für  $\tilde{\lambda} = \lambda_j$  wäre sie singular), weswegen die Kondition von  $(A - \tilde{\lambda}\text{Id})$  sehr groß wird. Wegen der besonderen Struktur des Algorithmus führt dies hier aber nicht auf numerische Probleme, da zwar die Lösung  $x^{(i+1)}$  des Gleichungssystems (3.3) mit großen Fehlern behaftet sein kann, sich diese Fehler aber in der hier eigentlich wichtigen *normierten Lösung*  $x^{(i+1)}/\|x^{(i+1)}\|$  nicht auswirken. Wir wollen dies an einem Beispiel illustrieren.

**Beispiel 3.2** Betrachte

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

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

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

fast singulär und man sieht leicht, dass für die Kondition z.B. in der Zeilensummennorm die Abschätzung  $\text{cond}_\infty(A - \tilde{\lambda}\text{Id}) > 1/\varepsilon$  gilt. Die Inverse besitzt aber eine typische spezielle Struktur: die großen Einträge, die der Grund für die große Kondition sind, entstehen lediglich durch einen skalaren Vorfaktor. Daher ist die Berechnung von  $y^{(i+1)} = x^{(i+1)}/\|x^{(i+1)}\|$  mittels (3.3) nicht stark anfällig für Rundungsfehler, denn es gilt

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

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

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

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

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

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

## 3.2 Der QR-Algorithmus

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

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

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

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

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

gilt, wobei die  $\lambda_i$  gerade die Eigenwerte von  $A$  sind. Wenn wir also eine orthogonale Matrix  $Q$  finden können, mit der  $A$  auf Diagonalgestalt konjugiert werden kann, so können wir die Eigenwerte direkt aus der resultierenden Diagonalmatrix  $\Lambda$  und die zugehörigen Eigenvektoren aus  $Q$  ablesen. Leider ist eine direkte Transformation auf Diagonalgestalt

nicht möglich. Zwar kann man z.B. orthogonale Householder–Matrizen dazu verwenden, Matrizen auf obere Dreiecksgestalt zu bringen, leider lässt sich dies nicht zur Konjugation auf Diagonalgestalt verwenden, da die positiven Effekte durch die Multiplikation von links mit  $H$  bei der Multiplikation von rechts mit  $H^T$  wieder zunichte gemacht werden.

Wir werden die benötigte Transformation  $Q$  daher iterativ konstruieren. Dabei empfiehlt es sich, die Matrix  $A$  in einem vorbereitenden Schritt zunächst auf eine möglichst einfache Form zu bringen, um die Anzahl der Rechenoperationen in der Iteration klein zu halten. Hierbei wählt man die *Tridiagonalgestalt* und nutzt aus, dass man Householder–Matrizen dazu verwenden kann, Matrizen auf Tridiagonalgestalt zu konjugieren. Grundlage dafür ist das folgende Lemma.

**Lemma 3.3** Sei  $A$  eine reelle symmetrische Matrix der Form

$$A = \begin{pmatrix} a_{11} & a_{12} & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ a_{21} & a_{22} & a_{23} & \ddots & & & & \vdots \\ 0 & \ddots & \ddots & \ddots & \ddots & & & \vdots \\ \vdots & \ddots & \ddots & a_{k-1\ k-1} & a_{k-1\ k} & 0 & \cdots & 0 \\ \vdots & & \ddots & a_{k\ k-1} & a_{k\ k} & \cdots & \cdots & a_{k\ n} \\ \vdots & & & 0 & \vdots & & & \vdots \\ \vdots & & & \vdots & \vdots & & & \vdots \\ 0 & \cdots & \cdots & 0 & a_{n\ k} & \cdots & \cdots & a_{n\ n} \end{pmatrix}$$

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

$$c = \text{sgn}(w_{k+1}) \sqrt{w_{k+1}^2 + w_{k+2}^2 + \cdots + w_n^2} \in \mathbb{R}$$

$$v = (0, \dots, 0, c + w_{k+1}, w_{k+2}, \dots, w_n)^T$$

und den Konventionen aus Lemma 2.14. Dann gilt

$$HAH^T = \begin{pmatrix} a_{11} & a_{12} & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ a_{21} & a_{22} & a_{23} & \ddots & & & & \vdots \\ 0 & \ddots & \ddots & \ddots & \ddots & & & \vdots \\ \vdots & \ddots & \ddots & a_{k\ k} & \tilde{a}_{k\ k+1} & 0 & \cdots & 0 \\ \vdots & & \ddots & \tilde{a}_{k+1\ k} & \tilde{a}_{k+1\ k+1} & \cdots & \cdots & \tilde{a}_{k+1\ n} \\ \vdots & & & 0 & \vdots & & & \vdots \\ \vdots & & & \vdots & \vdots & & & \vdots \\ 0 & \cdots & \cdots & 0 & \tilde{a}_{n\ k+1} & \cdots & \cdots & \tilde{a}_{n\ n} \end{pmatrix}$$

**Beweis:** Wir betrachten zunächst das Produkt  $HA$ . Es bezeichne  $a_{.j}$  die  $j$ -te Spalte von  $A$ . Aus Lemma 2.14 wissen wir, dass

$$Ha_{.j} = a_{.j}, \quad \text{für } j = 1, \dots, k-1,$$

also für die ersten  $k-1$  Spalten der Matrix  $A$ . Ebenfalls nach Lemma 2.14 gilt für die  $k$ -te Spalte

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

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

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

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

$$Ha_{.j} = (0, \dots, 0, a_{kj}, *, \dots, *)^T.$$

Insgesamt gilt also

$$HA = \begin{pmatrix} a_{11} & a_{12} & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ a_{21} & a_{22} & a_{23} & \ddots & & & & \vdots \\ 0 & \ddots & \ddots & \ddots & \ddots & & & \vdots \\ \vdots & \ddots & \ddots & a_{k-1k-1} & a_{k-1k} & 0 & \cdots & 0 \\ \vdots & & \ddots & a_{kk-1} & a_{kk} & \cdots & \cdots & a_{kn} \\ \vdots & & & 0 & -c & * & \cdots & * \\ \vdots & & & \vdots & 0 & \vdots & & \vdots \\ \vdots & & & \vdots & \vdots & \vdots & & \vdots \\ 0 & \cdots & \cdots & 0 & 0 & * & \cdots & * \end{pmatrix}$$

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

Die folgende Aussage ist nun ein einfaches Korollar.

**Korollar 3.4** Sei  $A$  eine reelle symmetrische Matrix. Dann existiert eine orthogonale Matrix  $P$ , so dass  $P^T A P$  symmetrisch und in Tridiagonalgestalt ist.

**Beweis:** Durch Anwendung von Lemma 3.3 für  $k = 1, \dots, n-2$  erhält man Matrizen  $H^{(1)}, \dots, H^{(n-2)}$ , für die  $P^T = H^{(n-2)} \cdots H^{(1)}$  die gewünschte Eigenschaft besitzt.  $\square$

Algorithmus 2.15 lässt sich leicht zur Berechnung dieser Transformationsmatrix  $P$  abändern, indem man den Index  $j$  in Schritt (1) nur von 2 bis  $n-1$  laufen lässt und alle *Spaltenindizes* in den folgenden Berechnungen um 1 erniedrigt, also  $a_{jj}$  und  $a_{ij}$  durch  $a_{j,j-1}$  bzw.  $a_{i,j-1}$  ersetzt. Die Berechnung von  $H^{(j)}y^{(j)}$  macht hier natürlich keinen Sinn,

die Berechnung von  $H^{(j)}A^{(j)}$  kann falls gewünscht auf die Berechnung von  $H^{(j)}A^{(j)}H^{(j)T}$  erweitert werden.

Die Iteration zur iterativen Berechnung von  $Q$  ist nun durch den folgenden Algorithmus gegeben.

**Algorithmus 3.5 (QR-Algorithmus zur Eigenwertberechnung)**

**Eingabe:** symmetrische, reelle Matrix  $A \in \mathbb{R}^{n \times n}$

- (0) Setze  $A^{(1)} := P^T A P$  mit  $P$  aus Korollar 3.4,  $i := 1$
- (1) Berechne eine QR-Zerlegung  $A^{(i)} = Q^{(i)} R^{(i)}$
- (2) Setze  $A^{(i+1)} := R^{(i)} Q^{(i)}$ ,  $i := i + 1$  und fahre fort bei (1)

**Ausgabe:** Approximation aller Eigenwerte von  $A$  als Diagonaleinträge von  $A^{(i)}$  und der Eigenvektoren als Spalten der Matrix  $PQ^{(1)} \dots Q^{(i)}$ , für Details siehe Bemerkung 3.9 (iii).  $\square$

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

Wir werden den Algorithmus 3.5 nun analysieren. Wir beginnen mit einigen Struktureigenschaften der Matrizen  $A^{(i)}$ .

**Lemma 3.6** Für alle  $i \geq 1$  gilt:

- (i)  $A^{(i)}$  ist konjugiert zu  $A$ . Genauer gilt  $A^{(i+1)} = Q_i^T P^T A P Q_i$  mit  $Q_i = Q^{(1)} \dots Q^{(i)}$  und  $P$  aus Korollar 3.4.
- (ii)  $A^{(i)}$  ist symmetrisch
- (iii)  $A^{(i)}$  ist eine Tridiagonalmatrix

**Beweis:** (i) Es gilt

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

Durch induktives Fortfahren erhält man so

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

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

(ii) Nach (i) gilt

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

da  $A$  symmetrisch ist.

(iii) Sei  $A^{(i)}$  tridiagonal. Durch explizites Ausrechnen der  $QR$ -Zerlegung mit Hilfe von Lemma 2.14 sieht man, dass  $(Q^{(i)})^T$  tridiagonal ist, also auch  $Q^{(i)}$ . Also ist  $A^{(i+1)} = R^{(i)}Q^{(i)}$  von der Form

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

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

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

**Lemma 3.7** Sei  $A \in \mathbb{R}^{n \times n}$  eine invertierbare Matrix und seien  $A = Q_1 R_1$  und  $A = Q_2 R_2$   $QR$ -Zerlegungen von  $A$ . Dann gilt  $Q_1 = Q_2 D$  und  $R_1 = D R_2$  mit  $D = \text{diag}(\mu_1, \dots, \mu_n)$  und  $\mu_k = \pm 1$ . Insbesondere existiert eine eindeutige  $QR$ -Zerlegung bei der alle Diagonalelemente von  $R$  positiv sind.

**Beweis:** Aus  $Q_1 R_1 = Q_2 R_2$  folgt  $Q_2^T Q_1 = R_2 R_1^{-1}$  (beachte, dass  $R_1$  und  $R_2$  invertierbar sind, da  $A$  invertierbar ist). Die Inverse einer oberen Dreiecksmatrix ist wieder eine obere Dreiecksmatrix, ebenso das Produkt zweier oberer Dreiecksmatrizen. Also ist  $R = R_2 R_1^{-1}$  eine obere Dreiecksmatrix, die wegen  $R = Q_2^T Q_1$  orthogonal ist. Aus der Orthogonalitätsbedingung  $\langle Rx, Ry \rangle = \langle x, y \rangle$  leitet man für  $x = e_i$  und  $y = e_j$  Bedingungen an die Koeffizienten von  $R$  ab, aus denen  $R = D$  mit der behaupteten Struktur folgt.  $\square$

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

**Satz 3.8** Sei  $A \in \mathbb{R}^{n \times n}$  eine symmetrische Matrix mit den Eigenwerten  $\lambda_1, \dots, \lambda_n$ , für die die Ungleichung

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

gelte. Seien  $A^{(i)}$ ,  $Q^{(i)}$  und  $R^{(i)}$  aus Algorithmus 3.5. Dann gilt

$$\lim_{i \rightarrow \infty} A^{(i)} = \Lambda = \text{diag}(\lambda_1, \dots, \lambda_n).$$

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

$$A^i = Q_i R_i \tag{3.4}$$

für die  $i$ -te Potenz von  $A$ , mit  $Q_i = Q^{(1)} \cdots Q^{(i)}$  und  $R_i = R^{(i)} \cdots R^{(1)}$ . Für  $i = 1$  ist (3.4) klar, denn

$$A^1 = A = A^{(1)} = Q^{(1)} R^{(1)} = Q_1 R_1.$$

Für den Schritt  $i \rightarrow i + 1$  verwenden wir  $A^{(i+1)} = Q_i^T A Q_i$  aus Lemma 3.6(i), aus dem wegen  $Q^{(i+1)} R^{(i+1)} = A^{(i+1)}$  die Gleichung  $A Q_i = Q_i Q^{(i+1)} R^{(i+1)}$  folgt. Damit und mit der Induktionsannahme  $A^i = Q_i R_i$  folgt

$$A^{i+1} = A A^i = A Q_i R_i = Q_i Q^{(i+1)} R^{(i+1)} R_i = Q_{i+1} R_{i+1},$$

also (3.4). Beachte, dass  $Q_i$  orthogonal und  $R_i$  eine obere Dreiecksmatrix ist, weswegen  $Q_i R_i$  eine  $QR$ -Zerlegung von  $A^i$  darstellt.

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

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

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

$$A^i = Q \Lambda^i L R = Q (\Lambda^i L \Lambda^{-i}) (\Lambda^i R). \quad (3.5)$$

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

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

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

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

Aus (3.5) erhalten wir so

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

Sei nun  $\tilde{Q}_i \tilde{R}_i$  die eindeutige  $QR$ -Zerlegung von  $\text{Id} + E_i$  mit positiven Vorzeichen der Diagonalelemente von  $\tilde{R}_i$ . Wegen der Eindeutigkeit dieser Zerlegung konvergieren diese Matrizenfolgen gegen die  $QR$ -Zerlegung von  $\text{Id} = \bar{Q} \bar{R}$  mit positiven Diagonalelementen von  $\bar{R}$ , welche gerade durch  $\text{Id} = \text{Id} \cdot \text{Id}$  gegeben ist<sup>2</sup>, also gilt

$$\tilde{Q}_i \rightarrow \text{Id} \text{ und } \tilde{R}_i \rightarrow \text{Id} \text{ für } i \rightarrow \infty$$

und damit auch

$$\tilde{Q}_i^T \rightarrow \text{Id} \text{ und } \tilde{R}_i^{-1} \rightarrow \text{Id} \text{ für } i \rightarrow \infty.$$

Die Matrizen  $Q \tilde{Q}_i$  und  $\tilde{R}_i \Lambda^i R$  bilden nun wegen (3.5) und

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

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



eine  $QR$ -Zerlegung von  $A^i$ . Da  $A^i = Q_i R_i$  eine weitere  $QR$ -Zerlegung ist, folgt mit Lemma 3.7 die Existenz von  $D_i = \text{diag}(\pm 1, \dots, \pm 1)$ , so dass

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

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

$$\begin{aligned} A^{(i)} &= Q^{(i)} R^{(i)} = Q_{i-1}^T Q_i R_i R_{i-1}^{-1} \\ &= D_{i-1} \tilde{Q}_{i-1}^T Q^T Q \tilde{Q}_i D_i D_i \tilde{R}_i \Lambda^i R R^{-1} \Lambda^{-(i-1)} \tilde{R}_{i-1}^{-1} D_{i-1} \\ &= D_{i-1} \tilde{Q}_{i-1}^T \tilde{Q}_i \tilde{R}_i \Lambda \tilde{R}_{i-1}^{-1} D_{i-1}. \end{aligned}$$

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

**Bemerkung 3.9** (i) Tatsächlich ist die Tatsache, dass  $A^{(1)}$  in Tridiagonalform vorliegt, nicht wesentlich für den korrekten Ablauf des Algorithmus, sie vereinfacht aber die  $QR$ -Zerlegung in Schritt (1) erheblich: Man sieht leicht, dass die in Algorithmus 2.15 auftretenden Summen in diesem Fall nur aus maximal 3 Summanden bestehen, weswegen sich der Aufwand eines Schrittes des Algorithmus auf  $O(n^2)$  reduziert. Die Transformation von  $A$  auf Tridiagonalgestalt besitzt zwar den Aufwand  $O(n^3)$ , ohne die vorhergehende Tridiagonalisierung wäre allerdings der Aufwand *jedes Schrittes* gleich  $O(n^3)$ .

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

(iii) Die approximativen Eigenwerte von  $A$  liest man bei diesem Algorithmus einfach aus der Diagonalen von  $A^{(i)}$  ab. Die zugehörigen Eigenvektoren werden approximiert durch die Spalten der Transformationsmatrix  $\bar{Q}_i$ , für die  $\bar{Q}_i^T A \bar{Q}_i = A^{(i)}$  gilt. Diese ist nach Lemma 3.6(i) gerade durch  $\bar{Q}_i = P Q_{i-1} = P Q^{(1)} \dots Q^{(i-1)}$  gegeben.

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

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

wobei die letzte “ungefähre” Gleichung aus der Konditionsanalyse des Eigenwertproblems folgt, die wir hier aus Zeitgründen nicht betrachtet haben. Für Diagonalmatrizen (bzw. allgemeiner für normale Matrizen) ist die Kondition des Eigenwertproblems bezüglich der 2-Norm nämlich gerade gleich 1, für Details siehe Deuffhard/Hohmann [2, Satz 5.2]. Die Matrixnorm  $\|B^{(i)}\|_2$  lässt sich aber durch die Größe der Einträge von  $B^{(i)}$  abschätzen.

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

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

**Algorithmus 3.10 (QR-Algorithmus zur Eigenwertberechnung mit Shift)**

**Eingabe:** symmetrische, reelle Matrix  $A$ , Folge von Shift-Parametern  $\sigma_i$

- (0) Setze  $A^{(1)} := P^T A P$  mit  $P$  aus Korollar 3.4,  $i := 1$
- (1) Berechne eine QR-Zerlegung  $A^{(i)} - \sigma_i \text{Id} = Q^{(i)} R^{(i)}$
- (2) Setze  $A^{(i+1)} := R^{(i)} Q^{(i)} + \sigma_i \text{Id}$ ,  $i := i + 1$  und fahre fort bei (1)

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

Analog zur einfachen Version des Algorithmus gilt

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

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

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

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

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

**Bemerkung 3.11** Für nichtsymmetrische Matrizen transformiert man  $A$  in Schritt (0) zunächst auf die sogenannte *Hessenberg-Gestalt*

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

statt auf Tridiagonalgestalt. Der QR-Algorithmus berechnet dann — unter geeigneten Voraussetzungen — ausgehend von der Hessenberg-Form iterativ eine obere Dreiecksmatrix,

deren Diagonaleinträge auf Grund des Satzes von Schur (siehe Schwarz/Köckler [8], Satz 5.12) die Eigenwerte approximieren. Für komplexe Eigenwerte  $\lambda_j = a + ib$  erhält man dabei einen  $2 \times 2$ -Block der Form

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

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



# Kapitel 4

## Interpolation

Die Interpolation von Funktionen oder Daten ist ein häufig auftretendes Problem sowohl in der Mathematik als auch in vielen Anwendungen.

Das allgemeine Problem, die sogenannte *Dateninterpolation*, entsteht, wenn wir eine Menge von Daten  $(x_i, f_i)$  für  $i = 0, \dots, n$  gegeben haben (z.B. Messwerte eines Experiments). Die Problemstellung ist nun wie folgt: Gesucht ist eine Funktion  $F$ , für die die Gleichung

$$F(x_i) = f_i \quad \text{für } i = 0, 1, \dots, n \quad (4.1)$$

gilt.

Ein wichtiger Spezialfall dieses Problems ist die *Funktionsinterpolation*: Nehmen wir an, dass wir eine reellwertige Funktion  $f : \mathbb{R} \rightarrow \mathbb{R}$  gegeben haben, die aber (z.B. weil keine explizite Formel bekannt ist) sehr kompliziert auszuwerten ist. Ein Beispiel einer solchen Funktion ist die in der Stochastik oft benötigte Gauß-Verteilungsfunktion

$$f(x) = \frac{1}{2\pi} \int_{-\infty}^x e^{-y^2/2} dy,$$

für die keine geschlossene Formel existiert.

Das Ziel der Interpolation liegt nun darin, eine Funktion  $F(x)$  zu bestimmen, die leicht auszuwerten ist, und für die für vorgegebene *Stützstellen*  $x_0, x_1, \dots, x_n$  die Gleichung

$$F(x_i) = f(x_i) \quad \text{für } i = 0, 1, \dots, n \quad (4.2)$$

gilt. Mit der Schreibweise

$$f_i = f(x_i),$$

erhalten wir hier wieder die Bedingung (4.1), weswegen (4.2) tatsächlich ein Spezialfall von (4.1) ist.

Wir werden in diesem Kapitel zum einen Verfahren zur Lösung von (4.1) entwickeln, die dann selbstverständlich auch auf den Spezialfall (4.2) anwendbar sind. Die Wichtigkeit dieses Spezialfalls liegt in diesem Zusammenhang darin, dass man bei der Interpolation einer Funktion  $f$  in natürlicher Weise einen *Interpolationsfehler* über den Abstand zwischen  $f$  und  $F$  definieren kann, und daher ein Maß für die Güte des Verfahrens erhält. Bei

der Dateninterpolation macht dies keinen rechten Sinn, das es ja keine Funktion  $f$  gibt, bezüglich der man einen Fehler messen könnte.

Zum anderen werden wir Verfahren betrachten, die speziell auf die Funktionsapproximation (4.2) zugeschnitten sind, da sich bei diesen die Wahl der Stützstellen  $x_i$  aus dem Verfahren ergibt, also nicht beliebig vorgegeben werden kann.

## 4.1 Polynominterpolation

Eine einfache aber oft sehr effektive Methode zur Interpolation ist die Wahl von  $F$  als Polynom, also als Funktion der Form

$$P(x) = a_0 + a_1x + a_2x^2 + \dots + a_mx^m. \quad (4.3)$$

Hierbei werden die Werte  $a_i$ ,  $i = 0, \dots, m$ , die *Koeffizienten* des Polynoms genannt. Die höchste auftretende Potenz (hier also  $m$ , falls  $a_m \neq 0$ ) heißt der *Grad* des Polynoms. Um zu betonen, dass wir hier Polynome verwenden, schreiben wir in diesem Abschnitt „ $P$ “ statt „ $F$ “ für die Interpolationsfunktion. Den Raum der Polynome vom Grad  $\leq m$  bezeichnen wir mit  $\mathcal{P}_m$ . Dieser Funktionenraum ist ein  $m+1$ -dimensionaler Vektorraum über  $\mathbb{R}$  bzw.  $\mathbb{C}$  mit Basis  $\mathcal{B} = \{1, x, \dots, x^m\}$ , da Addition von Polynomen und Multiplikation mit Skalaren wieder ein Polynom des selben Grads ergeben. Andere Basen dieses Vektorraums werden in den Übungen behandelt.

Das Problem der Polynominterpolation liegt nun darin, ein Polynom  $P$  zu bestimmen, das (4.1) erfüllt. Zunächst einmal müssen wir uns dazu überlegen, welchen Grad das gesuchte Polynom haben soll. Hier hilft uns der folgende Satz.

**Satz 4.1** Sei  $n \in \mathbb{N}$  und seien Daten  $(x_i, f_i)$  für  $i = 0, \dots, n$  gegeben, so dass die Stützstellen paarweise verschieden sind, d.h.  $x_i \neq x_j$  für alle  $i \neq j$ . Dann gibt es genau ein Polynom  $P \in \mathcal{P}_n$ , also vom Grad  $\leq n$ , das die Bedingung

$$P(x_i) = f_i \quad \text{für } i = 0, 1, \dots, n$$

erfüllt.

**Beweis:** Die Koeffizienten  $a_i$  des interpolierenden Polynoms erfüllen das lineare Gleichungssystem

$$\begin{pmatrix} 1 & x_0 & \dots & x_0^n \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & \dots & x_n^n \end{pmatrix} \begin{pmatrix} a_0 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} f_0 \\ \vdots \\ f_n \end{pmatrix}.$$

Die Determinante dieser Matrix ist

$$\prod_{i=0}^n \left( \prod_{j=i+1}^n (x_j - x_i) \right)$$

und ist damit ungleich Null, falls die  $x_i$  paarweise verschieden sind. Also ist die Matrix invertierbar und das Gleichungssystem besitzt eine eindeutige Lösung.  $\square$

Für  $n + 1$  gegebene Datenpunkte  $(x_i, f_i)$  “passt“ also gerade ein Polynom vom Grad  $n$ .

Nun ist es aus verschiedenen Gründen nicht besonders effizient, dieses lineare Gleichungssystem tatsächlich zu lösen, um die  $a_i$  zu bestimmen (wir erinnern daran, dass die direkte Lösung des linearen Gleichungssystems den Aufwand der Ordnung  $O(n^3)$  hat). Wir betrachten daher eine andere Technik zur Berechnung des Polynoms  $P$ . Beachte, dass diese das gleiche Polynom liefert, auch wenn es auf andere Art dargestellt wird.

#### 4.1.1 Lagrange–Polynome und baryzentrische Koordinaten

Die Idee der Lagrange–Polynome beruht auf einer geschickten Darstellung für Polynome. Für die vorgegebenen Stützstellen  $x_0, x_1, \dots, x_n$  definieren wir für  $i = 0, \dots, n$  die *Lagrange–Polynome*  $L_i$  als

$$L_i(x) := \prod_{\substack{j=0 \\ j \neq i}}^n \frac{x - x_j}{x_i - x_j}.$$

Man rechnet leicht nach, dass diese Polynome alle vom Grad  $n$  sind, und darüberhinaus die Gleichung

$$L_i(x_k) = \begin{cases} 1 & \text{für } i = k \\ 0 & \text{für } i \neq k \end{cases}$$

erfüllen. Mit Hilfe der  $L_i$  kann man das Interpolationspolynom einfach explizit berechnen.

**Satz 4.2** Seien Daten  $(x_i, f_i)$  für  $i = 0, \dots, n$  mit paarweise verschiedenen Stützstellen  $x_i$  gegeben. Dann ist das eindeutige Interpolationspolynom  $P(x)$  mit  $P(x_i) = f_i$  gegeben durch

$$P(x) = \sum_{i=0}^n f_i L_i(x).$$

**Beweis:** Offensichtlich ist die angegebene Funktion ein Polynom vom Grad  $\leq n$ . Darüberhinaus gilt

$$P(x_k) = \sum_{i=0}^n \underbrace{f_i L_i(x_k)}_{\substack{=0 \text{ falls } i \neq k \\ =f_k \text{ falls } i=k}} = f_k,$$

also gerade die gewünschte Bedingung (4.1).  $\square$

Die Lagrange–Polynome sind orthogonal (sogar orthonormal) bezüglich des Skalarproduktes

$$\langle P, Q \rangle := \sum_{i=0}^n P(x_i)Q(x_i)$$

auf dem Raum der Polynome  $\mathcal{P}_n$  und bilden damit eine Orthonormalbasis von  $\mathcal{P}_n$  bezüglich dieses Skalarproduktes, da sich jedes Polynom vom Grad  $\leq n$  mittels

$$P = \sum_{i=0}^n P(x_i)L_i = \sum_{i=0}^n \langle P, L_i \rangle L_i$$

als Summe der  $L_i$  schreiben lässt. Wir werden später sehen, dass Orthogonalität (allerdings bezüglich anderer Skalarprodukte) eine nützliche Eigenschaft bei der Funktionsinterpolation ist.

**Beispiel 4.3** Betrachte die Daten  $(3, 68)$ ,  $(2, 16)$ ,  $(5, 352)$ . Die zugehörigen Lagrange-Polynome sind gegeben durch

$$L_0(x) = \frac{x-2}{3-2} \frac{x-5}{3-5} = -\frac{1}{2}(x-2)(x-5),$$

$$L_1(x) = \frac{x-3}{2-3} \frac{x-5}{2-5} = \frac{1}{3}(x-3)(x-5),$$

$$L_2(x) = \frac{x-2}{5-2} \frac{x-3}{5-3} = \frac{1}{6}(x-2)(x-3).$$

Damit erhalten wir

$$P(x) = -68 \frac{1}{2}(x-2)(x-5) + 16 \frac{1}{3}(x-3)(x-5) + 352 \frac{1}{6}(x-2)(x-3).$$

Für  $x = 3$  ergibt sich  $P(3) = -68 \frac{1}{2}(3-2)(3-5) = 68$ , für  $x = 2$  berechnet man  $P(2) = 16 \frac{1}{3}(2-3)(2-5) = 16$  und für  $x = 5$  erhalten wir  $P(5) = 352 \frac{1}{6}(5-2)(5-3) = 352$ .  $\square$

Durch Abzählen der notwendigen Operationen sieht man, dass die direkte Auswertung des Polynoms  $P$  in dieser Form den Aufwand  $O(n^2)$  besitzt, also deutlich effizienter als die Lösung eines linearen Gleichungssystems ist. Für eine effiziente direkte Auswertung sollte man die Nenner der Lagrange-Polynome vorab berechnen und speichern, damit diese nicht bei jeder Auswertung von  $P$  erneut berechnet werden müssen.

Es geht aber noch effizienter, wenn wir die Auswertung der Lagrange-Polynome geschickt umformulieren. Dazu schreiben wir den Zähler von

$$L_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^n \frac{x - x_j}{x_i - x_j}.$$

als

$$\frac{\ell(x)}{x - x_i} \quad \text{mit} \quad \ell(x) := \prod_{j=0}^n x - x_j.$$

Der Nenner schreiben wir mittels der sogenannten *baryzentrischen Koordinaten*

$$w_i := \prod_{\substack{j=0 \\ j \neq i}}^n \frac{1}{x_i - x_j}.$$

Dann gilt

$$L_i(x) = \ell(x) \frac{w_i}{x - x_i}$$

und damit

$$P(x) = \sum_{i=0}^n L_i(x) f_i = \sum_{i=0}^n \ell(x) \frac{w_i}{x - x_i} f_i = \ell(x) \sum_{i=0}^n \frac{w_i}{x - x_i} f_i.$$



**Beispiel 4.4** Betrachte wiederum die Daten  $(3; 68)$ ,  $(2; 16)$ ,  $(5; 352)$ . Das zugehörige  $\ell$  ist gegeben durch

$$\ell(x) = (x - 2)(x - 3)(x - 5)$$

und die  $w_i$  berechnen sich zu

$$w_0 = \frac{1}{3-2} \frac{1}{3-5} = -\frac{1}{2},$$

$$w_1 = \frac{1}{2-3} \frac{1}{2-5} = \frac{1}{3},$$

$$w_3 = \frac{1}{5-2} \frac{1}{5-3} = \frac{1}{6}.$$

Damit erhalten wir

$$\begin{aligned} P(x) &= \ell(x) \left( \frac{-\frac{1}{2}}{x-3} 68 + \frac{\frac{1}{3}}{x-2} 16 + \frac{\frac{1}{6}}{x-5} 352 \right) \\ &= -\frac{1}{2}(x-2)(x-5) 68 + \frac{1}{3}(x-3)(x-5) 16 + \frac{1}{6}(x-2)(x-3) 352, \end{aligned}$$

also — wie zu erwarten — das gleiche Polynom wie oben.  $\square$

Um dieses Verfahren effizient zu implementieren, teilen wir die Berechnung in zwei Algorithmen auf.

**Algorithmus 4.5 (Berechnung der baryzentrischen Koordinaten)**

Eingabe: Stützstellen  $x_0, \dots, x_n$

- (1) für  $i$  von 0 bis  $n$ :
- (2)     setze  $w_i := 1$
- (3)     für  $j$  von 0 bis  $n$ :
- (4)         falls  $j \neq i$ , setze  $w_i := w_i / (x_i - x_j)$
- (6) Ende der Schleifen

Ausgabe: baryzentrische Koordinaten  $w_0, \dots, w_n$   $\square$

Durch Abzählen der Operationen sieht man leicht, dass die Berechnung der  $w_i$  gerade  $2(n+1)n = 2n^2 + 2n = O(n^2)$  Operationen benötigt. Dies entspricht der Ordnung des Aufwandes der direkten Auswertung von  $P$ . Der Trick liegt nun aber darin, die  $w_i$  einmal vorab zu berechnen und die gespeicherten Werte in der Auswertung von  $P$  zu verwenden.

**Algorithmus 4.6 (Auswertung des Interpolationspolynoms)**

Eingabe: Stützstellen  $x_0, \dots, x_n$ , Werte  $f_0, \dots, f_n$ , baryzentrische Koordinaten  $w_0, \dots, w_n$ , Auswertungsstelle  $x$

- (0) setze  $l := 1$ ,  $s := 0$  (Variablen für  $\ell$  und  $\sum_{i=0}^n \frac{w_i}{x-x_i} f_i$ )
- (1) für  $i$  von 0 bis  $n$

- (2) setze  $y := x - x_i$
- (3) falls  $y = 0$  ist, setze  $P := f_i$  und beende den Algorithmus
- (4) setze  $l := l * y$
- (5) setze  $s := s + w_i * f_i / y$
- (6) Ende der Schleife
- (7) Setze  $P := l * s$

Ausgabe: Polynomwert  $P = P(x)$  □

Durch Abzählen der Operationen sieht man: die Auswertung benötigt gerade  $5(n+1)+1 = 5n+6 = O(n)$  Operationen. Sind also die  $w_i$  einmal berechnet, so ist die Auswertung für ein gegebenes  $x$  dies deutlich weniger aufwändig als die direkte Auswertung von  $P$ . Dies ist z.B. bei der grafischen Darstellung des Polynoms ein wichtiger Vorteil, da das Polynom dabei für viele verschiedene  $x$  ausgewertet werden muss.

### 4.1.2 Kondition

In diesem Abschnitt wollen wir die Kondition der Polynominterpolation betrachten, wobei wir das Polynominterpolationsproblem für fest vorgegebene Stützstellen betrachten. In diesem Fall ist die Abbildung

$$\phi : (f_0, \dots, f_n) \mapsto \sum_{i=0}^n f_i L_i$$

des Datenvektors  $(f_0, \dots, f_n)$  auf das interpolierende Polynom  $P \in \mathcal{P}_n$  eine lineare Abbildung  $\phi : \mathbb{R}^{n+1} \rightarrow \mathcal{P}_n$ , weshalb wir die (absolute) Kondition  $\kappa_{abs}$  als induzierte Operatornorm

$$\kappa_{abs} := \|\phi\|_{\infty} = \sup_{\substack{f \in \mathbb{R}^{n+1} \\ f \neq 0}} \frac{\|\phi(f)\|_{\infty}}{\|f\|_{\infty}}$$

dieser linearen Abbildung berechnen können. Diese induzierte Operatornorm ist die Erweiterung der induzierten Matrixnorm auf lineare Abbildungen, die nicht notwendigerweise durch eine Matrix definiert sind. Im Gegensatz zu den linearen Gleichungssystemen verwenden wir hier die absolute Kondition, weil eine relative Definition hier keine anschauliche Interpretation besitzt. Auf dem Polynomraum  $\mathcal{P}_n$  verwenden wir dabei die Maximumsnorm

$$\|P\|_{\infty} := \max_{x \in [a, b]} |P(x)|,$$

des Raums der stetigen reellwertigen Funktionen  $C([a, b], \mathbb{R})$ , wobei wir  $a = \min_{i=0, \dots, n} x_i$  und  $b = \max_{i=0, \dots, n} x_i$  wählen (beachte, dass wir keine Ordnung der Stützstellen  $x_i$  vorausgesetzt haben).

**Satz 4.7** Seien  $x_0, x_1, \dots, x_n$  paarweise verschiedene Stützstellen und  $L_i$  die zugehörigen Lagrange–Polynome. Dann ist die absolute Kondition des Interpolationsproblems mit diesen Stützstellen gegeben durch

$$\kappa_{abs} = \Lambda_n := \left\| \sum_{i=0}^n |L_i| \right\|_{\infty},$$

wobei  $\Lambda_n$  als *Lebesgue-Konstante* bezeichnet wird.

**Beweis:** Es gilt

$$\begin{aligned} |\phi(f)(x)| &= \left| \sum_{i=0}^n f_i L_i(x) \right| \leq \sum_{i=0}^n |f_i| |L_i(x)| \\ &\leq \|f\|_\infty \max_{x \in [a,b]} \sum_{i=0}^n |L_i(x)| = \|f\|_\infty \left\| \sum_{i=0}^n |L_i| \right\|_\infty = \|f\|_\infty \Lambda_n, \end{aligned}$$

für alle  $x \in [a, b]$ , woraus  $\|\phi(f)\|_\infty \leq \|f\|_\infty \Lambda_n$  für alle  $f \in \mathbb{R}^{n+1}$  und damit  $\|\phi\| \leq \Lambda_n$  folgt. Für die umgekehrte Richtung konstruieren wir ein  $g \in \mathbb{R}^{n+1}$  so, dass

$$|\phi(g)(x^*)| = \|g\|_\infty \left\| \sum_{i=0}^n |L_i| \right\|_\infty$$

für ein  $x^* \in [a, b]$  gilt. Sei dazu  $x^* \in [a, b]$  die Stelle, an der die Funktion  $x \mapsto \sum_{i=0}^n |L_i(x)|$  ihr Maximum annimmt, also

$$\sum_{i=0}^n |L_i(x^*)| = \left\| \sum_{i=0}^n |L_i| \right\|_\infty = \Lambda_n.$$

Wir wählen  $g \in \mathbb{R}^{n+1}$  als  $g_i = \operatorname{sgn}(L_i(x^*))$ . Dann gilt  $\|g\|_\infty = 1$  und  $g_i L_i(x^*) = |L_i(x^*)|$ , also

$$\|\phi(g)\|_\infty \geq |\phi(g)(x^*)| = \left| \sum_{i=0}^n g_i L_i(x^*) \right| = \left| \sum_{i=0}^n |L_i(x^*)| \right| = \|g\|_\infty \left| \sum_{i=0}^n |L_i(x^*)| \right| = \|g\|_\infty \Lambda_n,$$

weswegen  $\|\phi\| \geq \Lambda_n$  ist. Zusammen erhalten wir also die Behauptung  $\kappa_{\text{abs}} = \|\phi\| = \Lambda_n$ .  $\square$

Die Zahl  $\Lambda_n$  hängt natürlich von der Anzahl und Lage der Stützstellen ab. In der folgenden Tabelle 4.1 sind die Konditionen für das Intervall  $[-1, 1]$  und verschiedene Anzahlen äquidistante Stützstellen  $x_i = -1 + 2i/n$  sowie für die sogenannten Tschebyscheff-Stützstellen  $x_i = \cos[(2i + 1)\pi/(2n + 2)]$ , die wir in Abschnitt 4.2 näher kennen lernen werden, dargestellt.

$n$	$\kappa_{\text{abs}}$ für äquidistante Stützstellen	$\kappa_{\text{abs}}$ für Tschebyscheff-Stützstellen
5	3.11	2.10
10	29.89	2.49
15	512.05	2.73
20	10986.53	2.90
60	$2.97 \cdot 10^{15}$	3.58
100	$1.76 \cdot 10^{27}$	3.90

Tabelle 4.1: Kondition  $\kappa_{\text{abs}}$  für verschiedene Stützstellen

Man sieht, dass das Problem für äquidistante Stützstellen und große  $n$  sehr schlecht konditioniert ist.

### 4.1.3 Fehlerabschätzungen

Wir betrachten in diesem Abschnitt das Problem der Funktionsinterpolation (4.2) für eine gegebene Funktion  $f : \mathbb{R} \rightarrow \mathbb{R}$ . Wir wollen abschätzen, wie groß der Abstand des interpolierenden Polynoms  $P$  von der Funktion  $f$  ist. Hierbei bezeichnen wir mit  $[a, b]$  ein Interpolationsintervall mit der Eigenschaft, dass alle Stützstellen  $x_i \in [a, b]$  erfüllen und wir verwenden wieder die Maximumsnorm

$$\|f\|_\infty := \max_{x \in [a, b]} |f(x)|.$$

Der folgende Satz gibt eine Abschätzung für den Abstand in Abhängigkeit von den Stützstellen an. Hierbei bezeichnet  $f^{(k)}$  die  $k$ -te Ableitung der Funktion  $f$ .

**Satz 4.8** Sei  $f$   $(n + 1)$ -mal stetig differenzierbar und sei  $P$  das Interpolationspolynom zu den paarweise verschiedenen Stützstellen  $x_0, \dots, x_n$ . Dann gelten die folgenden Aussagen.

(i) Für alle  $x \in [a, b]$  gibt es ein  $\xi \in [a, b]$ , so dass die Gleichung

$$f(x) - P(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!} (x - x_0)(x - x_1) \cdots (x - x_n)$$

gilt.

(ii) Für alle  $x \in [a, b]$  gilt die Abschätzung

$$|f(x) - P(x)| \leq \|f^{(n+1)}\|_\infty \left| \frac{(x - x_0)(x - x_1) \cdots (x - x_n)}{(n+1)!} \right|.$$

(iii) Es gilt die Abschätzung

$$\|f - P\|_\infty \leq \|f^{(n+1)}\|_\infty \frac{(b - a)^{n+1}}{(n+1)!}.$$

**Beweis:** (i) Wähle ein  $x \in [a, b]$  mit  $x \neq x_i$ ,  $i = 0, \dots, n$ , und setze

$$c = \frac{f(x) - P(x)}{(x - x_0) \cdots (x - x_n)}.$$

Mit diesem  $c$  definieren wir die Funktion

$$\Delta(y) = f(y) - P(y) - c(y - x_0) \cdots (y - x_n).$$

Dann ist  $\Delta(y)$  wieder  $(n + 1)$ -mal stetig differenzierbar und hat (mindestens) die  $n + 2$  Nullstellen  $y = x_0, \dots, x_n$  und  $y = x$ . Wir nummerieren diese Nullstellen aufsteigend mit der Bezeichnung  $y_0^{(0)} < y_1^{(0)} < \dots < y_{n+1}^{(0)}$ . Nach dem Satz von Rolle gilt: Zwischen je zwei Nullstellen der Funktion  $\Delta(y)$  liegt (mindestens) eine Nullstelle ihrer Ableitung  $\Delta'(y)$ . Also liegt für jedes  $i = 0, \dots, n$  zwischen den Werten  $y_i^{(0)}$  und  $y_{i+1}^{(0)}$  ein Wert  $y_i^{(1)}$  mit  $\Delta'(y_i^{(1)}) = 0$ , und wir erhalten  $n + 1$  Nullstellen  $y_0^{(1)} < y_1^{(1)} < \dots < y_n^{(1)}$  für die Ableitung  $\Delta'(y) = \Delta^{(1)}(y)$ .

Indem wir induktiv fortfahren, erhalten wir  $n+2-k$  Nullstellen  $y_0^{(k)} < y_1^{(k)} < \dots < y_{n+1-k}^{(k)}$  für die Ableitung  $\Delta^{(k)}$  und damit für  $k = n+1$  eine Nullstelle  $\xi = y_0^{(n+1)}$  für die Funktion  $\Delta^{(n+1)}$ .

Da  $P$  ein Polynom vom Grad  $\leq n$  ist, folgt  $P^{(n+1)}(y) \equiv 0$ , außerdem gilt  $[(y-x_0)\cdots(y-x_n)]^{(n+1)} = (n+1)!$  für alle  $y \in \mathbb{R}$ . Damit erhalten wir

$$0 = \Delta^{(n+1)}(\xi) = f^{(n+1)}(\xi) - c(n+1)! = f^{(n+1)}(\xi) - \frac{f(x) - P(x)}{(x-x_0)\cdots(x-x_n)}(n+1)!,$$

also

$$f^{(n+1)}(\xi) = \frac{f(x) - P(x)}{(x-x_0)\cdots(x-x_n)}(n+1)!.$$

Auflösen nach  $f(x) - P(x)$  liefert die Gleichung in (i).

(ii) Diese Abschätzung folgt aus (i) wegen

$$\begin{aligned} |f(x) - P(x)| &= \left| \frac{f^{(n+1)}(\xi)}{(n+1)!} (x-x_0)(x-x_1)\cdots(x-x_n) \right| \\ &\leq \max_{y \in [a,b]} \left| \frac{f^{(n+1)}(y)}{(n+1)!} (x-x_0)(x-x_1)\cdots(x-x_n) \right|, \end{aligned}$$

da  $\xi \in [a, b]$  ist.

(iii) Für alle  $x$  und  $x_i$  aus  $[a, b]$  gilt die Abschätzung

$$|x - x_i| \leq b - a.$$

Damit erhalten wir aus (ii)

$$\|f - P\|_\infty \leq \|f^{(n+1)}\|_\infty \frac{(b-a)^{n+1}}{(n+1)!},$$

also gerade die Behauptung.  $\square$

Wir illustrieren diese Abschätzung an 2 Beispielen.

**Beispiel 4.9** Betrachte die Funktion  $f(x) = \sin(x)$  auf dem Intervall  $[0, 2\pi]$ . Die Ableitungen von  $f$  sind

$$f^{(1)}(x) = \cos(x), \quad f^{(2)}(x) = -\sin(x), \quad f^{(3)}(x) = -\cos(x), \quad f^{(4)}(x) = \sin(x), \quad \dots$$

Für alle diese Funktionen gilt  $|f^{(k)}(x)| \leq 1$  für alle  $x \in \mathbb{R}$ . Mit äquidistanten Stützstellen  $x_i = 2\pi i/n$  ergibt sich damit die Abschätzung

$$|f(x) - P(x)| \leq \max_{y \in [a,b]} |f^{(n+1)}(y)| \frac{(b-a)^{n+1}}{(n+1)!} \leq \frac{(2\pi)^{n+1}}{(n+1)!}.$$

Dieser Term konvergiert für wachsende  $n$  sehr schnell gegen 0, weswegen man schon für kleine  $n$  eine sehr gute Übereinstimmung der Funktionen erwarten kann.  $\square$

**Beispiel 4.10** Betrachte die sogenannte *Runge-Funktion*  $f(x) = 1/(1+x^2)$  auf dem Intervall  $[-5, 5]$ . Die exakten Ableitungen führen zu ziemlich komplizierten Termen, man kann aber nachrechnen, dass für gerade  $n$  die Gleichung

$$\max_{y \in [a, b]} |f^{(n)}(y)| = |f^{(n)}(0)| = n!$$

gilt, für ungerade  $n$  gilt zumindest approximativ

$$\max_{y \in [a, b]} |f^{(n)}(y)| \approx n!.$$

Damit ergibt sich

$$|f(x) - P(x)| \leq \max_{y \in [a, b]} |f^{(n+1)}(y)| \frac{(b-a)^{n+1}}{(n+1)!} \approx (n+1)! \frac{(b-a)^{n+1}}{(n+1)!} = 10^{n+1}.$$

Dieser Term wächst für große  $n$  gegen unendlich, weswegen die Abschätzung hier keine brauchbare Fehlerschranke liefert, und tatsächlich zeigen sich bei dieser Funktion für äquidistante Stützstellen bei numerischen Tests große Probleme; insbesondere lässt sich für wachsende  $n$  keine Konvergenz erzielen, statt dessen stellt man für große  $n$  starke Schwankungen (“Oszillationen”) des interpolierenden Polynoms fest.  $\square$

Es gibt also zwei Gründe, aus denen die Polynominterpolation mit äquidistanten Stützstellen problematisch ist: Zum einen kann bei ungeeigneten Funktionen der Interpolationsfehler unabhängig von der Anzahl der Stützstellen groß sein, zum anderen neigen die numerisch erzeugten Interpolationspolynome wegen der extrem schlechten Kondition bei großer Stützstellenanzahl zu starken Oszillationen, selbst wenn die zu interpolierende Funktion gutartig ist.

Wir werden daher in den nächsten Abschnitten weitere Möglichkeiten zur Interpolation betrachten, die diese Probleme umgehen, indem sie entweder besser positionierte Stützstellen verwenden oder ohne Polynome hohen Grades auskommen.

## 4.2 Funktionsinterpolation und orthogonale Polynome

In diesem Abschnitt beschäftigen wir uns speziell mit der Frage der Funktionsinterpolation (4.2) durch Polynome. Wie bereits erwähnt, unterscheidet sich diese aus algorithmischer Sicht von der Dateninterpolation (4.1) dadurch, dass man die Stützstellen  $x_i$  frei wählen kann. Dies führt auf die Frage, wie man diese Stützstellen für ein gegebenes Interpolationsintervall  $[a, b]$  optimal wählen kann. Wir wollen dieses Problem lösen *ohne* die Kenntnis der zu interpolierenden Funktion  $f$  vorauszusetzen.

### 4.2.1 Orthogonale Polynome

Ein wesentliches Hilfsmittel hierbei sind *orthogonale Polynome*, die wir zunächst betrachten wollen. Orthogonalität ist immer bezüglich eines Skalarproduktes definiert, hier verwenden wir das folgende Skalarprodukt.

**Definition 4.11** Sei  $\omega : (a, b) \rightarrow \mathbb{R}^+$  eine positive und auf  $[a, b]$  Lebesgue-integrierbare Gewichtsfunktion<sup>1</sup>. Dann definieren wir auf dem Raum  $\mathcal{P}$  der Polynome das Skalarprodukt

$$\langle P_1, P_2 \rangle_\omega := \int_a^b \omega(x) P_1(x) P_2(x) dx$$

für  $P_1, P_2 \in \mathcal{P}$ . Mit

$$\|P\|_\omega := \sqrt{\langle P, P \rangle_\omega} = \sqrt{\int_a^b \omega(x) (P(x))^2 dx}$$

bezeichnen wir die zugehörige Norm. □

Mit diesem Skalarprodukt können wir nun Orthogonalität definieren.

**Definition 4.12** Eine Folge  $(P_k)_{k \in \mathbb{N}_0}$  von Polynomen mit  $P_k \in \mathcal{P}_k$  *exakt* vom Grad  $k$  (also mit führendem Koeffizienten  $\neq 0$ ) heißt *orthogonal* bezüglich  $\omega$ , falls

$$\langle P_i, P_j \rangle_\omega = 0 \text{ für } i \neq j \quad \text{und} \quad \langle P_i, P_i \rangle_\omega = \|P_i\|_\omega^2 =: \gamma_i > 0$$

gilt. □

Der folgende Satz zeigt, dass orthogonale Polynome immer existieren und darüberhinaus durch eine einfache Rekursionsformel berechnet werden können.

**Satz 4.13** Zu jeder Gewichtsfunktion  $\omega : [a, b] \rightarrow \mathbb{R}^+$  gibt es eindeutig bestimmte Orthogonalpolynome  $(P_k)_{k \in \mathbb{N}_0}$  gemäß Definition 4.12 mit führendem Koeffizienten = 1. Sie erfüllen die *Rekursionsgleichung*

$$P_k(x) = (x + b_k) P_{k-1}(x) + c_k P_{k-2}(x), \quad k = 1, 2, \dots$$

mit den Anfangswerten  $P_{-1} \equiv 0, P_0 \equiv 1$  sowie den Koeffizienten

$$b_k = -\frac{\langle x P_{k-1}, P_{k-1} \rangle_\omega}{\langle P_{k-1}, P_{k-1} \rangle_\omega} \quad \text{und} \quad c_k = -\frac{\langle P_{k-1}, P_{k-1} \rangle_\omega}{\langle P_{k-2}, P_{k-2} \rangle_\omega}.$$

**Beweis:** Wir beweisen den Satz per Induktion über  $k$ . Offenbar ist  $P_0 \equiv 1$  das einzige Polynom vom Grad 0 mit führendem Koeffizienten gleich 1.

Für den Induktionsschritt  $k-1 \rightarrow k$  seien  $P_0, P_1, \dots, P_{k-1}$  die ersten  $k$  orthogonalen Polynome, die die angegebene Rekursionsgleichung erfüllen. Sei  $P_k \in \mathcal{P}_k$  beliebig mit führendem Koeffizienten = 1. Da  $P_{k-1} \in \mathcal{P}_{k-1}$  ist und sich die führenden Koeffizienten aufheben, folgt  $P_k - x P_{k-1} \in \mathcal{P}_{k-1}$ . Da die  $P_0, P_1, \dots, P_{k-1}$  wegen der Orthogonalität linear unabhängig

<sup>1</sup>Beachte, dass  $\omega$  an den Randpunkten  $a$  und  $b$  nicht definiert sein muss, wenn wir das Lebesgue-Integral verwenden.

sind, bilden sie eine Basis von  $\mathcal{P}_{k-1}$ , sogar eine Orthogonalbasis bzgl.  $\langle \cdot, \cdot \rangle_\omega$ . Also gilt die Gleichung

$$P_k - xP_{k-1} = \sum_{j=0}^{k-1} d_j P_j \quad \text{mit} \quad d_j = \frac{\langle P_k - xP_{k-1}, P_j \rangle_\omega}{\langle P_j, P_j \rangle_\omega}.$$

Wir wollen nun Bedingungen an die Koeffizienten  $d_j$  ermitteln, unter der Annahme, dass  $P_k$  orthogonal zu  $P_j$ ,  $j = 0, \dots, k-1$  ist. Falls  $P_k$  diese Orthogonalitätseigenschaft besitzt, so muss notwendigerweise gelten

$$d_j = -\frac{\langle xP_{k-1}, P_j \rangle_\omega}{\langle P_j, P_j \rangle_\omega} = -\frac{\langle P_{k-1}, xP_j \rangle_\omega}{\langle P_j, P_j \rangle_\omega}.$$

Für  $j = 0, \dots, k-3$  liegt  $xP_j \in \mathcal{P}_{k-2}$ , lässt sich also als Linearkombination von  $P_0, \dots, P_{k-2}$  ausdrücken, weswegen  $\langle P_{k-1}, xP_j \rangle_\omega = 0$  und damit  $d_0 = d_1 = \dots = d_{k-3} = 0$  gelten muss. Für die verbleibenden Koeffizienten  $d_{k-1}$  und  $d_{k-2}$  muss gelten

$$d_{k-1} = -\frac{\langle P_{k-1}, xP_{k-1} \rangle_\omega}{\langle P_{k-1}, P_{k-1} \rangle_\omega}, \quad d_{k-2} = -\frac{\langle P_{k-1}, xP_{k-2} \rangle_\omega}{\langle P_{k-2}, P_{k-2} \rangle_\omega} = -\frac{\langle P_{k-1}, P_{k-1} \rangle_\omega}{\langle P_{k-2}, P_{k-2} \rangle_\omega},$$

wobei sich die letzte Gleichung aus der Induktionsannahme

$$P_{k-1}(x) = (x + b_{k-1})P_{k-2}(x) + c_{k-1}P_{k-3}(x)$$

mittels

$$\langle P_{k-1}, P_{k-1} \rangle_\omega = \langle P_{k-1}, xP_{k-2} \rangle_\omega + \underbrace{\langle P_{k-1}, b_{k-1}P_{k-2} + c_{k-1}P_{k-3} \rangle_\omega}_{= 0 \text{ wegen Orthogonalität}}$$

ergibt. Also folgt

$$P_k = xP_{k-1} + d_{k-1}P_{k-1} + d_{k-2}P_{k-2} = (x + b_k)P_{k-1} + c_kP_{k-2}.$$

Da die Koeffizienten  $b_k$  und  $c_k$  und das Polynom  $P_k$  hierdurch eindeutig bestimmt sind, folgt die Behauptung.  $\square$

**Bemerkung 4.14** Für numerische Zwecke ist die hier angegebene Rekursionsformel i.A. schlecht geeignet, da die Auswertung dieser Formel numerisch sehr instabil, d.h. anfällig gegenüber Rundungsfehlern ist. Numerisch stabile Algorithmen zur Berechnung orthogonaler Polynome werden z.B. in Kapitel 6 des Buches von Deuffhard/Hohmann [2] betrachtet.  $\square$

Satz 4.13 erlaubt es uns, bei der Konstruktion orthogonaler Polynome von der Rekursionsformel für gegebene Koeffizienten auszugehen und dann das zugehörige Skalarprodukt  $\langle \cdot, \cdot \rangle_\omega$  zu identifizieren. Beachte, dass nicht jede Rekursionsformel automatisch Polynome mit führendem Koeffizienten = 1 erzeugt. Man kann diese normierten Polynome aber leicht durch eine geeignete nachträgliche Skalierung erhalten.

Wir werden jetzt eine spezielle Rekursionsformel und die daraus entstehenden *Tschebyscheff-Polynome* näher betrachten, die bei der Interpolation eine wichtige Rolle spielen. Im Kapitel über numerische Integration werden wir weitere Familien orthogonaler Polynome kennen lernen.



Die für die Interpolation wichtige Rekursionsformel ist gegeben durch

$$T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x). \quad (4.4)$$

mit Anfangswerten  $T_0(x) = 1$ ,  $T_1(x) = x$  (wir skalieren hier etwas anders als in Satz 4.13, was im Folgenden einige Rechnungen vereinfacht). Der folgende Satz fasst die Eigenschaften der zugehörigen orthogonalen Polynome, der sogenannten *Tschebyscheff-Polynome* zusammen.

**Satz 4.15** Für die durch die Rekursion (4.4) gegebenen Tschebyscheff-Polynome  $T_k$ ,  $k = 0, 1, 2, \dots$ , gelten die folgenden Aussagen.

- (i) Die Polynome sind für  $x \in [-1, 1]$  gegeben durch  $T_k(x) = \cos(k \arccos(x))$ .  
(ii) Für  $k \geq 1$  sind die  $T_k$  von der Form

$$T_k(x) = 2^{k-1}x^k + \sum_{i=0}^{k-1} t_{k,i}x^i$$

- (iii) Jedes  $T_k$  besitzt genau die  $k$  Nullstellen

$$x_{k,l} = \cos\left(\frac{2l-1}{2k}\pi\right) \quad \text{für } l = 1, \dots, k.$$

Diese werden *Tschebyscheff-Knoten* genannt.

- (iv) Die  $T_k$  sind orthogonal bezüglich der auf  $(-1, 1)$  definierten Gewichtsfunktion

$$\omega(x) = \frac{1}{\sqrt{1-x^2}}.$$

Genauer gilt

$$\langle T_i, T_j \rangle_\omega = \begin{cases} 0, & \text{falls } i \neq j \\ \pi, & \text{falls } i = j = 0 \\ \pi/2, & \text{falls } i = j > 0 \end{cases}$$

**Beweis:** (i)–(iii) folgen durch Nachrechnen, vgl. Aufgabe 33 (a)–(c) auf dem 9. Übungsblatt.

- (iv): Wir betrachten das Skalarprodukt

$$\langle T_i, T_j \rangle_\omega = \int_{-1}^1 \omega(x)T_i(x)T_j(x)dx$$

Zum Lösen dieses Integrals verwenden wir die Variablensubstitution  $x = \cos(\alpha)$ , also  $dx = -\sin(\alpha)d\alpha$ , und die Darstellung der  $T_k$  aus (i). Damit folgt

$$\begin{aligned} \int_{-1}^1 \omega(x)T_i(x)T_j(x)dx &= \int_\pi^0 \frac{1}{\underbrace{\sqrt{1-\cos^2(\alpha)}}_{=\sin(\alpha)}} \cos(i\alpha) \cos(j\alpha)(-\sin(\alpha))d\alpha \\ &= \int_0^\pi \cos(i\alpha) \cos(j\alpha)d\alpha = \frac{1}{2} \int_{-\pi}^\pi \cos(i\alpha) \cos(j\alpha)d\alpha \\ &= \frac{1}{4} \int_{-\pi}^\pi \cos((i+j)\alpha) + \cos((i-j)\alpha)d\alpha, \end{aligned}$$

wobei wir im vorletzten Schritt die Gleichung  $\cos(a) = \cos(-a)$  und im letzten Schritt die Gleichung  $2\cos(a)\cos(b) = \cos(a+b) + \cos(a-b)$  verwendet haben.

Für  $i \neq j$  folgt wegen  $\sin(k\pi) = 0$  für  $k \in \mathbb{Z}$

$$\langle T_i, T_j \rangle_\omega = \frac{1}{4} \left[ \frac{1}{i+j} \sin((i+j)\alpha) + \frac{1}{i-j} \sin((i-j)\alpha) \right]_{-\pi}^{\pi} = 0,$$

für  $i = j = 0$  folgt

$$\langle T_0, T_0 \rangle_\omega = \frac{1}{4} \int_{-\pi}^{\pi} \cos(0) + \cos(0) d\alpha = \frac{1}{4} [2\alpha]_{-\pi}^{\pi} = \pi$$

und für  $i = j > 0$  folgt wiederum mit  $\sin(k\pi) = 0$

$$\langle T_i, T_i \rangle_\omega = \frac{1}{4} \int_{-\pi}^{\pi} \cos(2i\alpha) + \cos(0) d\alpha = \frac{1}{4} \left[ \frac{1}{2i} \sin(2i\alpha) + \alpha \right]_{-\pi}^{\pi} = \frac{\pi}{2}$$

□

Die Bedeutung der Tschebyscheff–Polynome für die Funktionsinterpolation ergibt sich aus der Fehlerabschätzung aus Satz 4.8(ii). Dort haben wir die Ungleichung

$$|f(x) - P(x)| \leq \|f^{(n+1)}\|_\infty \left| \frac{(x-x_0)(x-x_1)\cdots(x-x_n)}{(n+1)!} \right|. \quad (4.5)$$

für  $x \in [a, b]$  bewiesen. Wir wollen nun untersuchen, wie man die Stützstellen  $x_i$  wählen muss, so dass diese Fehlerschranke minimal wird. Da wir hierbei kein spezielles  $x$  vorgeben wollen (die Abschätzung soll für alle  $x$  optimal sein, also für  $\|f - P\|_\infty$ ), besteht die Aufgabe also darin, Stützstellen  $x_0, \dots, x_n$  zu finden, so dass der Ausdruck

$$\max_{x \in [a, b]} |(x-x_0)(x-x_1)\cdots(x-x_n)| \quad (4.6)$$

minimal wird.

O.B.d.A. betrachten wir nun das Intervall  $[a, b] = [-1, 1]$ , denn wenn wir auf  $[-1, 1]$  die optimalen Stützstellen  $x_i$  gefunden haben, so sind die mittels  $\tilde{x}_i = a + (x_i + 1)(b-a)/2$  definierten Stützstellen auf  $[a, b]$  ebenfalls optimal und es gilt

$$\max_{x \in [a, b]} |(x - \tilde{x}_0)(x - \tilde{x}_1)\cdots(x - \tilde{x}_n)| = \left(\frac{b-a}{2}\right)^{n+1} \max_{x \in [-1, 1]} |(x-x_0)(x-x_1)\cdots(x-x_n)|.$$

Definieren wir nun für beliebige Stützstellen  $x_0, \dots, x_n$   $R_{n+1}(x) = (x-x_0)(x-x_1)\cdots(x-x_n)$ , so definiert dies ein Polynom mit führendem Koeffizienten  $a_{n+1} = 1$ . Die Stützstellen  $x_i$  sind gerade die Nullstellen von  $R_{n+1}$  und der Ausdruck (4.6) ist gerade die Maximumsnorm  $\|R_{n+1}\|_\infty$ . Die Minimierung von (4.6) ist also äquivalent zur folgenden Problemstellung: Unter allen Polynomen  $R_{n+1}$  vom Grad  $n+1$  mit führendem Koeffizienten  $a_{n+1} = 1$  finde dasjenige mit kleinster Maximumsnorm auf  $[-1, 1]$ .

Der folgende Satz zeigt, dass das normierte Tschebyscheff–Polynom  $T_{n+1}/2^n$  gerade das gesuchte Polynom ist.

**Satz 4.16** Sei  $\|\cdot\|_\infty$  die Maximumsnorm auf  $[-1, 1]$ . Dann gilt

$$\|T_{n+1}/2^n\|_\infty \leq \|R_{n+1}\|_\infty$$

für jedes Polynom  $R_{n+1}$  der Form  $R_{n+1}(x) = (x - x_0)(x - x_1) \cdots (x - x_n)$  mit paarweise verschiedenen Nullstellen  $x_i$  im Intervall  $[-1, 1]$ .

Insbesondere minimieren die Tschebyscheff-Knoten

$$x_i = \cos\left(\frac{2i+1}{2n+2}\pi\right), \quad i = 0, \dots, n,$$

also die Nullstellen von  $T_{n+1}$ , den Ausdruck (4.6) und damit die Fehlerabschätzung (4.5).

**Beweis:** Aus der Darstellung  $T_{n+1}(x) = \cos((n+1)\arccos(x))$  folgt sofort  $\|T_{n+1}\|_\infty \leq 1$ . Außerdem gilt

$$\begin{aligned} |T_{n+1}(x)| = 1 &\Leftrightarrow (n+1)\arccos(x) = m\pi \text{ für ein } m \in \mathbb{N}_0 \\ &\Leftrightarrow x = \cos\left(\frac{m}{n+1}\pi\right) \text{ für ein } m \in \mathbb{N}_0. \end{aligned}$$

Wir schreiben  $\bar{x}_m := \cos\left(\frac{m}{n+1}\pi\right)$ . Beachte, dass dies genau  $n+2$  Stellen  $\bar{x}_0, \dots, \bar{x}_{n+1}$  definiert und dass  $T_{n+1}(\bar{x}_m) = 1$  ist, falls  $m$  gerade oder gleich 0 ist und  $T_{n+1}(\bar{x}_m) = -1$  gilt, falls  $m$  ungerade ist.

Wir zeigen nun, dass für jedes Polynom  $Q_{n+1}$  vom Grad  $n+1$  mit führendem Koeffizienten  $a_{n+1} = 2^n$  die Ungleichung

$$\|Q_{n+1}\|_\infty \geq \|T_{n+1}\|_\infty \quad (4.7)$$

Zum Beweis dieser Behauptung nehmen wir das Gegenteil an, d.h. es gelte  $\|Q_{n+1}\|_\infty < \|T_{n+1}\|_\infty$ . Wir betrachten die Differenz  $Q_{n+1} - T_{n+1}$ . Da sich die führenden Koeffizienten aufheben, ist dies ein Polynom vom Grad  $\leq n$ . An den  $n+2$  Punkten  $\bar{x}_m$  gilt nun

$$m \text{ gerade oder } 0: \quad T_{n+1}(\bar{x}_m) = 1, \quad Q_{n+1}(\bar{x}_m) < 1 \quad \Rightarrow \quad Q_{n+1}(\bar{x}_m) - T_{n+1}(\bar{x}_m) < 0$$

$$m \text{ ungerade:} \quad T_{n+1}(\bar{x}_m) = -1, \quad Q_{n+1}(\bar{x}_m) > -1 \quad \Rightarrow \quad Q_{n+1}(\bar{x}_m) - T_{n+1}(\bar{x}_m) > 0$$

Also wechselt  $Q_{n+1} - T_{n+1}$  in jedem der  $n+1$  Intervalle  $[\bar{x}_i, \bar{x}_{i+1}]$ ,  $i = 0, \dots, n$  sein Vorzeichen und besitzt damit (mindestens)  $n+1$  Nullstellen, was für ein Polynom vom Grad  $n$  nur möglich ist, wenn es konstant gleich Null ist. Also ist  $Q_{n+1} - T_{n+1} \equiv 0$ , damit  $Q_{n+1} = T_{n+1}$  was der Annahme  $\|Q_{n+1}\|_\infty < \|T_{n+1}\|_\infty$  widerspricht.

Die Behauptung ergibt sich nun sofort durch Skalierung von  $T_{n+1}$  und  $Q_{n+1}$  mit  $1/2^n$ , da sich jedes Polynom der im Satz angegebenen Form schreiben lässt als  $R_{n+1} = Q_{n+1}/2^n$  für ein  $Q_{n+1}$  aus (4.7).  $\square$

Für die Tschebyscheff-Stützstellen  $x_i$  erhält man damit in (4.6)

$$\max_{x \in [-1, 1]} |(x - x_0)(x - x_1) \cdots (x - x_n)| = \frac{1}{2^n}.$$

Für allgemeine Intervalle  $[a, b]$  ergibt sich daraus für die durch

$$\tilde{x}_i = a + (x_i + 1)(b - a)/2$$

gegebenen transformierten Stützstellen

$$\max_{x \in [a, b]} |(x - \tilde{x}_0)(x - \tilde{x}_1) \cdots (x - \tilde{x}_n)| = \frac{(b - a)^{n+1}}{2^{2n+1}} = 2 \left( \frac{b - a}{4} \right)^{n+1}.$$

Für die Runge-Funktion aus Beispiel 4.10 erhalten wir damit aus Satz 4.8(ii) auf dem Interpolationsintervall  $[-5, 5]$

$$\|f - P\|_\infty \leq 2 \left( \frac{5}{2} \right)^{n+1}.$$

Wiederum ergibt sich eine Fehlerschranke, die für  $n \rightarrow \infty$  divergiert. Überraschenderweise funktioniert die Interpolation der Runge-Funktion mit Tschebyscheff-Stützstellen aber trotzdem, vgl. Übungsaufgabe 34. Offenbar ist hier also die dieser Ungleichung zu Grunde liegende Abschätzung  $|f^{(n+1)}(\xi)| \leq \|f^{(n+1)}\|_\infty$  zu pessimistisch.

Beachte, dass die Randpunkte  $-1$  und  $1$  des Interpolationsintervalls keine Tschebyscheff-Knoten und damit keine Stützstellen sind. Wir interpolieren mit dieser Methode also auch außerhalb des durch die Stützstellen definierten Intervalls.

Die Implementierung dieser Stützstellen kann auf zwei verschiedene Weisen erfolgen: Zum einen kann man eines der Verfahren aus Abschnitt 4.1 mit den oben angegebenen Stützstellen  $x_i$  verwenden. Zum anderen ist es aber auch möglich, das interpolierende Polynom mit Hilfe der Basis  $\{T_0, \dots, T_n\}$  von  $\mathcal{P}_n$  auszudrücken, d.h. man kann Koeffizienten  $\gamma_k$  berechnen, so dass sich das interpolierende Polynom als

$$P_n(x) = \frac{1}{2}\gamma_0 T_0(x) + \sum_{k=1}^n \gamma_k T_k(x)$$

darstellen lässt, vgl. Übungsaufgabe 33(e).

### 4.3 Splineinterpolation

Wir haben gesehen, dass die Polynominterpolation aus Konditionsgründen problematisch ist, wenn wir viele Stützstellen gegeben haben und diese nicht — wie die Tschebyscheff-Stützstellen — optimal gewählt sind. Dies kann insbesondere bei der Dateninterpolation (4.1) auftreten, wenn die Stützstellen fest vorgegeben sind und nicht frei gewählt werden können. Wir behandeln daher in diesem Abschnitt eine alternative Interpolationstechnik, die auch bei einer großen Anzahl von Stützstellen problemlos funktioniert. Wir betrachten dazu paarweise verschiedene Stützstellen und nehmen an, dass diese aufsteigend angeordnet sind, also  $x_0 < x_1 < \dots < x_n$  gilt.

Die Grundidee der *Splineinterpolation* liegt darin, die interpolierende Funktion (die wir hier mit “ $S$ ” für “Spline” bezeichnen) nicht global, sondern nur auf jedem Teilintervall

$[x_i, x_{i+1}]$  als Polynom zu wählen. Diese Teilpolynome sollten dabei an den Intervallgrenzen nicht beliebig sondern möglichst glatt zusammenlaufen. Eine solche Funktion, die aus glatt zusammengeführten stückweisen Polynomen besteht, nennt man *Spline*.

Formal wird dies durch die folgende Definition präzisiert.

**Definition 4.17** Seien  $x_0 < x_1 < \dots < x_n$  Stützstellen und  $k \in \mathbb{N}$ . Eine stetige und  $(k-1)$ -mal stetig differenzierbare Funktion  $S : [x_0, x_n] \rightarrow \mathbb{R}$  heißt *Spline vom Grad  $k$* , falls  $S$  auf jedem Intervall  $I_i = [x_{i-1}, x_i]$  mit  $i = 1, \dots, n$  durch ein Polynom  $P_i$  vom Grad  $\leq k$  gegeben ist, d.h. für  $x \in I_i$  gilt

$$S(x) = P_i(x) = a_{i0} + a_{i1}(x - x_{i-1}) + \dots + a_{ik}(x - x_{i-1})^k = \sum_{j=0}^k a_{ij}(x - x_{i-1})^j.$$

Den Raum der Splines vom Grad  $k$  zur Stützstellenmenge  $\Delta = \{x_0, x_1, \dots, x_n\}$  bezeichnen wir mit  $S_{\Delta, k}$ . □

Ein solcher Spline aus Definition 4.17 löst dann das Interpolationsproblem, falls zusätzlich die Bedingung (4.1) erfüllt ist, also  $S(x_i) = f_i$  für alle  $i = 0, \dots, n$  gilt.

Bevor wir an die Berechnung von Splines gehen, zeigen wir eine Eigenschaft des Funktionenraums  $S_{\Delta, k}$ .

**Satz 4.18** Sei  $\Delta = \{x_0, x_1, \dots, x_n\}$  mit  $x_0 < x_1 < \dots < x_n$  und  $k \in \mathbb{N}$  gegeben. Dann ist der Raum der Splines  $S_{\Delta, k}$  ein  $k+n$ -dimensionaler Vektorraum über  $\mathbb{R}$ .

**Beweis:** Sicherlich ist mit  $S_1$  und  $S_2$  auch  $aS_1 + bS_2$  für  $a, b \in \mathbb{R}$  wieder ein Spline, also ist  $S_{\Delta, k}$  ein Vektorraum. Da die Splines linear von den Koeffizienten  $a_{ij}$  abhängen, genügt es zur Berechnung der Dimension die Anzahl der freien Parameter  $a_{ij}$  zu bestimmen. Auf dem ersten Intervall  $I_1$  haben wir freie Wahl für  $P_1$ , also gibt es genau  $k+1$  freie Parameter. Auf jedem weiteren Intervall  $I_i$  für  $i \geq 2$  sind die Werte der  $j$ -ten Ableitung  $P_i^{(j)}(x_{i-1}) = j!a_{ij}$  für  $j = 0, \dots, k-1$  bereits festgelegt, da die zusammengesetzte Funktion  $S$  in  $x_{i-1}$  ja stetig und  $k-1$ -mal stetig differenzierbar ist, es muss also

$$a_{ij} = P_{i-1}^{(j)}(x_{i-1})/j! \quad \text{für } j = 0, \dots, k-1$$

gelten. Daher ist hier nur noch der Koeffizient  $a_{ik}$  frei wählbar, was auf den  $n-1$  verbleibenden Intervallen gerade  $n-1$  weitere freie Parameter ergibt, also insgesamt  $k+n$ . □

**Bemerkung 4.19** Tatsächlich kann man beweisen, dass die Spline-Koeffizienten  $a_{ij}$  linear voneinander abhängen, statt der im Beweis des Satzes verwendeten Koeffizienten  $a_{10}, \dots, a_{1k}, a_{2k}, \dots, a_{nk}$  kann man also auch andere  $n+k$  Koeffizienten vorgeben und die übrigen daraus berechnen. Daher gilt: Zu je  $n+k$  beliebig festgelegten Koeffizienten  $a_{ij}$  existiert genau ein Spline  $S$ , dessen Koeffizienten mit den vorgegebenen übereinstimmen. □

Von besonderer praktischer Bedeutung in vielen Anwendungen sind die *kubischen Splines*, also der Splines vom Grad  $k = 3$ ; den Grund dafür werden wir etwas später besprechen. Zunächst wollen wir die Existenz und Eindeutigkeit des interpolierenden kubischen Splines  $S \in S_{\Delta,3}$  für gegebene Daten  $(x_i, f_i)$  mit  $x_0 < x_1 < \dots < x_n$  betrachten. Offenbar erfüllt der Spline genau dann das Interpolationsproblem (4.1), wenn die Bedingungen

$$\begin{aligned} a_{i0} &= f_{i-1} \text{ für } i = 1, \dots, n \\ \text{und} \\ a_{n1} &= \frac{f_n - a_{n0} - a_{n2}(x_n - x_{n-1})^2 - a_{n3}(x_n - x_{n-1})^3}{x_n - x_{n-1}} \end{aligned} \tag{4.8}$$

erfüllt sind, womit genau  $n + 1$  Koeffizienten festgelegt sind. Da wir nach Bemerkung 4.19 genau  $\dim S_{\Delta,k} = n + 3$  Koeffizienten festlegen müssen, verbleiben also 2 weitere Koeffizienten, die durch geeignete Bedingungen festgelegt werden müssen, um einen eindeutigen interpolierenden Spline zu erhalten. Diese werden typischerweise in Form von Randbedingungen, also Bedingungen an  $S$  oder an Ableitungen von  $S$  in den Punkten  $x_0$  und  $x_n$  formuliert. Das folgende Lemma stellt drei mögliche Bedingungen vor.

**Lemma 4.20** Gegeben seien Daten  $(x_i, f_i)$  für  $i = 0, \dots, n$  mit  $x_0 < x_1 < \dots < x_n$  und  $\Delta = \{x_0, x_1, \dots, x_n\}$ . Dann gibt es für jede der *Randbedingungen*

- (a)  $S''(x_0) = S''(x_n) = 0$  (“natürliche Randbedingungen”)
- (b)  $S'(x_0) = S'(x_n)$  und  $S''(x_0) = S''(x_n)$  (“periodische Randbedingungen”)
- (c)  $S'(x_0) = f'(x_0)$  und  $S'(x_n) = f'(x_n)$  (“hermite’sche Randbedingungen”, nur sinnvoll bei der Funktionsinterpolation)

genau einen kubischen Spline  $S \in S_{\Delta,3}$ , der das Interpolationsproblem (4.1) löst und die entsprechende Randbedingung (a), (b) oder (c) erfüllt.

**Beweis:** Jede der angegebenen Bedingungen lässt sich als Bedingung an zwei weitere Koeffizienten des Splines formulieren. Deswegen sind zusammen mit (4.8) genau  $n + 3$  Koeffizienten festgelegt, für die es nach Bemerkung 4.19 genau einen Spline gibt, der diese Bedingungen erfüllt.  $\square$

Kubische Splines werden in Anwendungen wie z.B. der Computergrafik bevorzugt verwendet, und wir wollen als nächstes den Grund dafür erläutern. Ein Kriterium zur Wahl der Ordnung eines Splines — speziell bei grafischen Anwendungen, aber auch bei “klassischen” Interpolationsproblemen — ist, dass die Krümmung der interpolierenden Kurve möglichst klein sein soll. Die Krümmung einer Kurve  $y(x)$  in einem Punkt  $x$  ist gerade gegeben durch die zweite Ableitung  $y''(x)$ . Die Gesamtkrümmung für alle  $x \in [x_0, x_n]$  kann nun auf verschiedene Arten gemessen werden, hier verwenden wir die  $L_2$ -Norm  $\|\cdot\|_2$  für quadratisch integrierbare Funktionen, die für  $g : [x_0, x_n] \rightarrow \mathbb{R}$  durch

$$\|g\|_2 := \left( \int_{x_0}^{x_n} g^2(x) dx \right)^{\frac{1}{2}}$$

gegeben ist. Die Krümmung einer zweimal stetig differenzierbaren Funktion  $y : [x_0, x_n] \rightarrow \mathbb{R}$  über dem gesamten Intervall kann also mittels  $\|y''\|_2$  gemessen werden. Hierfür gilt der folgende Satz.

**Satz 4.21** Sei  $S : [x_0, x_n] \rightarrow \mathbb{R}$  ein die Daten  $(x_i, f_i)$ ,  $i = 0, \dots, n$  interpolierender kubischer Spline, der eine der Randbedingungen (a)–(c) aus Lemma 4.20 erfüllt. Sei  $y : [x_0, x_n] \rightarrow \mathbb{R}$  eine zweimal stetig differenzierbare Funktion, die ebenfalls das Interpolationsproblem löst und die gleichen Randbedingungen wie  $S$  erfüllt. Dann gilt

$$\|S''\|_2 \leq \|y''\|_2.$$

**Beweis:** Setzen wir die offensichtliche Gleichung  $y'' = S'' + (y'' - S'')$  in die quadrierte Norm  $\|y''\|_2^2$  ein, so folgt

$$\begin{aligned} \|y''\|_2^2 &= \int_{x_0}^{x_n} (y''(x))^2 dx \\ &= \underbrace{\int_{x_0}^{x_n} (S''(x))^2 dx}_{=\|S''\|_2^2} + 2 \underbrace{\int_{x_0}^{x_n} S''(x)(y''(x) - S''(x)) dx}_{=:J} + \underbrace{\int_{x_0}^{x_n} (y''(x) - S''(x))^2 dx}_{\geq 0} \\ &\geq \|S''\|_2^2 + J. \end{aligned}$$

Wir betrachten nun den Term  $J$  genauer. Aus jeder der drei Randbedingungen folgt die Gleichung

$$\left[ S''(x)(y'(x) - S'(x)) \right]_{x=x_0}^{x_n} = S''(x_n)(y'(x_n) - S'(x_n)) - S''(x_0)(y'(x_0) - S'(x_0)) = 0. \quad (4.9)$$

Mit partieller Integration gilt

$$\int_{x_0}^{x_n} S''(x)(y''(x) - S''(x)) dx = \left[ S''(x)(y'(x) - S'(x)) \right]_{x=x_0}^{x_n} - \int_{x_0}^{x_n} S'''(x)(y'(x) - S'(x)) dx$$

Hierbei ist der erste Summand wegen (4.9) gleich Null. Auf jedem Intervall  $I_i = [x_{i-1}, x_i]$  ist  $S(x) = P_i(x)$  ein kubisches Polynom, weswegen  $S'''(x) \equiv d_i$  konstant für  $x \in I_i$  ist. Also folgt für den zweiten Summanden

$$\begin{aligned} \int_{x_0}^{x_n} S'''(x)(y'(x) - S'(x)) dx &= \sum_{i=1}^n \int_{x_{i-1}}^{x_i} d_i (y'(x) - S'(x)) dx \\ &= \sum_{i=1}^n d_i \int_{x_{i-1}}^{x_i} (y'(x) - S'(x)) dx \\ &= \sum_{i=1}^n d_i \underbrace{[(y(x_i) - y(x_{i-1})) - (S(x_i) - S(x_{i-1}))]}_{=0, \text{ da } y(x_i)=S(x_i) \text{ und } y(x_{i-1})=S(x_{i-1})} = 0 \end{aligned}$$

Damit erhalten wir  $J = 0$  und folglich die Behauptung.  $\square$

Diese Eigenschaft erklärt auch den Namen Spline: Ein ‘‘Spline’’ ist im Englischen eine dünne Holzlatte. Wenn man diese so verbiegt, dass sie vorgegebenen Punkten folgt (diese

also “interpoliert”), so ist auch bei dieser Latte die Krümmung, die hier näherungsweise die notwendige “Biegeenergie” beschreibt, minimal — zumindest für kleine Auslenkungen der Latte.

Wir kommen nun zur praktischen Berechnung der Spline-Koeffizienten für kubische Splines. Hierbei gibt es verschiedene Vorgehensweisen: man kann z.B. direkt ein lineares Gleichungssystem für die  $4n$  Koeffizienten  $a_{ij}$  für  $i = 1, \dots, n$  aufstellen, was aber wenig effizient ist. Alternativ kann man geschickt gewählte Basis-Funktionen für den Vektorraum  $S_{\Delta,3}$  verwenden (sogenannte B-Splines), und  $S$  in dieser Basis berechnen; dieser Ansatz wird im Buch von Deuffhard/Hohmann beschrieben. Dies führt auf ein  $n$ -dimensionales lineares Gleichungssystem mit tridiagonaler Matrix  $A$ . Es werden bei diesem Verfahren allerdings nicht die Koeffizienten  $a_{ij}$  berechnet, sondern die Koeffizienten bezüglich der B-Spline-Basis, die Auswertung von  $S$  muss demnach ebenfalls über diese Basisfunktionen erfolgen.

Hier stellen wir eine weitere Variante vor, mit der direkt die Koeffizienten  $a_{ij}$  berechnet werden, so dass  $S$  über die Darstellung in Definition 4.17 ausgewertet werden kann, was z.B. der Vorgehensweise in MATLAB entspricht. Wir kommen hierbei ebenfalls auf ein  $n$ -dimensionales lineares Gleichungssystem mit tridiagonaler Matrix  $A$ , so dass der Aufwand der Berechnung von der Ordnung  $O(n)$  ist. Wir betrachten zuerst die natürlichen Randbedingungen.

Hierzu definieren wir zunächst die Werte

$$f_i'' := S''(x_i) \quad \text{und} \quad h_i := x_i - x_{i-1}$$

für  $i = 0, \dots, n$  bzw.  $i = 1, \dots, n$ . Aus der Interpolationsbedingung und der geforderten Stetigkeit der zweiten Ableitung erhält man 4 Gleichungen für die Teilpolynome  $P_i$ :

$$P_i(x_{i-1}) = f_{i-1}, \quad P_i(x_i) = f_i, \quad P_i''(x_{i-1}) = f_{i-1}'', \quad P_i''(x_i) = f_i''. \quad (4.10)$$

Löst man diese — unter Ausnutzung der Ableitungsregeln für Polynome — nach den  $a_{ij}$  auf, so erhält man

$$\begin{aligned} a_{i0} &= f_{i-1} \\ a_{i1} &= \frac{f_i - f_{i-1}}{h_i} - \frac{h_i}{6}(f_i'' + 2f_{i-1}'') \\ a_{i2} &= \frac{f_{i-1}''}{2} \\ a_{i3} &= \frac{f_i'' - f_{i-1}''}{6h_i}. \end{aligned}$$

Da die Werte  $h_i$  und  $f_i$  ja direkt aus den Daten verfügbar sind, müssen lediglich die Werte  $f_i''$  berechnet werden. Da aus den natürlichen Randbedingungen sofort  $f_0'' = 0$  und  $f_n'' = 0$  folgt, brauchen nur die Werte  $f_1'', \dots, f_{n-1}''$  berechnet werden.

Beachte, dass wir in (4.10) bereits die Bedingungen an  $P_i$  und  $P_i''$  in den Stützstellen verwendet haben. Aus den noch nicht benutzten Gleichungen für die ersten Ableitungen erhält man nun die Gleichungen für die  $f_i''$ : Aus  $P_i'(x_i) = P_{i+1}'(x_i)$  erhält man

$$a_{i1} + 2a_{i2}(x_i - x_{i-1}) + 3a_{i3}(x_i - x_{i-1})^2 = a_{i+11}$$



für  $i = 1, \dots, n-1$ . Indem man hier die Werte  $f_i''$  und  $h_i$  gemäß den obigen Gleichungen bzw. Definitionen einsetzt, erhält man

$$h_i f_{i-1}'' + 2(h_i + h_{i+1})f_i'' + h_{i+1}f_{i+1}'' = 6 \frac{f_{i+1} - f_i}{h_{i+1}} - 6 \frac{f_i - f_{i-1}}{h_i} =: \delta_i$$

für  $i = 1, \dots, n-1$ . Dies liefert genau  $n-1$  Gleichungen für die  $n-1$  Unbekannten  $f_1'', \dots, f_{n-1}''$ . In Matrixform geschrieben erhalten wir so das Gleichungssystem

$$\begin{pmatrix} 2(h_1 + h_2) & h_2 & 0 & \cdots & \cdots & 0 \\ h_2 & 2(h_2 + h_3) & h_3 & \ddots & & \vdots \\ 0 & h_3 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & \ddots & \ddots & h_{n-1} \\ 0 & \cdots & \cdots & 0 & h_{n-1} & 2(h_{n-1} + h_n) \end{pmatrix} \begin{pmatrix} f_1'' \\ f_2'' \\ \vdots \\ \vdots \\ f_{n-1}'' \end{pmatrix} = \begin{pmatrix} \delta_1 \\ \delta_2 \\ \vdots \\ \vdots \\ \delta_{n-1} \end{pmatrix}$$

Zur Berechnung des Interpolationssplines löst man also zunächst dieses Gleichungssystem und berechnet dann gemäß der obigen Formel die Koeffizienten  $a_{ij}$  aus den  $f_k''$ .

Für äquidistante Stützstellen, also  $x_k - x_{k-1} = h_k = h$  für alle  $k = 1, \dots, n$ , kann man beide Seiten durch  $h$  teilen, und erhält so das Gleichungssystem

$$\begin{pmatrix} 4 & 1 & 0 & \cdots & \cdots & 0 \\ 1 & 4 & 1 & \ddots & & \vdots \\ 0 & 1 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 1 & 0 \\ \vdots & & \ddots & 1 & 4 & 1 \\ 0 & \cdots & \cdots & 0 & 1 & 4 \end{pmatrix} \begin{pmatrix} f_1'' \\ f_2'' \\ \vdots \\ \vdots \\ f_{n-1}'' \end{pmatrix} = \begin{pmatrix} \tilde{\delta}_1 \\ \tilde{\delta}_2 \\ \vdots \\ \vdots \\ \tilde{\delta}_{n-1} \end{pmatrix}$$

mit  $\tilde{\delta}_k = \delta_k/h$ , welches ein Beispiel für ein lineares Gleichungssystem mit (offensichtlich) diagonaldominanter Matrix ist.

Für andere Randbedingungen verändert sich dieses Gleichungssystem entsprechend, vgl. Übungsaufgabe 38.

Zum Abschluss wollen wir noch kurz auf den Interpolationsfehler bei der Splineinterpolation eingehen. Die Analyse dieses Fehlers ist recht langwierig, das Ergebnis, das wir hier ohne Beweis geben, allerdings sehr leicht darzustellen. Der folgende Satz wurde von C.A. Hall und W.W. Meyer [3] bewiesen.

**Satz 4.22** Sei  $S \in S_{\Delta,3}$  der interpolierende Spline einer 4 mal stetig differenzierbaren Funktion  $f$  mit hermite'schen Randbedingungen und Stützstellen  $\Delta = \{x_0, \dots, x_n\}$ . Dann gilt für  $h = \max_k(x_k - x_{k-1})$  die Abschätzung

$$\|f - S\|_\infty \leq \frac{5}{384} h^4 \|f^{(4)}\|_\infty$$



# Kapitel 5

## Integration

Die Integration von Funktionen ist eine elementare mathematische Operation, die in vielen Formeln benötigt wird. Im Gegensatz zur Ableitung, die für praktisch alle mathematischen Funktionen explizit analytisch berechnet werden kann, gibt es viele Funktionen, deren Integrale man nicht explizit angeben kann. Verfahren zur numerischen Integration (man spricht auch von *Quadratur*) spielen daher eine wichtige Rolle, sowohl als eigenständige Algorithmen als auch als Basis für andere Anwendungen wie z.B. der numerischen Lösung von Differentialgleichungen.

Das Grundproblem lässt sich hierbei ganz einfach beschreiben: Für eine Funktion  $f : \mathbb{R} \rightarrow \mathbb{R}$  soll das Integral

$$\int_a^b f(x)dx \tag{5.1}$$

auf einem Intervall  $[a, b]$  berechnet werden. Einige Verfahren werden auch auf unendliche Integrationsintervalle anwendbar sein.

Wir werden hier verschiedene Verfahren zur Integration kennen lernen: Die klassischen *Newton–Cotes–Formeln* und *zusammengesetzten Newton–Cotes–Formeln* (auch als *iterierte* oder *aufsummierte* Newton–Cotes–Formeln bekannt), welche auf der Polynominterpolation basieren, die *Gauss–Quadratur*, die auf den schon bekannten orthogonalen Polynomen beruht und die *Romberg–Quadratur*, bei welcher eine detaillierte und geschickte Analyse des numerischen Fehlers ausgenutzt wird.

### 5.1 Newton–Cotes–Formeln

Die Grundidee jeder numerischen Integrationsformel liegt darin, das Integral (5.1) durch eine Summe

$$\int_a^b f(x)dx \approx (b-a) \sum_{i=0}^n \alpha_i f(x_i) \tag{5.2}$$

zu approximieren. Hierbei heißen die  $x_i$  die *Stützstellen* und die  $\alpha_i$  die *Gewichte* der Integrationsformel.

Die Stützstellen  $x_i$  können hierbei beliebig vorgegeben werden, folglich benötigen wir eine Formel, mit der wir zu den  $x_i$  sinnvolle Gewichte  $\alpha_i$  berechnen können.

Die Idee der Newton–Cotes–Formeln liegt nun darin, die Funktion  $f$  zunächst durch ein Interpolationspolynom  $P \in \mathcal{P}_n$  zu den Stützstellen  $x_0, \dots, x_n$  zu approximieren und dann das *exakte* Integral über dieses Polynom zu berechnen. Wir führen diese Konstruktion nun durch:

Da wir einen expliziten Ausdruck in den  $f(x_i)$  erhalten wollen, bietet sich die Darstellung von  $P$  mittels der Lagrange–Polynome an, also

$$P(x) = \sum_{i=0}^n f(x_i)L_i(x) \quad \text{mit} \quad L_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^n \frac{x - x_j}{x_i - x_j},$$

vgl. Abschnitt 4.1.1. Das Integral über  $P$  ergibt sich dann zu

$$\begin{aligned} \int_a^b P(x)dx &= \int_a^b \sum_{i=0}^n f(x_i)L_i(x) dx \\ &= \sum_{i=0}^n f(x_i) \int_a^b L_i(x) dx. \end{aligned}$$

Um die Gewichte  $\alpha_i$  in (5.2) zu berechnen, setzen wir

$$(b-a) \sum_{i=0}^n \alpha_i f(x_i) = \sum_{i=0}^n f(x_i) \int_a^b L_i(x) dx.$$

Auflösen nach  $\alpha_i$  liefert dann

$$\alpha_i = \frac{1}{b-a} \int_a^b L_i(x) dx. \quad (5.3)$$

Diese  $\alpha_i$  können dann explizit berechnet werden, denn die Integrale über die Lagrange–Polynome  $L_i$  sind explizit lösbar. Hierbei hängen die Gewichte  $\alpha_i$  von der Wahl der Stützstellen  $x_i$  ab, nicht aber von den Funktionswerten  $f(x_i)$ . Für äquidistante Stützstellen

$$x_i = a + \frac{i(b-a)}{n}$$

sind die Gewichte aus (5.3) in Tabelle 5.1 für  $n = 0, \dots, 7$  angegeben.

Beachte, dass sich die Gewichte immer zu 1 aufsummieren und symmetrisch in  $i$  sind, d.h. es gilt  $\alpha_i = \alpha_{n-i}$ . Außerdem sind die Gewichte unabhängig von den Intervallgrenzen  $a$  und  $b$ . Einige dieser Formeln haben eigene Namen. So wird z.B. die Newton–Cotes–Formel für  $n = 0$  als *Rechteck–Regel*, für  $n = 1$  als *Trapez–Regel*, für  $n = 2$  als *Simpson–Regel* oder *Keplersche Fass–Regel*, und die Formel für  $n = 3$  als *Newtonsche 3/8–Regel* bezeichnet.

Aus der Abschätzung des Interpolationsfehlers kann man nun eine Abschätzung für den Integrationsfehler

$$F_n[f] := \int_a^b f(x)dx - (b-a) \sum_{i=0}^n \alpha_i f(x_i) = \int_a^b f(x)dx - \int_a^b P(x)dx$$

ableiten. Hierbei müssen die Stützstellen  $x_i$  nicht unbedingt äquidistant liegen.

$n$	$\alpha_0$	$\alpha_1$	$\alpha_2$	$\alpha_3$	$\alpha_4$	$\alpha_5$	$\alpha_6$	$\alpha_7$
0	1							
1	$\frac{1}{2}$	$\frac{1}{2}$						
2	$\frac{1}{6}$	$\frac{4}{6}$	$\frac{1}{6}$					
3	$\frac{1}{8}$	$\frac{3}{8}$	$\frac{3}{8}$	$\frac{1}{8}$				
4	$\frac{7}{90}$	$\frac{32}{90}$	$\frac{12}{90}$	$\frac{32}{90}$	$\frac{7}{90}$			
5	$\frac{19}{288}$	$\frac{75}{288}$	$\frac{50}{288}$	$\frac{50}{288}$	$\frac{75}{288}$	$\frac{19}{288}$		
6	$\frac{41}{840}$	$\frac{216}{840}$	$\frac{27}{840}$	$\frac{272}{840}$	$\frac{27}{840}$	$\frac{216}{840}$	$\frac{41}{840}$	
7	$\frac{751}{17280}$	$\frac{3577}{17280}$	$\frac{1323}{17280}$	$\frac{2989}{17280}$	$\frac{2989}{17280}$	$\frac{1323}{17280}$	$\frac{3577}{17280}$	$\frac{751}{17280}$

Tabelle 5.1: Gewichte der Newton-Cotes Formeln aus (5.3) für äquidistante Stützstellen  $x_i$ 

**Satz 5.1** Seien  $n \geq 1$  und  $\alpha_i$  die Gewichte, die gemäß (5.3) zu den Stützstellen  $a \leq x_0 < \dots < x_n \leq b$  berechnet wurden, sei  $h := (b-a)/n$  und  $z_i := n(x_i - a)/(b-a) \in [0, n]$  für  $i = 0, \dots, n$ . Dann gilt:

(i) Es gibt nur von  $z_0, \dots, z_n$  und  $n$  abhängende Konstanten  $c_n$ , so dass für alle  $f \in C^{n+1}([a, b])$  die Abschätzung

$$|F_n[f]| \leq c_n h^{n+2} \|f^{(n+1)}\|_\infty$$

gilt.

(ii) Für gerades  $n$ ,  $f \in C^{n+2}([a, b])$  und symmetrisch verteilte Stützstellen  $x_i$ , also  $x_i - a = b - x_{n-i}$  für  $i = 0, \dots, n$  (z.B. äquidistante Stützstellen) gibt es nur von  $z_0, \dots, z_n$  und  $n$  abhängende Konstanten  $d_n$ , so dass die Abschätzung

$$|F_n[f]| \leq d_n h^{n+3} \|f^{(n+2)}\|_\infty$$

gilt.

**Beweis:** (i) Aus Satz 4.8(ii) wissen wir

$$|f(x) - P(x)| \leq \|f^{(n+1)}\|_\infty \left| \frac{(x-x_0)(x-x_1)\cdots(x-x_n)}{(n+1)!} \right| = \frac{1}{(n+1)!} \|f^{(n+1)}\|_\infty \prod_{i=0}^n |x-x_i|.$$

Also folgt

$$|F_n[f]| \leq \frac{1}{(n+1)!} \|f^{(n+1)}\|_\infty \int_a^b \prod_{i=0}^n |x-x_i| dx,$$

Daraus folgt die behauptete Abschätzung mit

$$\begin{aligned} c_n &= \frac{1}{(n+1)!} \frac{1}{h^{n+2}} \int_a^b \prod_{i=0}^n |x-x_i| dx = \frac{1}{(n+1)!} \left(\frac{n}{b-a}\right)^{n+2} \int_a^b \prod_{i=0}^n |x-x_i| dx \\ &\left(\text{Substitution: } z = n\frac{x-a}{b-a}, \quad z_i = n\frac{x_i-a}{b-a}\right) = \frac{1}{(n+1)!} \int_0^n \prod_{i=0}^n |z-z_i| dz \end{aligned}$$

Beachte, dass die  $z_i$  im Intervall  $[0, n]$  liegen, der entstehende Ausdruck also unabhängig von  $a$  und  $b$  ist.

(ii) Aus der Konstruktion der Newton–Cotes–Formeln folgt sofort, dass Polynome  $Q \in \mathcal{P}_n$  exakt integriert werden, da das interpolierende Polynom  $P \in \mathcal{P}_n$ , über das integriert wird, in diesem Fall mit  $Q$  übereinstimmt. Der Beweis von (ii) folgt aus der — auf den ersten Blick etwas überraschenden — Eigenschaft, dass die Newton–Cotes–Formeln für gerades  $n$  und symmetrisch verteilte Stützstellen  $x_i$  auch für Polynome  $Q \in \mathcal{P}_{n+1}$  exakt sind. Zum Beweis dieser Eigenschaft sei  $Q \in \mathcal{P}_{n+1}$  und  $P \in \mathcal{P}_n$  das interpolierende Polynom an den Stützstellen  $x_i$ . Dann existiert nach Satz 4.8(i) für jedes  $x \in [a, b]$  eine Stelle  $\xi \in [a, b]$ , so dass

$$Q(x) = P(x) + \underbrace{\frac{Q^{(n+1)}(\xi)}{(n+1)!}}_{=: \gamma} (x - x_0)(x - x_1) \cdots (x - x_n)$$

gilt. Weil nun aber  $Q^{(n+1)}$  ein Polynom vom Grad 0 und damit konstant ist, ist  $\gamma$  unabhängig von  $\xi$  und damit auch von  $x$ . Folglich gilt

$$F_n[Q] = \int_a^b Q(x) dx - \int_a^b P(x) dx = \gamma \int_a^b \prod_{i=0}^n (x - x_i) dx.$$

Aus der Symmetrie der  $x_i$  folgt  $x - x_i = x - (-x_{n-i} + a + b)$  für  $x \in [a, b]$  und damit

$$\begin{aligned} \int_a^{(a+b)/2} \prod_{i=0}^n (x - x_i) dx &= \int_a^{(a+b)/2} \prod_{i=0}^n (x - (-x_i + a + b)) dx \\ \left( \text{Substitution: } x = -(x - a - b) \right) &= - \int_b^{(a+b)/2} \prod_{i=0}^n (-x + x_i) dx \\ &= - \int_{(a+b)/2}^b \prod_{i=0}^n (x - x_i) dx, \end{aligned}$$

also

$$\int_a^b \prod_{i=0}^n (x - x_i) dx = 0$$

und damit  $F_n[Q] = 0$ , weswegen  $Q$  exakt integriert wird. Zum Beweis der Abschätzung (ii) sei nun  $f$  aus der Behauptung gegeben. Sei  $Q \in \mathcal{P}_{n+1}$  ein Interpolationspolynom zu den Stützstellen  $x_0, x_1, \dots, x_n, x_{n+1}$ , wobei  $x_{n+1}$  eine von  $x_0, \dots, x_n$  verschiedene und ansonsten beliebige weitere Stützstelle im Intervall  $[x_0, x_n]$  ist. Dann stimmen die Interpolationspolynome  $P \in \mathcal{P}_n$  für  $f$  und  $Q$  zu den Stützstellen  $x_0, \dots, x_n$  überein und es gilt nach Satz 4.8(ii) (also analog zu Teil (i) dieses Beweises mit  $n + 1$  an Stelle von  $n$ ) und der oben bewiesenen Tatsache  $F_N[Q] = 0$

$$\begin{aligned} |F_n[f]| &= \left| \int_a^b f(x) dx - \int_a^b P(x) dx \right| \\ &= \left| \int_a^b f(x) dx - \int_a^b Q(x) dx + \underbrace{\int_a^b Q(x) dx - \int_a^b P(x) dx}_{=F_n[Q]=0} \right| \end{aligned}$$

$$\begin{aligned}
&= \left| \int_a^b f(x)dx - \int_a^b Q(x)dx \right| \\
&\leq \frac{1}{(n+2)!} \|f^{(n+2)}\|_\infty \int_a^b |x - x_{n+1}| \prod_{i=0}^n |x - x_i| dx.
\end{aligned}$$

Da die rechte Seite dieser Ungleichung stetig in  $x_{n+1}$  und die linke Seite unabhängig von  $x_{n+1}$  ist, gilt die Abschätzung auch für  $x_{n+1} = x_{n/2}$ . Daher kann dieser Integralausdruck nun analog zu Teil (i) durch

$$\frac{1}{(n+2)!} \|f^{(n+2)}\|_\infty \int_a^b |x - x_{n/2}| \prod_{i=0}^n |x - x_i| dx \leq d_n h^{n+3} \|f^{(n+2)}\|_\infty$$

mit

$$d_n = \frac{1}{(n+2)!} \int_0^n |z - z_{n/2}| \prod_{i=0}^n |z - z_i| dz$$

abgeschätzt werden.  $\square$

Beachte, dass Formulierung und Beweis des Satzes wegen der Division durch  $n$  nur für  $n \geq 1$  sinnvoll sind, man kann aber ähnliche Abschätzungen auch für  $n = 0$  erhalten.

Die Konstanten  $c_n$  und  $d_n$  hängen von  $n$  und der Lage der "skalierten" Stützstellen  $z_i$  ab und können für gegebene Werte explizit berechnet werden. Wir zeigen diese Rechnung für  $n = 1$  und die Stützstellen  $x_0 = a$ ,  $x_1 = b$ . In diesem Fall erhalten wir  $z_0 = 0$  und  $z_1 = 1$ . Damit ergibt sich

$$\begin{aligned}
c_1 &= \frac{1}{(1+1)!} \int_0^1 \prod_{i=0}^1 |z - z_i| dz = \frac{1}{2} \int_0^1 (z-0)(1-z) dz = \frac{1}{2} \int_0^1 z - z^2 dz \\
&= \frac{1}{2} \left[ \frac{1}{2} z^2 - \frac{1}{3} z^3 \right]_0^1 = \frac{1}{2} \left( \frac{1}{2} - \frac{1}{3} \right) = \frac{1}{12}.
\end{aligned}$$

In Tabelle 5.2 sind die mit dieser Technik berechneten Fehlerabschätzungen für  $n = 1, \dots, 7$  und äquidistante Stützstellen approximativ angegeben, wobei  $M_n := \|f^{(n)}\|_\infty$  ist.

$n$	1	2	3	4	5	6	7
	$\frac{(b-a)^3 M_2}{12}$	$\frac{(b-a)^5 M_4}{2880}$	$\frac{(b-a)^5 M_4}{6480}$	$\frac{5.2(b-a)^7 M_6}{10^7}$	$\frac{2.9(b-a)^7 M_6}{10^7}$	$\frac{6.4(b-a)^9 M_8}{10^{10}}$	$\frac{3.9(b-a)^9 M_8}{10^{10}}$

Tabelle 5.2: Fehlerabschätzungen der Newton-Cotes Formeln für äquidistante Stützstellen

Beachte, dass die Formeln mit ungeradem  $n = 2m + 1$  jeweils nur eine leichte Verbesserung gegenüber den Formeln mit geradem  $n = 2m$  liefern, dafür aber eine Funktionsauswertung mehr ausgeführt werden muss. Formeln mit geradzahligem  $n$  sind also vorzuziehen.

Wie zu erwarten erhöht sich die Genauigkeit mit wachsendem  $n$ , allerdings nur dann, wenn nicht zugleich mit wachsendem  $n$  die Norm der Ableitungen  $\|f^{(n)}\|_\infty$  zunimmt. Es tauchen also die gleichen prinzipiellen Probleme wie bei den Interpolationspolynomen auf, was nicht weiter verwunderlich ist, da diese ja dem Verfahren zu Grunde liegen. Hier

kommt aber noch ein weiteres Problem hinzu, nämlich kann man für  $n = 8$  und  $n \geq 10$  beobachten, dass einige der Gewichte  $\alpha_i$  negativ werden. Dies kann zu numerischen Problemen (z.B. Auslöschungen) führen, die man vermeiden möchte. Aus diesem Grunde ist es nicht ratsam, den Grad des zugrundeliegenden Polynoms immer weiter zu erhöhen. Trotzdem sind die Newton–Cotes–Formeln wichtig, da sie die Basis für eine ganze Reihe effizienterer Integrationsformeln liefern, die wir in den folgenden Abschnitten besprechen werden.

## 5.2 Zusammengesetzte Newton–Cotes–Formeln

Ein möglicher Ausweg aus den Problemen mit immer höheren Polynomgraden funktioniert bei der Integration mittels Newton–Cotes–Formeln ganz ähnlich wie bei der Interpolation — nur einfacher. Bei der Interpolation sind wir von Polynomen zu Splines, also stückweisen Polynomen übergegangen. Um dort weiterhin eine “schöne” Approximation zu erhalten, mussten wir Bedingungen an den Nahtstellen festlegen, die eine gewisse Glattheit der approximierenden Funktion erzwingen, weswegen wir die Koeffizienten recht aufwändig über ein lineares Gleichungssystem herleiten mussten.

Bei der Integration fällt diese Prozedur weg. Wie bei den Splines verwenden wir zur Herleitung der zusammengesetzten Newton–Cotes–Formeln stückweise Polynome, verzichten aber auf aufwändige Bedingungen an den Nahtstellen, da wir ja nicht an einer schönen Approximation der Funktion, sondern “nur” an einer guten Approximation des Integrals interessiert sind. In der Praxis berechnet man die zugrundeliegenden stückweisen Polynome nicht wirklich, sondern wendet die Newton–Cotes–Formeln wie folgt auf den Teilintervallen an:

Sei  $N$  die Anzahl von Teilintervallen, auf denen jeweils die Newton–Cotes–Formel vom Grad  $n$ , also mit  $n + 1$  Stützstellen verwendet werden soll. Wir setzen

$$x_i = a + ih, \quad i = 0, 1, \dots, nN, \quad h = \frac{b - a}{nN}$$

und zerlegen das Integral (5.1) mittels

$$\int_a^b f(x) dx = \int_{x_0}^{x_n} f(x) dx + \int_{x_n}^{x_{2n}} f(x) dx + \dots + \int_{x_{(N-1)n}}^{x_{Nn}} f(x) dx = \sum_{k=1}^N \int_{x_{(k-1)n}}^{x_{kn}} f(x) dx.$$

Auf jedem Teilintervall  $[x_{(k-1)n}, x_{kn}]$  wenden wir nun die Newton–Cotes–Formel an, d.h. wir approximieren

$$\int_{x_{(k-1)n}}^{x_{kn}} f(x) dx \approx nh \sum_{i=0}^n \alpha_i f(x_{(k-1)n+i})$$

und addieren die Teilapproximationen auf, also

$$\int_a^b f(x) dx \approx nh \sum_{k=1}^N \sum_{i=0}^n \alpha_i f(x_{(k-1)n+i}).$$



Der entstehende Integrationsfehler

$$F_{N,n}[f] := \int_a^b f(x)dx - nh \sum_{k=1}^N \sum_{i=0}^n \alpha_i f(x_{(k-1)n+i})$$

ergibt sich einfach als Summe der Fehler  $F_n[f]$  auf den Teilintervallen, weswegen man aus Satz 5.1(i) die Abschätzung

$$\begin{aligned} |F_{N,n}[f]| &\leq \sum_{k=1}^N c_n h^{n+2} \max_{y \in [x_{(k-1)n}, x_{kn}]} |f^{(n+1)}(y)| \\ &\leq N c_n h^{n+2} \|f^{(n+1)}\|_\infty = \frac{c_n}{n} (b-a) h^{n+1} \|f^{(n+1)}\|_\infty \end{aligned}$$

und aus Satz 5.1(ii) für gerades  $n$  die Abschätzung

$$|F_{N,n}[f]| \leq \frac{d_n}{n} (b-a) h^{n+2} \|f^{(n+2)}\|_\infty$$

erhält. Im Folgenden geben wir die zusammengesetzten Newton–Cotes–Formeln für  $n = 1, 2, 4$  mitsamt ihren Fehlerabschätzungen an; in allen Formeln sind die Stützstellen  $x_i$  als  $x_i = a + ih$  gewählt.

$n = 1$ , **Trapez–Regel:**

$$\begin{aligned} \int_a^b f(x)dx &\approx \frac{h}{2} \left( f(a) + 2 \sum_{i=1}^{N-1} f(x_i) + f(b) \right) \\ |F_{N,1}[f]| &\leq \frac{b-a}{12} h^2 \|f^{(2)}\|_\infty, \quad h = \frac{b-a}{N} \end{aligned}$$

$n = 2$ , **Simpson–Regel:**

$$\begin{aligned} \int_a^b f(x)dx &\approx \frac{h}{3} \left( f(a) + 2 \sum_{i=1}^{N-1} f(x_{2i}) + 4 \sum_{i=0}^{N-1} f(x_{2i+1}) + f(b) \right) \\ |F_{N,2}[f]| &\leq \frac{b-a}{180} h^4 \|f^{(4)}\|_\infty, \quad h = \frac{b-a}{2N} \end{aligned}$$

$n = 4$ , **Milne–Regel:**

$$\begin{aligned} \int_a^b f(x)dx &\approx \frac{2h}{45} \left( 7(f(a) + f(b)) + 14 \sum_{i=1}^{N-1} f(x_{4i}) \right. \\ &\quad \left. + 32 \sum_{i=0}^{N-1} (f(x_{4i+1}) + f(x_{4i+3})) + 12 \sum_{i=0}^{N-1} f(x_{4i+2}) \right) \\ |F_{N,4}[f]| &\leq \frac{2(b-a)}{945} h^6 \|f^{(6)}\|_\infty, \quad h = \frac{b-a}{4N} \end{aligned}$$

An einem Beispiel wollen wir die praktischen Auswirkungen der Fehlerabschätzungen illustrieren.

**Beispiel 5.2** Das Integral

$$\int_0^1 e^{-x^2/2} dx$$

soll mit einer garantierten Genauigkeit von  $\varepsilon = 10^{-10}$  numerisch approximiert werden. Die Ableitungen der Funktion  $f(x) = e^{-x^2/2}$  lassen sich relativ leicht berechnen; es gilt

$$f^{(2)}(x) = (x^2 - 1)f(x), \quad f^{(4)}(x) = (3 - 6x^2 + x^4)f(x), \quad f^{(6)}(x) = (-15 + 45x^2 - 15x^4 + x^6)f(x).$$

Mit etwas Rechnung (oder aus der grafischen Darstellung) sieht man, dass all diese Funktionen ihr betragsmäßiges Maximum auf  $[0, 1]$  in  $y = 0$  annehmen, woraus die Gleichungen

$$\|f^{(2)}\|_\infty = 1, \quad \|f^{(4)}\|_\infty = 3 \quad \text{und} \quad \|f^{(6)}\|_\infty = 15$$

folgen.

Löst man die oben angegebenen Fehlerabschätzungen  $F_{N,n}[f] \leq \varepsilon$  für die Trapez-, Simpson- und Milne-Regel nach  $h$  auf so erhält man die folgenden Bedingungen an  $h$

$$\begin{aligned} h &\leq \sqrt{\frac{12\varepsilon}{(b-a)\|f^{(2)}\|_\infty}} \approx \frac{1}{28867.51} && \text{(Trapez-Regel)} \\ h &\leq \sqrt[4]{\frac{180\varepsilon}{(b-a)\|f^{(4)}\|_\infty}} \approx \frac{1}{113.62} && \text{(Simpson-Regel)} \\ h &\leq \sqrt[6]{\frac{945\varepsilon}{2(b-a)\|f^{(6)}\|_\infty}} \approx \frac{1}{26.12} && \text{(Milne-Regel)} \end{aligned}$$

Der Bruch auf der rechten Seite gibt dabei die maximal erlaubte Größe für  $h$  vor. Um diese zu realisieren, muss  $1/(nN) \leq h$  gelten für die Anzahl  $nN + 1$  der Stützstellen. Da  $nN$  ganzzahliges Vielfaches von  $n$  ist, braucht man also 28869 Stützstellen für die Trapez-Regel, 115 Stützstellen für die Simpson-Regel und 29 Stützstellen für die Milne-Regel.  $\square$

### 5.3 Gauß-Quadratur

Bisher haben wir numerische Integrationsformeln erhalten, indem wir zu beliebig vorgegebenen Stützstellen geeignete Gewichte gesucht haben. Analog zur Interpolation können wir aber nun versuchen, auch die Stützstellen durch ein geeignetes Verfahren zu bestimmen und damit den numerischen Fehler zu verringern. In einer Newton-Cotes-Formel von Grad  $n$  gibt es  $n + 1$  Stützstellen und  $n + 1$  Gewichte, also  $2n + 2$  freie Parameter. Das Ziel der Gauß-Quadratur ist es nun, diese Parameter in den Newton-Cotes-Formeln optimal zu wählen, wobei "optimal" in diesem Zusammenhang bedeutet, dass Polynome möglichst hohen Grades exakt integriert werden. Die Newton-Cotes Formeln sind so konstruiert, dass die Formel vom Grad  $n$  für beliebige Stützstellen Polynome von Grad  $n$  exakt integriert, für symmetrische Stützstellen und gerades  $n$  werden sogar Polynome vom Grad  $n + 1$  exakt integriert. Wenn man "naiv" mit der Dimension der Räume und der Anzahl der freien Parameter argumentiert, könnte man vermuten, dass bei geschickter Wahl der Stützstellen

und Gewichte Polynome vom Grad  $2n + 1$  exakt integriert werden können, da dann die Dimension  $2n + 2$  des Polynomraums  $\mathcal{P}_{2n+1}$  gerade mit der Anzahl der freien Parameter übereinstimmt.

Bevor wir in die Theorie des Gauß-Quadratur einsteigen, wollen wir dies am Beispiel  $n = 1$  illustrieren. Wir wollen versuchen, Stützstellen  $x_0$  und  $x_1$  sowie Gewichte  $\alpha_0$  und  $\alpha_1$  zu bestimmen, so dass die Gleichung

$$\int_a^b Q(x)dx = (b-a)(\alpha_0 Q(x_0) + \alpha_1 Q(x_1))$$

für jedes  $Q \in \mathcal{P}_3$  erfüllt ist. Da beide Seiten dieser Gleichung linear in den Koeffizienten von  $Q$  sind, genügt es, die Parameter so zu bestimmen, dass die Gleichung für die Elemente der Monombasis  $\mathcal{B} = \{1, x, x^2, x^3\}$  von  $\mathcal{P}_3$  erfüllt ist. Also muss gelten

$$\begin{aligned} b-a &= (b-a)(\alpha_0 + \alpha_1) \\ \frac{b^2 - a^2}{2} &= (b-a)(\alpha_0 x_0 + \alpha_1 x_1) \\ \frac{b^3 - a^3}{3} &= (b-a)(\alpha_0 x_0^2 + \alpha_1 x_1^2) \\ \frac{b^4 - a^4}{4} &= (b-a)(\alpha_0 x_0^3 + \alpha_1 x_1^3) \end{aligned}$$

Dies ist ein (nichtlineares) Gleichungssystem mit 4 Gleichungen und 4 Unbekannten, das die Lösung

$$\begin{aligned} \alpha_0 &= \frac{1}{2}, & x_0 &= -\frac{b-a}{2} \frac{1}{\sqrt{3}} + \frac{b+a}{2} \\ \alpha_1 &= \frac{1}{2}, & x_1 &= +\frac{b-a}{2} \frac{1}{\sqrt{3}} + \frac{b+a}{2} \end{aligned}$$

besitzt. Die zugehörige Integrationsformel wird *Gauß-Legendre-Regel* genannt.

Tatsächlich stellt man fest, dass die so entstehenden Gleichungssysteme immer lösbar sind. Ein direkter Beweis ist recht umständlich; viel eleganter — und dazu noch allgemeiner — lässt sich dies mit orthogonalen Polynomen beweisen, was wir im Folgenden machen werden.

Wir betrachten dazu das allgemeinere numerische Integrationsproblem

$$\int_a^b \omega(x)f(x)dx \approx \sum_{i=0}^n \lambda_i f(x_i)$$

wobei  $\omega : (a, b) \rightarrow \mathbb{R}$  eine nichtnegative *Gewichtsfunktion* ist. Beachte, dass der Vorfaktor  $(b-a)$  aus den Newton-Cotes-Formeln (5.2) hier in die Gewichte  $\lambda_i$  einbezogen ist. Das bisher behandelte Integrationsproblem ohne Gewichtsfunktion ist hier durch den Spezialfall  $\omega(x) \equiv 1$  gegeben, es sind aber auch andere  $\omega$  möglich, z.B.  $\omega(x) = 1/(1-x^2)$  auf dem Intervall  $[-1, 1]$ . Falls  $f$  durch ein Polynom beschränkt ist, macht es mit exponentiell fallenden Gewichtsfunktionen auch Sinn, Integrationsprobleme auf unendlichen Integrationsintervallen zu betrachten, z.B. mit  $\omega(x) = e^{-x}$  auf  $[0, \infty)$  oder mit  $\omega(x) = e^{-x^2}$  auf  $(-\infty, \infty)$ .

Der folgende Satz zeigt, wie die Stützstellen und Gewichte optimal gewählt werden müssen, also so, dass Polynome vom Grad  $2n + 1$  exakt integriert werden.

**Satz 5.3** Die Gauß–Quadraturformel

$$\int_a^b \omega(x)f(x)dx \approx \sum_{i=0}^n \lambda_i f(x_i)$$

ist exakt für  $f = P \in \mathcal{P}_{2n+1}$  falls die folgenden zwei Bedingungen erfüllt sind.

- (1) Die Stützstellen  $x_0, \dots, x_n$  sind die Nullstellen des orthogonalen Polynoms  $P_{n+1}$  gemäß Definition 4.12 bezüglich der Gewichtsfunktion  $\omega$ .
- (2) Die Gewichte  $\lambda_i$  sind (analog zu den Newton–Cotes–Formeln) gegeben durch

$$\lambda_i = \int_a^b \omega(x)L_i(x)dx$$

mit den Lagrange–Polynomen

$$L_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^n \frac{x - x_j}{x_i - x_j}.$$

**Beweis:** Wir zeigen zunächst, dass die Formel exakt ist für  $P \in \mathcal{P}_n$ . Für  $P \in \mathcal{P}_n$  gilt

$$P(x) = \sum_{i=0}^n P(x_i)L_i(x)$$

und damit

$$\begin{aligned} \int_a^b \omega(x)P(x)dx &= \int_a^b \omega(x) \sum_{i=0}^n P(x_i)L_i(x)dx \\ &= \sum_{i=0}^n P(x_i) \int_a^b \omega(x)L_i(x)dx = \sum_{i=0}^n P(x_i)\lambda_i. \end{aligned}$$

Es bleibt zu zeigen, dass die Formel auch für  $P \in \mathcal{P}_{2n+1}$  exakt ist. Sei dazu  $P \in \mathcal{P}_{2n+1} \setminus \mathcal{P}_n$ . Dann lässt sich  $P$  schreiben als

$$P(x) = Q(x)P_{n+1}(x) + R(x)$$

mit  $Q, R \in \mathcal{P}_n$ . Damit gilt

$$\begin{aligned} &\int_a^b \omega(x)P(x)dx - \sum_{i=0}^n \lambda_i P(x_i) \\ &= \underbrace{\int_a^b \omega(x)Q(x)P_{n+1}(x)dx}_{= \langle Q, P_{n+1} \rangle_\omega} + \int_a^b \omega(x)R(x)dx - \sum_{i=0}^n \lambda_i Q(x_i)P_{n+1}(x_i) - \sum_{i=0}^n \lambda_i R(x_i). \end{aligned}$$

Da  $Q \in \mathcal{P}_n$  liegt, lässt sich  $Q$  als Linearkombination der Orthogonalpolynome  $P_0, \dots, P_n$  schreiben. Aus  $\langle P_k, P_{n+1} \rangle_\omega = 0$  für  $k = 0, \dots, n$  folgt also  $\langle Q, P_{n+1} \rangle_\omega = 0$ . Da die  $x_i$  gerade die Nullstellen von  $P_{n+1}$  sind, folgt

$$\sum_{i=1}^n \lambda_i Q(x_i) P_{n+1}(x_i) = 0.$$

Also ist

$$\int_a^b \omega(x) P(x) dx - \sum_{i=1}^n \lambda_i f(x_i) = \int_a^b \omega(x) R(x) dx - \sum_{i=1}^n \lambda_i R(x_i) = 0,$$

da  $R \in \mathcal{P}_n$  und die Formel für diese Polynome exakt ist.  $\square$

**Bemerkung 5.4** Analog zum Beweis von Satz 5.1 kann man auch hier den Integrationsfehler durch den Interpolationsfehler abschätzen, was auf die Ungleichung

$$|F_n[f, \omega]| \leq \frac{\langle P_{n+1}, P_{n+1} \rangle_\omega}{(2n+2)!} \|f^{(2n+1)}\|_\infty$$

führt.  $\square$

Wir beenden diesen Abschnitt mit einigen Beispielen von Gauß-Quadratur-Formeln.

**Beispiel 5.5** (i) Für  $\omega(x) = 1$  und Integrationsintervall  $[-1, 1]$  erhalten wir als orthogonale Polynome die *Legendre-Polynome*; für  $n = 1$  ergeben diese die Null- bzw. Stützstellen  $x_{0/1} = \pm 1/\sqrt{3}$  und Gewichte  $\lambda_{0/1} = 1$ . Die zugehörige Integrationsformel wird *Gauß-Legendre-Regel* genannt. Sie kann analog zu den zusammengesetzten Newton-Cotes-Formeln als zusammengesetzte Formel verwendet werden.

(ii) Für  $\omega(x) = 1/\sqrt{1-x^2}$  auf  $[-1, 1]$  erhalten wir die bekannten Tschebyscheff-Polynome  $T_n$ . Hier lässt sich die Integrationsformel explizit als

$$\frac{\pi}{n+1} \sum_{i=0}^n f\left(\cos\left(\frac{2i+1}{2n+2}\pi\right)\right)$$

angeben. Wegen der Singularitäten der Gewichtsfunktion in den Randpunkten kann diese Formel nicht zusammengesetzt werden.

(iii) Für  $\omega(x) = e^{-x}$  auf  $[0, \infty)$  bzw.  $\omega(x) = e^{-x^2}$  auf  $(-\infty, \infty)$  werden die zugehörigen Polynome *Laguerre-* bzw. *Hermite-Polynome* genannt. Für diese existieren keine geschlossenen Formeln für die Nullstellen und Gewichte, so dass diese numerisch bestimmt werden müssen. Diese numerische Berechnung kann mittels einer numerisch günstigen Auswertung der Rekursionsformeln aus Abschnitt 4.2 durchgeführt werden, vgl. Deuffhard/Hohmann [2], Abschnitt 9.3.2.  $\square$

Die Gauß-Legendre Integration aus (i) wird heutzutage eher selten verwendet, da Verfahren wie z.B. die Romberg-Extrapolation oft effizienter sind. Da sich die Romberg-Extrapolation auch zur Lösung von gewöhnlichen Differentialgleichungen eignet, wird diese Methode in der dortigen Vorlesung besprochen. Die Gauß-Quadratur hingegen ist nützlich, falls explizit mit vorgegebenen Gewichtsfunktionen wie in (ii) oder auf unendlichem Zeithorizont wie in (iii) integriert werden soll.



# Kapitel 6

## Nichtlineare Gleichungssysteme

Nichtlineare Gleichungen oder Gleichungssysteme müssen in vielen Anwendungen der Mathematik gelöst werden. Typischerweise werden die Lösungen nichtlinearer Gleichungen über die Nullstellen einer Funktion  $f : \mathbb{R}^n \rightarrow \mathbb{R}^n$  definiert, für die dann ein  $x^* \in \mathbb{R}^n$  mit

$$f(x^*) = 0$$

gesucht wird.

**Beispiel (Berechnung von Quadratwurzeln):** Berechne  $x^* \in \mathbb{R}$  mit  $f(x) = 0$  für  $f(x) = x^2 - 2$ . Die eindeutige positive reelle Lösung ist  $\sqrt{2}$ ; ein numerisches Verfahren zur Berechnung von Nullstellen kann also insbesondere zur Berechnung von Quadratwurzeln verwendet werden.

### 6.1 Fixpunktiteration

Die Fixpunktiteration ist eine recht einfache Methode, die auf der Idee beruht, die Lösung eines nichtlinearen Gleichungssystems als Fixpunktgleichung zu formulieren. Setzen wir

$$g(x) = f(x) + x,$$

so ist  $f(x^*) = 0$  äquivalent zu  $g(x^*) = x^*$ . Ein Punkt  $x \in \mathbb{R}^n$  mit  $g(x) = x$  heißt *Fixpunkt* von  $g$ . Statt eine Nullstelle von  $f$  zu suchen, können wir alternativ also einen Fixpunkt von  $g$  suchen.

Wir erinnern an den Banach'schen Fixpunktsatz Satz 2.18:

**Satz 2.18 (Banach'scher Fixpunktsatz)** Sei  $A$  eine abgeschlossene Teilmenge eines vollständigen normierten Raumes mit Norm  $\|\cdot\|$  und sei  $\Phi : A \rightarrow A$  eine Kontraktion, d.h. es existiere eine Konstante  $k \in (0, 1)$ , so dass die Ungleichung

$$\|\Phi(x) - \Phi(y)\| \leq k\|x - y\|$$

gilt. Dann existiert ein eindeutiger Fixpunkt  $x^* \in A$ , gegen den alle Folgen der Form  $x^{(i+1)} = \Phi(x^{(i)})$  mit beliebigen  $x^{(0)} \in A$  konvergieren. Darüberhinaus gelten die *a priori*

und *a posteriori* Abschätzungen

$$\|x^{(i)} - x^*\| \leq \frac{k^i}{1-k} \|x(1) - x^{(0)}\| \quad \text{und} \quad \|x^{(i)} - x^*\| \leq \frac{k}{1-k} \|x^{(i)} - x^{(i-1)}\|.$$

Die einfachste Idee zur Bestimmung eines Fixpunktes  $x^*$  von  $g$  liegt nun in der Iteration dieser Abbildung:

**Algorithmus 6.1 (Fixpunktiteration)** Gegeben seien eine Funktion  $g : \mathbb{R}^n \rightarrow \mathbb{R}^n$  und ein Startwert  $x^{(0)} \in \mathbb{R}^n$ . Weiterhin sei eine Abbruchgenauigkeit  $\varepsilon > 0$  gegeben.

(0) Setze  $i = 0$  (Zählindex).

(1) Setze  $x^{(i+1)} := g(x^{(i)})$ .

(2) Falls  $\|x^{(i+1)} - x^{(i)}\| \leq \varepsilon$  ist oder eine maximal erlaubte Iterationsanzahl überschritten ist, beende den Algorithmus;  
sonst setze  $i := i + 1$  und gehe zu (1)

□

Falls  $g$  die Bedingungen des Banach'schen Fixpunktsatzes auf einer Umgebung  $A$  von  $x^*$  erfüllt, so konvergiert dieses Verfahren für alle  $x^{(0)} \in A$ . Der Banach'sche Fixpunktsatz garantiert dann die Genauigkeit

$$\|x^{(i+1)} - x^*\| \leq \frac{k}{1-k} \varepsilon.$$

Natürlich erfüllt nicht jede Abbildung  $g$  die Voraussetzungen dieses Satzes. Falls  $g$  stetig differenzierbar ist, lässt sich die Kontraktionseigenschaft aber relativ leicht überprüfen.

**Satz 6.2** Es seien  $D \subset \mathbb{R}^n$  und  $g \in C^1(D, \mathbb{R}^n)$  mit Fixpunkt  $x^* \in D$ . Für eine beliebige Vektornorm  $\|\cdot\|$  sei  $A = \{x \in \mathbb{R}^n \mid \|x - x^*\| \leq \delta\} \subset D$  für ein  $\delta > 0$ . Dann erfüllt  $g$  die Voraussetzungen des Banach'schen Fixpunktsatzes bezüglich dieser Norm, falls

$$\max_{x \in A} \|Dg(x)\| =: k < 1$$

ist, wobei  $Dg(x) \in \mathbb{R}^{n \times n}$  die Jacobi-Matrix, also die matrixwertige Ableitung der Funktion  $g$  an der Stelle  $x$  bezeichnet und  $\|\cdot\|$  die von der gegebenen Vektornorm induzierte Matrixnorm ist. Insbesondere konvergiert also die Fixpunktiteration für alle Startwerte  $x^{(0)} \in A$ .

**Beweis:** Aus dem Mittelwertsatz der Differentialrechnung folgt für alle  $x, y \in A$  die Ungleichung

$$\|g(x) - g(y)\| \leq \sup_{z \in A} \|Dg(z)\| \|x - y\| \leq k \|x - y\|.$$

Daher ist  $g$  eine Kontraktion mit Kontraktionskonstante  $k$ .



Es bleibt zu zeigen, dass  $g$  die Menge  $A$  nach  $A$  abbildet. Sei dazu  $x \in A$ , also  $\|x - x^*\| \leq \delta$ . Es ist zu zeigen, dass dann auch  $g(x) \in A$ , also  $\|g(x) - x^*\| \leq \delta$  gilt. Dies folgt, denn

$$\|g(x) - x^*\|_\infty = \|g(x) - g(x^*)\|_\infty \leq k\|x - x^*\|_\infty \leq k\delta \leq \delta.$$

□

Für das folgende Korollar, in dem ein weiteres hinreichendes Kriterium für die Konvergenz der Fixpunktiteration hergeleitet wird, erinnern wir an den in Lemma 2.24 eingeführten Spektralradius  $\rho(A) = \max_i |\lambda_i(A)|$  einer Matrix  $A$ .

**Korollar 6.3** Es seien  $D \subset \mathbb{R}^n$  und  $g \in C^1(D, \mathbb{R}^n)$  mit Fixpunkt  $x^* \in D$ . Es gelte  $\rho(Dg(x^*)) < 1$  für den Spektralradius der Matrix  $Dg(x^*)$ . Dann existiert eine Umgebung  $A$  von  $x^*$ , so dass die Fixpunktiteration für alle Startwerte  $x^{(0)} \in A$  gegen  $x^*$  konvergiert.

**Beweis:** Wie im Beweis von Lemma 2.24 folgt die Existenz einer Vektornorm  $\|\cdot\|$  und zugehöriger induzierter Matrixnorm mit

$$k_\varepsilon := \|Dg(x^*)\| \leq \rho(Dg(x^*)) + \varepsilon < 1.$$

Für jede hinreichend kleine Umgebung  $A$  von  $x^*$  gilt dann wegen der Stetigkeit von  $Dg$  die Ungleichung  $\sup_{x \in A} \|Dg(x)\| \leq k_A < 1$ . Damit folgt die Behauptung mit Satz 6.2. □

Mit Korollar 6.3 kann für  $n = 1$  zudem das folgende Korollar bewiesen werden. Hierbei bezeichnet  $g^{-1}(x)$  die Umkehrabbildung der Funktion  $g(x)$ .

**Korollar 6.4** Sei  $g \in C^1(\mathbb{R}, \mathbb{R})$  mit  $g(x^*) = x^*$ . Es gelte  $|g'(x^*)| \neq 1$ . Dann gibt es eine Umgebung  $A$  von  $x^*$ , so dass eine der beiden Iterationen

- i)  $x^{(i+1)} = g(x^{(i)})$
- ii)  $x^{(i+1)} = g^{-1}(x^{(i)})$

für alle Startwerte  $x^{(0)} \in A$  gegen  $x^*$  konvergiert.

**Beweis:** Für eindimensionale Funktionen gilt  $\rho(g'(x^*)) = |g'(x^*)|$ . Da aus der Voraussetzung nun entweder

$$|g'(x^*)| < 1 \quad \text{oder} \quad |(g^{-1})'(x^*)| = \frac{1}{|g'(x^*)|} < 1$$

folgt, ergibt sich die Behauptung mit Korollar 6.3. □

Verfahren, die nur für Anfangswerte in einer Umgebung des gesuchten Wertes konvergieren, bezeichnen wir als *lokal konvergent*. Beispiele, in denen die Fixpunkt-Iteration gut funktioniert, finden sich in den Übungsaufgaben.

Das folgende Beispiel zeigt, dass der im vorhergehenden Satz beschriebene Übergang zur Umkehrabbildung leider nicht immer praktikabel ist und stellt zugleich eine weitere Methode vor, mit der das Problem der Nicht-Konvergenz behoben werden kann.

**Beispiel 6.5** Betrachte die Wurzelberechnung aus dem einführenden Beispiel mit  $f(x) = x^2 - 2$ . Hier ist die zugehörige Fixpunktabbildung gegeben durch  $g(x) = x^2 + x - 2$ , die Ableitung in  $x^* = \sqrt{2}$  ist  $g'(x^*) = 2x^* + 1 \approx 3.8284271$ , also ist der Banach'sche Fixpunktsatz nicht anwendbar. Die Umkehrfunktion lässt sich hier zwar theoretisch leicht berechnen, es gilt  $g^{-1}(x) = \sqrt{x + 9/4} - 1/2$ , und der Banach'sche Fixpunktsatz wäre auch anwendbar. Wenn wir aber diese Abbildung als Iterationsvorschrift verwenden, müssen wir in jedem Schritt eine Wurzel berechnen: wir haben also die Berechnung *einer* Wurzel  $\sqrt{2}$  durch eine Berechnung *mehrerer* Wurzeln ersetzt, was sicherlich wenig effizient ist.

Eine Abhilfe bietet hier aber die einfache Skalierung der ursprünglichen Funktion mit  $-1/2$ : Die Funktion  $f(x) = -x^2/2 + 1$  besitzt offenbar die gleichen Nullstellen wie das ursprüngliche  $f$ . Für die Fixpunktabbildung  $g(x) = -x^2/2 + x + 1$  gilt nun aber in  $x^* = \sqrt{2}$  gerade  $|g'(x^*)| = |-x^* + 1| \approx 0.41421356$ , weswegen das Verfahren hier lokal konvergiert, wie die Ergebnisfolge für Startwert  $x^{(0)} = 1$  zeigt:

```
x( 0) = 1.0000000000000000
x( 1) = 1.5000000000000000
x( 2) = 1.3750000000000000
x( 3) = 1.4296875000000000
x( 4) = 1.40768432617188
x( 5) = 1.41689674509689
x( 6) = 1.41309855196381
x( 7) = 1.41467479318270
x( 8) = 1.41402240794944
x( 9) = 1.41429272285787
x(10) = 1.41418076989350
```

Eine solche Skalierung funktioniert im eindimensionalen Fall “fast” immer, wie wir in einer Übungsaufgabe genauer untersuchen werden.  $\square$

Festzuhalten bleibt, dass die Fixpunktiteration eines der wenigen Verfahren ist, mit denen man allgemeine nichtlineare Gleichungssysteme im  $\mathbb{R}^n$  direkt lösen kann, ohne weitere Informationen wie Ableitungen etc. zu benötigen. Allerdings werden wir später sehen, dass das Verfahren vergleichsweise langsam konvergiert.

## 6.2 Das Bisektionsverfahren

Wir werden in diesem Kapitel auch Verfahren betrachten, die nur im  $\mathbb{R}^1$  funktionieren und die sich nicht auf höhere Dimensionen verallgemeinern lassen. Das nun folgende Bisektionsverfahren ist ein solches Verfahren, das sehr einfach und anschaulich ist und darüberhinaus unter minimalen Bedingungen global konvergiert.

Wir betrachten also eine stetige reelle Funktion  $f : \mathbb{R} \rightarrow \mathbb{R}$  und suchen eine Nullstelle, also einen Wert  $x^* \in \mathbb{R}$  mit  $f(x^*) = 0$ . Der folgende Algorithmus berechnet solch eine Nullstelle.

**Algorithmus 6.6 (Bisektionsverfahren)** Gegeben seien eine Funktion  $f : \mathbb{R} \rightarrow \mathbb{R}$  und Werte  $a < b$ , so dass  $f(a)f(b) < 0$  gilt (d.h.,  $f(a)$  und  $f(b)$  haben unterschiedliches Vorzeichen). Weiterhin sei eine gewünschte Genauigkeit  $\varepsilon > 0$  gegeben.

- (0) Setze  $i = 0$  (Zählindex) und  $a_0 = a$ ,  $b_0 = b$ .
- (1) Setze  $x^{(i)} = a_i + (b_i - a_i)/2$ .
- (2) Falls  $f(x^{(i)}) = 0$  oder  $(b_i - a_i)/2 < \varepsilon$  beende den Algorithmus
- (3) Falls  $f(x^{(i)})f(a_i) < 0$  ist setze  $a_{i+1} = a_i$ ,  $b_{i+1} = x^{(i)}$   
 Falls  $f(x^{(i)})f(a_i) > 0$  ist setze  $a_{i+1} = x^{(i)}$ ,  $b_{i+1} = b_i$   
 Setze  $i = i + 1$  und gehe zu Schritt (1).

□

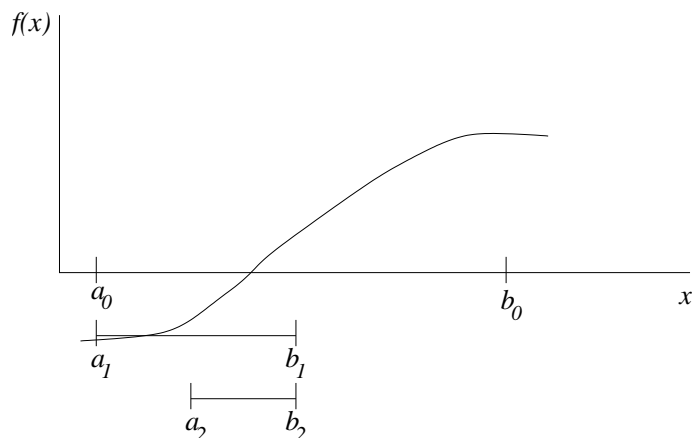


Abbildung 6.1: Bisektionsverfahren

Abbildung 6.1 illustriert dieses Verfahren. Man kann die Punkte  $a_i$ ,  $b_i$  als Intervallgrenzen der Intervalle  $[a_i, b_i]$  verstehen, mit denen die Nullstelle durch immer weitere Halbierung eingeschachtelt wird. Daher stammt der Name “Bisektion” (= Zweiteilung).

Die Auswahlbedingung der neuen Werte  $a_{i+1}$  und  $b_{i+1}$  stellt sicher, dass  $f(a_{i+1})$  und  $f(b_{i+1})$  unterschiedliches Vorzeichen haben; deswegen muss (da  $f$  stetig ist) sich eine Nullstelle zwischen diesen Werten befinden. Wenn die Abbruchbedingung  $(x^{(i)} - a_i) < \varepsilon$  erreicht ist, ist also sichergestellt, dass es eine Nullstelle  $x^*$  mit  $|x^* - x^{(i)}| < \varepsilon$  gibt, dass also  $x^{(i)}$  eine approximative Nullstelle ist.

Das Bisektionsverfahren hat einige sehr vorteilhafte Eigenschaften:

- Es funktioniert für allgemeine stetige Funktionen.
- Es liefert immer ein Ergebnis, vorausgesetzt, dass man geeignete Startwerte  $a$  und  $b$  finden kann (man sagt, dass das Verfahren “global konvergiert”).

- Die Anzahl der Schritte, nach der die gewünschte Genauigkeit erreicht ist, hängt nur von  $a$  und  $b$  aber nicht von  $f$  ab.

Bei der Anwendung auf die Wurzelberechnung mit  $f(x) = x^2 - 2$  und Startintervall  $[1, 2]$  erhält man die folgenden Werte:

```

i= 0: [1.000000, 2.000000], x( 0)=1.500000
i= 1: [1.000000, 1.500000], x( 1)=1.250000
i= 2: [1.250000, 1.500000], x( 2)=1.375000
i= 3: [1.375000, 1.500000], x( 3)=1.437500
i= 4: [1.375000, 1.437500], x( 4)=1.406250
i= 5: [1.406250, 1.437500], x( 5)=1.421875
i= 6: [1.406250, 1.421875], x( 6)=1.414062
i= 7: [1.414062, 1.421875], x( 7)=1.417969
i= 8: [1.414062, 1.417969], x( 8)=1.416016
i= 9: [1.414062, 1.416016], x( 9)=1.415039
i=10: [1.414062, 1.415039], x(10)=1.414551
i=11: [1.414062, 1.414551], x(11)=1.414307
i=12: [1.414062, 1.414307], x(12)=1.414185
i=13: [1.414185, 1.414307], x(13)=1.414246
i=14: [1.414185, 1.414246], x(14)=1.414215
i=15: [1.414185, 1.414215], x(15)=1.414200
i=16: [1.414200, 1.414215], x(16)=1.414207
i=17: [1.414207, 1.414215], x(17)=1.414211
i=18: [1.414211, 1.414215], x(18)=1.414213
i=19: [1.414213, 1.414215], x(19)=1.414214
i=20: [1.414213, 1.414214], x(20)=1.414214
i=21: [1.414213, 1.414214], x(21)=1.414213
i=22: [1.414213, 1.414214], x(22)=1.414214
i=23: [1.414214, 1.414214], x(23)=1.414214

```

Der Grund, warum in der Praxis auch im eindimensionalen Fall trotzdem oft andere Verfahren eingesetzt werden, liegt darin, dass das Verfahren — ebenso wie die Fixpunktiteration — relativ langsam gegen den gesuchten Wert  $x^*$  konvergiert. Um dies zu verstehen, müssen wir zunächst geeignete Konzepte zum Messen von Konvergenzgeschwindigkeiten einführen.

### 6.3 Konvergenzordnung

Der Begriff der Konvergenzordnung liefert eine Möglichkeit, iterative Verfahren auf ihre Geschwindigkeit hin zu untersuchen. Wir werden hier drei verschiedene Konvergenzordnungen betrachten: Lineare Konvergenz, superlineare Konvergenz und quadratische Konvergenz.

Iterative Verfahren liefern eine Folge approximativer Lösungen  $x^{(i)}$ , die gegen die exakte Lösung  $x^*$  konvergieren. Die Konvergenzordnung wird über den Fehler

$$\|x^{(i)} - x^*\|$$

definiert und gibt an, wie schnell dieser Fehler gegen Null konvergiert.

Die folgende Definition beschreibt die drei Arten der Konvergenzordnung, die wir hier betrachten wollen.

**Definition 6.7** Betrachte ein iteratives Verfahren, das eine Folge von approximativen Lösungen  $x^{(i)}$  für die exakte Lösung  $x^*$  liefert. Dann definieren wir die folgenden Konvergenzordnungen:

- (i) Das Verfahren heißt *linear konvergent*, falls eine Konstante  $c \in (0, 1)$  existiert, so dass die Abschätzung

$$\|x^{(i+1)} - x^*\| \leq c\|x^{(i)} - x^*\| \text{ für alle } i = 0, 1, 2, \dots$$

gilt.

- (ii) Das Verfahren heißt *superlinear konvergent*, falls Konstanten  $c_i \in (0, 1)$  für  $i = 0, 1, 2, \dots$  existieren, so dass die Bedingungen

$$c_{i+1} \leq c_i, \quad i = 0, 1, 2, \dots, \quad \lim_{i \rightarrow \infty} c_i = 0$$

und die Abschätzung

$$\|x^{(i+1)} - x^*\| \leq c_i\|x^{(i)} - x^*\| \text{ für alle } i = 0, 1, 2, \dots$$

gelten.

- (iii) Das Verfahren heißt *quadratisch konvergent*, falls eine Konstante  $q > 0$  existiert, so dass die Abschätzung

$$\|x^{(i+1)} - x^*\| \leq q\|x^{(i)} - x^*\|^2 \text{ für alle } i = 0, 1, 2, \dots$$

gilt.

□

**Bemerkung 6.8** Durch iterative Anwendung dieser Ungleichungen erhält man die Fehlerabschätzungen

$$\begin{aligned} \|x^{(i)} - x^*\| &\leq c^i \|x^{(0)} - x^*\| \\ \|x^{(i)} - x^*\| &\leq \prod_{k=0}^{i-1} c_k \|x^{(0)} - x^*\| \\ \|x^{(i)} - x^*\| &\leq \frac{1}{q} (q \|x^{(0)} - x^*\|)^{2^i}. \end{aligned}$$

Beachte, dass die dritte Ungleichung nur dann eine sinnvolle Fehlerabschätzung liefert, wenn  $q\|x^{(0)} - x^*\| < 1$  bzw.  $\|x^{(0)} - x^*\| < 1/q$  gilt, d.h. wenn der Anfangswert  $x^{(0)}$  bereits nahe genug am exakten Ergebnis  $x^*$  liegt. □

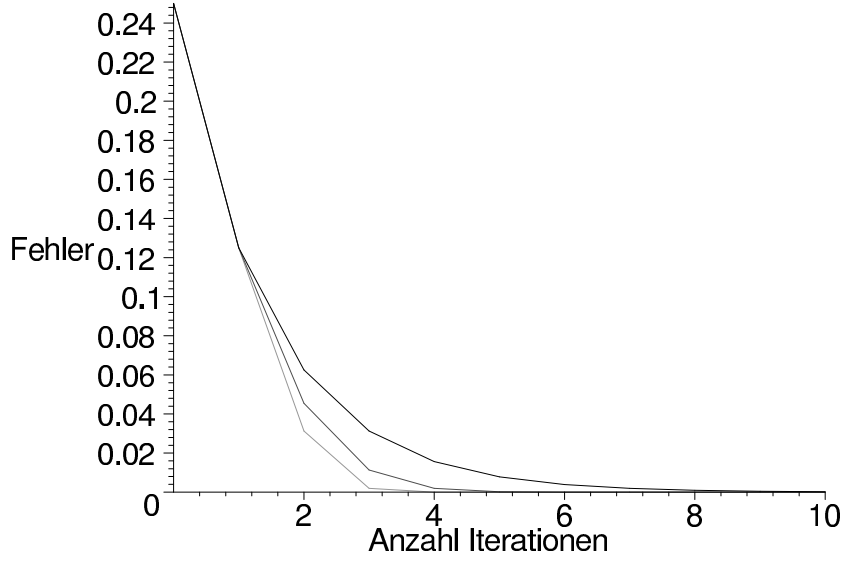


Abbildung 6.2: Konvergenzordnungen: linear, superlinear und quadratisch (von oben nach unten)

Abbildung 6.2 zeigt die Abhängigkeit der Fehler für  $c = 0.5$ ,  $c_i = \frac{4}{(i+1)^{2+7}}$ ,  $q = 2$  und  $\|x^{(0)} - x^*\| = 1/4$ .

Zwar kann man erkennen, dass die quadratische Konvergenz schneller gegen Null tendiert als die superlineare, und diese wiederum als die lineare, allerdings lässt sich aus dieser Grafik nicht direkt erkennen, welche Konvergenzordnung einer bestimmten Kurve zu Grunde liegt.

Dies lässt sich viel leichter feststellen, wenn man statt des Fehlers den *Logarithmus des Fehlers* betrachtet. Für den Logarithmus gelten die Rechenregeln

$$\log(ab) = \log(a) + \log(b), \quad \log(1/q) = -\log(q) \quad \text{und} \quad \log(c^d) = d \log(c).$$

Damit erhalten wir für die drei Konvergenzarten aus Definition 6.7 die Ungleichungen

$$\begin{aligned} \log(\|x^{(i+1)} - x^*\|) &\leq \log(\|x^{(i)} - x^*\|) + \log(c) \\ \log(\|x^{(i+1)} - x^*\|) &\leq \log(\|x^{(i)} - x^*\|) + \log(c_i) \\ \log(\|x^{(i+1)} - x^*\|) &\leq 2 \log(\|x^{(i)} - x^*\|) + \log(q). \end{aligned} \tag{6.1}$$

und mit der Abkürzung  $K = \log(\|x^{(0)} - x^*\|)$  aus Bemerkung 6.8 die Abschätzungen

$$\begin{aligned} \log(\|x^{(i)} - x^*\|) &\leq i \log(c) + K \\ \log(\|x^{(i)} - x^*\|) &\leq \sum_{k=0}^{i-1} \log(c_k) + K \\ \log(\|x^{(i)} - x^*\|) &\leq 2^i (\log(q) + K) - \log(q). \end{aligned}$$

Beachte, dass der Logarithmus dabei gegen minus unendlich strebt, wenn der Fehler gegen Null konvergiert.

Wenn man nun diese letzten drei Abschätzungen grafisch darstellt, so erhält man im linearen Fall eine Gerade mit negativer Steigung, im quadratischen Fall eine Kurve der Form  $i \mapsto -C2^i + D$  für  $C > 0$ ,  $D \in \mathbb{R}$  und im superlinearen Fall eine dazwischenliegende Kurve, deren Neigung immer weiter zunimmt, die also negative Krümmung besitzt. Abbildung 6.3 zeigt das typische Verhalten dieser Kurven für den Logarithmus der Basis 10.

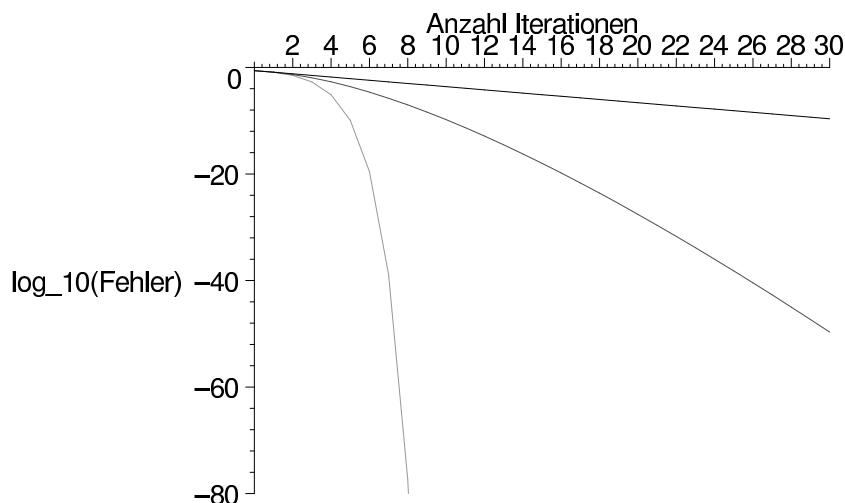


Abbildung 6.3: Konvergenzordnungen: linear, superlinear und quadratisch (von oben nach unten)

Tatsächlich würden diese Kurven für jeden anderen Logarithmus ähnlich aussehen, der Logarithmus zur Basis 10 hat aber die spezielle Eigenschaft, dass man die Genauigkeit direkt ablesen kann, wenn man sie in der Anzahl der korrekten Nachkommastellen des Ergebnisses misst, zumindest im eindimensionalen Fall, also für  $x \in \mathbb{R}$ :

Sei dazu  $d = \log_{10}(|x^{(i)} - x^*|)$ . Wir nehmen an, dass  $d$  negativ ist, was genau dann der Fall ist, wenn  $|x^{(i)} - x^*| < 1$  ist (die folgenden Überlegungen gelten also nur, wenn  $x^{(i)}$  bereits hinreichend nahe an  $x^*$  liegt). Sei nun  $m > 0$  die größte ganze Zahl, die echt kleiner als  $-d$  ist. Dann gilt

$$|x^{(i)} - x^*| = 10^d < 10^{-m} = \underbrace{0.0 \dots 0}_{(m-1)\text{-mal}}1.$$

Jede Zahl, die kleiner als  $10^{-m}$  ist, ist von der Form

$$\underbrace{0.0 \dots 0}_{m\text{-mal}} * * * \dots,$$

wobei der Stern “\*” beliebige Ziffern symbolisiert. Also ist

$$|x^{(i)} - x^*| \leq \underbrace{0.0 \dots 0}_{m\text{-mal}} * * * \dots,$$

weswegen  $x^{(i)}$  und  $x^*$  mindestens in den ersten  $m$  Nachkommastellen übereinstimmen müssen. Auf diese Weise lässt sich aus  $d$  die Anzahl der korrekten Nachkommastellen des Ergebnisses direkt ablesen.

Aus den Ungleichungen (6.1) kann man daraus die Konsequenzen ableiten, dass sich die Anzahl der korrekten Stellen bei linearer Konvergenz etwa alle  $1/(-\log(c))$  Schritte um 1 erhöht<sup>1</sup> und bei quadratischer Konvergenz (wenn man den  $\log(q)$  Summanden vernachlässigt) in jedem Schritt in etwa verdoppelt. Superlineare Konvergenz liegt auch hier zwischen diesen Werten: wie bei der linearen Konvergenz erhöht sich die Anzahl der korrekten Stellen jeweils nach einer bestimmten Schrittzahl um 1, allerdings nimmt die Anzahl der für die Erhöhung benötigten Schritte mit zunehmender Iterationsdauer immer weiter ab.

In Tabelle 6.1 werden die Charakteristika der verschiedenen betrachteten Konvergenzordnungen noch einmal zusammengefasst.

Konvergenzordnung	linear	superlinear	quadratisch
Definition	$\ x^{(i+1)} - x^*\  \leq c\ x^{(i)} - x^*\ $ für ein $c \in (0, 1)$	$\ x^{(i+1)} - x^*\  \leq c_i\ x^{(i)} - x^*\ $ für $c_i \in (0, 1)$ , $c_i \searrow 0$	$\ x^{(i+1)} - x^*\  \leq q\ x^{(i)} - x^*\ ^2$ für ein $q > 0$
Kurve im log-Diagramm	Gerade	Kurve mit negativer Krümmung	$\approx i \mapsto -C2^i + D$
Anzahl korrekter Nachkommastellen	erhöht sich um 1 nach ca. $1/(-\log(c))$ Schritten	wie linear, aber mit immer schnellerer Zunahme	verdoppelt sich ca. nach jedem Schritt

Tabelle 6.1: Charakteristika verschiedener Konvergenzordnungen

Wir wollen nun die Konvergenzordnung der bisher betrachteten Verfahren bestimmen.

Für die Fixpunktiteration erhalten wir aus den Voraussetzungen des Banach'schen Fixpunktsatzes die Fehlerabschätzung

$$\|x^{(i+1)} - x^*\| = \|g(x^{(i)}) - g(x^*)\| \leq k\|x^{(i)} - x^*\|,$$

also lineare Konvergenz mit  $c = k$ . Ein Sonderfall ergibt sich, falls  $Dg(x^*) = 0$  gilt und  $g$  zweimal stetig differenzierbar ist. Dann gilt mit Taylor-Entwicklung um  $x^*$  für die Komponenten  $g_j$  von  $g$  die Gleichung

$$g_j(x) - x_j^* = g_j(x) - g_j(x^*) = \frac{1}{2} \sum_{k,l=1}^n \frac{\partial^2 g_j(\xi_j)}{\partial x_k \partial x_l} (x_k - x_k^*)(x_l - x_l^*),$$

wobei  $x_j^*$  die  $j$ -te Komponente von  $x^*$  bezeichnet und  $\xi_j$  ein Punkt auf der Verbindungsgeraden von  $x$  nach  $x^*$  ist. Da die zweiten Ableitungen von jeder Komponente  $g_j$  nach Voraussetzung stetig sind, sind diese für  $x$  in einer Umgebung  $N$  von  $x^*$  durch eine Konstante  $r > 0$  beschränkt. Damit gilt mit  $q = r/2$

$$\begin{aligned} \|x^{(i+1)} - x^*\|_\infty &= \|g(x^{(i)}) - x^*\|_\infty = \max_{j=1,\dots,n} |g_j(x^{(i)}) - x_j^*| \\ &\leq \max_{j=1,\dots,n} q(x_j^{(i)} - x_j^*)^2 \leq q\|x^{(i)} - x^*\|_\infty^2, \end{aligned}$$

<sup>1</sup>Wenn  $1/(-\log(c)) < 1$  ist, ist dies so zu verstehen, dass die Anzahl korrekter Stellen pro Schritt im Mittel um  $-\log(c)$  zunimmt.



also quadratische Konvergenz.

Für das Bisektionsverfahren aus Algorithmus 6.6 müssen wir den Fehler etwas anders definieren. Bei diesem Verfahren kann der Fall eintreten, dass der Wert  $x^{(i)}$  zufällig sehr nahe an der gesuchten Nullstelle liegt und sich in weiteren Iterationsschritten zunächst wieder entfernt, bevor er letztendlich konvergiert. Tatsächlich sollte man hier den Fehler nicht über den Abstand  $|x^{(i)} - x^*|$  sondern über die Intervallgröße  $(b_i - a_i)$  definieren, da aus der Konstruktion sofort die Abschätzung  $|x^{(i)} - x^*| \leq (b_i - a_i)/2$  folgt. Diese Intervallgröße halbiert sich in jedem Schritt; man erhält also

$$(b_{i+1} - a_{i+1}) \leq \frac{1}{2}(b_i - a_i)$$

und damit lineare Konvergenz mit  $c = 1/2$ .

In den folgenden Abschnitten werden wir Verfahren erläutern, die quadratisch oder zumindest superlinear konvergieren.

**Bemerkung 6.9** Wir haben bereits im Kapitel über lineare Gleichungssysteme iterative Verfahren, nämlich das Jacobi-, das Gauss-Seidel-Verfahren und das CG-Verfahren kennen gelernt. Alle diese Verfahren haben ebenfalls lineare Konvergenzordnung.  $\square$

## 6.4 Das Newton-Verfahren

In diesem Abschnitt werden wir ein weiteres Verfahren zur Lösung nichtlinearer Gleichungssysteme betrachten, das Newton-Verfahren. Im Vergleich zu den bisher betrachteten Verfahren (mit Ausnahme der Fixpunktiteration mit verschwindender Ableitung) konvergiert dieses deutlich schneller, es ist quadratisch konvergent.

Im Gegensatz zum Bisektions-Verfahren oder der Fixpunktiteration wird hier nicht allerdings nur die Funktion  $f$  selbst sondern auch ihre Ableitung benötigt. Wir beschreiben das Verfahren zunächst im  $\mathbb{R}^1$  und gehen danach zum  $\mathbb{R}^n$  über.

Die Idee des Newton-Verfahrens ist wie folgt: Berechne die Tangente  $g(x)$  von  $f$  im Punkt  $x^{(i)}$ , d.h. die Gerade

$$g(x) = f(x^{(i)}) + f'(x^{(i)})(x - x^{(i)})$$

und wähle  $x^{(i+1)}$  als Nullstelle von  $g$ , also

$$\begin{aligned} f(x^{(i)}) + f'(x^{(i)})(x^{(i+1)} - x^{(i)}) = 0 &\Leftrightarrow f'(x^{(i)})x^{(i+1)} = f'(x^{(i)})x^{(i)} - f(x^{(i)}) \\ &\Leftrightarrow x^{(i+1)} = x^{(i)} - \frac{f(x^{(i)})}{f'(x^{(i)})}. \end{aligned}$$

Die Idee ist in Abbildung 6.4 grafisch dargestellt.

Formal lässt sich der Algorithmus im  $\mathbb{R}^1$  wie folgt beschreiben.

**Algorithmus 6.10 (Newton-Verfahren)** Gegeben sei eine Funktion  $f : \mathbb{R} \rightarrow \mathbb{R}$ , ihre Ableitung  $f' : \mathbb{R} \rightarrow \mathbb{R}$  sowie ein Startwert  $x^{(0)} \in \mathbb{R}$  und eine gewünschte Genauigkeit  $\varepsilon > 0$ . Setze  $i = 0$ .

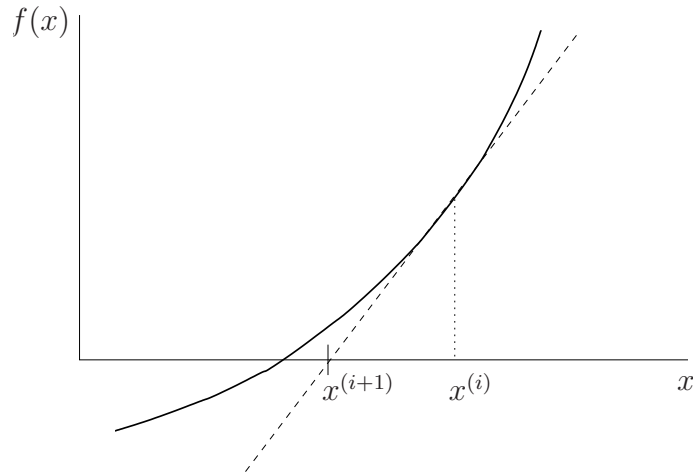


Abbildung 6.4: Newton-Verfahren

- (1) Berechne  $x^{(i+1)} = x^{(i)} - f(x^{(i)})/f'(x^{(i)})$
- (2) Falls  $|x^{(i+1)} - x^{(i)}| < \varepsilon$ , beende den Algorithmus, ansonsten setze  $i = i + 1$  und gehe zu (1)

□

Das folgende Beispiel zeigt den Verlauf des Newton-Verfahrens für das bereits bekannte Beispiel 6.5.

**Beispiel 6.11** Betrachte die Funktion  $f(x) = x^2 - 2$  mit  $f'(x) = 2x$  und Nullstelle  $x^* = \sqrt{2} \approx 1.4142135623731$ . Die Iterationsvorschrift des Newton-Verfahrens ergibt hier

$$x^{(i+1)} = x^{(i)} - f(x^{(i)})/f'(x^{(i)}) = x^{(i)} - \frac{(x^{(i)})^2 - 2}{2x^{(i)}} = \frac{1}{2}x^{(i)} + \frac{1}{x^{(i)}}$$

Wir wählen den Startwert  $x^{(0)} = 2$ . Damit erhalten wir (korrekte Nachkommastellen sind unterstrichen)

$$\begin{aligned} x^{(1)} &= \frac{1}{2} \cdot 2 + \frac{1}{2} &= \frac{3}{2} &= 1.5 \\ x^{(2)} &= \frac{1}{2} \cdot \frac{3}{2} + \frac{1}{\frac{3}{2}} &= \frac{17}{12} &= 1.\underline{41}\bar{6} \\ x^{(3)} &= \frac{1}{2} \cdot \frac{17}{12} + \frac{1}{\frac{17}{12}} &= \frac{577}{408} &\approx 1.\underline{41421}56862745 \\ x^{(4)} &= \frac{1}{2} \cdot \frac{577}{408} + \frac{1}{\frac{577}{408}} &= \frac{665857}{470832} &\approx 1.\underline{41421356237}46 \end{aligned}$$

□

Im Gegensatz zum Bisektionsverfahren lässt sich das Newton-Verfahren auf nichtlineare Gleichungssysteme im  $\mathbb{R}^n$  verallgemeinern. Wenn wir die Iterationsvorschrift des Newton-Verfahrens für  $f : \mathbb{R} \rightarrow \mathbb{R}$

$$x^{(i+1)} = x^{(i)} - \frac{f(x^{(i)})}{f'(x^{(i)})}$$

betrachten, stellt sich die Frage, wie eine geeignete Verallgemeinerung aussehen kann.

Sei dazu  $f : \mathbb{R}^n \rightarrow \mathbb{R}^n$  nun eine vektorwertige Funktion. Die Ableitung an einer Stelle  $x \in \mathbb{R}^n$ , die wir wie üblich mit  $Df(x)$  bezeichnen, ist jetzt keine reelle Zahl mehr, sondern eine Matrix.

Natürlich können wir diese Ableitung  $Df(x)$  nicht einfach in die Iterationsvorschrift für  $x^{(i+1)}$  einsetzen, da man ja durch eine Matrix nicht teilen kann. Man kann also nicht einfach  $f(x^{(i)})/f'(x^{(i)})$  durch  $f(x^{(i)})/Df(x^{(i)})$  ersetzen, sondern muss, um denselben Effekt zu erzielen, die entsprechende Operation für Matrizen verwenden. Statt durch  $Df(x^{(i)})$  zu teilen, multiplizieren wir nun mit  $[Df(x^{(i)})]^{-1}$ , berechnen also  $[Df(x^{(i)})]^{-1}f(x^{(i)})$ . Dies führt zum folgenden Algorithmus

**Algorithmus 6.12 (Newton-Verfahren im  $\mathbb{R}^n$ , Version 1)**

Gegeben sei eine Funktion  $f : \mathbb{R}^n \rightarrow \mathbb{R}^n$ , ihre Ableitung  $Df : \mathbb{R}^n \rightarrow \mathbb{R}^{n \times n}$  sowie ein Startwert  $x^{(0)} \in \mathbb{R}^n$  und eine gewünschte Genauigkeit  $\varepsilon > 0$ . Setze  $i = 0$ .

- (1) Berechne  $x^{(i+1)} = x^{(i)} - [Df(x^{(i)})]^{-1}f(x^{(i)})$
- (2) Falls  $\|x^{(i+1)} - x^{(i)}\| < \varepsilon$ , beende den Algorithmus, ansonsten setze  $i = i + 1$  und gehe zu (1)

□

Wir illustrieren den Ablauf dieses Algorithmus an dem obigen Beispiel.

**Beispiel 6.13** Gesucht ist eine Lösung des nichtlinearen Gleichungssystems

$$\begin{aligned} x_1^2 + x_2^2 &= 1 \\ x_1 &= 0 \end{aligned}$$

Dies ist äquivalent zum Suchen einer Nullstelle  $x^* \in \mathbb{R}^2$  der Funktion  $f : \mathbb{R}^2 \rightarrow \mathbb{R}^2$  gegeben durch

$$f(x) = \begin{pmatrix} f_1(x) \\ f_2(x) \end{pmatrix} = \begin{pmatrix} x_1^2 + x_2^2 - 1 \\ x_1 \end{pmatrix}.$$

Für die gesuchte Lösung  $x^*$  muss also gleichzeitig  $f_1(x^*) = 0$  und  $f_2(x^*) = 0$  gelten. Für die hier gegebene Funktion  $f$  ist die Lösung leicht zu sehen: Die Funktion  $f_1$  ist genau dann gleich Null, wenn  $x_1^2 + x_2^2 = 1$ , also  $\|x\| = 1$  ist. Die Funktion  $f_2$  ist gleich Null, wenn  $x_1 = 0$  ist. Die Menge der möglichen Lösungen bestehe also aus allen Vektoren  $(x_1, x_2)^T$  der Länge  $\|x\| = 1$ , für die  $x_1 = 0$  ist, also  $x^* = (0, 1)^T$  oder  $x^* = (0, -1)^T$ .

Die partiellen Ableitungen von  $f_1$  und  $f_2$  lauten

$$\frac{\partial f_1}{\partial x_1}(x) = 2x_1, \quad \frac{\partial f_1}{\partial x_2}(x) = 2x_2, \quad \frac{\partial f_2}{\partial x_1}(x) = 1, \quad \frac{\partial f_2}{\partial x_2}(x) = 0$$

also ist

$$Df(x) = \begin{pmatrix} 2x_1 & 2x_2 \\ 1 & 0 \end{pmatrix}$$

Damit ist z.B. für  $x = (0, 1)^T$  die Ableitung gegeben durch

$$Df(x) = \begin{pmatrix} 0 & 2 \\ 1 & 0 \end{pmatrix}.$$

Die Inverse der Ableitung

$$Df(x) = \begin{pmatrix} 2x_1 & 2x_2 \\ 1 & 0 \end{pmatrix}$$

ist gegeben durch

$$Df(x)^{-1} = \begin{pmatrix} 0 & 1 \\ \frac{1}{2x_2} & -\frac{x_1}{x_2} \end{pmatrix}.$$

Die Iterationsvorschrift ergibt sich mit  $x^{(i)} = (x_1^{(i)}, x_2^{(i)})^T$  also zu

$$x^{(i+1)} = x^{(i)} - \begin{pmatrix} 0 & 1 \\ \frac{1}{2x_2^{(i)}} & -\frac{x_1^{(i)}}{x_2^{(i)}} \end{pmatrix} \begin{pmatrix} (x_1^{(i)})^2 + (x_2^{(i)})^2 - 1 \\ x_1^{(i)} \end{pmatrix}.$$

Mit  $x^{(0)} = (1, 1)$  ergibt sich so (korrekte Nachkommastellen sind wieder unterstrichen)

$$\begin{aligned} x^{(1)} &= \begin{pmatrix} 1 \\ 1 \end{pmatrix} - \begin{pmatrix} 0 & 1 \\ \frac{1}{2} & -1 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ \frac{3}{2} \end{pmatrix} = \begin{pmatrix} 0 \\ 1.5 \end{pmatrix} \\ x^{(2)} &= \begin{pmatrix} 0 \\ \frac{3}{2} \end{pmatrix} - \begin{pmatrix} 0 & 1 \\ \frac{1}{3} & 0 \end{pmatrix} \begin{pmatrix} \frac{5}{4} \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ \frac{13}{12} \end{pmatrix} = \begin{pmatrix} 0 \\ 1.\underline{08\bar{3}} \end{pmatrix} \\ x^{(3)} &= \begin{pmatrix} 0 \\ \frac{13}{12} \end{pmatrix} - \begin{pmatrix} 0 & 1 \\ \frac{6}{13} & 0 \end{pmatrix} \begin{pmatrix} \frac{25}{144} \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ \frac{313}{312} \end{pmatrix} \approx \begin{pmatrix} 0 \\ 1.\underline{0032051282} \end{pmatrix} \\ x^{(4)} &= \begin{pmatrix} 0 \\ \frac{313}{312} \end{pmatrix} - \begin{pmatrix} 0 & 1 \\ \frac{156}{313} & 0 \end{pmatrix} \begin{pmatrix} \frac{625}{97344} \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ \frac{195313}{195312} \end{pmatrix} \approx \begin{pmatrix} 0 \\ 1.\underline{0000051200} \end{pmatrix} \end{aligned}$$

□

In der Praxis wird man  $[Df(x)]^{-1}$  natürlich nicht “per Hand” berechnen, sondern eine numerische Routine verwenden. Tatsächlich ist es numerisch nicht besonders effizient, die Inverse der Matrix  $Df(x^{(i)})$  wirklich zu berechnen, statt dessen löst man das lineare Gleichungssystem  $Df(x^{(i)})\Delta x^{(i)} = f(x^{(i)})$ , das einen Vektor  $\Delta x^{(i)}$  mit  $\Delta x^{(i)} = [Df(x^{(i)})]^{-1}f(x^{(i)})$  liefert. Dies führt zu der folgenden effizienteren Version des Newton-Verfahrens.

**Algorithmus 6.14 (Newton-Verfahren im  $\mathbb{R}^n$ , Version 2)**

Gegeben sei eine Funktion  $f : \mathbb{R}^n \rightarrow \mathbb{R}^n$ , ihre Ableitung  $Df : \mathbb{R}^n \rightarrow \mathbb{R}^{n \times n}$  sowie ein Startwert  $x^{(0)} \in \mathbb{R}^n$  und eine gewünschte Genauigkeit  $\varepsilon > 0$ . Setze  $i = 0$ .

- (1) Löse das lineare Gleichungssystem  $Df(x^{(i)})\Delta x^{(i)} = f(x^{(i)})$  und berechne  $x^{(i+1)} = x^{(i)} - \Delta x^{(i)}$
- (2) Falls  $\|\Delta x^{(i)}\| < \varepsilon$ , beende den Algorithmus, ansonsten setze  $i = i + 1$  und gehe zu (1)

□

Der folgende Satz zeigt die Konvergenzeigenschaften des Newton–Verfahrens.

**Satz 6.15** Sei  $D \subset \mathbb{R}^n$  eine offene und konvexe Menge und sei  $f : D \rightarrow \mathbb{R}^n$  eine stetig differenzierbare Funktion mit invertierbarer Jacobi–Matrix  $Df(x)$  für alle  $x \in D$ . Für ein  $\omega > 0$  gelte die folgende *affin–invariante Lipschitz–Bedingung*

$$\|Df(x)^{-1}(Df(x+sv) - Df(x))v\| \leq s\omega\|v\|^2$$

für alle  $s \in [0, 1]$ , alle  $x \in D$  und alle  $v \in \mathbb{R}^n$  mit  $x+v \in D$ . Sei  $x^* \in D$  eine Nullstelle von  $f$ .

Dann gilt: Für alle Startwerte  $x^{(0)} \in \mathbb{R}^n$  mit

$$\rho := \|x^* - x^{(0)}\| < \frac{2}{\omega} \quad \text{und} \quad B_\rho(x^*) = \{x \in \mathbb{R}^n \mid \|x - x^*\| < \rho\} \subseteq D$$

bleibt die durch das Newton–Verfahren definierte Folge  $x^{(i)}$  für  $i > 0$  im Ball  $B_\rho(x^*)$  und konvergiert gegen  $x^*$ , d.h.

$$\|x^{(i)} - x^*\| < \rho \quad \text{für alle } i > 0 \quad \text{und} \quad \lim_{i \rightarrow \infty} x^{(i)} = x^*.$$

Die Konvergenzordnung lässt sich dabei abschätzen durch

$$\|x^{(i+1)} - x^*\| \leq \frac{\omega}{2} \|x^{(i)} - x^*\|^2,$$

d.h. das Verfahren konvergiert lokal quadratisch. Darüberhinaus folgt aus den angegebenen Bedingungen, dass  $x^*$  die eindeutige Nullstelle in  $B_{2/\omega}(x^*)$  ist.

**Beweis:** Wir zeigen zunächst die folgende Hilfsaussage: Für alle  $x, y \in D$  gilt

$$\|Df(x)^{-1}(f(y) - f(x) - Df(x)(y-x))\| \leq \frac{\omega}{2} \|y-x\|^2 \quad (6.2)$$

Um (6.2) zu beweisen, benutzen wir den Mittelwertsatz der Integralrechnung im  $\mathbb{R}^n$ . Nach diesem gilt

$$f(y) - f(x) - Df(x)(y-x) = \int_0^1 (Df(x+s(y-x)) - Df(x))(y-x) ds.$$

Unter Ausnutzung der affin–invarianten Lipschitz–Bedingung gilt damit

$$\begin{aligned} & \|Df(x)^{-1}(f(y) - f(x) - Df(x)(y-x))\| \\ &= \left\| Df(x)^{-1} \left( \int_0^1 (Df(x+s(y-x)) - Df(x))(y-x) ds \right) \right\| \\ &\leq \int_0^1 s\omega \|y-x\|^2 ds = \frac{\omega}{2} \|y-x\|^2, \end{aligned}$$

also (6.2).

Aus der Iterationsvorschrift und  $f(x^*) = 0$  erhalten wir nun

$$\begin{aligned} x^{(i+1)} - x^* &= x^{(i)} - Df(x^{(i)})^{-1}f(x^{(i)}) - x^* \\ &= x^{(i)} - x^* - Df(x^{(i)})^{-1}(f(x^{(i)}) - f(x^*)) \\ &= Df(x^{(i)})^{-1}(f(x^*) - f(x^{(i)})) - Df(x^{(i)})(x^* - x^{(i)}). \end{aligned}$$

Mit (6.2) ergibt dies

$$\|x^{(i+1)} - x^*\| \leq \frac{\omega}{2} \|x^{(i)} - x^*\|^2,$$

also die behauptete Abschätzung. Falls  $\|x^{(i)} - x^*\| \leq \rho$  gilt, folgt daraus

$$\|x^{(i+1)} - x^*\| \leq \frac{\omega}{2} \|x^{(i)} - x^*\| \|x^{(i)} - x^*\| < \underbrace{\rho\omega/2}_{<1} \|x^{(i)} - x^*\| < \rho,$$

weswegen die Folge für  $\|x^{(i)} - x^*\| \leq \rho$  für alle  $i > 0$  in  $B_\rho(x^*)$  bleibt und gegen  $x^*$  konvergiert.

Zum Beweis der Eindeutigkeit von  $x^*$  in  $B_{2/\omega}(x^*)$  sei  $x^{**}$  eine weitere Nullstelle von  $f$  in diesem Ball. Dann gilt  $\|x^{**} - x^*\| < 2/\omega$  und eingesetzt in (6.2) erhalten wir

$$\|x^{**} - x^*\| = \|Df(x^*)^{-1}(0 - 0 - Df(x^*)(x^{**} - x^*))\| \leq \underbrace{\frac{\omega}{2} \|x^{**} - x^*\|}_{<1} \|x^{**} - x^*\|,$$

was nur für  $x^{**} = x^*$  möglich ist. □

**Bemerkung 6.16** Falls  $f$  zweimal stetig differenzierbar mit invertierbarer Jacobi-Matrix ist, so ist die affin-invariante Lipschitz-Bedingung in einer (hinreichend kleinen) Umgebung der Nullstelle immer erfüllt, da dann  $\|Df(x + sv) - Df(x)\| \leq Cs\|v\|$  gilt, woraus die angegebene Bedingung mit  $\omega = \|Df(x)^{-1}\|C$  folgt. In diesem Fall ist die lokale quadratische Konvergenz (d.h. die quadratische Konvergenz für  $x^{(0)}$  hinreichend nahe bei  $x^*$ ) also sicher gestellt. □

In Satz 6.15 haben wir die Voraussetzung " $\|x^* - x^{(0)}\| \leq 2/\omega$ " verwendet. Dies ist keine Bedingung, die nur aus beweistechnischen Gründen eingeführt wurde: Tatsächlich kann es Situationen geben, in denen das Newton-Verfahren bei ungeeignetem Anfangswert  $x^{(0)}$  nicht oder nur sehr langsam konvergiert. Das folgende Beispiel soll dies verdeutlichen.

**Beispiel 6.17** Betrachte die Funktion  $f(x) = 5x/4 - x^3/4$  mit Ableitung  $f'(x) = 5/4 - 3x^2/4$ . Offenbar hat diese Funktion eine Nullstelle in  $x^* = 0$  mit  $f'(0) = 5/4 \neq 0$  und ist beliebig oft stetig differenzierbar. Die Iterationsvorschrift ergibt sich hier als

$$x^{(i+1)} = x^{(i)} - f(x^{(i)})/f'(x^{(i)}) = x^{(i)} - \frac{5x^{(i)}/4 - (x^{(i)})^3/4}{5/4 - 3(x^{(i)})^2/4} = \frac{2(x^{(i)})^3}{-5 + 3(x^{(i)})^2}.$$

Mit Startwert  $x^{(0)} = 1$  erhalten wir daraus

$$\begin{aligned} x^{(1)} &= \frac{2 \cdot 1}{-5 + 3 \cdot 1} = \frac{2}{-2} = -1 \\ x^{(2)} &= \frac{2 \cdot (-1)}{-5 + 3 \cdot 1} = \frac{-2}{-2} = 1 \\ x^{(3)} &= \frac{2 \cdot 1}{-5 + 3 \cdot 1} = \frac{2}{-2} = -1 \\ x^{(4)} &= \frac{2 \cdot (-1)}{-5 + 3 \cdot 1} = \frac{-2}{-2} = 1 \\ &\vdots \end{aligned}$$

Das Verfahren springt also für alle Zeiten zwischen 1 und  $-1$  hin und her und konvergiert nicht.  $\square$

Das Verfahren ist also tatsächlich nur lokal konvergent, es ist also wichtig, bereits einen brauchbaren Startwert  $x^{(0)}$  für die Iteration zu haben. Im Buch von Deuffhard/Hohmann ist ein Verfahren beschrieben, mit dem bereits nach wenigen Schritten erkannt werden kann, dass voraussichtlich keine Konvergenz zu erwarten ist.

Ein wesentlicher Nachteil des Newton-Verfahrens ist, dass die Ableitung der Funktion  $f$  in der Iterationsvorschrift benötigt wird. Tatsächlich kann man die Ableitung durch eine geeignete numerische Näherung ersetzen und damit die explizite Berechnung von  $Df(x^{(i)})$  vermeiden. Ersetzt man z.B.  $f'(x)$  durch  $(f(x + f(x)) - f(x))/f(x)$ , so erhält man im  $\mathbb{R}^1$  das sogenannte *Steffensen-Verfahren*.

Das Newton-Verfahren kann auch zur Bestimmung von Extremalstellen von Funktionen  $g : \mathbb{R}^n \rightarrow \mathbb{R}$  verwendet werden, wenn man  $f := \nabla g : \mathbb{R}^n \rightarrow \mathbb{R}^n$  setzt. In diesem Fall ist die Jacobi-Matrix  $Df$  gerade die Hesse-Matrix  $D^2g$  und man kann zur Vermeidung der exakten Berechnung von  $D^2g$  das sogenannte *BFGS-Verfahren* verwenden. Hierbei werden während der Iteration aus einer Startschätzung  $B^{(0)} \approx D^2g(x^{(0)})$  iterativ Näherungen  $B^{(i)} \approx D^2g(x^{(i)})$  erzeugt.

Da die numerische Approximation von Ableitungen im Allgemeinen sehr anfällig gegenüber Rundungsfehlern ist, muss die numerische Stabilität solcher Verfahren — möglichst unter Hinzunahme weiterer Informationen über  $f$  — genau überprüft werden.

Eine ableitungsfreie Alternative zum Newton-Verfahren, die im  $\mathbb{R}^1$  funktioniert, wollen wir im folgenden Abschnitt beispielhaft betrachten.

## 6.5 Das Sekanten-Verfahren

Das Sekanten-Verfahren ist wieder ein Verfahren, das nur im  $\mathbb{R}^1$  funktioniert. Es unterscheidet sich in der Konzeption leicht vom Newton-Verfahren, da hier die neue Näherung  $x^{(i+1)}$  nicht nur aus  $x^{(i)}$  sondern aus den zwei vorhergehenden Werten  $x^{(i-1)}$  und  $x^{(i)}$  berechnet wird.

Wir geben wiederum zunächst eine anschauliche und dann die formale Beschreibung.

Die Idee hinter diesem Verfahren ist wie folgt: man betrachtet zunächst die Sekante  $g(x)$  durch  $f(x^{(i-1)})$  und  $f(x^{(i)})$ , d.h. die Gerade

$$g(x) = f(x^{(i)}) + \frac{(x^{(i)} - x)}{x^{(i)} - x^{(i-1)}} \left( f(x^{(i-1)}) - f(x^{(i)}) \right)$$

und wählt  $x^{(i+1)}$  als Nullstelle dieser Geraden, also

$$\begin{aligned} 0 &= f(x^{(i)}) + \frac{(x^{(i)} - x^{(i+1)})}{x^{(i)} - x^{(i-1)}} \left( f(x^{(i-1)}) - f(x^{(i)}) \right) \\ \Leftrightarrow x^{(i)} - x^{(i+1)} &= -f(x^{(i)}) \frac{x^{(i)} - x^{(i-1)}}{f(x^{(i-1)}) - f(x^{(i)})} \\ \Leftrightarrow x^{(i+1)} &= x^{(i)} - f(x^{(i)}) \frac{x^{(i)} - x^{(i-1)}}{f(x^{(i)}) - f(x^{(i-1)})} \end{aligned}$$

Die Idee ist in Abbildung 6.5 grafisch veranschaulicht.

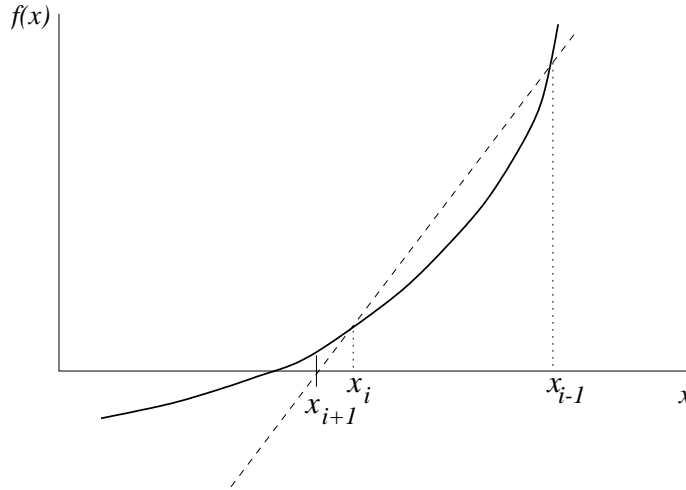


Abbildung 6.5: Sekanten-Verfahren

Formal lässt sich das Verfahren wie folgt beschreiben.

**Algorithmus 6.18 (Sekanten-Verfahren)** Gegeben sei eine Funktion  $f : \mathbb{R} \rightarrow \mathbb{R}$  sowie zwei Startwerte  $x^{(0)} \in \mathbb{R}$  und  $x^{(1)} \in \mathbb{R}$  und eine gewünschte Genauigkeit  $\varepsilon > 0$ . Setze  $i = 1$ .

- (1) Berechne  $x^{(i+1)} = x^{(i)} - \frac{(x^{(i)} - x^{(i-1)})f(x^{(i)})}{f(x^{(i)}) - f(x^{(i-1)})}$
- (2) Falls  $|x^{(i+1)} - x^{(i)}| < \varepsilon$ , beende den Algorithmus, ansonsten setze  $i = i + 1$  und gehe zu (1)

□



Der folgende Satz zeigt die Konvergenzordnungen dieses Verfahrens.

**Satz 6.19** Sei  $f : \mathbb{R} \rightarrow \mathbb{R}$  zweimal stetig differenzierbar und seien  $x^{(0)}$  und  $x^{(1)}$  hinreichend nahe bei  $x^*$ . Weiterhin gelte  $f'(x^*) \neq 0$ . Dann konvergiert das Sekanten-Verfahren superlinear.

**Beweisskizze:** Wenn  $x^{(i)}$  und  $x^{(i-1)}$  hinreichend nahe bei  $x^*$  liegen, folgt aus der Taylor-Entwicklung von  $f$  in  $x^*$  die Abschätzung

$$f(x^{(i)}) - f(x^{(i-1)}) \approx f'(x^*)(x^{(i)} - x^{(i-1)}) \quad (6.3)$$

sowie für  $j = i - 1$  und  $j = i$  die Abschätzungen

$$f(x^{(j)}) \approx f'(x^*)(x^{(j)} - x^*) + \frac{1}{2}f''(x^*)(x^{(j)} - x^*)^2. \quad (6.4)$$

Aus der Iterationsvorschrift ergibt sich die Gleichung

$$x^{(i+1)} - x^* = \frac{(x^{(i-1)} - x^*)f(x^{(i)}) - (x^{(i)} - x^*)f(x^{(i-1)})}{f(x^{(i)}) - f(x^{(i-1)})}.$$

Aus (6.3) erhalten wir für den Nenner die Abschätzung

$$f(x^{(i)}) - f(x^{(i-1)}) \approx f'(x^*)(x^{(i)} - x^{(i-1)})$$

und mittels (6.4) erhalten wir für den Zähler

$$\begin{aligned} & (x^{(i-1)} - x^*)f(x^{(i)}) - (x^{(i)} - x^*)f(x^{(i-1)}) \\ & \approx (x^{(i-1)} - x^*) \left( f'(x^*)(x^{(i)} - x^*) + \frac{1}{2}f''(x^*)(x^{(i)} - x^*)^2 \right) \\ & \quad - (x^{(i)} - x^*) \left( f'(x^*)(x^{(i-1)} - x^*) + \frac{1}{2}f''(x^*)(x^{(i-1)} - x^*)^2 \right) \\ & = \frac{1}{2}f''(x^*)(x^{(i-1)} - x^*)(x^{(i)} - x^*) \left( (x^{(i)} - x^*) - (x^{(i-1)} - x^*) \right) \\ & = \frac{1}{2}f''(x^*)(x^{(i-1)} - x^*)(x^{(i)} - x^*)(x^{(i)} - x^{(i-1)}). \end{aligned}$$

Also folgt

$$|x^{(i+1)} - x^*| \approx \underbrace{\frac{1}{2} \left| \frac{f''(x^*)}{f'(x^*)} \right|}_{\approx c_i} |x^{(i-1)} - x^*| |x^{(i)} - x^*|$$

und damit die behauptete superlineare Konvergenz.  $\square$

Wir wiederholen Beispiel 6.5 bzw. 6.11 für dieses Verfahren.

**Beispiel 6.20** Betrachte wiederum die Funktion  $f(x) = x^2 - 2$  mit Nullstelle  $x^* = \sqrt{2} \approx 1.4142135623731$ . Die Iterationsvorschrift des Sekanten-Verfahrens ergibt hier

$$\begin{aligned} x^{(i+1)} &= x^{(i)} - \frac{(x^{(i)} - x^{(i-1)})((x^{(i)})^2 - 2)}{(x^{(i)})^2 - (x^{(i-1)})^2} \\ &= x^{(i)} - \frac{(x^{(i)} - x^{(i-1)})((x^{(i)})^2 - 2)}{(x^{(i)} - x^{(i-1)})(x^{(i)} + x^{(i-1)})} = x^{(i)} - \frac{(x^{(i)})^2 - 2}{x^{(i)} + x^{(i-1)}}. \end{aligned}$$

Mit  $x^{(0)} = 2$  und  $x^{(1)} = 1$  ergibt sich (korrekte Nachkommastellen sind unterstrichen)

$$\begin{aligned} x^{(2)} &= 1 - \frac{1^2 - 2}{1 + 2} = \frac{4}{3} = 1.\underline{3} \\ x^{(3)} &= \frac{4}{3} - \frac{(\frac{4}{3})^2 - 2}{\frac{4}{3} + 1} = \frac{10}{7} = 1.\underline{4285714} \\ x^{(4)} &= \frac{10}{7} - \frac{(\frac{10}{7})^2 - 2}{\frac{10}{7} + \frac{4}{3}} = \frac{41}{29} \approx 1.\underline{4137931034483} \\ x^{(5)} &= \frac{41}{29} - \frac{(\frac{41}{29})^2 - 2}{\frac{41}{29} + \frac{10}{7}} = \frac{816}{577} \approx 1.\underline{4142114384749} \\ x^{(6)} &= \frac{816}{577} - \frac{(\frac{816}{577})^2 - 2}{\frac{816}{577} + \frac{41}{29}} = \frac{66922}{47321} \approx 1.\underline{4142135626889} \end{aligned}$$

Das Sekanten-Verfahren konvergiert also deutlich schneller als linear aber langsamer als das Newton-Verfahren.  $\square$

Auch das Sekanten-Verfahren ist nur lokal konvergent, d.h. die Anfangswerte müssen genügend nahe an  $x^*$  liegen, um die Konvergenz des Verfahrens zu garantieren. Hier bietet sich das Bisektions-Verfahren an, um gute Anfangswerte in der Nähe von  $x^*$  zu finden. Eine Strategie dieser Art wird z.B. in der Nullstellenroutine `fzero` in MATLAB benutzt.

## 6.6 Das Gauß-Newton-Verfahren für nichtlineare Ausgleichsprobleme

Zum Abschluss dieses Kapitels wollen wir noch einmal zum Ausgleichsproblem zurück kommen, das wir in Abschnitt 2.1 eingeführt haben. Wir haben dort das lineare Ausgleichsproblem betrachtet, das wir hier mit leicht anderer Notation zunächst noch einmal zusammenfassen wollen:

Zu einer Matrix  $A \in \mathbb{R}^{m \times n}$  mit  $m > n$  und einem Vektor  $z \in \mathbb{R}^m$  finde den Vektor  $x \in \mathbb{R}^n$ , der die (quadrierte) Norm

$$\|Ax - z\|_2^2$$

minimiert.

In der theoretischen Betrachtung haben wir gesehen, dass dieser Vektor  $x$  gerade die Lösung der Normalgleichungen

$$A^T Ax = A^T z$$

ist, die ein "gewöhnliches" lineares Gleichungssystem im  $\mathbb{R}^n$  darstellen, das z.B. mit dem Choleski-Verfahren gelöst werden kann.

Zur numerischen Lösung haben wir in Algorithmus 2.16 alternativ die  $QR$ -Zerlegung verwendet, die auf eine Zerlegung  $A = QR$  der Form

$$R = \begin{pmatrix} \bar{R} \\ 0 \end{pmatrix}$$

führt, wobei  $\bar{R} \in \mathbb{R}^{n \times n}$  eine obere Dreiecksmatrix ist. Hiermit kann man dann den Vektor  $x$  durch Rückwärtseinsetzen berechnen. Für die numerische Lösung ist dieses Vorgehen i.A. günstiger als die Lösung der Normalgleichungen, da die Matrix  $A^T A$  schlecht konditioniert sein kann.

Die Normalgleichungen liefern allerdings eine Möglichkeit, die Lösungen des linearen Ausgleichsproblems theoretisch zu analysieren. Hierzu verwenden wir die folgende Definition.

**Definition 6.21** Es sei  $m \geq n$  und  $A \in \mathbb{R}^{m \times n}$  eine Matrix mit vollem Rang. Dann definieren wir zu  $A$  die *Pseudoinverse*  $A^+ \in \mathbb{R}^{n \times m}$  als

$$A^+ = (A^T A)^{-1} A^T$$

□

Man sieht leicht, dass die Lösung des linearen Ausgleichsproblems  $\min \|Ax - z\|_2^2$  gerade durch

$$x = A^+ z$$

gegeben ist, denn

$$x = A^+ z \Leftrightarrow x = (A^T A)^{-1} A^T z \Leftrightarrow A^T A x = A^T z,$$

d.h.  $x = A^+ z$  löst gerade die Normalgleichungen.

**Bemerkung 6.22** Man rechnet leicht nach, dass  $A^+ = (A^T A)^{-1} A^T \in \mathbb{R}^{n \times m}$  die Eigenschaften

$$\begin{aligned} \text{(i)} \quad (A^+ A)^T &= A^+ A & \text{(ii)} \quad (A A^+)^T &= A A^+ \\ \text{(iii)} \quad A^+ A A^+ &= A^+ & \text{(iv)} \quad A A^+ A &= A \end{aligned}$$

erfüllt. Diese Eigenschaften (i)–(iv) heißen auch *Penrose-Axiome*.

Tatsächlich ist  $A^+$  sogar die *eindeutige*  $n \times m$ -Matrix, die diese Axiome erfüllt: Für jede Matrix  $A^+$ , die diese Axiome erfüllt, sind die linearen Abbildungen  $P := A^+ A$  und  $\bar{P} := A A^+$  orthogonale Projektionen, d.h. sie erfüllen  $P^T = P = P^2$  und  $\bar{P}^T = \bar{P} = \bar{P}^2$ . Darüberhinaus ist  $A^+$  injektiv, denn aus (iv) folgt

$$n = \dim \operatorname{im} A = \dim \operatorname{im} A A^+ A \leq \dim \operatorname{im} A^+,$$

also ist  $\dim \operatorname{im} A^+ = n$  weswegen die Dimension des Kerns von  $A^+$  gleich Null sein muss. Aus (iii) und der Injektivität von  $A^+$  folgt nun, dass der Kern von  $P$  gerade gleich dem Kern von  $A$  ist, und aus (iv) folgt (da  $A$  nach Annahme vollen Rang hat, also surjektiv ist), dass das Bild von  $\bar{P}$  gerade gleich dem Bild von  $A$  ist. Damit sind  $P$  und  $\bar{P}$  eindeutig bestimmt. Sind nun  $A_1^+$  und  $A_2^+$  zwei Matrizen, die die vier Axiome erfüllen, so muss demnach  $A_1^+ A = P = A_2^+ A$  und  $A A_1^+ = \bar{P} = A A_2^+$  gelten. Daraus folgt

$$A_1^+ \stackrel{\text{(iii)}}{=} \underbrace{A_1^+ A}_{=A_2^+ A} A_1^+ = A_2^+ \underbrace{A A_1^+}_{=A A_2^+} \stackrel{\text{(iii)}}{=} A_2^+.$$

□

Die Pseudoinverse  $A^+$  vereinfacht in erster Linie die Darstellung der Lösung des Ausgleichsproblems. Sie spielt damit eine wichtige Rolle in der nun folgenden Verallgemeinerung des Problems auf das nichtlineare Ausgleichsproblem.

Das nichtlineare Ausgleichsproblem ist, wie im linearen Fall, als ein Minimierungsproblem gegeben. Hierzu betrachten wir für  $D \subseteq \mathbb{R}^n$  und  $m > n$  eine zweimal stetig differenzierbare Abbildung

$$f : D \rightarrow \mathbb{R}^m$$

und wollen für diese Abbildung das Problem

$$\text{minimiere } g(x) := \|f(x)\|_2^2 \text{ über } x \in D \quad (6.5)$$

lösen.

Wie beim linearen Ausgleichsproblem ist die Interpretation und Anwendung dieses Problems wie folgt: Seien  $z_i$  Messwerte zu Parametern  $t_i$  für  $i = 1, \dots, m$ . Auf Grund theoretischer Überlegungen (z.B. eines zu Grunde liegenden physikalischen Gesetzes) weiß man, dass eine Funktion  $h : \mathbb{R} \times \mathbb{R}^n \rightarrow \mathbb{R}^m$ ,  $h(t, x) = z$  existiert, so dass die (idealen) Messwerte für einen geeigneten Parametervektor  $x^* \in \mathbb{R}^n$  die Gleichung  $h(t_i, x^*) = z_i$  erfüllen. Mit

$$f(x) = \begin{pmatrix} h(t_1, x) - z_1 \\ \vdots \\ h(t_m, x) - z_m \end{pmatrix}$$

würde also  $f(x^*) = 0$  gelten. Da wir hier Messfehler einkalkulieren müssen, wird diese Gleichung üblicherweise nicht exakt sondern nur approximativ erfüllt sein, weswegen wir also nach einer Lösung des Ausgleichsproblems (6.5) suchen.

Ein Beispiel für eine solche Anwendung ist z.B. durch Daten für ein Populationswachstum gegeben. Hier ist (unter idealen Bedingungen) ein theoretisches Wachstum der Form  $z = h(t, x) = x_1 e^{x_2 t}$  zu erwarten, also eine Funktion, die nichtlinear in  $x = (x_1, x_2)^T$  ist. Ein MATLAB M-File, in dem diese Anwendung als Beispiel mit realen Daten berechnet wird, findet sich auf der Vorlesungshomepage.

Wir leiten den Algorithmus zur Lösung nichtlinearer Ausgleichsprobleme zunächst informell her, schreiben ihn dann formal auf und formulieren und beweisen schließlich die exakten Konvergenzeigenschaften.

Wir wollen bei der Lösung des Problems (6.5) nur lokale Minima im Inneren von  $D$  betrachten, also Minimalstellen auf dem Rand  $\partial D$  nicht berücksichtigen. Darüberhinaus wollen wir uns auf lokale Minimalstellen  $x^* \in D$  von  $g : \mathbb{R}^n \rightarrow \mathbb{R}$  beschränken, die die hinreichenden Bedingungen

$$Dg(x^*) = 0 \quad \text{und} \quad D^2g(x^*) \text{ ist positiv definit} \quad (6.6)$$

erfüllen. Die Ableitung von  $g(x) = \|f(x)\|_2^2 = f(x)^T f(x)$  erfüllt

$$Dg(x)^T = 2Df(x)^T f(x),$$

also müssen wir zum Finden von Kandidaten von Minimalstellen das  $n \times n$ -nichtlineare Gleichungssystem

$$G(x) := Df(x)^T f(x) = 0 \quad (6.7)$$

lösen. Wenn wir hierfür das Newton-Verfahren einsetzen, so erhalten wir mit der Schreibweise aus Algorithmus 6.14 die Iteration  $x^{(i+1)} = x^{(i)} - \Delta x^{(i)}$  mit den iterativ zu lösenden Gleichungssystemen

$$DG(x^{(i)})\Delta x^{(i)} = G(x^{(i)}), \quad i = 0, 1, 2, \dots, \quad (6.8)$$

Falls ein lokales Minimum  $x^*$  mit (6.6) existiert, so ist

$$DG(x) = D^2g(x) = Df(x)^T Df(x) + D^2f(x)^T f(x)$$

in  $x = x^*$  positiv definit, also auch für alle  $x$  in einer Umgebung von  $x^*$ , weswegen  $DG(x)$  insbesondere invertierbar und das Newton-Verfahren anwendbar ist.

Falls das Ausgleichsproblem tatsächlich ein lösbares Gleichungssystem ist (wenn also in der obigen Interpretation keine Messfehler vorliegen), so gilt  $f(x^*) = 0$ ; das Problem heißt dann *kompatibel*. Im diesem Fall gilt

$$DG(x^*) = Df(x^*)^T Df(x^*).$$

Auch wenn Kompatibilität ein Idealfall ist und in der Praxis kaum auftritt, so werden wir doch vereinfachend annehmen, dass  $f(x^*)$  für  $x$  nahe bei  $x^*$  nahe bei Null liegt, also

$$DG(x) \approx Df(x)^T Df(x)$$

gilt. Auf Basis dieser informellen Überlegung ersetzen wir in der Iterationsvorschrift (6.8) die Ableitung  $DG(x^{(i)})$  durch  $Df(x^{(i)})^T Df(x^{(i)})$ . Damit vermeiden wir die Verwendung der zweiten Ableitung und erhalten

$$Df(x^{(i)})^T Df(x^{(i)})\Delta x^{(i)} = G(x^{(i)}) = Df(x^{(i)})^T f(x^{(i)}), \quad i = 0, 1, 2, \dots \quad (6.9)$$

Dies sind gerade die Normalgleichungen zu dem linearen Ausgleichsproblem

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

mit der Lösung

$$\Delta x^{(i)} = Df(x^{(i)})^+ f(x^{(i)}),$$

die wir für die Iteration nun verwenden. Insgesamt führt dies auf den folgenden Algorithmus.

**Algorithmus 6.23 (Gauß-Newton-Verfahren)**

Gegeben sei eine Funktion  $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ , ihre Ableitung  $Df : \mathbb{R}^n \rightarrow \mathbb{R}^{m \times n}$  sowie ein Anfangswert  $x^{(0)} \in \mathbb{R}^n$  und eine gewünschte Genauigkeit  $\varepsilon > 0$ . Setze  $i := 0$ .

- (1) Löse das lineare Ausgleichsproblem  $\|Df(x^{(i)})\Delta x^{(i)} - f(x^{(i)})\|_2^2 = \min$  und berechne  $x^{(i+1)} = x^{(i)} - \Delta x^{(i)}$
- (2) Falls  $\|\Delta x^{(i)}\| < \varepsilon$ , beende den Algorithmus, ansonsten setze  $i := i + 1$  und gehe zu (1)

□

**Bemerkung 6.24** Ebenso, wie wir im Newton–Verfahren eine Folge von linearen Gleichungssystemen zur Lösung des nichtlinearen Gleichungssystems lösen müssen, lösen wir hier eine Folge linearer Ausgleichsprobleme zur Lösung des nichtlinearen Ausgleichsproblems.  $\square$

Der folgende Satz, der eine Verallgemeinerung des Konvergenzsatzes 6.15 des Newton–Verfahrens darstellt, zeigt, dass der Algorithmus trotz der oben gemachten Vereinfachungen konvergiert — allerdings wird nur bei kompatiblen Problemen quadratische Konvergenz erreicht.

**Satz 6.25** Sei  $D \subset \mathbb{R}^n$  eine offene und konvexe Menge und sei  $f : D \rightarrow \mathbb{R}^m$  mit  $m > n$  eine stetig differenzierbare Funktion, deren Jacobi–Matrix  $Df(x)$  für alle  $x \in D$  vollen Rang habe. Es sei  $x^* \in D$  eine Lösung des Ausgleichsproblems  $\|f(x)\|_2^2 = \min$  in  $D$ . Für ein  $\omega > 0$  gelte die folgende *affin-invariante Lipschitz-Bedingung*

$$\|Df(x)^+(Df(x+sv) - Df(x))v\| \leq s\omega\|v\|^2$$

für alle  $s \in [0, 1]$ , alle  $x \in D$  und alle  $v \in \mathbb{R}^n$  mit  $x+v \in D$ . Des Weiteren gelte für ein  $0 \leq \kappa^* < 1$  die Ungleichung

$$\|Df(x)^+f(x^*)\| \leq \kappa^*\|x - x^*\| \quad (6.10)$$

für alle  $x \in D$ .

Dann gilt: Für alle Anfangswerte  $x^{(0)} \in \mathbb{R}^n$  mit

$$\rho := \|x^* - x^{(0)}\| < \frac{2(1 - \kappa^*)}{\omega} \quad \text{und} \quad B_\rho(x^*) \subseteq D$$

bleibt die durch das Gauß–Newton–Verfahren definierte Folge  $x^{(i)}$  für  $i > 0$  im Ball  $B_\rho(x^*)$  und konvergiert gegen  $x^*$ , d.h.

$$\|x^{(i)} - x^*\| < \rho \quad \text{für alle } i > 0 \quad \text{und} \quad \lim_{i \rightarrow \infty} x^{(i)} = x^*.$$

Die Konvergenzordnung lässt sich dabei abschätzen durch

$$\|x^{(i+1)} - x^*\| \leq \left( \frac{\omega}{2} \|x^{(i)} - x^*\| + \kappa^* \right) \|x^{(i)} - x^*\| = \frac{\omega}{2} \|x^{(i)} - x^*\|^2 + \kappa^* \|x^{(i)} - x^*\|,$$

d.h. das Verfahren konvergiert lokal linear. Falls das Problem kompatibel ist, gilt  $\kappa^* = 0$  und das Verfahren konvergiert lokal quadratisch. Darüberhinaus folgt aus den angegebenen Bedingungen, dass  $x^*$  die eindeutige Lösung in  $B_{2(1-\kappa^*)/\omega}(x^*)$  ist.

**Beweis:** Der Beweis verläuft analog zum Beweis für das Newton–Verfahren (Satz 6.15). Wie dort erhält man aus der affin-invarianten Lipschitz–Bedingung die Ungleichung

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

für alle  $x, y \in D$ .

Da die Jacobi-Matrix  $Df$  auf  $D$  nach Annahme vollen Rang besitzt, ist die Pseudoinverse  $Df(x)^+ = (Df(x)^T Df(x))^{-1} Df(x)^T$  wohldefiniert und es folgt

$$Df(x)^+ Df(x) = (Df(x)^T Df(x))^{-1} Df(x)^T Df(x) = \text{Id}_{\mathbb{R}^n}$$

für all  $x \in D$ . Zudem gilt für die Lösung  $x^*$  des Ausgleichsproblems  $0 = Dg(x^*) = 2Df(x^*)^T f(x^*)$  und daher auch

$$Df(x^*)^+ f(x^*) = (Df(x^*)^T Df(x^*))^{-1} Df(x^*)^T f(x^*) = 0.$$

(Dies zeigt nebenbei, dass die Ungleichung (6.10) für  $x = x^*$  für alle  $\kappa^* \geq 0$  erfüllt ist und somit nur für  $x \neq x^*$  eine echte Bedingung ist.)

Damit erhalten wir

$$\begin{aligned} x^{(i+1)} - x^* &= x^{(i)} - x^* + \Delta x^{(i)} = x^{(i)} - x^* - Df(x^{(i)})^+ f(x^{(i)}) \\ &= Df(x^{(i)})^+ (f(x^*) - f(x^{(i)}) - Df(x^{(i)})(x^* - x^{(i)})) - Df(x^{(i)})^+ f(x^*). \end{aligned}$$

Aus den Voraussetzungen des Satzes folgt also

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

Für  $\|x^{(i)} - x^*\| \leq \rho$  folgt daraus

$$\|x^{(i+1)} - x^*\| \leq \left( \frac{\omega}{2} \rho + \kappa^* \right) \|x^{(i)} - x^*\|$$

mit

$$k := \frac{\omega}{2} \rho + \kappa^* < \frac{\omega}{2} \frac{2(1 - \kappa^*)}{\omega} + \kappa^* = 1.$$

Mittels Induktion ergibt sich damit

$$\|x^{(i)} - x^*\| \leq k^i \|x^{(0)} - x^*\| < \rho,$$

weswegen die Folge für alle  $i \geq 1$  in  $B_\rho(x^*)$  bleibt und wegen  $k^i \rightarrow 0$  für  $i \rightarrow \infty$  gegen  $x^*$  konvergiert.

Für kompatible Probleme gilt  $f(x^*) = 0$  und damit  $\kappa^* = 0$  und quadratische Konvergenz. Die Eindeutigkeit folgt analog zum Beweis von Satz 6.15.  $\square$

**Bemerkung 6.26** Der Parameter  $\kappa^*$  kann als Maß für die ‘‘Nichtkompatibilität’’ des nicht-linearen Ausgleichsproblems aufgefasst werden. Beweistechnisch ist leicht einzusehen, dass  $\kappa^* < 1$  notwendig für die Konvergenz ist, da dieses  $\kappa^*$  gerade den Faktor des linearen Konvergenzanteils ausmacht. Eine genaue Analyse des Verfahrens mit statistischen Methoden zeigt jedoch auch, dass für  $\kappa^* \geq 1$  die Lösung  $\tilde{x} = x + \Delta x$  zu gestörten Messwerten  $\tilde{z} = z + \Delta z$  durch beliebig kleine Störungen  $\|\Delta z\|$  unbeschränkt verändert werden kann, das Problem für  $\kappa^* \geq 1$  also extrem schlecht konditioniert ist. Falls das Verfahren also in diesem Falle trotzdem konvergiert, so ist die erhaltene Lösung mit großer Wahrscheinlichkeit praktisch unbrauchbar.  $\square$





# Literaturverzeichnis

- [1] BRYAN, K. ; LEISE, T.: The \$25,000,000,000 eigenvector: the linear algebra behind Google. In: *SIAM Rev.* 48 (2006), Nr. 3, S. 569–581. – ISSN 0036–1445
- [2] DEUFLHARD, P. ; HOHMANN, A.: *Numerische Mathematik. I: Eine algorithmisch orientierte Einführung.* 3. Auflage. Berlin : de Gruyter, 2002
- [3] HALL, C. A. ; MEYER, W. W.: Optimal Error Bounds for Cubic Spline Interpolation. In: *J. Approx. Theory* 16 (1976), S. 105–122
- [4] KLOEDEN, P. E.: *Einführung in die Numerische Mathematik.* Vorlesungsskript, J.W. Goethe–Universität Frankfurt am Main, 2002. – Erhältlich im WWW unter [http://www.math.uni-frankfurt.de/~numerik/lehre/Vorlesungen/EinN\\_2007/](http://www.math.uni-frankfurt.de/~numerik/lehre/Vorlesungen/EinN_2007/)
- [5] LEMPIO, F.: *Numerische Mathematik I: Methoden der Linearen Algebra.* Bayreuther Mathematische Schriften, Band 51, 1997
- [6] LEMPIO, F.: *Numerische Mathematik II: Methoden der Analysis.* Bayreuther Mathematische Schriften, Band 56, 1998
- [7] OEVEL, W.: *Einführung in die Numerische Mathematik.* Spektrum Verlag, Heidelberg, 1996
- [8] SCHWARZ, H. R. ; KÖCKLER, N.: *Numerische Mathematik.* 5. Auflage. Stuttgart : B. G. Teubner, 2004
- [9] STOER, J.: *Numerische Mathematik I.* 9. Auflage. Springer Verlag, Heidelberg, 2005

# Index

- absoluter Fehler, 15
- affin-invariante Lipschitz-Bed., 103, 112
- Aufwandsabschätzung, 26–29, 37–38
- Ausgleichsrechnung
  - linear, 6–7, 25
  - nichtlinear, 110
- Auslöschung, 18
  
- Banach'scher Fixpunktsatz, 30, 89
- baryzentrische Koordinaten, 58
- BFGS-Verfahren, 105
- Bisektionsverfahren, 92
  
- charakteristisches Polynom, 41
  
- Defekt, 13
- Dreiecksmatrix
  - obere, 7
  - untere, 10
  
- Eigenvektor, 41
- Eigenwert, 41
- Einzelschrittverfahren, 34
- Eliminationsverfahren, 9
  - mit Pivotierung, 9
  - mit Pivotsuche, 18
  
- Fehler
  - absolut, 15
  - relativ, 15
- Fixpunkt, 89
- Fixpunktiteration, 90
  
- Gauß'sches Eliminationsverfahren, 9
  - mit Pivotierung, 9
  - mit Pivotsuche, 18
- Gauß-Legendre-Regel, 85, 87
- Gauß-Newton-Verfahren, 111
- Gauß-Quadratur, 84
  - Fehler, 87
  - Wahl der Gewichte, 86
- Gauß-Seidel-Verfahren, 34
- Gesamtschrittverfahren, 33
- Gewichte, 77
- Gewichtsfunktion, 64, 85
- Gleichungssystem
  - gestört, 13
  - linear, 5
  - nichtlinear, 89
- Gleitkommaoperationen, 26
- Grad eines Polynoms, 56
  
- Hermite-Polynome, 87
- Hessenberg-Matrix, 52
- Householder-Algorithmus, 23
  
- induzierte Matrixnorm, 13
- Integration, 77
  - Fehler, 78, 83
- Interpolation
  - Fehler, 55, 62, 78
  - für Splines, 75
  - von Daten, 55
  - von Funktionen, 55
- inverse power iteration, 43
- inverse Vektoriteration, 43
- iterative Verfahren, 29
  
- Jacobi-Verfahren, 33
  
- Koeffizienten
  - Polynom, 56
- Kondition
  - der Polynominterpolation, 60
  - einer Matrix, 15
  - nichtlineares Ausgleichsproblem, 113
  - numerisch, 19
- Konvergenz
  - linear, 95
  - quadratisch, 95

- superlinear, 95
- Konvergenzordnung
  - Anzahl korrekter Stellen, 97
  - Definition, 95
  - grafische Darstellung, 97
  - Übersicht, 98
- Lagrange–Polynome, 57, 78
  - effiziente Implementierung, 58
- Laguerre–Polynome, 87
- Lebesgue–Konstante, 61
- Legendre–Polynome, 87
- lineare Konvergenz, 95
- lineares Gleichungssystem, 5
- lokale Konvergenz, 91
- LR–Faktorisierung, 10
- LR–Zerlegung, 10
- maschinendarstellbare Zahlen, 12
- Matrixnorm, 13
  - induziert, 13
- Methode der kleinsten Quadrate, 7
- Milne–Regel, 83
- Newton–Cotes–Formeln, 77
  - Fehler, 78
  - zusammengesetzt, 82
    - Fehler, 83
- Newton–Verfahren
  - eindimensional, 99
  - mehrdimensional, 101, 102
- nichtlineares Ausgleichsproblem, 110
- nichtlineares Gleichungssystem, 89
- Normalengleichungen, 7
- Normen, 13
- numerische Kondition, 19
- obere Dreiecksmatrix, 7
- Ordnung
  - Aufwand, 29
  - Konvergenz,  $\rightarrow$  Konvergenzordnung
- orthogonale Matrizen, 19
- orthogonale Polynome, 64
  - Rekursionsgleichung, 65
- Penrose–Axiome, 109
- Permutationsmatrix, 11
- Pivotelement, 10
- Pivotierung, 9
- Pivotsuche, 18
- Polynominterpolation, 56,  $\rightarrow$  Interpolation
- power iteration, 42
  - inverse, 43
- Präkonditionierung, 19
- Pseudoinverse, 109
- QR–Algorithmus, 48
  - mit Shift, 52
- QR–Faktorisierung,  $\rightarrow$  QR–Zerlegung
- QR–Zerlegung, 23
  - für Ausgleichsrechnung, 25
- quadratische Konvergenz, 95
- Quadratur, 77
- Quadratwurzelberechnung, 89
- Rayleigh’scher Quotient, 42
- relativer Fehler, 15
- Relaxationsverfahren, 38
- Residuum, 13
- Rückwärtseinsetzen, 7
- Rundungsfehler, 12
- Runge–Funktion, 64, 70
- schlecht konditionierte Matrizen, 16
- schwach besetzte Matrix, 38
- Sekanten–Verfahren, 106
- Shift–Strategien, 52
- Simpson–Regel, 83
- Skalarprodukt
  - für Polynome, 64
- SOR–Verfahren, 39
- Spaltensummennorm, 14
- Spektralnorm, 14
- Spline, 70
  - kubisch, 72
  - Randbedingungen, 72
- Splineinterpolation, 70
- Steffensen–Verfahren, 105
- stückweise Polynome, 70
- Stützstellen, 55, 77
- superlineare Konvergenz, 95
- Trapez–Regel, 83
- Tridiagonalmatrix, 46
- Tschebyscheff–Knoten, 67

Tschebyscheff-Polynome, 67, 87

überbestimmtes Gleichungssystem, 6  
untere Dreiecksmatrix, 10

Vektoriteration, 42

    inverse, 43

Vektornormen, 13

von Mises-Iteration, 42

Vorwärtseinsetzen, 10

Zeilensummennorm, 14