Numerische Mathematik I

Lars Grüne
Mathematisches Institut
Fakultät für Mathematik und Physik
Universität Bayreuth
95440 Bayreuth
lars.gruene@uni-bayreuth.de
www.math.uni-bayreuth.de/∼lgruene/

Vorlesungsskript

Dritte Auflage

Wintersemester 2007/2008

Vorwort

Dieses Skript ist im Rahmen einer gleichnamigen Vorlesung entstanden, die ich im Wintersemester 2007/2008 an der Universität Bayreuth gehalten habe. Es ist die dritte Auflage einer gleichnamigen Ausarbeitung aus dem Wintersemester 2002/2003, die gegenüber der zweiten Auflage korrigiert, aktualisiert und in einigen Punkten erweitert wurde. Eine elektronische Version dieses Skripts sowie die zu dieser Vorlesung gehörigen Übungsaufgaben sind im WWW auf der Seite `http://www.math.uni-bayreuth.de/~lgruene/` unter dem Link `Lehrveranstaltungen` erhältlich.

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 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, März 2008 Lars Grüne

Inhaltsverzeichnis

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[1], daher der Name „Numerische Mathematik“.

In der Grundvorlesung zur Numerik werden traditionell viele verschidedene 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 Mathematik II“ im 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.

[1] 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 Problemdaten tatsächlich das richtige Ergebnis berechnet wird. Diese Überprüfung soll natürlich mit mathematischen Methoden durgeführt 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 $e \in \mathbb{N}_0$ der *Exponent* genannt wird. 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*

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

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 $\mathcal{A} : D \to L$ der Problemdaten 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 $x_1, \ldots, x_n \in \mathbb{R}$ zu bestimmen, für die das Gleichungssystem

$$\begin{array}{rcrcrcrcl} a_{1\,1}x_1 & + & a_{1\,2}x_2 & + & \ldots & + & a_{1\,n}x_n & = & b_1 \\ & & & & & & \vdots & & \\ a_{n\,1}x_1 & + & a_{n\,2}x_2 & + & \ldots & + & a_{n\,n}x_n & = & b_n \end{array} \tag{2.1}$$

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

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

mit

$$A = \begin{pmatrix} a_{1\,1} & a_{1\,2} & \ldots & a_{1\,n} \\ \vdots & \vdots & & \vdots \\ a_{n\,1} & a_{n\,2} & \ldots & a_{n\,n} \end{pmatrix}, \quad x = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \quad \text{und} \quad b = \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix}. \tag{2.3}$$

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.

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

Da die Anzahl n der Gleichungen in (2.1) gleich der Anzahl der Unbekannten $x_1, \ldots, x_n$ 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 Anwendungen linearer Gleichungssysteme

2.1.1 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 $t_1, t_2, \ldots, t_m$ Messwerte $z_1, z_2, \ldots, z_m$ erhalten. Auf Grund von theoretischen Überlegungen (z.B. auf Grund eines zugrundeliegendes physikalisches Gesetzes), kennt man eine Funktion $f(t)$, für die $f(t_i) = z_i$ gelten sollte. Diese Funktion wiederum hängt aber nun von unbekannten Parametern $x_1, \ldots, x_n$ ab; wir schreiben $f(t; x)$ für $x = (x_1, \ldots, x_n)^T$, 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 $\widetilde{A}x = z$ mit

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

ergibt. Diese linearen Gleichungssysteme haben m Gleichungen (eine für jedes Wertepaar (t_i, z_i)) und n Unbekannte (nämlich gerade die unbekannten Parameter x_i), wobei m üblicherweise sehr viel größer als n ist. Man sagt, dass das Gleichungsystem *überbestimmt* ist. Da Messwerte eines Versuchs praktisch immer mit Fehlern behaftet sind, ist es sicherlich zu optimistisch, anzunehmen, dass das Gleichungssystem $\widetilde{A}x = z$ 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 $\widetilde{A}x = z$ nicht lösbar ist, wollen wir zumindest ein x finden, so dass $\widetilde{A}x$ 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 $x = (x_1, \ldots, x_n)^T$, so dass $\varphi(x) := \|z - \widetilde{A}x\|^2$ 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 φ zu minimieren, setzten wir den Gradienten $\nabla\varphi(x)$ gleich Null. Beachte dazu, dass der Gradient der Funktion $g(x) := \|f(x)\|^2$ für beliebiges $f : \mathbb{R}^n \to \mathbb{R}^n$ durch $\nabla g(x) = 2Df(x)^T f(x)$ gegeben ist. Wir erhalten also

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

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

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

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

2.1.2 Diskrete L_2–Approximation

Auch bei diesem Problem geht es darum, Parameter einer Funktion zu bestimmen, um eine möglichst gute Approximation zu erhalten. Während wir beim Ausgleichsproblem aber den Abstand zwischen den Funktionswerten $f(t_j)$ und den Messwerten z_j klein halten wollten, geht es nun darum, Parameter so zu wählen, dass das Integral einer Funktion möglichst gut approximiert wird. Sei dazu $z : [c, d] \to \mathbb{R}$ eine integrierbare Funktion auf einem Intervall $[c, d]$. Für vorgegebene integrierbare Funktionen $v_1, \ldots, v_n : [c, d] \to \mathbb{R}$ wollen wir einen Parametervektor $x = (x_1, \ldots, x_n)^T \in \mathbb{R}^n$ bestimmen, so dass das Integral

$$\varphi(x) := \int_c^d |z(t) - \sum_{i=1}^{n} x_i v_i(t)|^2 dt$$

minimal wird.

Wiederum berechnet man hier den Gradienten $\nabla\varphi$ und leitet so geeignete *Normalengleichungen* her, die auch hier auf ein lineares Gleichungssystem $Ax = b$ führen, wobei die Einträge a_{ij} von A und b_i von b gegeben sind durch

$$a_{ij} = \int_c^d v_j(t)v_i(t)dt \quad \text{und} \quad b_i = \int_c^d z(t)v_i(t)dt.$$

2.1.3 Randwertaufgaben gewöhnlicher Differentialgleichungen

Differentialgleichungen sind Gleichungen, bei denen eine unbekannte differenzierbare Funktion dadurch charakterisiert ist, dass ihre Werte mit ihren Ableitungen in Beziehung gesetzt ist. Wir werden diese erst im zweiten Teil dieser Vorlesung genauer betrachten, wollen hier aber kurz ein spezielles Problem skizzieren, das wiederum auf ein lineares Gleichungssystem führt.

Gegeben sei ein Intervall $[c, d]$, reelle Zahlen y_c, y_d und eine stetige Funktion $g : [c, d] \to \mathbb{R}$. Gesucht ist eine differenzierbare Funktion $y : [c, d] \to \mathbb{R}$, welche die Gleichungen

$$\begin{aligned} y''(t) + y'(t) + y(t) &= g(t) \\ y(c) &= y_c \\ y(d) &= y_d \end{aligned}$$

erfüllt. Zur Lösung des Problems kann man die *konsistente Differenzenapproximation* verwenden. Hierbei zerlegt man das Intervall $[c, d]$ in gleichlange Teilintervalle

$$c = t_0 < t_1 < \ldots < t_N = d$$

mit $t_i - t_{i-1} = h := (d - c)/N$ für ein $N \in \mathbb{N}$, also $t_i = c + i(d - c)/N$, und ersetzt die Ableitungen y' und y'' an den Punkten t_i durch Differenzenquotienten

$$y'(t_i) \approx \frac{y(t_{i+1}) - y(t_{i-1})}{2h}, \quad y''(t_i) \approx \frac{y(t_{i+1}) - 2y(t_i) + y(t_{i-1})}{h^2}.$$

Mit der Schreibweise $\eta_i = y(t_i)$ und $g_i = g(t_i)$ erhalten wir damit aus den obigen Gleichungen für die inneren Gitterpunkte die Differenzengleichungen

$$\frac{\eta_{i+1} - 2\eta_i + \eta_{i-1}}{h^2} + \frac{\eta_{i+1} - \eta_{i-1}}{2h} + \eta_i = g_i, \quad i = 1, \ldots, N-1$$

und für die Randpunkte die Gleichungen

$$\eta_0 = y_c, \quad \eta_N = y_d$$

aus den Randbedingungen. Insgesamt erhalten wir so das lineare Gleichungssystem

$$A_h \eta = b$$

mit

$$A_h = \begin{pmatrix} 1 & & & & \\ \frac{1}{h^2} - \frac{1}{2h} & -\frac{2}{h^2} + 1 & \frac{1}{h^2} + \frac{1}{2h} & & \\ & \ddots & \ddots & \ddots & \\ & & \frac{1}{h^2} - \frac{1}{2h} & -\frac{2}{h^2} + 1 & \frac{1}{h^2} + \frac{1}{2h} \\ & & & & 1 \end{pmatrix} \quad \text{und} \quad b = \begin{pmatrix} y_c \\ g_1 \\ \vdots \\ g_{N-1} \\ y_d \end{pmatrix}.$$

Die Einträge $\eta_0, \ldots, \eta_n$ von η liefern dann eine Approximation der Funktionswerte $y(t_0)$, $\ldots$, $y(t_N)$.

2.1.4 Weitere Anwendungen

Es gibt 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.,

$$A = \begin{pmatrix} a_{1\,1} & a_{1\,2} & \dots & a_{1\,n} \\ 0 & a_{2\,2} & \dots & a_{2\,n} \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & a_{n\,n} \end{pmatrix}.$$

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

$$x_n = \frac{b_n}{a_{n\,n}}, \quad x_{n-1} = \frac{b_{n-1} - a_{n-1\,n}x_n}{a_{n-1\,n-1}}, \dots, \quad x_1 = \frac{b_1 - a_{1\,2}x_2 - \dots - a_{1\,n}x_n}{a_{1\,1}}$$

oder, kompakt geschrieben,

$$x_i = \frac{b_i - \sum_{j=i+1}^{n} a_{i\,j}x_j}{a_{i\,i}}, \qquad i = n,\, n-1, \dots,\, 1 \tag{2.5}$$

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

Die Idee des Gauß'schen Eliminationsverfahrens liegt nun darin, das Gleichungssystem $Ax = b$ in ein Gleichungssystem $\widetilde{A}x = \tilde{b}$ umzuformen, so dass die Matrix $\widetilde{A}$ 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 $\widetilde{A} = A_3$ und $\tilde{b} = b_3$. Rückwärtseinsetzen gemäß (2.5) liefert dann

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

□

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 $b \in \mathbb{R}^n$.

(0) Setze $i = n$ und $j = 1$ (Zeilen– und Spaltenindex des zu eliminierenden Eintrags)

(1) Subtrahiere a_{ij}/a_{jj}–mal die j–te Zeile von der i–ten Zeile:
Setze $\alpha := a_{ij}/a_{jj}$ und berechne

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

(2) Falls $i \geq j + 2$ ist, setze $i := i - 1$ und fahre fort bei (1), sonst:
Falls $j \leq n - 2$ ist, setze $j := j + 1$ und $i := n$ und fahre fort bei (1), sonst:
Ende des Algorithmus

□

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 $b \in \mathbb{R}^n$.

(0) Setze $i = n$ und $j = 1$ (Zeilen– und Spaltenindex des zu eliminierenden Eintrags)

(1a) Falls $a_{ij} = 0$ gehe zu (2), sonst:
Falls $a_{jj} = 0$ vertausche a_{jk} und a_{ik} für $k = j, \ldots, n$ sowie b_i und b_j und gehe zu (1b)

(1b) Subtrahiere a_{ij}/a_{jj}–mal die j–te Zeile von der i–ten Zeile:
Setze $\alpha := a_{ij}/a_{jj}$ und berechne

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

(2) Falls $i \geq j + 2$ ist, setze $i := i - 1$ und fahre fort bei (1a), sonst:
Falls $j \leq n - 2$ ist, setze $j := j + 1$ und $i := n$ und fahre fort bei (1a), sonst:
Ende des Algorithmus

□

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 und das Choleski–Verfahren

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*

$$L = \begin{pmatrix} l_{1\,1} & 0 & \dots & 0 \\ \vdots & \ddots & \ddots & \vdots \\ l_{n-1\,1} & \dots & l_{n-1\,n-1} & 0 \\ l_{n\,1} & l_{n\,2} & \dots & l_{n\,n} \end{pmatrix}.$$

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:

$$y_i = \frac{b_i - \sum_{j=1}^{i-1} l_{i\,j} y_j}{l_{i\,i}}, \qquad i = 1, 2, \dots, n \tag{2.6}$$

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 – α–mal j–te Zeile“ lässt sich durch Multiplikation von links mit der Matrix

$$\begin{pmatrix} 1 & & & & & \\ & \ddots & & & & \\ & & \ddots & & & \\ & -\alpha & & \ddots & & \\ & & & & \ddots & \\ & & & & & 1 \end{pmatrix}$$

erzielen, hierbei steht α 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 $F = F_k \cdot F_{k-1} \cdots F_1$ aufmultipliziert, so erhält man $A = LR$ mit $L = F^{-1}$, also die LR–Faktorisierung (beachte, dass die F–Matrix als Produkt unterer Dreiecksmatrizen eine untere Dreiecksmatrix ist, weswegen auch ihre Inverse eine untere Dreiecksmatrix ist).

Sind allerdings wegen der Pivotierung Zeilenvertauschungen nötig, so lässt sich i.A. keine LR–Zerlegung erhalten.[1] Die Vertauschung der i–ten und der j–ten Zeile in A lässt sich durch die Multiplikation von links mit Matrizen der Form

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

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 $P_1, \ldots, P_l$ ebenfalls zu einer Matrix $P = P_l \cdots P_1$ aufmultipizieren und erhält dann eine Zerlegung der Form $PA = LR$ mit $L = F^{-1}$ von oben (P nennt man *Permutationsmatrix*).

Wir wollen auf die Details dieser Erweiterung der Gauß–Elimination hier nicht näher eingehen (sie finden sich in jedem einführenden Numerik Lehrbuch, z.B. in den Büchern von Deuflhard/Hohmann [2], Schwarz/Köckler [8] oder Stoer [9]), sondern statt dessen ein anderes Verfahren zur LR–Zerlegung betrachten, das *Choleski–Verfahren*. Dieses funktioniert zwar nicht für allgemeine Matrizen, sondern nur für *symmetrische, positiv definite* Matrizen, liefert dafür aber eine besonders schöne Form der LR–Faktorisierung.

Definition 2.4 (i) Eine Matrix $A \in \mathbb{R}^{n\times n}$ heißt *symmetrisch*, falls $A^T = A$ ist.

(ii) Eine Matrix $A \in \mathbb{R}^{n\times n}$ heißt *positiv definit*, falls $\langle x, Ax\rangle > 0$ ist für alle Vektoren $x \in \mathbb{R}^n$ mit $x \neq (0 \ldots 0)^T$.

Hierbei bezeichnet $\langle x, y\rangle$ das *euklidische* oder *Standard–Skalarprodukt* im $\mathbb{R}^n$, d.h., für Vektoren $x, y \in \mathbb{R}^n$ ist

$$\langle x, y\rangle = x^T y = \sum_{i=1}^{n} x_i y_i.$$

□

Diese Bedingung ist natürlich recht einschränkend; trotzdem ist sie in vielen Anwendungen erfüllt. So führt z.B. das Ausgleichsproblem i.A. auf ein Gleichungssystem mit symmetrischer und positiv definiter Matrix A.

Für solche Matrizen kann man zeigen, dass immer eine LR–Zerlegung existiert, die zusätzlich die schöne Form $R = L^T$ besitzt, insbesondere genügt es also, die untere Dreiecksmatrix L zu berechnen. Der Beweis folgt dabei konstruktiv aus dem folgenden Algorithmus.

[1] tatsächlich existiert für allgemeine invertierbare Matrizen im Allgemeinen keine LR–Zerlegung

Algorithmus 2.5 (Choleski–Verfahren) Gegeben sei eine Matrix $A \in \mathbb{R}^{n\times n}$.

(0) Setze $i = 1$ und $j = 1$ (Zeilen– und Spaltenindex des aktuellen Eintrags l_{ij} von L)

(1) (a) Falls $i > j$ setze

$$l_{ij} = \frac{a_{ij} - \sum_{k=1}^{j-1} l_{ik} l_{jk}}{l_{jj}}$$

(b) Falls $i = j$ setze

$$l_{ii} = \sqrt{a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2}$$

(c) Falls $i < j$ setze

$$l_{ij} = 0$$

(2) Falls $i \leq n-1$ ist, setze $i := i+1$ und fahre fort bei (1), sonst:
Falls $j \leq n-1$ ist, setze $j := j+1$ und $i := 1$ und fahre fort bei (1), sonst:
Ende des Algorithmus

□

Offenbar ist dieser Algorithmus nicht so anschaulich wie die Gauß–Elimination. Um zu beweisen, dass dieser Algorithmus das richtige Ergebnis liefert, schreiben wir die Gleichung $A = LL^T$ explizit auf und lösen nach L auf. Dies ist jedoch direkt für beliebige Dimensionen sehr unübersichtlich, weswegen wir per Induktion über die Dimension der Matrix A vorgehen. Zum **Induktionsanfang** betrachten wir zunächst 2×2 Matrizen. Hier ergibt sich

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} = \begin{pmatrix} l_{11} & 0 \\ l_{21} & l_{22} \end{pmatrix} \begin{pmatrix} l_{11} & l_{21} \\ 0 & l_{22} \end{pmatrix} = \begin{pmatrix} l_{11}^2 & l_{11} l_{21} \\ l_{21} l_{11} & l_{21}^2 + l_{22}^2 \end{pmatrix}. \tag{2.7}$$

Daraus erhalten wir

$$\begin{aligned} l_{11} &= \sqrt{a_{11}} \\ l_{21} &= a_{21}/l_{11} \\ l_{22} &= \sqrt{a_{22} - l_{21}^2} \end{aligned}$$

was genau den Berechnungen im Algorithmus 2.5 für $i = 1, 2$ und $j = 1, 2$ entspricht. Die erste und letzte Gleichung sind dabei reell lösbar, denn weil A positiv definit ist, gilt $a_{11} > 0$ und

$$a_{22} - l_{21}^2 = a_{22} - \frac{a_{21}^2}{l_{11}^2} = a_{22} - \frac{a_{21}^2}{a_{11}} = \frac{\det(A)}{a_{11}} > 0.$$

Für größere Matrizen gehen wir per Induktion vor. Wir schreiben

$$A = \begin{pmatrix} A_{n-1} & \bar{a}_n \\ \bar{a}_n^T & a_{nn} \end{pmatrix}$$

Falls A symmetrisch und positiv definit ist, so hat auch die Matrix A_{n-1} diese Eigenschaft. Als **Induktionsannahme** nehmen wir an, dass der Choleski–Algorithmus 2.5 für die $(n-1) \times (n-1)$–Matrix A_{n-1} die LR Faktorisierung der Form $A_{n-1} = L_{n-1}L_{n-1}^T$ korrekt berechnet.

Wir schreiben die gesuchte untere Dreiecksmatrix L nun als

$$L = \begin{pmatrix} L_{n-1} & 0 \\ \bar{l}_n^T & l_{n\,n} \end{pmatrix}$$

mit $\bar{l}_n = (l_{n\,1} \ldots l_{n\,n-1})^T$.

Zum **Induktionsschluss** müssen wir nun die Gleichungen im Choleski–Algorithmus für $i = n$ und $j = 1, \ldots, n$ überprüfen (die Gleichungen für $i \leq n-1$ liefern nach der Induktionsannahme bereits die richtige Matrix L_{n-1}). Dazu betrachten wir die Gleichung $LL^T = A$, die sich mit der obigen Notation als

$$\begin{pmatrix} A_{n-1} & \bar{a}_n \\ \bar{a}_n^T & a_{n\,n} \end{pmatrix} = \begin{pmatrix} L_{n-1} & 0 \\ \bar{l}_n^T & l_{n\,n} \end{pmatrix} \begin{pmatrix} L_{n-1}^T & \bar{l}_n \\ 0 & l_{n\,n} \end{pmatrix} = \begin{pmatrix} L_{n-1}L_{n-1}^T & L_{n-1}\bar{l}_n \\ (L_{n-1}\bar{l}_n)^T & \bar{l}_n^T\bar{l}_n + l_{n\,n}^2 \end{pmatrix}. \tag{2.8}$$

schreiben lässt. Bestimmt man nun die Einträge im Vektor $\bar{l}_n$ durch Vorwärtseinsetzen aus dem Gleichungssystem $L_{n-1}\bar{l}_n = \bar{a}_n$, so erhält man gerade die Gleichungen für $l_{n,j}$, $j = 1, \ldots, n-1$ aus Algorithmus 2.5. Löst man dann noch die Gleichung $\bar{l}_n^T\bar{l}_n + l_{n\,n}^2 = a_{n\,n}$, so ergibt sich auch die Gleichung für $j = n$ in Algorithmus 2.5. Dass diese Gleichung rell lösbar ist, folgt aus $\det(A) = \det(L)^2$ (wegen $A = LL^T$), $\det(A_{n-1}) = \det(L_{n-1})^2$ (wegen $A_{n-1} = L_{n-1}L_{n-1}^T$) und $\det(L)^2 = \det(L_{n-1})^2 l_{n\,n}^2$ (wegen der Form von L), womit

$$l_{n\,n}^2 = \det(L)^2 / \det(L_{n-1})^2 = \det(A) / \det(A_{n-1})$$

wegen der positiven Definitheit von A (und damit auch von A_{n-1}) reell und positiv ist.

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“ Gleichungssystems

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

ein, für das $\tilde{x}$ gerade die exakte Lösung ist. Der Vektor Δb 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 Residuum $\|\Delta b\|$ 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 ∞*–Norm*

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

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 2.6 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 2.7 Für die zu den obigen Vektornormen gehörigen induzierten Matrixnormen und $A \in \mathbb{R}^{n\times n}$ gelten die folgenden Gleichungen

$$\begin{aligned}
\|A\|_1 &= \max_{j=1,\dots,n} \sum_{i=1}^n |a_{i\,j}| && \text{(Spaltensummennorm)}\\
\|A\|_\infty &= \max_{i=1,\dots,n} \sum_{j=1}^n |a_{i\,j}| && \text{(Zeilensummennorm)}\\
\|A\|_2 &= \sqrt{\rho(A^TA)} && \text{(Spektralnorm),}
\end{aligned}$$

wobei $\rho(A^TA)$ den maximalen Eigenwert der symmetrischen Matrix A^TA bezeichnet.

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

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

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

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

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

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

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

woraus, da x beliebig war, die Ungleichung

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

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

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

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^TA 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 $i \ne j$. 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 λ_i die zu den v_i gehörigen Eigenwerte von A^TA und sei λ_{i^*} der maximale Eigenwert, also $\lambda_{i^*} = \rho(A^TA)$. Dann gilt

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

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

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

Andererseits gilt die Ungleichung

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

□

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

Definition 2.8 Für eine gegebene Matrixnorm $\|\cdot\|_p$ ist die *Kondition* einer invertierbaren Matrix A (bzgl. $\|\cdot\|_p$) definiert durch

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

□

Wenn wir das Verhältnis zwischen dem Fehler Δx und dem Residuum Δb 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 anschätzen kann.

Satz 2.9 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

$$\|\Delta x\|_p \leq \|A^{-1}\|_p\|\Delta b\|_p \tag{2.9}$$

und

$$\frac{\|\Delta x\|_p}{\|x\|_p} \leq \mathrm{cond}_p(A)\frac{\|\Delta b\|_p}{\|b\|_p}. \tag{2.10}$$

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)

$$\|Cy\|_p \leq \|C\|_p \|y\|_p. \tag{2.11}$$

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 (2.11) mit $C = A^{-1}$ und $y = \Delta b$ benutzt haben. Dies zeigt (2.9). Aus (2.11) 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 (2.9) anwenden, erhalten wir

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

also (2.10). □

Für Matrizen, deren Kondition $\mathrm{cond}_p(A)$ 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.

Ein wichtiges Kriterium beim Entwurf eines Lösungsverfahrens ist es, dass das Verfahren auch für schlecht konditionierte Matrizen noch zuverlässig funktioniert. Beim Gauß–Verfahren kann z.B. die Auswahl der Pivotelemente so gestaltet werden, dass sich schlechte Konditionierung weniger stark auswirkt.

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 $j, \ldots, n$ gilt. Dies gibt uns die Freiheit, den *Zeilen*index „j“ durch einen beliebigen Index $p \in \{j, \ldots, n\}$ zu ersetzen, was im Algorithmus durch Tauschen der j–ten mit der p–ten Zeile realisiert wird. Beachte, dass das „j“ in „$a_{i\,j}$“ der *Spalten*index des zu eliminierenden Elementes ist, der sich durch die Vertauschung nicht ändert; wir können also durch Zeilenvertauschung nur die Elemente $a_{j\,j}$, $a_{j\,k}$ und b_j oder anders gesagt gerade die Brüche $a_{j\,k}/a_{j\,j}$ und $b_j/a_{j\,j}$ 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“ gennant). 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, dass man *Pivotsuche* nennt. Da wir nur die Brüche $a_{j\,k}/a_{j\,j}$ $(k = j, \ldots, n)$ und $b_j/a_{j\,j}$ 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* $a_{p\,j}$ (also den Nenner der Brüche) so zu wählen, dass $|a_{p\,j}|$ maximal wird. Im folgenden Algorithmus 2.10 verwenden wir eine etwas aufwändigere Strategie, bei der auch die Zähler der auftauchenden Brüche berücksichtigt werden.

Algorithmus 2.10 (Gauß–Elimination mit Pivotsuche)
Gegeben sei eine Matrix $A \in \mathbb{R}^{n\times n}$ und ein Vektor $b \in \mathbb{R}^n$.

(0) Setze $i = n$ und $j = 1$ (Zeilen– und Spaltenindex des zu eliminierenden Eintrags)

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

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

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

(1b) Subtrahiere $a_{i\,j}/a_{j\,j}$–mal die j–te Zeile von der i–ten Zeile:
Setze $\alpha := a_{i\,j}/a_{j\,j}$ und berechne

$$a_{i\,k} := a_{i\,k} - \alpha\, a_{j\,k} \text{ für } k = j, \ldots, n, \quad b_i := b_i - \alpha\, b_j$$

(2) Falls $i \geq j + 2$ ist, setze $i := i - 1$ und fahre fort bei (1b), sonst:
Falls $j \leq n - 2$ ist, setze $j := j + 1$ und $i := n$ und fahre fort bei (1a), sonst:
Ende des Algorithmus

□

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 werden immer solche Pivotsuchmethoden verwenden. 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 sogannante *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, das die einzelnen Matrizen L und R der Zerlegung deutlich größere Kondition haben als die zerlegte Matrix A.

Beispiel 2.11 Betrachte die Matrix

$$A = \begin{pmatrix} 0,001 & 0,001 \\ 1 & 2 \end{pmatrix}$$

mit LR–Faktorisierung

$$L = \begin{pmatrix} 1 & 0 \\ 1000 & 1 \end{pmatrix}, \quad R = \begin{pmatrix} 0,001 & 0,001 \\ 0 & 1 \end{pmatrix}.$$

Hier gilt

$$\mathrm{cond}_2(A) \approx 5000, \ \ \mathrm{cond}_2(L) \approx 1000000 \ \text{ und } \ \mathrm{cond}_2(R) \approx 1000$$

die 2–Kondition von L ist also etwa 200–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 Δx 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 2.12 Eine Matrix $Q \in \mathbb{R}^{n\times n}$ heißt *orthogonal*, falls $QQ^T = 1$ ist[2]. □

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 2.13 Sei $A \in \mathbb{R}^{n\times n}$ eine invertierbare Matrix mit QR–Zerlegung. Dann gilt

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

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

$$\mathrm{cond}_2(Q) = \|Q\|_2\|Q^{-1}\|_2 = \|Q\mathrm{Id}\|_2\|Q^{-1}\mathrm{Id}\|_2 = \|\mathrm{Id}\|_2\|\mathrm{Id}\|_2 = 1$$

und

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

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

□

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.

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 2.13 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[3]. Wir veranschaulichen die Idee zunächst geometrisch: Sei $a_{\cdot j} \in \mathbb{R}^n$ die j–te Spalte der Matrix A. Wir wollen eine Spiegelung $H^{(j)}$ finden, die $a_{\cdot j}$ auf einen Vektor der Form $a^{(j)}_{\cdot j} = (\underbrace{*, *, \ldots, *}_{j \text{ Stellen}}, 0)^T$ bringt. Der Vektor soll also in die Ebene $E_j = \text{span}(e_1, \ldots, e_j)$ 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.

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

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

[3]ein Algorithmus auf Basis von Drehungen ist der sogenannte Givens–Algorithmus

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

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

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

Beweis: Falls $v \neq 0$ ist gilt

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

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 $w_{j+1} = \ldots = w_n = 0$ 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 $w_j = w_{j+1} = \ldots = w_n = 0$, so dass die Behauptung mit $H = \mathrm{Id}$ 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

$$A^{(2)} = H^{(1)}A^{(1)} = \begin{pmatrix} a_{1\,1}^{(2)} & a_{1\,2}^{(2)} & \cdots & a_{1\,n}^{(2)} \\ 0 & a_{2\,2}^{(2)} & \cdots & a_{2\,n}^{(2)} \\ \vdots & \vdots & & \vdots \\ 0 & a_{n\,2}^{(2)} & \cdots & a_{n\,n}^{(2)} \end{pmatrix}.$$

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

$$A^{(3)} = H^{(2)}A^{(2)} = \begin{pmatrix} a_{1\,1}^{(3)} & a_{1\,2}^{(3)} & a_{1\,3}^{(3)} & \cdots & a_{1\,n}^{(3)} \\ 0 & a_{2\,2}^{(3)} & a_{2\,3}^{(3)} & \cdots & a_{2\,n}^{(3)} \\ 0 & 0 & a_{3\,3}^{(3)} & \cdots & a_{3\,n}^{(3)} \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & a_{n\,3}^{(3)} & \cdots & a_{n\,n}^{(3)} \end{pmatrix}.$$

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

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

denn es gilt

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

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 $A = (a_{i\,j})$, Vektor $b = (b_i)$

(0) für i von 1 bis n
setze $y_i := b_i$
für j von 1 bis n
setze $q_{i\,j} := 1$, falls $i = j$; $q_{i\,j} := 0$, 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 $H^{(j)} \cdot (H^{(j-1)} \cdots H^{(1)})$:
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 $q_{ik} := q_{ik} - e a_{ij}$
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 $y_i := y_i - e a_{ij}$
Ende der i Schleife

Berechnung von $H^{(j)} A^{(j)}$ für die Spalten $j+1, \ldots, n$:
(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 $a_{ik} := a_{ik} - e a_{ij}$
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 2.1.1 das lineare Ausgleichsproblem kennen gelernt, bei dem $x \in \mathbb{R}^n$ gesucht ist, so dass der Vektor

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

minimale 2-Norm $\|r\|_2$ hat, und haben gesehen, dass dieses Problem durch Lösen der Normalengleichungen $\widetilde{A}^T \widetilde{A} x = \widetilde{A}^T z$ gelöst werden kann. Gerade diese Matrix $\widetilde{A}^T \widetilde{A}$ kann aber (bedingt durch ihre Struktur, vgl. Übungsblatt 3) sehr große Kondition haben, so dass es hier erstrebenswert ist, (a) ein robustes Verfahren zu verwenden und (b) die explizite Lösung der Normalengleichungen 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 $\widetilde{A}$ 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 $\widetilde{A} = QR$ 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 Normalengleichungen genau dann lösbar sind, wenn $\widetilde{A}$ 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

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

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 $s^1 = (s_1, \ldots, s_n)^T$ und $s^2 = (s_{n+1}, \ldots, s_m)^T$. Wegen der Form von $R = Q^T\widetilde{A}$ 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 miminal, 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 $\widetilde{A} \in \mathbb{R}^{m\times n}$ mit $m > n$ und (maximalem) Spaltenrang n, Vektor $z \in \mathbb{R}^m$

(1) Berechne Zerlegung $\widetilde{A} = QR$ 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 $\|\widetilde{A}x - z\|_2$ minimal wird. □

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

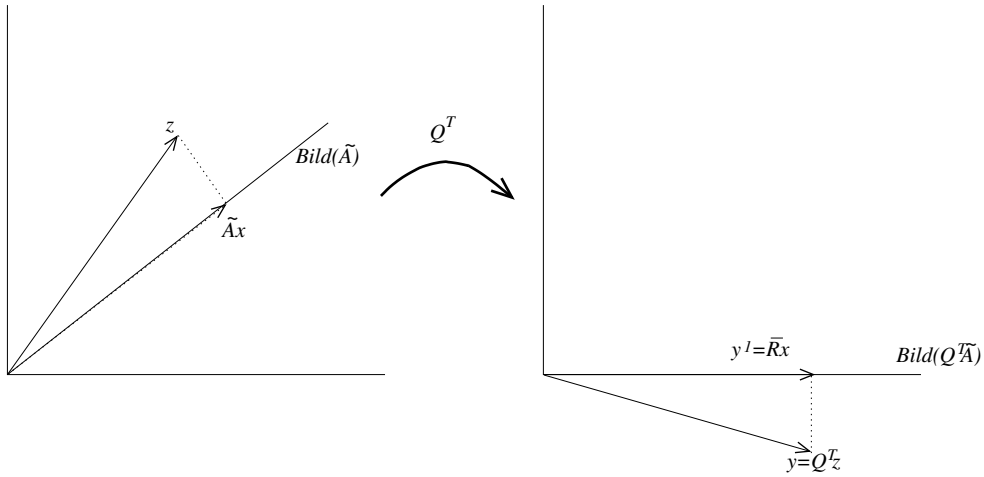

Abbildung 2.1: Veranschaulichung von Algorithmus 2.16

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[4].

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

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

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

[4]Tatsächlich sind Multiplikation, Division und die Berechnung von Wurzeln etwas aufwändiger als Addition und Subtraktion, was wir hier aber vernachlässigen werden.

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 α, d.h., $2(n+2-j)+1$ Operationen. In der j–ten Spalte müssen dabei $n-j$ Einträge eliminiert werden, nämlich für $i=n,\ldots,j+1$. 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 $j=1,\ldots,n-1$ durchgeführt werden, womit wir auf

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

Operationen kommen.

Beim **Choleski Verfahren** kann man wieder direkt abzählen: Für jedes i muss man für $j<i$ je $j-1$ Multiplikationen und Additionen durchführen, dazu eine Division, also insgesamt

$$\sum_{j=1}^{i-1}(2(j-1)+1)=2\sum_{j=1}^{i-1} j+\sum_{j=1}^{i-1}(-1)=i(i-1)-(i-1)=i^2-2i+1$$

Operationen (beachte, dass diese Formel auch für $i = 1$ stimmt). Für $i = j$ ergeben sich $i-1$ Additionen und Multiplikationen (zum Quadrieren der $l_{j\,j}$) sowie einmal Wurzelziehen, also $2(i-1)+1$ Operationen. Insgesamt erhalten wir also für jedes i

$$i^2 - 2i + 1 + 2(i-1) + 1 = i^2 - 2i + 1 + 2i - 2 + 1 = i^2$$

Operationen. Damit ergibt sich die Gesamtzahl der Operationen als

$$\sum_{i=1}^{n} i^2 = \frac{1}{3}n^3 + \frac{1}{2}n^2 + \frac{1}{6}n.$$

Für die QR–**Faktorisierung mittels Householder–Spiegelungen** betrachten wir hier nur die Berechnung von R und $y = Q^T b$. Für jedes $j = 1, \ldots, n-1$ muss c berechnet werden ($2(n-j+1)$ Operationen, wobei die Berechnung von sgn vernachlässigbar schnell ist), sowie $a_{j\,j}$ und d (weitere 3 Operationen). Für die Berechnung von y muss zunächst e berechnet werden ($2n$ Operationen) und dann y ($2(n-j+1)$ Operationen). Für die Berechnung von R schließlich müssen die gleichen Berechnungen $(n-j)$–mal durchgeführt werden, also $(n-j)2n = 2n^2 - 2nj$ Operationen und $(n-j)2(n-j+1) = 2n^2 + 2j^2 - 4nj + 2n - 2j$ Operationen. Insgesamt kommt man so für jedes j auf

$$\begin{aligned}&2(n-j+1) + 3 + 2n + 2(n-j+1) + 2n^2 - 2nj + 2n^2 + 2j^2 - 4nj + 2n - 2j\\ &= 2j^2 - 6(n+1)j + 4n^2 + 8n + 7\end{aligned}$$

Operationen, insgesamt also

$$\begin{aligned}&\sum_{j=1}^{n-1}(2j^2 - 6(n+1)j + 8n + 7)\\ &= \frac{2}{3}(n-1)^3 + (n-1)^2 + \frac{1}{3}(n-1)\\ &\qquad -6(n+1)\frac{n(n-1)}{2} + 4n^2(n-1) + 8n(n-1) + 7(n-1)\\ &= \frac{5}{3}n^3 + 3n^2 + \frac{7}{3}n - 7\end{aligned}$$

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, für das Choleski–Verfahren auf

$$\frac{1}{3}n^3 + \frac{1}{2}n^2 + \frac{1}{6}n + 2n^2 = \frac{1}{3}n^3 + \frac{5}{2}n^2 + \frac{1}{6}n$$

Operationen und für die QR–Zerlegung auf

$$\frac{5}{3}n^3 + 3n^2 + \frac{7}{3}n - 7 + n^2 = \frac{5}{3}n^3 + 4n^2 + \frac{7}{3}n - 7$$

Operationen. Berücksichtigt man, dass für große n die führenden "n^3–Terme" dominant werden, so ergibt sich, dass das Choleski–Verfahren etwa doppelt so schnell wie die Gauß–Elimination ist und diese wiederum etwa $2,5$–mal so schnell 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 Dual Core Prozessor etwa eine Leistung von 20 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	Choleski	Gauß	QR
1000	16.79 ms	33.38 ms	83.54 ms
10000	16.68 s	33.34 s	83.35 s
100000	4.63 h	9.26 h	23.15 h
500000	24.11 d	48.23 d	120.56 d

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

Zum Abschluss dieses Abschnitts wollen wir noch ein gröberes 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 $n \to \infty$ 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 $\Phi : \mathbb{R}^n \to \mathbb{R}^n$ iterativ eine Folge von Vektoren

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

die für $i \to \infty$ gegen die Lösung x^* des Gleichungssystems $Ax = b$ konvergieren, also $\lim_{i\to\infty} \|x^{(i)} - x^*\|_p = 0$. 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 $\Phi : A \to A$ 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

$$\|x^{(j+1)} - x^{(j)}\| \leq k^{j-i}\|x^{(i+1)} - x^{(i)}\| \quad \text{und} \quad \|x^{(i+1)} - x^{(i)}\| \leq k^i\|x^{(1)} - x^{(0)}\| \tag{2.12}$$

Damit gilt

$$\begin{aligned}
\|x^{(i+n)} - x^{(i)}\| &= \left\| \sum_{j=i}^{i+n-1} \left(x^{(j+1)} - x^{(j)}\right) \right\| \\
&\leq \sum_{j=i}^{i+n-1} \|x^{(j+1)} - x^{(j)}\| \leq \sum_{j=i}^{i+n-1} k^{j-i}\|x^{(i+1)} - x^{(i)}\| \\
&= \frac{1-k^n}{1-k}\|x^{(i+1)} - x^{(i)}\| \leq \frac{1}{1-k}\|x^{(i+1)} - x^{(i)}\| \\
&\leq \frac{k}{1-k}\|x^{(i)} - x^{(i-1)}\| \leq \frac{k^i}{1-k}\|x^{(1)} - x^{(0)}\|,
\end{aligned} \tag{2.13}$$

weswegen diese Folge wegen $k^i \to 0$ eine Cauchy–Folge ist.

Wir zeigen nun, dass $x^{(i)}$ gegen einen Fixpunkt von Φ 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 $\lim_{i\to\infty} x^{(i)} = x^* \in A$. Da Φ eine Kontraktion, also insbesondere stetig ist, folgt

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

also ist x^* ein Fixpunkt von Φ. Es folgt also, dass jede Folge der angegebenen Form gegen einen Fixpunkt von Φ konvergiert. Es bleibt die Eindeutigkeit des Fixpunktes zu zeigen: Wir nehmen dazu an, dass zwei Fixpunkte $x^*, x^{**} \in A$ von Φ 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 (2.13) mittels

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

□

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

$$x^{(i+1)} = M^{-1}Nx^{(i)} + M^{-1}b, \quad i = 0, 1, 2, \ldots. \tag{2.14}$$

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 (2.14) 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 $\Phi : \mathbb{R}^n \to \mathbb{R}^n$ 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 (2.14) 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. □

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

$$\|x^{(i+1)} - x^{(i)}\| < \varepsilon \tag{2.15}$$

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 ε 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

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

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 (2.14) die Vektoren $x^{(1)}, \ldots, x^{(10)}$, 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 (2.14) 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

$$M_J = \begin{pmatrix} a_{1\,1} & 0 & \ldots & 0 \\ 0 & a_{2\,2} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \ldots & 0 & a_{n\,n} \end{pmatrix}$$

und $N = N_J$ als $N_J = M_J - A$. Dann ergibt sich (2.14) zu

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

□

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

$$M_{GS} = \begin{pmatrix} a_{1\,1} & 0 & \ldots & 0 \\ a_{2\,1} & a_{2\,2} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ a_{n\,1} & \ldots & a_{n\,n-1} & a_{n\,n} \end{pmatrix}$$

und $N = N_{GS}$ als $N_{GS} = M_{GS} - A$. Wenn wir (2.14) 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

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

□

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

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

für alle $i = 1, \ldots, n$ erfüllt.

Dann ist die Voraussetzung von Lemma 2.19 für die Zeilensummennorm $\|\cdot\|_\infty$ erfüllt und die Konstante k lässt sich durch

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

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 $M = M_J = \mathrm{diag}(a_{1\,1}, \ldots, a_{n\,n})$ und $N = N_J = M_J - A$. Wegen $M_J^{-1} = \mathrm{diag}(a_{1\,1}^{-1}, \ldots, a_{n\,n}^{-1})$ folgt

$$M_J^{-1}N_J = \begin{pmatrix} 0 & -\frac{a_{1\,2}}{a_{1\,1}} & \cdots & -\frac{a_{1\,n}}{a_{1\,1}} \\ -\frac{a_{2\,1}}{a_{2\,2}} & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & -\frac{a_{n-1\,n}}{a_{n-1\,n-1}} \\ -\frac{a_{n\,1}}{a_{n\,n}} & \cdots & -\frac{a_{n\,n-1}}{a_{n\,n}} & 0 \end{pmatrix},$$

also

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

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 $i = 1, \ldots, n$:

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

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

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

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

folgt

$$\begin{aligned} |y_i| &\leq \frac{1}{|a_{i\,i}|}\left(\sum_{j=1}^{i-1}|a_{i\,j}||y_j| + |[N_{GS}x]_i|\right) \leq \frac{1}{|a_{i\,i}|}\left(\sum_{j=1}^{i-1}|a_{i\,j}||y_j| + \sum_{j=i+1}^{n}|a_{i\,j}||x_j|\right) \\ &= \sum_{j=1}^{i-1}\frac{|a_{i\,j}|}{|a_{i\,i}|}|y_j| + \sum_{j=i+1}^{n}\frac{|a_{i\,j}|}{|a_{i\,i}|}|x_j| \leq \sum_{j=1}^{i-1}\frac{|a_{i\,j}|}{|a_{i\,i}|}\underbrace{k_J}_{<1} + \sum_{j=i+1}^{n}\frac{|a_{i\,j}|}{|a_{i\,i}|}\underbrace{\|x\|_\infty}_{=1} \end{aligned}$$

$$< \left(\sum_{j=1}^{i-1} \frac{|a_{ij}|}{|a_{ii}|} + \sum_{j=i+1}^{n} \frac{|a_{ij}|}{|a_{ii}|} \right) \leq \max_{q=1,\ldots,n} \sum_{\substack{j=1 \\ j\neq q}}^{n} \frac{|a_{qj}|}{|a_{qq}|}$$
$$= \|M_J^{-1} N_J\|_\infty = k_J.$$

□

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 $\lambda_1(E), \ldots, \lambda_d(E)$ einer Matrix $E \in \mathbb{R}^{n\times n}$ bezeichnet. Dann konvergiert Verfahren (2.14) 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

$$\|E\|_{E,\varepsilon} \leq \rho(E) + \varepsilon \tag{2.16}$$

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

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

ist. Für das gegebene $\varepsilon \in (0,1)$ setzen wir $C_\varepsilon := SD_\varepsilon$ mit S von oben und $D_\varepsilon = \mathrm{diag}(1, \varepsilon, \varepsilon^2, \ldots, \varepsilon^{n-1})$. Wir schreiben $R_\varepsilon = C_\varepsilon^{-1}EC_\varepsilon$. Man rechnet leicht nach, dass für die Elemente $r_{\varepsilon,ij}$ von R_ε 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_ε gleich Null sind. Damit gilt

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

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 (2.16) eine Norm $\|\cdot\|_{M^{-1}N,\varepsilon}$ mit $\|M^{-1}N\|_{M^{-1}N,\varepsilon} < \rho(M^{-1}N) + \varepsilon < 1$. 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 unbekanten Norm $\|\cdot\|_{M^{-1}N,\varepsilon}$. □

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 (2.14) 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,

$$2Nz = 2\lambda Mz. \tag{2.17}$$

Wir müssen zeigen, dass $|\lambda| < 1$ ist. Wir setzen nun $D := \mathrm{diag}(a_{1\,1}, \ldots, a_{n\,n})$. Für alle $i = 1, \ldots, n$ gilt wegen der positiven Definitheit von A die Ungleichung $a_{i\,i} = e_i^T A e_i > 0$, 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 (2.17) ein und multiplizieren die Gleichung dann von links mit $\bar{z}^T$, so erhalten wir

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

Da A und D symmetrisch und positiv definit sind, sind die Werte $a = \bar{z}^T Az$ sowie $d = \bar{z}^T Dz$ 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. □

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_ε bis zum Erreichen einer vorgegebenen Genauigkeit ε 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*, siehe Abschnitt 2.1.3 für ein Beispiel. 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 8 (Übungsblatt 3).

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 uns dann $PAx = Pb$ gelöst wird. Eine mögliche Strategie dafür wird auf dem 5. Übungsblatt behandelt.

2.9 Weitere iterative Verfahren

Wir wollen in diesem letzten Abschnitt zwei weitere iterative Verfahren kurz erläutern, ohne allzutief in die theoretischen Grundlagen einzusteigen.

2.9.1 Relaxation

Das erste Verfahren 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 Grundidee ist wie folgt: Beim Jacobi–Verfahren wählt man einen reellen Parameter $\omega > 0$ und ändert die Iteration

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

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 Φ gelieferten neuen Wert.

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

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

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*.

Wir betrachten die Variante auf Basis des Gauß–Seidel–Verfahrens etwas genauer: Wir zerlegen zunächst die Matrix A wie folgt in $A = D - L - R$

$$A = \underbrace{\begin{pmatrix} a_{1\,1} & 0 & \dots & 0 \\ 0 & a_{2\,2} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & a_{n\,n} \end{pmatrix}}_{=:D} - \underbrace{\begin{pmatrix} 0 & 0 & \dots & 0 \\ -a_{2\,1} & 0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ -a_{n\,1} & \dots & -a_{n\,n-1} & 0 \end{pmatrix}}_{=:L} - \underbrace{\begin{pmatrix} 0 & -a_{1\,2} & \dots & -a_{1\,n} \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & 0 & -a_{n-1\,n} \\ 0 & \dots & 0 & 0 \end{pmatrix}}_{=:R}$$

Die oben angegebene relaxierte Rechenvorschrift lässt sich damit als

$$Dx^{(i+1)} = (1-\omega) Dx^{(i)} + \omega L x^{(i+1)} + \omega R x^{(i)} + \omega b \tag{2.18}$$

schreiben. Um das Ganze in der bekannten Form (2.14) zu schreiben, skalieren wir das gegebene Gleichungssystem zu $\omega Ax = \omega b$ (was nichts an der Lösung ändert) und definieren ein Iterationsverfahren der Form (2.14) mit $M = D - \omega L$ und $N = (1-\omega)D + \omega R$, ($\Rightarrow M - N = \omega A$) bzw. ausgeschrieben als

$$x^{(i+1)} = (D - \omega L)^{-1}((1-\omega)D + \omega R)x^{(i)} + (D - \omega L)^{-1}\omega b, \qquad i = 0, 1, 2, \ldots \tag{2.19}$$

Beachte, dass diese Vorschrift äquivalent zu der Gleichung (2.18) ist. Für $\omega = 1$ erhalten wir gerade das ursprüngliche Gauß–Seidel–Verfahren. Da in vielen Fällen $\omega > 1$ die schnellere Konvergenz liefert, wird das Verfahren (2.19) als *SOR–Verfahren* (SOR="successive overrelaxation") bezeichnet, selbst wenn $\omega < 1$ theoretisch zulässig ist.

Für verschiedene Bereiche von ω und verschiedene Strukturannahmen an A kann man die Konvergenz dieses Verfahrens zeigen. Ein Beispiel für ein solches Resultat liefert der folgende Satz.

Satz 2.27 Sei $A \in \mathbb{R}^{n\times n}$ eine symmetrische, positiv definite Matrix. Dann konvergiert das SOR–Verfahren (2.19) mit $\omega \in (0, 2)$ für alle Startwerte $x^{(0)} \in \mathbb{R}^n$ gegen die Lösung x^* des Gleichungssystems $Ax = b$.

Beweisskizze: Der Beweis verläuft analog zum Beweis von Satz 2.26, wobei Gleichung (2.17) nun

$$2((1-\omega)D + \omega R)z = 2\lambda(D - \omega L)z$$

lautet. Mit ähnlichen Umformungen wie in diesem Beweis kommt man schließlich auf die Gleichung

$$\lambda = \frac{(2-\omega)d - \omega a + i\omega b}{(2-\omega)d + \omega a + i\omega b}$$

aus der man für $\omega \in (0, 2)$ auf $|\lambda| < 1$ schließt. □

Die besondere "Kunst" bei diesem Verfahren besteht nun darin, $\omega > 0$ so auszuwählen, dass das Verfahren möglichst schnell konvergiert. Mit Blick auf Bemerkung 2.25 bietet es sich hierbei an, ω so zu wählen, dass der Spektralradius $\rho((D+\omega L)^{-1}((1-\omega)D+\omega R)$ möglichst klein wird. Bei besonderer Struktur von A kann man hierfür explizite Formeln herleiten; oft ist man jedoch auf "try–and–error" Verfahren angewiesen, bei denen geeignete Werte für ω auf Grund numerischer Erfahrung gewählt werden. Ein Beispiel, bei dem die Anzahl der Iterationen bei optimaler Wahl von ω drastisch gesenkt werden kann, findet sich in Schwarz/Köckler [8], Bsp. 11.7 (Bsp. 11.5 in der 4. Auflage).

2.9.2 Das konjugierte Gradientenverfahren

Die bisherigen Verfahren basieren alle auf einer additiven Zerlegung der Matrix A. Eine weitere Klasse iterativer Verfahren folgt einer ganz anderen Idee, sie basieren nämlich auf Optimierungsmethoden. Zum Abschluss dieses Abschnitts wollen wir einen einfachen Vertreter dieser Klasse, das *konjugierte Gradientenverfahren* oder *CG–Verfahren* (CG="conjugate gradient"), kurz betrachten. Dies ist wiederum ein Verfahren für symmetrische und positiv definite[5] Matrizen A.

[5] Ähnliche Verfahren für allgemeine Matrizen existieren ebenfalls, z.B. das CGS– oder das BiCGstab–Verfahren, sind aber komplizierter

Statt das Gleichungssystem $Ax = b$ zu lösen, löst man das Minimierungsproblem

$$\text{minimiere} \quad f(x) = \frac{1}{2}x^T Ax - b^T x.$$

Für eine Lösung dieses Minimierungsproblems gilt

$$0 = \nabla f(x) = Ax - b,$$

weswegen dies eine Lösung des ursprünglichen linearen Gleichungssystems liefert. Das CG–Verfahren ist eigentlich ein direktes Verfahren, da es (zumindest in der Theorie, also ohne Rundungsfehler) nach endlich vielen Schritten ein exaktes Ergebnis liefert. Trotzdem zählt man es zu den iterativen Verfahren, da die Zwischenergebnisse des Verfahrens bereits Näherungslösungen darstellen, so dass man das Verfahren in der Praxis vor dem Erreichen der exakten Lösung abbricht. Die Näherung $x^{(i+1)}$ wird hierbei aus der vorhergehenden bestimmt, indem eine Suchrichtung $d^{(i)} \in \mathbb{R}^n$ und eine Schrittweite $\alpha^{(i)} \in \mathbb{R}$ ermittelt wird, und dann

$$x^{(i+1)} = x^{(i)} + \alpha^{(i)} d^{(i)}$$

gesetzt wird, wobei $d^{(i)}$ und $\alpha^{(i)}$ so gewählt werden, dass $f(x^{(i+1)})$ möglichst klein wird und $f(x^{(i+1)}) < f(x^{(i)})$ gilt.

Zur Wahl der Schrittweite: Für eine gegebene Suchrichtung $d^{(i)}$ soll die Schrittweite $\alpha^{(i)}$ so gewählt werden, dass $h(\alpha) = f(x^{(i)} + \alpha d^{(i)})$ minimal wird. Dies ist ein eindimensionales Optimierungsproblem, da h eine Abbildung von $\mathbb{R}$ nach $\mathbb{R}$ ist. Bedingt durch die Struktur von h bzw. f kann man ausrechnen, dass das Minimum für

$$\alpha = \frac{(b - Ax^{(i)})^T d^{(i)}}{{d^{(i)}}^T A d^{(i)}}$$

angenommen wird.

Zur Wahl der Suchrichtung: Die Suchrichtung wird in verschiedenen Schritten auf unterschiedliche Weise gewählt. Im ersten Schritt wird die Richtung des steilsten Abstiegs verwendet. Da der Gradient ∇f in Richtung des steilsten Anstiegs zeigt, wählt man

$$d^{(0)} = -\nabla f(x^{(0)}) = -(Ax^{(0)} + b) = b - Ax^{(0)}.$$

Die weiteren Suchrichtungen für $i \geq 1$ werden nun so gewählt, dass sie bzgl. des Skalarproduktes $\langle x, y \rangle_A = x^T Ay$ orthogonal zu der vorhergehenden Richtung liegen, womit sichergestellt ist, dass "gleichmäßig" in alle Richtungen des $\mathbb{R}^n$ gesucht wird. Formal wählt man dazu $d^{(i)}$ so, dass

$$\langle d^{(i)}, d^{(i-1)} \rangle_A = 0$$

ist. Zusätzlich zu dieser Bedingung (die von vielen Vektoren $d^{(i)}$ erfüllt ist) wird der Ansatz

$$d^{(i)} = r^{(i)} + \beta^{(i)} d^{(i-1)} \quad \text{mit} \quad r^{(i)} = -\nabla f(x^{(i)}) = b - Ax^{(i)}$$

gemacht. Wir gehen also in Richtung des negativen Gradienten (der zugleich das Residuum des linearen Gleichungssystems ist), modifizieren diese Richtung aber durch Addition von $\beta_i d^{(i-1)}$. Diese spezielle Korrektur erlaubt eine einfache Berechnung von $\beta^{(i)}$ als

$$\beta^{(i)} = -\frac{{r^{(i)}}^T A d^{(i-1)}}{{d^{(i-1)}}^T A d^{(i-1)}}.$$

Der so konstruierte Vektor $d^{(i)}$ steht tatsächlich auf allen vorhergehenden Suchrichtungen $d^{(0)}, \ldots, d^{(i-1)}$ senkrecht, was nicht direkt zu sehen ist, aber mit etwas Aufwand bewiesen werden kann. Beachte, dass in all diesen Berechnungen der Nenner der auftretenden Brüche ungleich Null ist, da A positiv definit ist.

Man kann beweisen, dass das Verfahren (in der Theorie, also ohne Rundungsfehler) nach spätestens n Schritten eine exakte Lösung des Problems findet. Bei großem n wird man die Iteration typischerweise bereits früher, d.h. nach Erreichen einer vorgegebenen Genauigkeit abbrechen wollen, was möglich ist, da die Folge $x^{(i)}$ schon während des Iterationsprozesses gegen x^* konvergiert. Als Abbruchkriterium wird hier üblicherweise die Ungleichung $\|r^{(i)}\| \leq \varepsilon$ verwendet. Wegen $r^{(i)} = b - Ax^{(i)}$ wird hiermit die Größe des Residuums abgeschätzt, so dass wir mittels Satz 2.9 eine Abschätzung für den tatsächlichen Fehler erhalten.

Kapitel 3

Eigenwerte

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. 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.

3.1 Kondition des Eigenwertproblems

Bei den linearen Gleichungssystemen haben wir den Fehler analysiert, indem wir das Residuum betrachtet haben, womit wir alle möglichen Fehlerquellen in die gestörte rechte Seite $b + \Delta b$ "verschoben" haben. Damit hängt der Fehler Δx linear vom Residuum Δb ab, was eine Fehlerbetrachtung allein über die Matrixnorm ermöglicht hat.

Bei den Eigenwertproblemen ist dies nicht möglich, da ja nur die Einträge der Matrix als Problemdaten vorhanden sind. Wir müssen also explizit untersuchen, wie sich ein Eigenwert $\lambda_0(A)$ in Abhängigkeit von Änderungen in A verändert, also die Größe

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

berechnen. Da die Eigenwerte nicht linear von der Matrix A abhängen, ist dieser Ausdruck im Allgemeinen schwer zu berechnen; wie allgemein bei nichtlinearen Problemen üblich,

beschränken wir uns daher auf eine lineare Approximation der Änderung von $\lambda_0(A)$, welche gerade durch die Ableitung $D\lambda_0(A)$ gegeben ist. Diese Ableitung ist dabei für festes A als lineare Abbildung $D\lambda_0(A) : \mathbb{C}^{n\times n} \to \mathbb{C}$ aufzufassen. Wir betrachten für kleine $\|\Delta A\|$ also die Näherung

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

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

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

Definition 3.1 Die (absolute) Kondition der Berechnung eines Eigenwerts $\lambda_0(A)$ für eine Matrix $A \in \mathbb{C}^{n\times n}$ ist definiert durch

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

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

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

Beispiel 3.2 Betrachte die Matrizen

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

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

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

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

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

also

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

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

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

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

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

Satz 3.3 Die (absolute) Kondition der Berechnung eines einfachen Eigenwertes $\lambda_0(A)$ einer Matrix $A \in \mathbb{C}^{n\times n}$ gemessen in der 2–Norm ist gegeben durch

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

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

Beweis: Wir zeigen zunächst die Gleichung

$$|D\lambda_0(A)C| = \frac{|\langle Cx_0, y_0\rangle|}{|\langle x_0, y_0\rangle|} \tag{3.1}$$

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

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

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

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

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

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

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

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

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

Mit

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

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

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

und damit die behauptete Gleichung (3.1).

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

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

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

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

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

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

$$|\langle Cx_0, y_0\rangle| \leq \|Cx_0\|_2\|y_0\|_2 \leq \|C\|_2\|x_0\|_2\|y_0\|_2. \tag{3.2}$$

Für $C = y_0\bar{x}_0^T$ gilt hierbei

$$|\langle Cx_0, y_0\rangle| = |\langle y_0, Cx_0\rangle| = |\bar{y}_0^T Cx_0| = |\bar{y}_0^T y_0\bar{x}_0^T x_0| = \|y_0\|_2^2\|x_0\|_2^2 = \|C\|_2\|y_0\|_2\|x_0\|_2, \tag{3.3}$$

also Gleichheit. Aus (3.1) und (3.2) folgt also für beliebige $C \in \mathbb{C}^{n\times n}$

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

wobei für $C = y_0\bar{x}_0^T$ wegen (3.3) Gleichheit gilt. Dies zeigt die Behauptung. □

Bemerkung 3.4 (i) Der Ausdruck für κ_{abs} hat eine geometrische Interpretation: Für reelle Vektoren x_0, $y_0 \in \mathbb{R}^n$ gilt nämlich

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

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

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

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

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

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

3.2 Vektoriteration

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

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

$$x^{(i+1)} = Ax^{(i)} \tag{3.4}$$

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

Satz 3.5 Sei $A \in \mathbb{R}^{n\times n}$ eine symmetrische Matrix und $\lambda_1 = \lambda_1(A)$ ein einfacher Eigenwert, für den die Ungleichung

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

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

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 (3.4) gegen $\pm v_1$, also gegen einen normierten Eigenvektor zum Eigenwert λ_1. Insbesondere konvergiert damit der sogenannte *Rayleigh'sche Quotient*

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

gegen den Eigenwert λ_1.

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

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

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

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

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

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

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

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

□

Beachte, dass die Symmetrie der Matrix A hier nur eine hinreichende aber keine notwendige Bedingung für die Konvergenz des Verfahrens ist. So ist z.B. in [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 ein reelle symmetrische Matrix. Wir setzen voraus, dass wir einen Schätzwert $\tilde{\lambda} \in \mathbb{R}$ für einen Eigenwert $\lambda_j = \lambda_j(A)$ kennen, für den die Ungleichung

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

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

Die inverse Vektoriteration ist nun gegeben durch

$$x^{(i+1)} = (A - \tilde{\lambda}\mathrm{Id})^{-1} x^{(i)} \tag{3.5}$$

Aus Satz 3.5 (angewendet auf $(A - \tilde{\lambda}\mathrm{Id})^{-1}$ an Stelle von A) folgt, dass diese Iteration gegen einen normierten Eigenvektor v_j von $(A - \tilde{\lambda}\mathrm{Id})^{-1}$ zum Eigenwert $1/(\lambda_j - \tilde{\lambda})$ konvergiert. Wegen

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

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

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

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

Die tatsächliche Implementierung der Iteration (3.5) ist hier etwas komplizierter als bei der direkten Iteration (3.4). Während dort in jedem Schritt eine Matrix–Vektor Multiplikation mit Aufwand $O(n^2)$ durchgeführt werden muss, geht hier die Inverse $(A - \tilde{\lambda}\mathrm{Id})^{-1}$ ein. In der Praxis berechnet man nicht die Inverse (weil dies numerisch sehr aufwändig ist), sondern löst das lineare Gleichungssystem

$$(A - \tilde{\lambda}\mathrm{Id}) x^{(i+1)} = x^{(i)}, \tag{3.6}$$

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

Für sehr gute Schätzwerte $\tilde{\lambda} \approx \lambda_j$ wird die Matrix $(A - \tilde{\lambda}\mathrm{Id})$ "fast" singulär (für $\tilde{\lambda} = \lambda_j$ wäre sie singulär), weswegen die Kondition von $(A - \tilde{\lambda}\mathrm{Id})$ sehr groß wird. Wegen der besonderen Struktur des Algorithmus führt dies hier aber nicht auf numerische Probleme, da zwar die Lösung $x^{(i+1)}$ des Gleichungssystems (3.6) mit großen Fehlern behaftet sein kann, sich diese Fehler aber in der hier eigentlich wichtigen *normierten Lösung* $x^{(i+1)}/\|x^{(i+1)}\|$ nicht auswirken. Wir wollen dies an einem Beispiel illustrieren.

Beispiel 3.6 Betrachte

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

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

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

fast singulär und man sieht leicht, dass für die Kondition z.B. in der Zeilensummennorm die Abschätzung $\mathrm{cond}_\infty(A - \tilde{\lambda}\mathrm{Id}) > 1/\varepsilon$ gilt. Die Inverse besitzt aber eine typische spezielle Struktur: die großen Einträge, die der Grund für die große Kondition sind, entstehen lediglich durch einen skalaren Vorfaktor. Daher ist die Berechnung von $y^{(i+1)} = x^{(i+1)}/\|x^{(i+1)}\|$ mittels (3.6) nicht stark anfällig für Rundungsfehler, denn es gilt

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

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

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

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

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

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. □

3.3 Der QR–Algorithmus

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

Wir wollen daher nun 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 $v_1, v_2, \ldots, v_n$ besteht, so dass für die orthogonale Matrix $Q = (v_1, \ldots, v_n) \in \mathbb{R}^{n\times n}$ die Gleichung

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

gilt, wobei die λ_i gerade die Eigenwerte von A sind. Wenn wir also eine orthogonale Matrix Q finden können, mit der A auf Diagonalgestalt konjugiert werden kann, so können wir die Eigenwerte direkt aus der resultierenden Diagonalmartix Λ 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 3.7 Sei A eine reelle symmetrische Matrix der Form

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

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

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

und den Konventionen aus Lemma 2.14. Dann gilt

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

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

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

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

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

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

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

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

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

Insgesamt gilt also

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

Da A symmetrisch ist, gilt $(HAH^T)^T = (H^T)^T A^T H^T = HAH^T$, d.h., HAH^T ist ebenfalls symmetrisch. Mit dem gleichen Argument wie oben folgt für Zeilenvektoren y die Gleichung

$[yH^T]_i = y_i$ für $i = 1, \ldots, k$, weswegen die ersten k Spalten von HA und HAH^T übereinstimmen. Die behauptete Form ergibt sich damit aus der Symmetrie von HAH^T. □

Die folgende Aussage ist nun ein einfaches Korollar.

Korollar 3.8 Sei A eine reelle symmetrische Matrix. Dann existiert eine orthogonale Matrix P, so dass P^TAP symmetrisch und in Tridiagonalgestalt ist.

Beweis: Durch Anwendung von Lemma 3.7 für $k = 1, \ldots, n-2$ erhält man Matrizen $H^{(1)}$,..., $H^{(n-2)}$, für die $P^T = H^{(n-2)} \cdots H^{(1)}$ die gewünschte Eigenschaft besitzt. □

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_{jj-1} bzw. a_{ij-1} ersetzt. Die Berechnung von $H^{(j)}y^{(j)}$ macht hier natürlich keinen Sinn, die Berechung von $H^{(j)}A^{(j)}$ kann falls gewünscht auf die Berechnung von $H^{(j)}A^{(j)}H^{(j)^T}$ erweitert werden.

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

Algorithmus 3.9 (QR–Algorithmus zur Eigenwertberechnung)
Eingabe: symmetrische, reelle Matrix $A \in \mathbb{R}^{n\times n}$

(0) Setze $A^{(1)} := P^TAP$ mit P aus Korollar 3.8, $i := 1$

(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 $A^{(i)}$, Details siehe unten in Bemerkung 3.13 (iii). □

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

Wir werden den Algorithmus 3.9 nun analysieren. Wir beginnen mit einigen Struktureigenschaften der Matrizen $A^{(i)}$.

Lemma 3.10 Für alle $i \geq 1$ gilt:

(i) $A^{(i)}$ ist konjugiert zu A. Genauer gilt $A^{(i+1)} = Q_i^T P^T A P Q_i$ mit $Q_i = Q^{(1)} \cdots Q^{(i)}$ und P aus Korollar 3.8.

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

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

Beweis: (i) Es gilt

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

Durch induktives Fortfahren erhält man so

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

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

(ii) Nach (i) gilt

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

da A symmetrisch ist.

(iii) Sei $A^{(i)}$ tridiagonal. Durch explizites Ausrechnen der QR–Zerlegung mit Hilfe von Lemma 2.14 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 $A^{(i)}$. □

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

Lemma 3.11 Sei $A \in \mathbb{R}^{n\times n}$ eine invertierbare Matrix und seien $A = Q_1R_1$ und $A = Q_2R_2$ QR–Zerlegungen von A. Dann gilt $Q_1 = DQ_2$ und $R_1 = DR_2$ mit $D = \mathrm{diag}(\mu_1, \ldots, \mu_n)$ und $\mu_k = \pm 1$. Insbesondere existiert eine eindeutige QR–Zerlegung bei der alle Diagonalelemente von R positiv sind.

Beweis: Aus $Q_1R_1 = Q_2R_2$ folgt $Q_2^TQ_1 = R_2R_1^{-1}$ (beachte, dass R_1 und R_2 invertierbar sind, da A invertierbar ist). Die Inverse einer oberen Dreiecksmatrix ist wieder eine obere Dreiecksmatrix, ebenso das Produkt zweier oberer Dreiecksmatrizen. Also ist $R = R_2R_1^{-1}$ eine obere Dreiecksmatrix, die wegen $R = Q_2^TQ_1$ orthogonal ist. 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. □

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

Satz 3.12 Sei $A \in \mathbb{R}^{n\times n}$ eine symmetrische Matrix mit den Eigenwerten $\lambda_1, \ldots, \lambda_n$, für die die Ungleichung

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

gelte. Seien $A^{(i)}$, $Q^{(i)}$ und $R^{(i)}$ aus Algorithmus 3.9. Dann gilt

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

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

$$A^i = Q_i R_i \tag{3.7}$$

für die i–te Potenz von A, mit $Q_i = Q^{(1)} \cdots Q^{(i)}$ und $R_i = R^{(i)} \cdots R^{(1)}$. Für $i = 1$ ist (3.7) klar, denn

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

Für den Schritt $i \to i+1$ verwenden wir $A^{(i+1)} = Q_i^T A Q_i$ aus Lemma 3.10(i), aus dem wegen $Q^{(i+1)} R^{(i+1)} = A^{(i+1)}$ die Gleichung $AQ_i = Q_i Q^{(i+1)} R^{(i+1)}$ folgt. Damit und mit der Induktionsannahme $A^i = Q_i R_i$ folgt

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

also (3.7). 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

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

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

$$A^i = Q \Lambda^i L R = Q(\Lambda^i L \Lambda^{-i})(\Lambda^i R). \tag{3.8}$$

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

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

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 $l_{k\,j} \neq 0$. Also folgt $(\lambda_k/\lambda_j)^i \to 0$ und damit

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

Aus (3.8) erhalten wir so

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

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

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

und damit auch

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

Die Matrizen $Q\widetilde{Q}_i$ und $\widetilde{R}_i\Lambda^i R$ bilden nun wegen (3.8) und

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

eine QR–Zerlegung von A^i. Da $A^i = Q_iR_i$ eine weitere QR–Zerlegung ist, folgt mit Lemma 3.11 die Existenz von $D_i = \mathrm{diag}(\pm 1, \ldots, \pm 1)$, so dass

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

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

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

Da jede der vier von i abhängigen Matrizen gegen Id konvergiert, konvergiert das Produkt also gegen Λ, womit die Behauptung gezeigt ist. □

Bemerkung 3.13 (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 λ_i und λ_j mit $\lambda_i = -\lambda_j$ so kann in $A^{(i)}$ ein 2×2–Block stehen bleiben.

(iii) Die approximativen Eigenwerte von A liest man bei diesem Algorithmus einfach aus der Diagonalen von $A^{(i)}$ ab. Die zugehörigen Eigenvektoren werden approximiert durch die Spalten der Transformationsmatrix $\overline{Q}_i$, für die $\overline{Q}_i^T A \overline{Q}_i = A^{(i)}$ gilt. Diese ist nach Lemma 3.10(i) gerade durch $\overline{Q}_i = PQ_{i-1} = PQ^{(1)} \cdots Q^{(i-1)}$ gegeben.

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

(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)

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

wobei hier $\kappa_{\text{abs}}(\lambda_j(\Lambda^{(i)})) = 1$ gilt, da $\Lambda^{(i)}$ eine Diagonalmatrix und damit normal ist. Die Matrixnorm $\|B^{(i)}\|_2$ lässt sich aber durch die Größe der Einträge von $B^{(i)}$ abschätzen.

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

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

Algorithmus 3.14 (QR–Algorithmus zur Eigenwertberechnung mit Shift)
Eingabe: symmetrische, reelle Matrix A, Folge von Shift–Parametern σ_i

(0) Setze $A^{(1)} := P^T A P$ mit P aus Korollar 3.8, $i := 1$

(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

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

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

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

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

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

Bemerkung 3.15 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 $\lambda_j = a + bi$ erhält man dabei einen 2×2–Block der Form

$$\begin{pmatrix} a & b \\ -b & a \end{pmatrix}.$$

□

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 $i = 0, \ldots, n$ gegeben haben (z.B. Messwerte eines Experiments). Die Problemstellung ist nun wie folgt: Gesucht ist eine Funktion F, für die die Gleichung

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

gilt.

Ein wichtiger Spezialfall dieses Problems ist die *Funktionsinterpolation*: Nehmen wir an, dass wir eine reellwertige Funktion $f : \mathbb{R} \to \mathbb{R}$ 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, $x_1, \ldots, x_n$ die Gleichung

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

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. Für die Funktionsapproximation werden wir zusätzlich die sogenannte *Hermite–Interpolation* betrachten, bei der zusätzlich zu den Funktionswerten auch Ableitungen vorgegeben werden.

4.1 Polynominterpolation

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

$$P(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_m x^m. \tag{4.3}$$

Hierbei werden die Werte a_i, $i = 0, \ldots, m$, 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 $\mathcal{B} = \{1, x, \ldots, x^m\}$, 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 $i = 0, \ldots, n$ 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

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

erfüllt.

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

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

Die Determinante dieser Matrix ist

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

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

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 zwei andere Techniken zur Berechnung des Polynoms P. Beachte, dass beide Techniken das gleiche Polynom berechnen, auch wenn dieses auf verschiedene Arten 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 $x_0, x_1 \ldots, x_n$ definieren wir für $i = 0, \ldots, n$ 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 $i = 0, \ldots, n$ 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). □

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 $P(5) = 352\frac{1}{6}(5-2)(5-3) = 352$. □

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 ℓ 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. □

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

Algorithmus 4.5 (Berechnung der baryzentrischen Koordinaten)
Eingabe: Stützstellen $x_0,\ldots,x_n$

(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 $w_0,\ldots,w_n$ □

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 $x_0,\ldots,x_n$, Werte $f_0,\ldots,f_n$, baryzentrische Koordinaten $w_0,\ldots,w_n$, Auswertungsstelle x

(0) setze $l := 1$, $s := 0$ (Variablen für ℓ 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

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

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

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

dieser Abbildung berechnen können, also keine Ableitung berechnen müssen. 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 $a = \min_{i=0,\ldots,n} x_i$ und $b = \max_{i=0,\ldots,n} x_i$ wählen (beachte, dass wir keine Ordnung der Stützstellen x_i vorausgesetzt haben).

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

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

wobei Λ_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 $g_i = \text{sgn}(L_i(x^*))$. 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 $\kappa_{\text{abs}} = \|\phi\| = \Lambda_n$. □

Die Zahl Λ_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.

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

n	κ_{abs} für äquidistante Stützstellen	κ_{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 κ_{abs} für verschiedene Stützstellen

4.1.3 Das Newton–Schema

Wir betrachten nun ein weiteres Verfahren zur Berechnung von Interpolationspolynomen, das sogenannte *Newton–Schema*, dessen Vorteil darin liegt, dass es sich auch für allgemeinere Interpolationsaufgaben eignet, die wir im nachfolgenden Abschnitt einführen werden. Für das Newton-Schema definiert man zunächst die folgenden Werte:

$$f_{[x_i]} = f_i, \quad i = 0, \ldots, n$$

und berechnet daraus für Stützstellen–Mengen der Form $\{x_l, x_{l+1}, \ldots, x_{l+k}\}$ mit $0 \leq l < l + k \leq n$ rekursiv die sogenannten *dividierten Differenzen*

$$f_{[x_l, x_{l+1}, \ldots, x_{l+k}]} := \frac{f_{[x_l, x_{l+1}, \ldots, x_{l+k-1}]} - f_{[x_{l+1}, x_{l+2}, \ldots, x_{l+k}]}}{x_l - x_{l+k}}.$$

Abbildung 4.1 veranschaulicht diese rekursive Berechnung

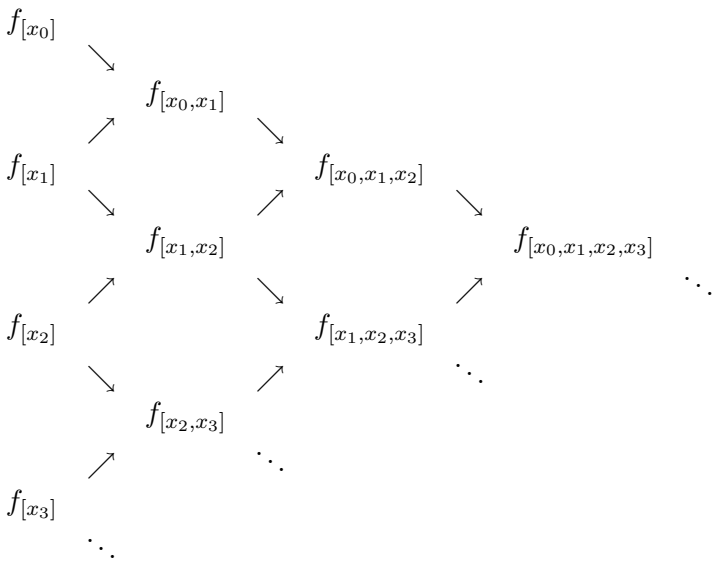

Abbildung 4.1: Illustration des Newton–Schemas

Der folgende Satz zeigt, wie man aus den dividierten Differenzen das Interpolationspolynom berechnen kann.

Satz 4.8 Seien Daten (x_i, f_i) für $i = 0, \ldots, n$ mit paarweise verschiedenen Stützstellen x_i gegeben. Dann ist das eindeutige Interpolationspolynom $P(x)$ mit $P(x_i) = f_i$ für $i = 0, \ldots, n$ gegeben durch

$$\begin{aligned} P(x) &= \sum_{k=0}^{n} \left(f_{[x_0,\ldots,x_k]} \prod_{j=0}^{k-1} (x - x_j) \right) \\ &= f_{[x_0]} + f_{[x_0,x_1]}(x - x_0) + f_{[x_0,x_1,x_2]}(x - x_0)(x - x_1) \\ &\quad + \ldots + f_{[x_0,\ldots,x_n]}(x - x_0)(x - x_1) \cdots (x - x_{n-1}). \end{aligned}$$

Beweis: Wir zeigen zunächst, dass das Interpolationspolynom $P(x) = a_n x^n + \ldots + a_1 x + a_0$ die Gleichung

$$a_n = f_{[x_0,\ldots,x_n]} \tag{4.4}$$

erfüllt. Für $n = 0$ ist die Behauptung klar, um (4.4) für $n \geq 1$ zu beweisen, betrachten wir die Interpolationspolynome P^0_{n-1} und P^n_{n-1} zu den Daten $\{(x_k, f_k) : k = 1, \ldots, n\}$ bzw. $\{(x_k, f_k) : k = 0, \ldots, n-1\}$. Für diese gilt die Gleichung

$$P(x) = \frac{(x_0 - x)P^0_{n-1}(x) - (x_n - x)P^n_{n-1}(x)}{x_0 - x_n}, \tag{4.5}$$

da die rechte Seite dieser Gleichung ein Polynom vom Grad $\leq n$ definiert, dass die Interpolationsbedingung erfüllt. Der Beweis von (4.4) ergibt sich nun durch Induktion über n: Für $n - 1 \to n$ betrachten wir (4.5). Aus dieser Gleichung folgt $a_n = (a^n_{n-1} - a^0_{n-1})/(x_0 - x_n)$, wobei a^n_{n-1} und a^0_{n-1} die führenden Koeffizienten von P^n_{n-1} bzw. P^0_{n-1} sind, für die nach Induktionsannahme $a^n_{n-1} = f_{[x_0,\ldots,x_{n-1}]}$ und $a^0_{n-1} = f_{[x_1,\ldots,x_n]}$ gilt. Also ist

$$a_n = \frac{a^n_{n-1} - a^0_{n-1}}{x_0 - x_n} = \frac{f_{[x_0,\ldots,x_{n-1}]} - f_{[x_1,\ldots,x_n]}}{x_0 - x_n} = f_{[x_0,\ldots,x_n]}.$$

Wir zeigen die Behauptung des Satzes nun durch Induktion über n. Für $n = 0$ ist die Aussage klar. Für $n - 1 \to n$ gilt nach Induktionsannahme

$$P_{n-1}(x) = \sum_{k=0}^{n-1} \left(f_{[x_0,\ldots,x_k]} \prod_{j=0}^{k-1} (x - x_j) \right)$$

für das Interpolationspolynom P_{n-1} zu den Daten $\{(x_i, f_i) : i = 0, \ldots, n-1\}$. Bezeichne P_n das Interpolationspolynom zu $\{(x_i, f_i) : i = 0, \ldots, n\}$. Dann gilt für P_n nach (4.4)

$$\begin{aligned} P_n(x) &= f_{[x_0,\ldots,x_n]} x^n + a_{n-1} x^{n-1} + \ldots + a_0 \\ &= f_{[x_0,\ldots,x_n]} \left(\prod_{j=0}^{n-1} (x - x_j) \right) + Q_{n-1}(x) \end{aligned}$$

für ein Polynom $Q_{n-1} \in \mathcal{P}_{n-1}$. Dieses Polynom

$$Q_{n-1}(x) = P_n(x) - f_{[x_0,\ldots,x_n]} \left(\prod_{j=0}^{n-1} (x - x_j) \right)$$

erfüllt aber gerade die Bedingungen $Q_{n-1}(x_i) = f_i$ für $i = 0, \ldots, n-1$, weswegen $Q_{n-1} = P_{n-1}$ gelten muss, woraus die behauptete Gleichung

$$\begin{aligned} P_n(x) &= f_{[x_0,\ldots,x_n]} \left(\prod_{j=0}^{n-1} (x - x_j) \right) + P_{n-1}(x) \\ &= f_{[x_0,\ldots,x_n]} \left(\prod_{j=0}^{n-1} (x - x_j) \right) + \sum_{k=0}^{n-1} \left(f_{[x_0,\ldots,x_k]} \prod_{j=0}^{k-1} (x - x_j) \right) \end{aligned}$$

folgt. □

Beachte, dass wir für dieses Polynom nur jeweils die Werte aus den ersten Zeilen des Schemas benötigen. Die anderen werden jedoch zur Berechnung dieser Werte gebraucht.

Wir wiederholen unser Beispiel aus dem letzten Abschnitt.

Beispiel 4.9 Betrachte die Daten (3; 68), (2; 16), (5; 352). Die dividierten Differenzen ergeben sich als

$$\begin{array}{ccccc} f_{[x_0]} = 68 & & & & \\ & \searrow & & & \\ & & f_{[x_0,x_1]} = \frac{68-16}{3-2} = 52 & & \\ & \nearrow & & \searrow & \\ f_{[x_1]} = 16 & & & & f_{[x_0,x_1,x_2]} = \frac{52-112}{3-5} = 30 \\ & \searrow & & \nearrow & \\ & & f_{[x_1,x_2]} = \frac{16-352}{2-5} = 112 & & \\ & \nearrow & & & \\ f_{[x_2]} = 352 & & & & \end{array}$$

Damit erhalten wir das Interpolationspolynom

$$P(x) = P_2(x) = 68 + 52(x-3) + 30(x-3)(x-2).$$

Für $x = 3$ erhalten wir $P(3) = 68$, für $x = 2$ ergibt sich $P(2) = 68+52(2-3) = 68-52 = 16$ und für $x = 5$ berechnet man $P(5) = 68 + 52(5-3) + 30(5-3)(5-2) = 68 + 104 + 180 = 352$. □

Auch wenn die Berechnung des Interpolationspolynoms über das Newton–Schema kompliziert aussieht, hat sie mehrere Vorteile. Zum einen lassen sich weitere Datenpunkte (x_i, f_i) leicht hinzufügen, da sich die Polynome für mehr Stützstellen einfach durch Addition zusätzlicher Terme aus denen für weniger Stützstellen ergeben. (Dafür müssen hier schon bei Änderung eines einzigen Wertes f_i alle von f_i abhängigen dividierten Differenzen neu berechnet werden.)

Zum anderen lassen sich sowohl die Berechnung der dividierten Differenzen als auch die Auswertung des Polynoms sehr effizient implementieren. Die Berechnung der dividierten Differenzen kann mit dem folgenden Algorithmus durchgeführt werden, wobei die Schreibweise

$$F_0 = f_{[x_0]}, \;\; F_1 = f_{[x_0,x_1]}, \ldots, \;\; F_n = f_{[x_0,x_1,\ldots,x_n]}$$

verwendet wird.

Algorithmus 4.10 (Berechnung der dividierten Differenzen)
Eingabe: Daten $(x_0, f_0), \ldots, (x_n, f_n)$.

(0) Für i von 0 bis n setze $F_i := f_i$; Ende der Schleife

(1) Für k von 1 bis n

(2) Für i von n bis k (nach unten zählend)

(3) berechne $F_i := \frac{F_{i-1}-F_i}{x_{i-k}-x_i}$

(4) Ende der Schleifen

Ausgabe: Dividierte Differenzen $F_0, \ldots, F_n$ □

Dieser Algorithmus hat einen Aufwand von der Ordnung $O(n^2)$. Sind diese F_i einmal berechnet, so lässt sich das Interpolationspolynom mit der folgenden Formel, dem sogenannten *Horner–Schema* auswerten.

Algorithmus 4.11 (Berechnung von $P(x)$ mittels Horner–Schema)
Eingabe: Stützstellen $x_0, \ldots, x_n$, dividierte Differerenzen $F_0, \ldots, F_n$ aus Algorithmus 4.10, $x \in \mathbb{R}$.

(0) Setze $P := F_n$

(1) Für i von $n-1$ bis 0 (nach unten zählend)

(2) berechne $P := F_i + (x - x_i)P$

(3) Ende der Schleife

Ausgabe: $P = P(x)$ □

Analog zur Berechnung von $P(x)$ besitzt dieser Algorithmus einen Aufwand der Ordnung $O(n)$.

Beachte, dass es bei den baryzentrischen Koordinaten leicht ist, Polynome für verschiedene f_i aber gleichbleibende Stützstellen x_i auszuwerten, weil sich die baryzentrischen Koordinaten w_i nicht ändern. Beim Newton-Schema hingegen müssen die F_i für geänderte f_i komplett neu berechnet werden. Andererseits ist es beim Newton-Schema leichter, eine neue Stützstelle x_{n+1} hinzuzufügen (wenn man das komplette Schema gespeichert hat), da die $F_0, \ldots, F_n$ gleich bleiben und nur die unterste Diagonale des Schemas neu berechnet werden muss. Bei den baryzentrischen Koordinaten hingegen müssen in diesem Fall alle w_i neu berechnet werden.

4.1.4 Hermite–Interpolation

Wie bereits angekündigt, wollen wir nun eine verallgemeinerte Interpolationsaufgabe betrachten, die *Hermite–Interpolation*[1], bei der zusätzlich zu den Funktionswerten auch Ableitungen vorgegeben werden können.

Formal fassen wir dies, indem wir die bisher gemachte Annahme, dass die Stützstellen paarweise verschieden sind, aufgeben. Zur einfacheren Notation nehmen wir an, dass die Stützstellen aufsteigend angeordnet sind. So wäre z.B.

$$x_0 < x_1 = x_2 = x_3 < x_4 < x_5 = x_6$$

nun eine erlaubte Stützstellenfolge. Zu jeder Stützstelle x_i definieren wir mit d_i ihre Vielfachheit, beginnend bei 0. Die oben angegebenen Stützstellenfolge gemeinsam mit ihren Vielfachheiten wäre z.B.

x_i	x_0	$<$	x_1	$=$	x_2	$=$	x_3	$<$	x_4	$<$	x_5	$=$	x_6
d_i	0		0		1		2		0		0		1

Das Problem der Hermite–Interpolation besteht nun darin, zu gegebenen Stützstellen $x_0, \ldots, x_n$ mit Vielfachheiten $d_0, \ldots, d_n$ und gegebenen Werten f_i^j ein Polynom P zu finden, so dass die Bedingung

$$P^{(d_i)}(x_i) = f_i^{(d_i)} \quad \text{für } i = 0, 1, \ldots, n \tag{4.6}$$

erfüllt ist, wobei $P^{(j)}$ die j–te Ableitung bezeichnet. Im Spezialfall der Funktionsinterpolation sind diese Werte durch

$$f_i^{d_i} = f^{(d_i)}(x_i) \tag{4.7}$$

gegeben, wobei f als hinreichend oft differenzierbar vorausgesetzt wird. Wir bezeichnen das Interpolationsintervall hier mit $[a, b]$, und nehmen an, dass alle $x_i \in [a, b]$ liegen. Die Randpunkte a und b müssen jedoch keine Stützstellen sein.

Der folgende Satz zeigt, dass das Problem wohldefiniert ist.

Satz 4.12 Zu jeder Stützstellenfolge $x_0, \ldots, x_n$ mit Vielfachheiten $d_i \leq r$ für $i = 0, \ldots, n$ gibt es genau ein Polynom $P \in \mathcal{P}_n$, welches (4.6) erfüllt.

Beweis: Betrachte die lineare Abbildung

$$\begin{aligned} \mu : \mathcal{P}_n &\to \mathbb{R}^{n+1} \\ P &\mapsto (P^{(d_0)}(x_0), P^{(d_1)}(x_1), \ldots, P^{(d_n)}(x_n)) \end{aligned}$$

Diese Abbildung ist injektiv, d.h. $\mu(P) = 0$ impliziert $P \equiv 0$. Dies gilt, da $\mu(P) = 0$ bedeutet, dass P mindestens $n + 1$ Nullstellen besitzt (mit Vielfachheiten gezählt), was für ein Polynom vom Grad n nur für das Nullpolynom gelten kann. Da $\dim\mathcal{P}_n = n + 1 = \dim\mathbb{R}^{n+1}$ ist, ist die Abbildung μ also auch surjektiv, damit invertierbar, woraus die eindeutige Existenz von P folgt. □

Zwei Spezialfälle dieses Problems sollen besonders erwähnt werden:

[1] Benannt nach dem französischen Mathematiker Charles Hermite, 1822–1901.

(1) Falls die x_i paarweise verschieden sind, so erhalten wir das bekannte Interpolationsproblem.

(2) Falls alle x_i übereinstimmen, so besteht das Hermite–Funktionsinterpolationspolynom mit (4.6), (4.7) gerade aus den ersten $n+1$ Termen der Taylorentwicklung von f, also

$$P(x) = \sum_{j=0}^{n} \frac{(x-x_0)^j}{j!} f^{(j)}(x_0). \tag{4.8}$$

Man prüft leicht nach, dass dieses P die Bedingung (4.6) erfüllt.

Im allgemeinen Fall kann man die Hermite–Interpolationspolynome mit Hilfe der dividierten Differenzen aus Abschnitt 4.1.3 berechnen. Wir müssen deren Definition nur für den Fall übereinstimmender Stützstellen erweitern. Dazu definieren wir

$$f_{[x_i]} = f^0_{i-d_i}, \quad i = 0, \ldots, n$$

und

$$\begin{aligned} f_{[x_l, x_{l+1}, \ldots, x_{l+k}]} &:= \frac{f_{[x_l, x_{l+1}, \ldots, x_{l+k-1}]} - f_{[x_{l+1}, x_{l+2}, \ldots, x_{l+k}]}}{x_l - x_{l+k}}, && \text{falls } x_l \neq x_{l+k} \\ f_{[x_l, x_{l+1}, \ldots, x_{l+k}]} &:= \frac{f_l^{(k)}}{k!}, && \text{falls } x_l = \ldots = x_{l+k} \end{aligned}$$

Diese Definition ist nur für aufsteigend angeordnete Stützstellen sinnvoll, da ansonsten mit der obigen Fallunterscheidung nicht alle möglichen Fälle abgedeckt sind. Falls die Stützstellen nicht aufsteigend angeordnet sind, wendet man die obige Definition nach vorheriger Sortierung der x_i an.

Mit dieser Definition können wir den folgenden Satz formulieren, der völlig analog zu Satz 4.8 ist.

Satz 4.13 Seien Stützstellen x_0, x_1, …, x_n mit Vielfachheiten d_i und zugehörigen Werten $f_i^{(d_i)}$ gegeben. Dann ist das eindeutige Interpolationspolynom $P(x)$ mit $P^{(d_i)}(x_i) = f_i^{(d_i)}$ für $i = 0, \ldots, n$ gegeben durch

$$\begin{aligned} P(x) &= \sum_{k=0}^{n} \left(f_{[x_0, \ldots, x_k]} \prod_{j=0}^{k-1} (x - x_j) \right) \\ &= f_{[x_0]} + f_{[x_0, x_1]}(x - x_0) + f_{[x_0, x_1, x_2]}(x - x_0)(x - x_1) \\ &\quad + \ldots + f_{[x_0, \ldots, x_n]}(x - x_0)(x - x_1) \cdots (x - x_{n-1}). \end{aligned}$$

Beweis: Völlig analog zum Beweis des Satzes 4.8, wobei im Beweis von (4.4) der (einfache) Fall $x_0 = x_n$ separat betrachtet werden muss. □

4.1.5 Fehlerabschätzungen

Wir betrachten in diesem Abschnitt das Problem der Funktionsinterpolation (4.2) für eine gegebene Funktion $f : \mathbb{R} \to \mathbb{R}$, bzw. dessen Hermite-Erweiterung (4.6) mit $f_i^j = f^{(j)}(x_i)$. 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.

Wir verwenden für die Fehleranalyse die dividierten Differenzen und beginnen mit einigen vorbereitenden Lemmata.

Lemma 4.14 Es seien die Voraussetzungen von Satz 4.13 erfüllt. Darüberhinaus gelte $f \in C^{r+1}[a, b]$. Dann folgt

$$f(x) = P(x) + f_{[x_0,\ldots,x_n,x]} \prod_{j=0}^{n} (x - x_j)$$

für alle $x \in [a, b]$

Beweis: Die Funktion

$$\widetilde{P}(x) = P(x) + f_{[x_0,\ldots,x_n,x]} \prod_{j=0}^{n} (x - x_j)$$

ist nach Satz 4.13 gerade das Hermite–Interpolationspolynom durch die Stützstellen x_0, $\ldots$, x_i, x, x_{i+1}, $\ldots$, x_n. Also gilt insbesondere $\widetilde{P}(x) = f(x)$, d.h. die Behauptung. □

Wir wollen dieses Lemma benutzen, um die Größe des Interpolationsfehlers über die Größe des Terms $f_{[x_0,\ldots,x_i,x,x_{i+1},\ldots,x_n]} \prod_{j=0}^{n}(x-x_j)$ abzuschätzen. Hierfür benötigen wir die folgende Formel.

Lemma 4.15 Für die n–te dividierte Differenz einer n–mal stetig differenzierbaren Funktion gilt die *Hermite–Genocchi–Formel*

$$f_{[x_0,\ldots,x_n]} = \int_{\Sigma^n} f^{(n)} \left(x_0 + \sum_{i=1}^{n} s_i (x_i - x_0) \right) ds,$$

wobei für $n \geq 1$

$$\Sigma^n := \left\{ s = (s_1, \ldots, s_n) \in \mathbb{R}^n \;\middle|\; \sum_{i=1}^{n} s_i \leq 1 \text{ und } s_i \geq 0 \right\}$$

den *n–dimensionalen Standardsimplex* bezeichnet und wir für $n = 0$

$$\int_{\Sigma^0} f(x_0) ds = f(x_0)$$

definieren.

Beweis: Wir beweisen die Formel per Induktion über n. Für $n = 0$ folgt die Behauptung aus $f_{[x_0]} = f(x_0)$ und der Definition des Integrals über Σ^0.

Für den Induktionsschritt $n \to n+1$ betrachten wir zunächst den Spezialfall $x_0 = x_1 = \ldots = x_{n+1}$. In diesem Fall gilt

$$f_{[x_0,\ldots,x_{n+1}]} := \frac{f^{(n+1)}(x_0)}{(n+1)!} \quad \text{und} \quad x_0 + \sum_{i=1}^{n+1} s_i(x_i - x_0) = x_0$$

woraus sofort die Behauptung folgt, da für alle $n \geq 1$ die Gleichung

$$\int_{\Sigma^n} 1 \, ds = \operatorname{Vol} \Sigma_n = \frac{1}{n!} \tag{4.9}$$

gilt

Falls $x_0 \neq x_{n+1}$ ist, bezeichnen wir für $s = (s_1, \ldots, s_{n+1}) \in \Sigma^{n+1}$ und $1 \leq i \leq j \leq n+1$ die Vektoren $s^{i,j} = (s_i, \ldots, s_j)$. Mit dieser Bezeichnung folgt

$$\begin{aligned}
&\int\limits_{\sum_{i=1}^{n+1} s_i \leq 1} f^{(n+1)}\left(x_0 + \sum_{i=1}^{n+1} s_i(x_i - x_0)\right) ds^{1,n+1} \\
&= \int\limits_{\sum_{i=1}^{n} s_i \leq 1} \int\limits_{s_{n+1}=0}^{1-\sum_{i=1}^{n} s_i} f^{(n+1)}\left(x_0 + \sum_{i=1}^{n} s_i(x_i - x_0) + s_{n+1}(x_{n+1} - x_0)\right) ds_{n+1}\, ds^{1,n} \\
&= \frac{1}{x_{n+1} - x_0} \int\limits_{\sum_{i=1}^{n} s_i \leq 1} \left[f^{(n)}\left(x_{n+1} + \sum_{i=1}^{n} s_i(x_i - x_{n+1})\right) \right. \\
&\qquad\qquad \left. - f^{(n)}\left(x_0 + \sum_{i=1}^{n} s_i(x_i - x_0)\right)\right] ds^{1,n} \\
&= \frac{1}{x_{n+1} - x_0}\Big(f_{[x_1,\ldots,x_{n+1}]} - f_{[x_0,\ldots,x_n]}\Big) \;=\; f_{[x_0,\ldots,x_{n+1}]}
\end{aligned}$$

wobei wir im vorletzten Schritt die Induktionsannahme und im letzten Schritt die Definition der dividierten Differenzen verwendet haben. □

Diese Formel kann als alternative Definition der dividierten Differenzen verwendet werden. Beachte, dass sich der Integralausdruck bei Umnummerierung der Stützstellen nicht ändert, bei dieser alternativen Definition also keine aufsteigende Sortierung benötigt wird.

Wir formulieren zwei Konsequenzen aus Lemma 4.15.

Korollar 4.16 Für $f \in C^n[a,b]$ gelten die folgenden Aussagen.

(i) Die Funktion $g : \mathbb{R}^{n+1} \to \mathbb{R}$ gegeben durch

$$g(x_0, \ldots, x_n) = f_{[x_0,\ldots,x_n]}$$

ist stetig.

(ii) Es existiert ein $\xi \in [a, b]$, so dass

$$f_{[x_0,\ldots,x_n]} = \frac{f^{(n)}(\xi)}{n!}.$$

Beweis: (i) Die Stetigkeit ist aus der Integraldarstellung aus Lemma 4.15 unmittelbar klar.

(ii) Aus der Integraldarstellung 4.15 folgt mit dem Mittelwertsatz der Integralrechnung die Gleichung

$$f_{[x_0,\ldots,x_n]} = \int_{\Sigma^n} f^{(n)} \left(x_0 + \sum_{i=1}^{n} s_i(x_i - x_0) \right) ds = f^{(n)}(\xi) \int_{\Sigma^n} 1 \, ds$$

für ein geeignetes $\xi \in [a, b]$. Hieraus folgt die Behauptung mit (4.9). □

Nun ist es leicht, den Interpolationsfehler abzuschätzen.

Satz 4.17 Sei f $(n+1)$–mal stetig differenzierbar und sei P das Hermite–Interpolationspolynom zu den Stützstellen $x_0, \ldots, x_n$. Dann gelten die folgenden Ausagen.

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

$$f(x) - P(x) = f_{[x_0,\ldots,x_n,x]} \prod_{j=0}^{n} (x - x_j) = \frac{f^{(n+1)}(\xi)}{(n+1)!} \prod_{j=0}^{n} (x - x_j)$$

gilt.

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

$$|f(x) - P(x)| \leq \|f^{(n+1)}\|_\infty \left| \frac{\prod_{j=0}^{n}(x - x_j)}{(n+1)!} \right|.$$

(iii) Es gilt die Abschätzung

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

Beweis: Die Gleichungen in (i) folgen direkt aus Lemma 4.14 und Korollar 4.16(ii). Die Ungleichungen (ii) und (iii) folgen mit der Definition der Maximumsnorm. □

Bemerkung 4.18 Für den Fall, dass alle x_i übereinstimmen, erhält man aus Satz 4.17(i) und (4.8) die Gleichung

$$f(x) - \sum_{j=0}^{n} \frac{(x-x_0)^j}{j!} f^{(j)}(x_0) = \frac{f^{(n+1)}(\xi)}{(n+1)!}(x-x_0)^{n+1},$$

also gerade das Lagrange'sche Restglied der Taylorentwicklung. □

Wir illustrieren Satz 4.17 an 2 Beispielen.

Beispiel 4.19 Betrachte die Funktion $f(x) = \sin(x)$ auf dem Intervall $[0, 2\pi]$. Die Ableitungen von "f" sind

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

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

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

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

Beispiel 4.20 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

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

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. □

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 gegebenenes 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 4.21 Sei $\omega : (a, b) \to \mathbb{R}^+$ eine positive und auf $[a, b]$ Lebesgue–integrierbare *Gewichtsfunktion*[2]. 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 4.22 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 ω, 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. □

[2] Beachte, dass ω an den Randpunkten a und b nicht definiert sein muss, wenn wir das Lebesgue–Integral verwenden.

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

Satz 4.23 Zu jeder Gewichtsfunktion $\omega : [a,b] \to \mathbb{R}^+$ gibt es eindeutig bestimmte Orthogonalpolynome $(P_k)_{k\in\mathbb{N}_0}$ gemäß Definition 4.22 mit führendem Koeffizienten = 1. Sie erfüllen die *Rekursionsgleichung*

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

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

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

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 $k-1 \to k$ seien $P_0, P_1, \ldots, P_{k-1}$ 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 $P_k - xP_{k-1} \in \mathcal{P}_{k-1}$. Da die $P_0, P_1, \ldots, P_{k-1}$ wegen der Orthgonalität linear unabhängig 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, $j = 0, \ldots, k-1$ 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 $j = 0, \ldots, k-3$ liegt $xP_j \in \mathcal{P}_{k-2}$, lässt sich also als Linearkombination von $P_0, \ldots, P_{k-2}$ ausdrücken, weswegen $\langle P_{k-1},\, xP_j\rangle_\omega = 0$ und damit $d_0 = d_1 = \ldots = d_{k-3} = 0$ 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. □

Bemerkung 4.24 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 Deuflhard/Hohmann [2] betrachtet. □

Satz 4.23 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

$$T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x). \tag{4.10}$$

mit Anfangswerten $T_0(x) = 1$, $T_1(x) = x$ (wir skalieren hier etwas anders als in Satz 4.23, was im Folgenden einige Rechnungen vereinfacht). Das folgende Lemma fasst die Eigenschaften der zugehörigen orthogonalen Polynome, der sogenannten *Tschebyscheff–Polynome* zusammen.

Satz 4.25 Für die durch die Rekursion (4.10) gegebenen Tschebyscheff–Polynome T_k, $k = 0, 1, 2, \ldots$, 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

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

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

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

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 4.17(ii). Dort haben wir die Ungleichung

$$|f(x) - P(x)| \leq \|f^{(n+1)}\|_\infty \left| \frac{(x-x_0)(x-x_1)\cdots(x-x_n)}{(n+1)!} \right|. \tag{4.11}$$

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 $x_0, \ldots, x_n$ zu finden, so dass der Ausdruck

$$\max_{x\in[a,b]} |(x-x_0)(x-x_1)\cdots(x-x_n)| \tag{4.12}$$

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 $x_0, \ldots, x_n$ $R_{n+1}(x) = (x-x_0)(x-x_1)\cdots(x-x_n)$, 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 (4.12) ist gerade die Maximumsnorm $\|R_{n+1}\|_\infty$. Die Minimierung von (4.12) 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 4.26 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

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

also die Nullstellen von T_{n+1}, den Ausdruck (4.12) und damit die Fehlerabschätzung (4.11).

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 $\bar{x}_0, \ldots, \bar{x}_{n+1}$ 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

$$\|Q_{n+1}\|_\infty \geq \|T_{n+1}\|_\infty \tag{4.13}$$

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

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

Also wechselt $Q_{n+1} - T_{n+1}$ in jedem der $n+1$ Intervalle $[\bar{x}_i, \bar{x}_{i+1}]$, $i = 0, \ldots, n$ 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 (4.13). □

Für die Tschebyscheff–Stützstellen x_i erhält man damit in (4.12)

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

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 4.20 erhalten wir damit aus Satz 4.17(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 $n \to \infty$ 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 $\{T_0, \ldots, T_n\}$ von $\mathcal{P}_n$ auszudrücken, d.h. man kann Koeffizienten γ_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 $x_0 < x_1 < \ldots < x_n$ 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 zusammengefügten stückweisen Polynomen besteht, nennt man *Spline*.

Formal wird dies durch die folgende Definition präzisiert.

Definition 4.27 Seien $x_0 < x_1 < \ldots < x_n$ Stützstellen und $k \in \mathbb{N}$. Eine stetige und $(k-1)$–mal stetig differenzierbare Funktion $S : [x_0, x_n] \to \mathbb{R}$ heißt *Spline vom Grad k*, falls S auf jedem Intervall $I_i = [x_{i-1}, x_i]$ mit $i = 1, \ldots, n$ durch ein Polynom P_i vom Grad $\leq k$ gegeben ist, d.h. für $x \in I_i$ gilt

$$S(x) = P_i(x) = a_{i\,0} + a_{i\,1}(x - x_{i-1}) + \ldots + a_{i\,k}(x - x_{i-1})^k = \sum_{j=0}^{k} a_{i\,j}(x - x_{i-1})^j.$$

Den Raum der Splines vom Grad k zur Stützstellenmenge $\Delta = \{x_0, x_1, \ldots, x_n\}$ bezeichnen wir mit $S_{\Delta,k}$. □

Ein solcher Spline aus Definition 4.27 löst dann das Interpolationsproblem, falls zusätzlich die Bedingung (4.1) erfüllt ist, also $S(x_i) = f_i$ für alle $i = 0, \ldots, n$ gilt.

Bevor wir an die Berechnung von Splines gehen, zeigen wir eine Eigenschaft des Funktionenraums $S_{\Delta,k}$.

Satz 4.28 Sei $\Delta = \{x_0, x_1, \ldots, x_n\}$ mit $x_0 < x_1 < \ldots < x_n$ 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 $j = 0, \ldots, k-1$ bereits festgelegt, da die zusammengesetzte Funktion S in x_{i-1} ja stetig und $k-1$–mal stetig differenzierbar ist, es muss also

$$a_{ij} = P_{i-1}^{(j)}(x_{i-1})/j! \quad \text{für } j = 0, \ldots, k-1$$

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 4.29 Tatsächlich kann man beweisen, dass die Spline–Koeffizienten a_{ij} linear voneinander abhängen, statt der im Beweis des Satzes verwendeten Koeffizienten $a_{10}, \ldots, a_{1k}, a_{2k}, \ldots, a_{nk}$ 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 $x_0 < x_1 < \ldots < x_n$ betrachten. Offenbar erfüllt der Spline genau dann das Interpolationsproblem (4.1), wenn die Bedingungen

$$\begin{aligned} &a_{i0} = f_{i-1} \text{ für } i = 1, \ldots, n \\ &\text{und} \\ &a_{n1} = \frac{f_n - a_{n0} - a_{n2}(x_n - x_{n-1})^2 - a_{n3}(x_n - x_{n-1})^3}{x_n - x_{n-1}} \end{aligned} \tag{4.14}$$

erfüllt sind, womit genau $n+1$ Koeffizienten festgelegt sind. Da wir nach Bemerkung 4.29 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 4.30 Gegeben seien Daten (x_i, f_i) für $i = 0, \ldots, n$ mit $x_0 < x_1 < \ldots < x_n$ und $\Delta = \{x_0, x_1, \ldots, x_n\}$. 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 (4.14) genau $n+3$ Koeffizienten festgelegt, für die es nach Bemerkung 4.29 genau einen Spline gibt, der diese Bedingungen erfüllt. □

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 $g : [x_0, x_n] \to \mathbb{R}$ 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 $y : [x_0, x_n] \to \mathbb{R}$ über dem gesamten Intervall kann also mittels $\|y''\|_2$ gemessen werden. Hierfür gilt der folgende Satz.

Satz 4.31 Sei $S : [x_0, x_n] \to \mathbb{R}$ ein die Daten (x_i, f_i), $i = 0, \ldots, n$ interpolierender kubischer Spline, der eine der Randbedingungen (a)–(c) aus Lemma 4.30 erfüllt. Sei $y : [x_0, x_n] \to \mathbb{R}$ 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

$$\begin{aligned} \|y''\|_2^2 &= \int_{x_0}^{x_n} (y''(x))^2 dx \\ &= \underbrace{\int_{x_0}^{x_n} (S''(x))^2 dx}_{=\|S''\|_2^2} + \underbrace{2\int_{x_0}^{x_n} S''(x)(y''(x) - S''(x))dx}_{=:J} + \underbrace{\int_{x_0}^{x_n} (y''(x) - S''(x))^2 dx}_{\geq 0} \\ &\geq \|S''\|_2^2 + J. \end{aligned}$$

Wir betrachten nun den Term J genauer. Aus jeder der drei Randbedingungen folgt die Gleichung

$$\Big[S''(x)(y'(x)-S'(x))\Big]_{x=x_0}^{x_n} = S''(x_n)(y'(x_n)-S'(x_n))-S''(x_0)(y'(x_0)-S'(x_0)) = 0. \quad (4.15)$$

Mit partieller Integration gilt

$$\int_{x_0}^{x_n} S''(x)(y''(x)-S''(x))dx = \Big[S''(x)(y'(x)-S'(x))\Big]_{x=x_0}^{x_n} - \int_{x_0}^{x_n} S'''(x)(y'(x)-S'(x))dx$$

Hierbei ist der erste Summand wegen (4.15) 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. □

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 $i = 1, \ldots, n$ 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 Deuflhard/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 4.27 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 $i = 0, \ldots, n$ bzw. $i = 1, \ldots, n$. Aus der Interpolationsbedingung und der geforderten Stetigkeit der zweiten Ableitung erhält man 4 Gleichungen für die Teilpolynome P_i:

$$P_i(x_{i-1}) = f_{i-1}, \;\; P_i(x_i) = f_i, \;\; P_i''(x_{i-1}) = f_{i-1}'', \;\; P_i''(x_i) = f_i''. \tag{4.16}$$

Löst man diese — unter Ausnutzung der Ableitungsregeln für Polynome — nach den $a_{i\,j}$ auf, so erhält man

$$\begin{aligned} a_{i\,0} &= f_{i-1} \\ a_{i\,1} &= \frac{f_i - f_{i-1}}{h_i} - \frac{h_i}{6}(f_i'' + 2f_{i-1}'') \\ a_{i\,2} &= \frac{f_{i-1}''}{2} \\ a_{i\,3} &= \frac{f_i'' - f_{i-1}''}{6h_i}. \end{aligned}$$

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 $f_1'', \ldots, f_{n-1}''$ berechnet werden.

Beachte, dass wir in (4.16) 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

$$a_{i\,1} + 2a_{i\,2}(x_i - x_{i-1}) + 3a_{i\,3}(x_i - x_{i-1})^2 = a_{i+1\,1}$$

für $i = 1, \ldots, n-1$. 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 $i = 1, \ldots, n-1$. Dies liefert genau $n-1$ Gleichungen für die $n-1$ Unbekannten $f_1'', \ldots, f_{n-1}''$. 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 $a_{i\,j}$ aus den f_k''.

Für äquidistante Stützstellen, also $x_k - x_{k-1} = h_k = h$ für alle $k = 1, \ldots, n$, 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 4.32 Sei $S \in S_{\Delta,3}$ der interpolierende Spline einer 4 mal stetig differenzierbaren Funktion f mit hermite'schen Randbedingungen und Stützstellen $\Delta = \{x_0, \ldots, x_n\}$. 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 $f : \mathbb{R} \to \mathbb{R}$ 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 α_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 α_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 $x_0, \ldots, x_n$ 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 α_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 α_i liefert dann

$$\alpha_i = \frac{1}{b-a} \int_a^b L_i(x) dx. \tag{5.3}$$

Diese α_i können dann explizit berechnet werden, denn die Integrale über die Lagrange–Polynome L_i sind explizit lösbar. Hierbei hängen die Gewichte α_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 $n = 0, \ldots, 7$ 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	α_0	α_1	α_2	α_3	α_4	α_5	α_6	α_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 α_i die Gewichte, die gemäß (5.3) zu den Stützstellen $a \leq x_0 < \ldots < x_n \leq b$ berechnet wurden, sei $h := (b-a)/n$ und $z_i := n(x_i - a)/(b-a) \in [0, n]$ für $i = 0, \ldots, n$. Dann gilt:

(i) Es gibt nur von $z_0, \ldots, z_n$ 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 $i = 0, \ldots, n$ (z.B. äquidistante Stützstellen) gibt es nur von $z_0, \ldots, z_n$ 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 Lemma 4.14 wissen wir

$$f(x) = P(x) + f_{[x_0,\ldots,x_n,x]} \prod_{i=0}^{n} (x - x_i),$$

wobei aus Korollar 4.16(ii) für alle $x \in [a,b]$ die Abschätzung

$$|f_{[x_0,\ldots,x_n,x]}| \leq \frac{\|f^{(n+1)}\|_\infty}{(n+1)!}$$

folgt. Also folgt

$$\begin{aligned} |F_n[f]| &= \left| \int_a^b f_{[x_0,\ldots,x_n,x]} \prod_{i=0}^{n} (x - x_i) dx \right| \\ &\leq \frac{1}{(n+1)!} \left\| f^{(n+1)} \right\|_\infty \int_a^b \prod_{i=0}^{n} |x - x_i| \, dx, \end{aligned}$$

Daraus folgt die behauptete Abschätzung mit

$$\begin{aligned} c_n &= \frac{1}{(n+1)!}\frac{1}{h^{n+2}}\int_a^b \prod_{i=0}^n |x-x_i|\,dx &&= \frac{1}{(n+1)!}\left(\frac{n}{b-a}\right)^{n+2}\int_a^b \prod_{i=0}^n |x-x_i|\,dx \\ \left(\text{Substitution: } z = n\tfrac{x-a}{b-a},\ \ z_i = n\tfrac{x_i-a}{b-a}\right) &&= \frac{1}{(n+1)!}\int_0^n \prod_{i=0}^n |z-z_i|\,dz \end{aligned}$$

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 gilt

$$Q(x) = P(x) + Q_{[x_0,\ldots,x_n,x]}\prod_{i=0}^n (x-x_i).$$

Nach Korollar 4.16(ii) existert für jedes $x \in [a,b]$ eine Stelle $\xi \in [a,b]$, so dass

$$Q_{[x_0,\ldots,x_n,x]} = \frac{Q^{(n+1)}(\xi)}{(n+1)!} =: \gamma.$$

Weil nun aber $Q^{(n+1)}$ ein Polynom vom Grad 0 und damit konstant ist, ist γ unabhängig von ξ und ξ ist für jedes x beliebig. Folglich ist γ unabhängig von x und es 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 $Q \in \mathcal{P}_{n+1}$ das hermit'sche Interpolationspolynom zu den Stützstellen $x_0,\ldots,x_{n/2-1},x_{n/2},x_{n/2},x_{n/2+1},\ldots,x_n$, also mit doppelter

Stützstelle $x_{n/2}$. Dann stimmen die Interpolationspolynome $P \in \mathcal{P}_n$ für f und Q zu den (einfachen) Stützstellen x_i überein und es gilt

$$\begin{aligned} F_n[f] &= \int_a^b f(x)dx - \int_a^b P(x)dx \\ &= \int_a^b f(x)dx - \int_a^b Q(x)dx + \underbrace{\int_a^b Q(x)dx - \int_a^b P(x)dx}_{=0} \\ &= \int_a^b f(x)dx - \int_a^b Q(x)dx \\ &= \int_a^b f_{[x_0,\ldots,x_{n/2-1},x_{n/2},x_{n/2},x_{n/2+1},\ldots,x_n,x]}(x-x_{n/2})\prod_{i=0}^{n}(x-x_i)dx \end{aligned}$$

Dieser Integralausdruck kann nun analog zu Teil (i) durch

$$\left|\int_a^b f_{[x_0,\ldots,x_{n/2-1},x_{n/2},x_{n/2},x_{n/2+1},\ldots,x_n,x]}(x-x_{n/2})\prod_{i=0}^{n}(x-x_i)dx\right| \leq d_n h^{n+3}\|f^{(n+2)}\|_\infty$$

mit

$$d_n = \frac{1}{(n+2)!}\int_0^n |z-z_{n/2}|\prod_{i=0}^{n}|z-z_i|dz$$

abgeschätzt werden. □

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

$$\begin{aligned} c_1 &= \frac{1}{(1+1)!}\int_0^1 \prod_{i=0}^{1}|z-z_i|\,dz = \frac{1}{2}\int_0^1 (z-0)(1-z)dz = \frac{1}{2}\int_0^1 z-z^2dz \\ &= \frac{1}{2}\left[\frac{1}{2}z^2 - \frac{1}{3}z^3\right]_0^1 = \frac{1}{2}\left(\frac{1}{2}-\frac{1}{3}\right) = \frac{1}{12}. \end{aligned}$$

In Tabelle 5.2 sind die mit dieser Technik berechneten Fehlerabschätzungen für $n = 1, \ldots, 7$ und äquidistante Stützstellen approximativ angegeben, wobei $M_n := \|f^{(n)}\|_\infty$ ist.

n	1	2	3	4	5	6	7
	$\frac{(b-a)^3M_2}{12}$	$\frac{(b-a)^5M_4}{2880}$	$\frac{(b-a)^5M_4}{6480}$	$\frac{5.2(b-a)^7M_6}{10^7}$	$\frac{2.9(b-a)^7M_6}{10^7}$	$\frac{6.4(b-a)^9M_8}{10^{10}}$	$\frac{3.9(b-a)^9M_8}{10^{10}}$

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 α_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–Formel 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

$$x_i = a + ih, \;\; i = 0, 1, \ldots, nN, \quad h = \frac{b-a}{nN}$$

und zerlegen das Integral (5.1) mittels

$$\int_a^b f(x)dx \;\; = \;\; \int_{x_0}^{x_n} f(x)dx + \int_{x_n}^{x_{2n}} f(x)dx + \cdots + \int_{x_{(N-1)n}}^{x_{Nn}} f(x)dx = \sum_{k=1}^{N} \int_{x_{(k-1)n}}^{x_{kn}} f(x)dx.$$

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 Approximationsfehler

$$F_{N,n}[f] := \int_a^b f(x)dx - nh\sum_{k=1}^{N}\sum_{i=0}^{n}\alpha_i f(x_{(k-1)n+i})$$

ergibt sich einfach als Summe der Fehler $F_n[f]$ auf den Teilintervallen, weswegen man aus Satz 5.1(i) die Abschätzung

$$\begin{aligned}|F_{N,n}[f]| &\le \sum_{k=1}^{N} c_n h^{n+2}\max_{y\in[x_{(k-1)n},x_{kn}]}|f^{(n+1)}(y)|\\ &\le Nc_nh^{n+2}\|f^{(n+1)}\|_\infty \;=\; \frac{c_n}{n}(b-a)h^{n+1}\|f^{(n+1)}\|_\infty\end{aligned}$$

und aus Satz 5.1(ii) für gerades n die Abschätzung

$$|F_{N,n}[f]| \le \frac{d_n}{n}(b-a)h^{n+2}\|f^{(n+2)}\|_\infty$$

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:**

$$\int_a^b f(x)dx \;\approx\; \frac{h}{2}\left(f(a) + 2\sum_{i=1}^{N-1} f(x_i) + f(b)\right)$$

$$|F_{N,1}[f]| \le \frac{b-a}{12}h^2\|f^{(2)}\|_\infty, \quad h = \frac{b-a}{N}$$

$n = 2$**, Simpson–Regel:**

$$\int_a^b f(x)dx \;\approx\; \frac{h}{3}\left(f(a) + 2\sum_{i=1}^{N-1} f(x_{2i}) + 4\sum_{i=0}^{N-1} f(x_{2i+1}) + f(b)\right)$$

$$|F_{N,2}[f]| \le \frac{b-a}{180}h^4\|f^{(4)}\|_\infty, \quad h = \frac{b-a}{2N}$$

$n = 4$**, Milne–Regel:**

$$\begin{aligned}\int_a^b f(x)dx \;\approx\; \frac{2h}{45}\Bigg(&7\Big(f(a)+f(b)\Big) + 14\sum_{i=1}^{N-1} f(x_{4i})\\ &+\; 32\sum_{i=0}^{N-1}\Big(f(x_{4i+1})+f(x_{4i+3})\Big) + 12\sum_{i=0}^{N-1} f(x_{4i+2})\Bigg)\end{aligned}$$

$$|F_{N,4}[f]| \leq \frac{2(b-a)}{945} h^6 \|f^{(6)}\|_\infty, \quad h = \frac{b-a}{4N}$$

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

$$f^{(2)}(x) = (x^2-1)f(x), \;\; f^{(4)}(x) = (3-6x^2+x^4)f(x), \;\; f^{(6)}(x) = (-15+45x^2-15x^4+x^6)f(x).$$

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

$$\|f^{(2)}\|_\infty = 1, \;\; \|f^{(4)}\|_\infty = 3 \;\; \text{und } \|f^{(6)}\|_\infty = 15$$

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. □

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 α_0 und α_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

$$\begin{aligned}
\alpha_0 &= \frac{1}{2}, \qquad x_0 = -\frac{b-a}{2}\frac{1}{\sqrt{3}} + \frac{b+a}{2}\\
\alpha_1 &= \frac{1}{2}, \qquad x_1 = +\frac{b-a}{2}\frac{1}{\sqrt{3}} + \frac{b+a}{2}
\end{aligned}$$

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 $\omega : (a,b) \to \mathbb{R}$ eine nichtnegative *Gewichtsfunktion* ist. Beachte, dass der Vorfaktor $(b-a)$ aus den Newton–Cotes–Formeln (5.2) hier in die Gewichte λ_i einbezogen ist. Das

bisher behandelte Integrationsproblem ohne Gewichtsfunktion ist hier durch die Spezialfall $\omega(x) \equiv 1$ gegeben, es sind aber auch andere ω 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 $x_0, \ldots, x_n$ sind die Nullstellen des orthogonalen Polynoms P_{n+1} gemäß Definition 4.22 bezüglich der Gewichtsfunktion ω.

(2) Die Gewichte λ_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 $P_0, \ldots, P_n$ schreiben. Aus $\langle P_k, P_{n+1}\rangle_\omega = 0$ für $k = 0, \ldots, n$ 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. □

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. □

Wir beenden diesen Abschnitt mit einigen Beispielen von Gauß–Quadratur–Formeln.

Beispiel 5.5 (i) Für $\omega(x) = 1$ und Integrationsintevall $[-1, 1]$ 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. Deuflhard/Hohmann [2], Abschnitt 9.3.2. □

Die Gauß–Legendre Integration aus (i) wird heutzutage eher selten verwendet, da die Romberg–Extrapolation aus dem nachfolgenden Abschnitt meist effizienter ist. Die Gauß–Quadratur ist aber nützlich, falls explizit mit vorgegebenen Gewichtsfunktionen wie in (ii) oder auf unendlichem Zeithorizont wie in (iii) integriert werden soll.

5.4 Romberg–Extrapolation

Bisher haben wir Integrationsformeln betrachtet, bei denen wir explizit die Anzahl der Stützstellen vorgegeben haben. In diesem und dem folgenden Abschnitt werden wir Verfahren betrachten, bei der die Anzahl der Stützstellen variabel ist.

Diese Verfahren beruhen auf der (zusammengesetzten) Trapez–Regel, die wir hier für $h = (b-a)/N$ mit $T(h)$ bezeichnen:

$$T(h) = \frac{h}{2}\left(f(a) + 2\sum_{j=1}^{N-1} f(a+jh) + f(b) \right).$$

Für diese Formel gilt ein von Euler und McLaurin (unabhängig voneinander) bewiesener Satz, der über die bisher betrachteten Fehlerabschätzungen deutlich hinaus geht.

Satz 5.6 Sei $f \in C^{2m+1}([a,b])$ und $h = (b-a)/N$ für ein $N \in \mathbb{N}$. Dann gilt für die Trapez–Regel die Gleichung

$$T(h) = \int_a^b f(x)dx + \tau_2 h^2 + \tau_4 h^4 + \ldots + \tau_{2m} h^{2m} + R_{2m+2}(h)h^{2m+2}$$

mit den Koeffizienten

$$\tau_{2k} = \frac{B_{2k}}{(2k)!}\left(f^{(2k-1)}(b) - f^{(2k-1)}(a) \right),$$

wobei B_{2k} die sogenannten *Bernoulli–Zahlen* sind. Hierbei ist der Restterm gleichmäßig beschränkt durch

$$|R_{2m+2}(h)| \leq C_{2m+2}(b-a)\,\|f^{(2m)}\|_\infty,$$

wobei C_{2m+2} eine von h unabhängige Konstante ist.

Der formale Beweis dieses Satzes ist recht kompliziert, weswegen wir ihn hier nicht ausführen. Eine Beweisskizze findet sich z.B. im Buch von Stoer [9], Abschnitt 3.3.

Der wichtige Aspekt in dieser Formel ist die Tatsache, dass die Koeffizienten τ_{2k} nicht von h (und damit nicht von der Anzahl der Stützstellen) abhängen, lediglich der Restterm hängt von h ab. Die Funktion $T(h)$ kann also als ein (durch den Restterm $R_{2m+2}(h)h^{2m+2}$) gestörtes Polynom aufgefasst werden. Die explizite Gestalt der Bernoulli–Zahlen werden wir bei der Anwendung dieser Formel nicht benötigen. Für große k kann gezeigt werden, dass

$$B_{2k} \approx (2k)!$$

gilt, weswegen die Reihe (im Gegensatz z.B. zur Taylor–Reihe) für $m \to \infty$ i.A. nicht konvergiert, wenn die höheren Ableitungen von f wachsen und/oder h groß ist. Der obige Ausdruck ist aber sinnvoll für endliches m und kleine h.

Um das Extrapolationsverfahren in etwas allgemeinerem Rahmen darzustellen, verwenden wir die folgende Definition.

Definition 5.7 Es sei $T(h)$ ein numerisches Verfahren zur Approximation des Wertes

$$\tau_0 = \lim_{h \to 0} T(h).$$

Eine *asymptotische Entwicklung in h^p* dieses Verfahrens bis zur *Ordnung pm* ist eine Darstellung der Form

$$T(h) = \tau_0 + \tau_p h^p + \tau_{2p} h^{2p} + \ldots + \tau_{mp} h^{mp} + O(h^{(m+1)p})$$

mit Konstanten τ_{ip}, $i = 0, \ldots, m$. Hierbei bezeichnet das *Landau–Symbol* $O(h^k)$ einen beliebigen Ausdruck mit der Eigenschaft, dass $O(h^k)/h^k \leq C$ ist für ein $C > 0$ und alle hinreichend kleinen $h > 0$. □

Nach Satz 5.6 besitzt die Trapezregel eine solche asymptotische Entwicklung in h^2 bis zur Ordnung $2m$.

Eine solche asymptotische Entwicklung ist die Grundlage für die *Extrapolation*. Extrapolation bezeichnet das Verfahren, bei dem man ein Interpolationspolynom zu einer Funktion $g(x)$ und zu Stützstellen auf einem Intervall $[a, b]$ an einer Stelle $x^* \notin [a, b]$ auswertet. Dieses Verfahren werden wir wie folgt auf die Funktion $T(h)$ anwenden:

(1) Berechne die Approximationswerte $T(h_j)$ für m verschiedene Schrittweiten $h_1, \ldots, h_m$

(2) Berechne das Interpolationspolynom $P(h^p)$ zu den Daten $(h_i^p, T(h_i))$, $i = 1, \ldots, m$, und werte dieses in $h^p = 0$ aus.

Wir wollen also aus Werten von T zu großen Schrittweiten (also "groben" Approximationen des Integrals) einen approximativen Wert von T zur Schrittweite $h = 0$ berechnen, in der Hoffnung, dass dies eine genauere Approximation des Integrals liefert.

Wir illustrieren das Verfahren an einem einfachen Beispiel mit $m = 2$. Betrachte die Funktion $f(x) = x^4$ auf $[0, 1]$. Offenbar ist das gesuchte Integral gerade $\int_0^1 x^4 dx = 1/5 =: \tau_0$. Wir betrachten nun die Trapez–Regel für $N_1 = 1$ und $N_2 = 2$, also $h_1 = 1$ und $h_2 = 1/2$. Diese ergibt

$$T(1) = \frac{h_1}{2}(f(0) + f(1)) = \frac{1}{2}(0 + 1) = \frac{1}{2}$$

und

$$T\left(\frac{1}{2}\right) = \frac{h_2}{2}\left(f(0) + 2f\left(\frac{1}{2}\right) + f(1)\right) = \frac{1}{4}\left(0 + \frac{2}{16} + 1\right) = \frac{9}{32}.$$

Der Fehler ist also $|1/5 - 1/2| = 3/10 = 0.3$ bzw. $|1/5 - 9/32| = 13/160 = 0.08125$.

Wenn wir T durch ein Polynom P in h^2 an den Stellen h_1^2 und h_2^2 interpolieren, so ergibt sich P gerade zu

$$P(h^2) = T(h_1) + \frac{7}{24}(h^2 - h_1^2),$$

wie man durch Nachrechnen leicht überprüft. Ausgewertet in $h^2 = 0$ erhalten wir

$$P(0) = \frac{1}{2} + \frac{7}{24}(-1) = \frac{5}{24},$$

was wegen $|1/5 - 5/24| = 1/120 = 0.008\overline{3}$ eine deutlich bessere Approximation des Integralwertes als $T(h_1)$ bzw. $T(h_2)$ liefert.

Bevor wir die so erzielbare Verbesserung in Satz 5.8 genau untersuchen, wollen wir uns mit der Implementierung beschäftigen. Um dieses Verfahren effizient programmieren zu können, benötigen wir einen Algorithmus, mit dem der Wert eines Interpolationspolynoms an einer vorgegebenen Stelle schnell berechnet werden kann. Durch Überprüfen der Interpolationseigenschaft beweist man für $k \geq 2$ die Gleichung

$$P_{i,k}(x) = \frac{(x_{i-k+1} - x)P_{i,k-1}(x) - (x_i - x)P_{i-1,k-1}(x)}{x_{i-k+1} - x_i},$$

die auch als *Lemma von Aitken* bekannt ist. Hierbei ist $P_{i,k}(x)$ das Interpolationspolynom durch die Daten $(x_{i-k+1}, f_{i-k+1}), \ldots, (x_i, f_i)$. Mit dieser Formel angewendet mit $x = 0$ und den Werten $(x_i, f_i) = (h_i^p, T(h_i))$ rechnet man mittels Induktion leicht nach, dass die Werte $T_{i,k} = P_{i,k}(0)$ durch die rekursive Vorschrift

$$\begin{aligned} T_{i,1} &= T(h_i), \qquad i = 1, 2, \ldots \\ T_{i,k} &= T_{i,k-1} + \frac{T_{i,k-1} - T_{i-1,k-1}}{\left(\frac{h_{i-k+1}}{h_i}\right)^p - 1}, \quad k = 2, 3, \ldots; \;\; i = k, k+1, \ldots \end{aligned}$$

gegeben sind. Die Berechnung, die als *Extrapolationsschema* bezeichnet wird, lässt sich ganz analog zur Berechnung der dividierten Differenzen grafisch darstellen, siehe Abb. 5.1 (vgl. auch Abb. 4.1).

Beachte, dass k hier durch Anhängen weiterer Zeilen beliebig erhöht werden kann, wobei die zuvor berechneten Werte für größere k weiterhin benutzt werden können. Der nächste Satz gibt die Genauigkeit der Approximation $T_{k,k}$ an.

Satz 5.8 Sei $T(h)$ ein Verfahren mit einer asymptotischen Entwicklung gemäß Definition 5.7 in h^p bis zur Ordnung pm. Dann gilt für alle hinreichend kleinen Schrittweiten $h_1, \ldots, h_m$ für den Approximationsfehler

$$\varepsilon_{i,k} := |T_{i,k} - \tau_0|$$

$$
\begin{array}{ccccccccc}
T_{1,1} & & & & & & & & \\
 & \searrow & & & & & & & \\
T_{2,1} & \rightarrow & T_{2,2} & & & & & & \\
 & \searrow & & \searrow & & & & & \\
T_{3,1} & \rightarrow & T_{3,2} & \rightarrow & T_{3,3} & & & & \\
 & \searrow & & \searrow & & \searrow & & & \\
T_{4,1} & \rightarrow & T_{4,2} & \rightarrow & T_{4,3} & \rightarrow & T_{4,4} & & \\
\vdots & & \vdots & & \vdots & & \vdots & \ddots &
\end{array}
$$

Abbildung 5.1: Illustration des Extrapolationsschemas

die Gleichung

$$\varepsilon_{i,k} = |\tau_{kp}| h_{i-k+1}^p \dots h_i^p + \sum_{j=i-k+1}^{i} O(h_j^{(k+1)p}).$$

Beweis: Wir beweisen zunächst die folgende Gleichung für die Lagrange–Polynome L_i zu Stützstellen $x_1, \dots, x_n$.

$$\sum_{i=1}^{n} L_i(0) x_i^m = \begin{cases} 1, & \text{falls } m = 0, \\ 0, & \text{falls } 1 \le m \le n-1 \\ (-1)^{n-1} x_1 \cdots x_n, & \text{falls } m = n \end{cases} \tag{5.4}$$

Zum Beweis von (5.4) betrachten wir das Polynom $P(x) = x^m$. Offenbar ist dies für $m \le n-1$ gerade das Interpolationspolynom zu den Daten (x_i, x_i^m), $i = 1, \dots, n$: das Polynom verläuft durch diese $n-1$ Punkte und ist vom Grad $\le n-1$. Daher gilt

$$P(x) = x^m = \sum_{i=1}^{n} L_i(x) P(x_i) = \sum_{i=1}^{n} L_i(x) x_i^m.$$

Die Behauptung (5.4) für $m = 1, \dots, n-1$ folgt hieraus durch Einsetzen von $x = 0$.

Für $m = n$ verläuft das Polynom $x^m = x^n$ zwar ebenfalls durch die angegebenen $n-1$ Punkte, ist aber vom Grad n und ist damit, weil es zu hohen Grad besitzt, nicht das eindeutige Interpolationspolynom. Daher betrachten wir in diesem Fall die Differenz zwischen x^n und dem eindeutigen Interpolationspolynom vom Grad $\le n-1$, also

$$Q(x) = x^n - \sum_{i=1}^{n} L_i(x) x_i^n.$$

Dieses Polynom hat den führenden Koeffizienten 1 und wegen $\sum_{i=1}^{n} L_i(x_j) x_i^n = x_j^n$ genau die n Nullstellen $x_1, \dots, x_n$, also folgt

$$Q(x) = (x - x_1) \cdots (x - x_n),$$

und damit

$$\sum_{i=1}^{n} L_i(0) x_i^n = -Q(0) = -(-x_1)\cdots(-x_n) = (-1)^{n-1} x_1 \cdots x_n,$$

also (5.4).

Zum Beweis des Satzes betrachten wir die asymptotische Entwicklung von $T(h)$. Daraus folgt für $j = 1, \ldots, m$ die Gleichung

$$T_{j,1} = T(h_j) = \tau_0 + \tau_p h_j^p + \ldots + \tau_{kp} h_j^{kp} + O(h_j^{(k+1)p}). \tag{5.5}$$

Wir zeigen die Behauptung für $i = k$, die Abschätzungen für $i > k$ folgen durch Umnummerierung der Schrittweiten h_i. Sei $P(h^p)$ das Interpolationspolynom in h^p zu den Daten $(h_1^p, T(h_1)), \ldots, (h_k^p, T(h_k))$. Seien darüberhinaus $L_1(x), \ldots, L_k(x)$ die Lagrange–Polynome zu den Stützstellen $h_1^p, \ldots, h_k^p$. Dann gilt

$$P(h^p) = \sum_{i=1}^{k} L_i(h^p) T_{i,1}.$$

Mit (5.4), angewendet auf $x_i = h_i^p$ für $i = 1, \ldots, k$ und (5.5) folgt

$$\begin{aligned} T_{k,k} &= P(0) = \sum_{i=1}^{k} L_i(0) T_{i,1} \\ &= \sum_{i=1}^{k} L_i(0) \Big(\tau_0 + \tau_p h_i^p + \ldots + \tau_{kp} h_i^{kp} + O(h_i^{(k+1)p}) \Big) \\ &= \tau_0 + \underbrace{\tau_{kp} (-1)^{k-1} h_1^p \cdots h_k^p + \sum_{i=1}^{k} O(h_i^{(k+1)p})}_{|\cdot| = \varepsilon_{k,k}}, \end{aligned}$$

und damit die Behauptung. □

Der Satz besagt: Wenn wir eine Zeile mit Schrittweite h_m zum Extrapolationsschema hinzufügen, so können wir erwarten, dass der Fehler um einen Faktor der Ordnung h_m^p abnimmt, im Trapezschema also um die Ordnung h_m^2. Beachte, dass diese Aussage nur dann gilt, wenn alle verwendeten Schrittweiten hinreichend klein sind. Anschaulich gesprochen müssen die Schrittweiten h_i in einem Bereich liegen, in dem die Werte $T(h_i)$ hinreichend viele Informationen über den Wert $\tau_0 = \lim_{h \to 0} T(h)$ liefern.

In der praktischen Implementierung wählt man die Schrittweitenfolge zweckmäßigerweise absteigend, also $h_{i+1} < h_i$. Darüberhinaus wird h_i typischerweise als Teil einer Basisschrittweite $h = (b-a)/N$ gewählt, also $h_i = h/N_i$ für $N_i \in \mathbb{N}$. Dies führt für das Trapezverfahren mit $p = 2$ zu dem folgenden Algorithmus.

Algorithmus 5.9 (Romberg–Extrapolationsverfahren)

(0) Wähle eine maximale Iterationszahl $i_{\max}$, eine Basisschrittweite $h = (b-a)/N$ und eine Folge von Schrittweiten $h_1, h_2, \ldots$ mit $h_i = h/N_i$, $N_{i+1} > N_i$. Setze $i := 1$.

(1) Berechne $T_{i,1} := T(h_i) = \frac{h_i}{2}\left(f(a) + 2\sum_{j=1}^{N\cdot N_i - 1} f(a + jh_i) + f(b)\right)$.

(2) Berechne

$$T_{i,k} := T_{i,k-1} + \frac{T_{i,k-1} - T_{i-1,k-1}}{\left(\frac{N_i}{N_{i-k+1}}\right)^2 - 1} \quad \text{für} \quad k = 2, \ldots, i$$

(3) Falls $i \geq i_{\max}$ oder $T_{i,i}$ genau genug ist, beende den Algorithmus, ansonsten setze $i = i + 1$ und gehe zu (1)

□

Das Abbruchkriterium "genau genug" ist natürlich nicht sehr präzise. Üblicherweise wird eine gewünschte Genauigkeit ε vorgegeben und so lange iteriert, bis $|T_{i+1,i+1} - T_{i,i}| < \varepsilon$ gilt. Ein relatives Abbruchkriterium ist z.B. gegeben durch $|T_{i+1,i+1} - T_{i,i}| < \varepsilon \cdot |T_{i+1,i+1}|$. Kennt man eine Schranke für das Integral

$$absint := \int_a^b |f(x)|dx,$$

so kann man alternativ iterieren, bis $|T_{i+1,i+1} - T_{i,i}| < \varepsilon \cdot absint$ gilt. Der Unterschied liegt darin, dass in dem zweiten Kriterium die integrierte Größe von $|f|$ und nicht der Wert des Integrals zur Gewichtung von ε dient, was bei wechselndem Vorzeichen von f ein leichter erfüllbares Kriterium liefert. Diese Abbruchkriterien funktionieren in der Praxis recht gut, gewährleisten aber nur bei hinreichend kleiner Basisschrittweite h, dass die gewünschte Genauigkeit auch tatsächlich erreicht wurde.

In der Implementierung spielt auch die Wahl der Folgen N_i eine wichtige Rolle. Für geschickt gewähltes N_i kann man bei der Berechnung von $T(h_i)$ auf vorher berechnete Funktionswerte $f(x_j)$ zurückgreifen. So gilt für jede Schrittweite $h = (b-a)/N$ die Gleichung

$$\begin{aligned} T(h/2) &= \frac{h}{4}\left(f(a) + 2\sum_{j=1}^{2N-1} f(a + jh/2) + f(b)\right) \\ &= \underbrace{\frac{h}{4}\left(f(a) + 2\sum_{j=1}^{N-1} f(a + jh) + f(b)\right)}_{=T(h)/2} + \frac{h}{2}\sum_{j=1}^{N} f(a + (2j-1)h/2), \end{aligned}$$

so dass zur Berechnung von $T(h/2)$ nur N Auswertungen von f nötig sind, wenn die übrigen $N-1$ Werte durch Verwendung von $T(h)$ "abgedeckt" werden. Dies kann man durch die Wahl $N_i = 2^{i-1}$ ausnutzen; diese Schrittweitenfolge wird als *Romberg–Folge* bezeichnet und üblicherweise in der Implementierung des Extrapolationsverfahrens verwendet. Bei dieser Wahl der N_i ersetzt man die Gleichung in Schritt (1) für $i \geq 2$ durch

$$T_{i,1} := \frac{1}{2}T_{i-1,1} + h_i \sum_{k=1}^{N\cdot N_{i-1}} f(a + (2k-1)h_i)$$

5.5 Adaptive Romberg–Quadratur

In allen bisherigen Verfahren haben wir die Stützstellen unabhängig vom Integranden f gewählt. Es ist jedoch naheliegend, dass für bestimmte f eine von f abhängige geschickte Platzierung der Stützstellen sehr viel bessere Ergebnisse liefert. Als Beispiel betrachte die *Nadelimpulsfunktion*

$$f(x) = \frac{1}{10^{-4} + x^2}$$

in Abbildung 5.2.

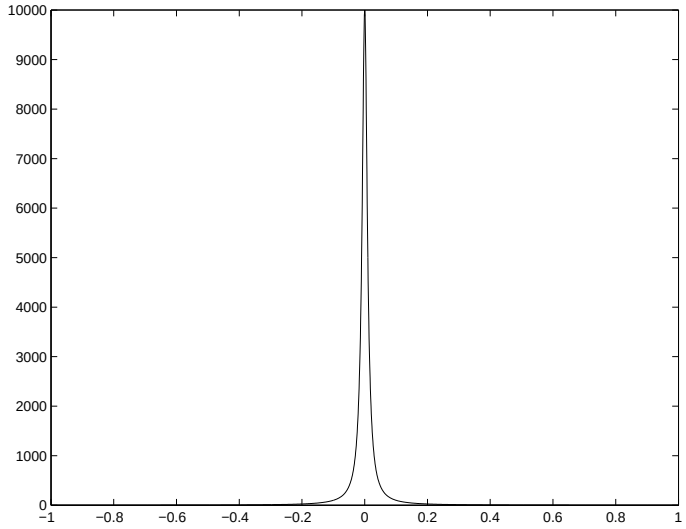

Abbildung 5.2: Plot der Nadelimpulsfunktion

Hier ist es offenbar sinnvoll, viele Stützstellen im Bereich nahe 0 zu konzentrieren und außerhalb dieses Bereichs nur relativ wenige Stützstellen zu verwenden.

Um Stützstellen variabel verteilen zu können, unterteilt man das Integrationsintervall in Teilintervalle und approximiert das Integral auf jedem Teilintervall durch eine numerische Formel. Bereiche, in denen die Teilintervalle klein sind, erhalten so mehr Stützstellen als Bereiche, in denen die Teilintervalle groß sind.

Die Idee der *adaptiven Integration* ist es nun, die Größe dieser Teilintervalle abhängig von f möglichst effizient zu wählen, sie also angepasst (= adaptiv) an f zu wählen. Dies soll ohne vorherige aufwändige Analyse von f und ohne weitere Informationen (wie Ableitungen von f etc.) geschehen, darüberhinaus soll die Routine schnell sein, damit der Aufwand zur Berechnung der Intervallgröße nicht den gewonnenen Vorteil zunichte macht, und sie soll zuverlässig, d.h., es soll garantiert sein, dass eine vorgegebene Fehlerschranke eingehalten wird.

Diese zwei letzten Kriterien — geringer Aufwand und hohe Zuverlässigkeit — widersprechen sich offenbar, denn je sicherer ein solches Verfahren sein soll, desto mehr Aufwand muss man bei der Berechnung der Intervallgröße betreiben. Ein guter Kompromiss zwischen Aufwand und Zuverlässigkeit wird durch die Idee der *a posteriori–Fehlerschätzung*

erzielt, die wir im Folgenden bespechen werden. Formal ist ein Fehlerschätzer durch die folgende Definition gegeben.

Definition 5.10 Eine numerisch berechenbare Größe $\bar{\varepsilon}$ heißt *Fehlerschätzer* für den tatsächlichen Fehler ε eines numerischen Verfahrens, falls von $\bar{\varepsilon}$ und ε unabhängige Konstanten $\kappa_1, \kappa_2 > 0$ existieren, so dass die Abschätzung

$$\kappa_1 \varepsilon \leq \bar{\varepsilon} \leq \kappa_2 \varepsilon$$

gilt. □

Solche Fehlerschätzer werden dann iterativ für alle Teilintervalle des Integrationsintervalls berechnet; dort, wo der geschätzte Fehler groß ist, werden kleine Teilintervalle gewählt, so dass sich dort mehr Stützstellen befinden und der Fehler dort kleiner wird.

Wir werden nun für das Extrapolationsverfahren einen solchen Fehlerschätzer $\bar{\varepsilon}$ herleiten. Wir betrachten dazu das Romberg–Verfahren mit Integrationsintervall $[a, b] = [x, x + h]$ und Basisschrittweite $h = b - a$, also $N = 1$. Auf diesem Intervall gilt nach Satz 5.8 die Approximation

$$\varepsilon_{i,k} = \left| T_{i,k} - \int_x^{x+h} f(y)dy \right| \approx |\tau_{2k}| h_{i-k+1}^2 \dots h_i^2. \tag{5.6}$$

Nach Satz 5.6 gilt für die Koeffizienten

$$\tau_{2k} = \frac{B_{2k}}{(2k)!} \underbrace{\left(f^{(2k-1)}(x+h) - f^{(2k-1)}(x) \right)}_{\approx f^{(2k)}(x) h} \approx \underbrace{\frac{B_{2k}}{(2k)!} f^{(2k)}(x)\, h}_{=:\bar{\tau}_{2k}}$$

wobei diese Approximation für kleine h gültig ist. Die approximativen Konstanten $\bar{\tau}_{2k}$ hängen dabei vom Integranden f ab. Setzen wir diese Approximation für τ_{2k} in (5.6) ein, so erhalten wir

$$\varepsilon_{i,k} \approx |\tau_{2k}| h_{i-k+1}^2 \dots h_i^2 \approx |\bar{\tau}_{2k}| \gamma_{i,k} h^{2k+1} \tag{5.7}$$

mit $\gamma_{i,k} = (N_{i-k+1} \cdots N_i)^{-2}$. Die hier auftretende Potenz $2k + 1$ von h hängt dabei gerade vom Spaltenindex k des Wertes im Extrapolationsschema ab. Innerhalb einer Spalte gilt die Abschätzung

$$\frac{\varepsilon_{i+1,k}}{\varepsilon_{i,k}} \approx \frac{\gamma_{i+1,k}}{\gamma_{i,k}} = \left(\frac{N_{i-k+1}}{N_{i+1}} \right)^2 << 1$$

(die Schreibweise $a << b$ bedeutet hierbei "a ist deutlich kleiner als b"). Mit anderen Worten, für kleines h (denn nur dafür gelten alle verwendeten Approximationen) nehmen die Integrationsfehler innerhalb einer Spalte schnell ab: es gilt

$$\varepsilon_{i+1,k} << \varepsilon_{i,k}. \tag{5.8}$$

Wir machen nun die *Annahme*, dass das Gleiche innerhalb der Zeilen gilt, wir nehmen also

$$\varepsilon_{i,k+1} << \varepsilon_{i,k}. \tag{5.9}$$

an. Die Annahme bedeutet, dass der Übergang zu einer höheren Ordnung $k + 1$ im Schema zu einem kleineren Fehler führt. Auch diese Ungleichung ist nach Satz 5.8 für hinreichend

kleine h erfüllt; sie gilt aber für in der Praxis verwendete h nicht unbedingt, was eine mögliche Fehlerquelle des Verfahrens darstellt.

Die Konstruktion eines Fehlerschätzers für einen Fehler $\varepsilon_{i,k}$ beruht nun darauf, dass man die Werte $T_{i,k-1}$ und $T_{i,k}$ vergleicht. Relativ zu der groben Approximation $T_{i,k-1}$ ist die genauere Approximation $T_{i,k}$ annähernd exakt, so dass

$$\varepsilon_{i,k-1} = \left| T_{i,k-1} - \int_x^{x+h} f(y)dy \right| \approx |T_{i,k-1} - T_{i,k}| \tag{5.10}$$

gelten sollte. Dieses Verfahren hat den großen Vorteil, dass man den Fehler aus den Werten berechnet, die im Schema sowieso schon vorhanden sind und daher keinen zusätzlichen Berechnungsaufwand benötigt.

Nehmen wir nun an, dass wir ein Extrapolationsdiagramm mit k Zeilen und k Spalten berechnet haben. Unter den Annahmen (5.8) und (5.9) sieht man leicht, dass in diesem Diagramm der Fehler $\varepsilon_{k,k}$ am kleinsten ist, also $T_{k,k}$ die genaueste Approximation darstellt. Es wäre also sinnvoll, einen Fehlerschätzer für $\varepsilon_{k,k}$ zu erhalten. Mit der obigen Idee ist dies aber nicht möglich, da wir dafür den Wert $T_{k,k+1}$ benötigen würden, der im Diagramm nicht enthalten ist und erst mit viel Aufwand berechnet werden müsste. Wir begnügen uns daher mit einem Schätzer für den "zweitbesten" Fehler $\varepsilon_{k,k-1}$. Das folgende Lemma zeigt, dass sich die informelle Approximation (5.10) formal in den Rahmen von Definition 5.10 fassen lässt.

Lemma 5.11 Es gelte Annahme (5.9) für $\varepsilon_{k,k-1}$ und $\varepsilon_{k,k}$, genauer existiere $\alpha < 1$, so dass die Ungleichung

$$\varepsilon_{k,k} \leq \alpha \varepsilon_{k,k-1}$$

Dann ist der Wert

$$\bar{\varepsilon}_{k,k-1} := |T_{k,k-1} - T_{k,k}|$$

ein Fehlerschätzer für $\varepsilon_{k,k-1}$ im Sinn von Definition 5.10 mit $\kappa_1 = 1 - \alpha$ und $\kappa_2 = 1 + \alpha$.

Beweis: Wir schreiben kurz $I = \int_x^{x+h} f(y)dy$. Damit gilt

$$\bar{\varepsilon}_{k,k-1} = |(T_{k,k-1} - I) - (T_{k,k} - I)| \leq \varepsilon_{k,k-1} + \varepsilon_{k,k}.$$

Andererseits folgt aus $\varepsilon_{k,k} < \varepsilon_{k,k-1}$

$$\bar{\varepsilon}_{k,k-1} = |(T_{k,k-1} - I) - (T_{k,k} - I)| \geq \varepsilon_{k,k-1} - \varepsilon_{k,k}.$$

Zusammen erhalten wir mit $\alpha \geq \varepsilon_{k,k}/\varepsilon_{k,k-1}$ die gewünschten Abschätzungen

$$(1-\alpha)\varepsilon_{k,k-1} \leq \left(1 - \frac{\varepsilon_{k,k}}{\varepsilon_{k,k-1}}\right) \varepsilon_{k,k-1} = \varepsilon_{k,k-1} - \varepsilon_{k,k} \leq \bar{\varepsilon}_{k,k-1}$$

und

$$\bar{\varepsilon}_{k,k-1} \leq \varepsilon_{k,k-1} + \varepsilon_{k,k} = \left(1 + \frac{\varepsilon_{k,k}}{\varepsilon_{k,k-1}}\right) \varepsilon_{k,k-1} \leq (1+\alpha)\varepsilon_{k,k-1}.$$

□

Auch wenn dieser Fehlerschätzer den Fehler für $T_{k,k-1}$ schätzt, sollte man in der praktischen Rechnung den Wert $T_{k,k}$ als Ergebnis des Algorithmus verwenden, da dieser nach Annahme (5.9) genauer als $T_{k,k-1}$ ist.

Auf Basis dieses Fehlerschätzers beschreiben wir nun den adaptiven Algorithmus. Wir betrachten hier eine einfache Variante, die mit fester vorgegebener Ordnung k arbeitet. Eine verfeinerte Variante, bei der auch die Ordnung k adaptiv gewählt wird, findet sich z.B. im Buch von Deuflhard und Hohmann [2].

Wir wollen das Integral

$$\int_a^b f(x)dx$$

mit der Extrapolationsformel für ein vorgegebenes k schrittweise berechnen, wobei wir auf jedem Teilintervall der Berechnung eine vorgegebene Genauigkeit *tol* einhalten wollen.[1] Dazu wählen wir eine Anfangsschrittweite $h_1 \leq b - a$, setzen $i := 1$ und $x_i := a$ und berechnen die Approximationen

$$T^i_{k,k-1} \approx \int_{x_i}^{x_i+h_i} f(x)dx \quad \text{und} \quad T^i_{k,k} \approx \int_{x_i}^{x_i+h_i} f(x)dx \tag{5.11}$$

des Teilintegrals auf $[x_i, x_i + h_i]$. Wir nehmen zunächst an, dass wir den Integrationsfehler $\varepsilon^i_{k,k-1}$ für $T^i_{k,k-1}$ kennen. Nach (5.7) gilt

$$\varepsilon^i_{k,k-1} \approx \delta_i h_i^{2k+1}, \quad \text{also} \quad \delta_i \approx \varepsilon^i_{k,k-1} h_i^{-(2k+1)}.$$

Um also $\varepsilon^i_{k,k-1} \leq tol$ zu garantieren, müsste die Berechnung mit der *idealen Schrittweite*

$$\tilde{h}_i = \sqrt[2k+1]{\frac{tol}{\varepsilon^i_{k,k-1}}} h_i$$

durchgeführt werden. Um diese ideale Schrittweite ohne den unbekannten Wert $\varepsilon^i_{k,k-1}$ zu berechnen, ersetzen wir diesen durch den Fehlerschätzer

$$\bar{\varepsilon}^i_{k,k-1} = |T^i_{k,k-1} - T^i_{k,k}|.$$

Da dieser nicht exakt ist, führen wir einen "Sicherheitsparameter" $\rho < 1$ ein, und berechnen die neue Schrittweite mittels

$$\tilde{h}_i \approx \sqrt[2k+1]{\frac{\rho\, tol}{\bar{\varepsilon}^i_{k,k-1}}} h_i.$$

Falls nun $\tilde{h}_i$ kleiner als h_i ist, wiederholen wir die Berechnung mit $h_i = \tilde{h}_i$. Wenn (eventuell nach Wiederholung der Berechnung) $\tilde{h}_i \geq h_i$ gilt, so gehen wir zum nächsten Teilintervall über. Als neue Schrittweite wählen wir nun $\tilde{h}_i$, wir setzen also $x_{i+1} = x_i + h_i$ und $h_{i+1} = \tilde{h}_i$. Dies ist sinnvoll, da wegen der Stetigkeit von $f^{(2k)}$ für kleine h_i

$$\delta_{i+1} \approx \delta_i$$

[1] Auch hier kann man natürlich wahlweise ein relatives Abbruchkriterium verwenden.

gilt, also

$$\varepsilon_{k,k-1}^{i+1} = \delta_{i+1} h_{i+1}^{2k+1} \approx \delta_i h_{i+1}^{2k+1} = \delta_i \tilde{h}_i^{2k+1} = tol$$

als Fehler mit dieser Schrittweite zu erwarten ist. Falls $x_{i+1} + h_{i+1} > b$ gilt, verringern wir die neue Schrittweite zu $h_{i+1} = b - x_{i+1}$, um nicht über die obere Integrationsgrenze hinweg zu integrieren. Ebenso empfiehlt es sich, eine obere Schranke $h_{\max}$ für die erlaubte maximale Schrittweite einzuführen und h_{i+1} durch $h_{\max}$ zu beschränken.

Falls $x_{i+1} < b$ ist, setzen wir $i := i + 1$ und fahren wir fort bei (5.11).

Falls $x_{i+1} = b$ ist, haben wir die obere Integrationsgrenze erreicht, die gesuchte Integralapproximation können wir dann — da wir hier auf die sowieso berechneten genaueren Werte $T_{k,k}^i$ zurückgreifen können — als $\sum_{j=1}^{i} T_{k,k}^j$ berechnen.

Abbildung 5.3 zeigt die Anwendung dieses Algorithmus auf die Nadelimpulsfunktion $f(x) = \frac{1}{10^{-4}+x^2}$, mit den Parametern $[a, b] = [-1, 1]$, $k = 3$, $tol = 10^{-5}$, $\rho = 0.8$ und Basisschrittweite $h = 1$.

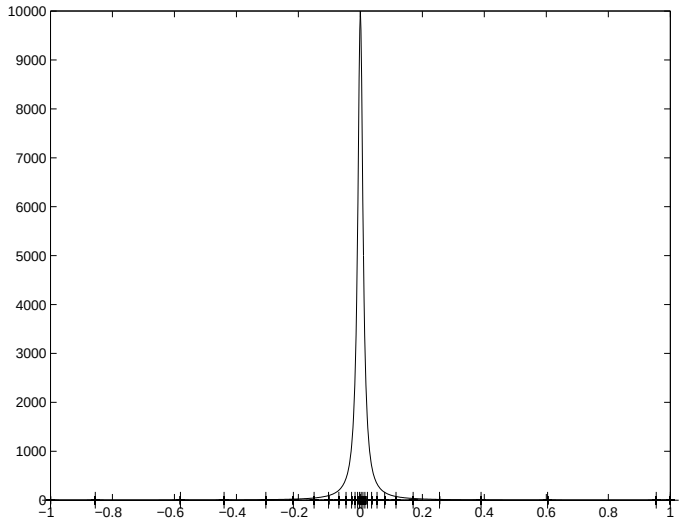

Abbildung 5.3: Adaptive Teilintervalle bei der Integration der Nadelimpulsfunktion

Das Integrationsintervall wurde dabei durch den adaptiven Algorithmus in 27 Einzelintervalle aufgeteilt, die minimale Schrittweite war 0.005532, die maximale 0.350629.[2]

Die insgesamt zu erwartende Genauigkeit beträgt bei diesem Verfahren $m \cdot tol$, wobei m die Anzahl der durchgeführten Schritte ist; diese Genauigkeit lässt sich also naturgemäß erst nach Ablauf des Algorithmus berechnen.

Trotz der sehr effizienten "Selbstanpassung" an den Integranden, können — wie bei allen numerischen Methoden — auch bei der adaptiven Romberg–Quadratur Probleme auftreten, denn bei zu großen Schrittweiten oder wenn die gemachten Voraussetzungen nicht erfüllt sind, liefern die Fehlerschätzer unzuverlässige Werte. Tatsächliche Fehler können

[2] MATLAB Files für diesen Algorithmus werden am Ende des Semesters auf der Vorlesungshomepage bereit gestellt.

dann “übersehen” werden, wodurch eine falsche Genauigkeit berechnet wird und das Verfahren mit falschen Werten konvergiert. Man spricht in diesem Fall von *Pseudokonvergenz*. Strategien zur Erkennung dieser Situation finden sich ebenfalls im Buch von Deuflhard/Hohmann [2].

Ob man für ein gegebenes Problem eine adaptive oder eine nicht–adaptive Methode vorzieht, hängt alles in allem von der Aufgabenstellung und dem Einsatzbereich der Integrationsroutine ab. Adaptive Verfahren liefern für viele Integranden f, speziell wenn diese nicht gleichmäßig sind oder keine a priori–Analyse durchgeführt werden soll, eine zuverlässige und effiziente Strategie zur Lösung von Integrationsproblemen. Falls viele Informationen (z.B. über die Ableitungen) des Integranden vorhanden sind oder relativ zum Aufwand des Integrationsproblems günstig berechnet werden können, z.B. wenn eine bestimmte Funktion im Rahmen eines Algorithmus oft integriert werden muss, werden nicht–adaptive Verfahren im Allgemeinen effizienter sein, falls die nötigen Parameter (Ordnung, Schrittweite, etc.) gut eingestellt sind.

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 $f : \mathbb{R}^n \to \mathbb{R}^n$ 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 $\Phi : A \to A$ 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 $g : \mathbb{R}^n \to \mathbb{R}^n$ 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 $A = \{x \in \mathbb{R}^n \,|\, \|x - x^*\| \leq \delta\} \subset D$ 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 Spektrakradius $\rho(A) = \max_i |\lambda_i(A)|$ 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

$$|g'(x^*)| < 1 \quad \text{oder } |(g^{-1})'(x^*)| = \frac{1}{|g'(x^*)|} < 1$$

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) = 1.00000000000000
x( 1) = 1.50000000000000
x( 2) = 1.37500000000000
x( 3) = 1.42968750000000
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. □

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 $f : \mathbb{R} \to \mathbb{R}$ 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 $f : \mathbb{R} \to \mathbb{R}$ 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).

□

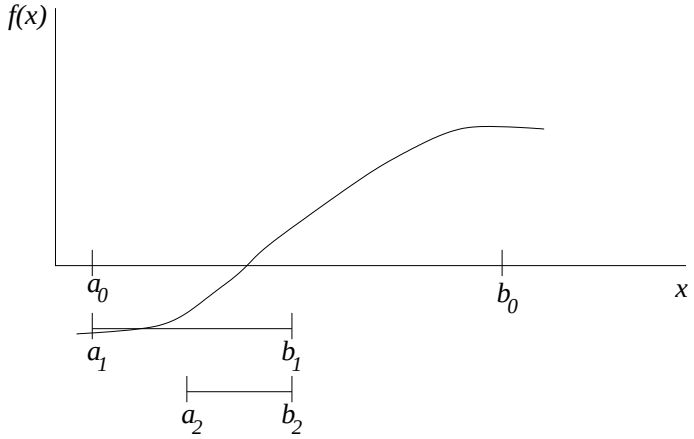

Abbildung 6.1: Bisektionsverfahren

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

$$\|x^{(i+1)} - x^*\| \leq c\|x^{(i)} - x^*\| \text{ für alle } i = 0, 1, 2, \ldots$$

gilt.

(ii) Das Verfahren heißt *superlinear konvergent*, falls Konstanten $c_i \in (0,1)$ für $i = 0, 1, 2, \ldots$ existieren, so dass die Bedingungen

$$c_{i+1} \leq c_i,\ i = 0, 1, 2, \ldots, \quad \lim_{i \to \infty} c_i = 0$$

und die Abschätzung

$$\|x^{(i+1)} - x^*\| \leq c_i\|x^{(i)} - x^*\| \text{ für alle } i = 0, 1, 2, \ldots$$

gelten.

(iii) Das Verfahren heißt *quadratisch konvergent*, falls eine Konstante $q > 0$ existiert, so dass die Abschätzung

$$\|x^{(i+1)} - x^*\| \leq q\|x^{(i)} - x^*\|^2 \text{ für alle } i = 0, 1, 2, \ldots$$

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. □

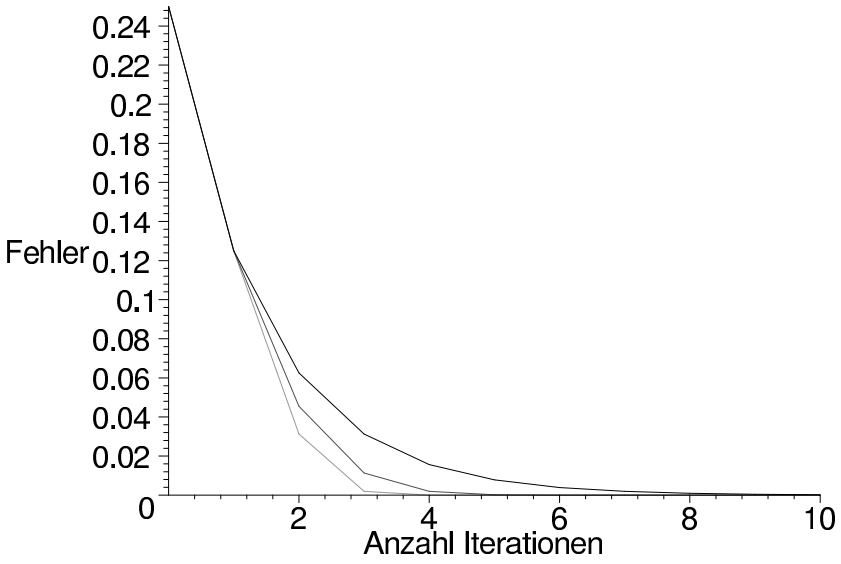

Abbildung 6.2: Konvergenzordnungen: linear, superlinear und quadratisch (von oben nach unten)

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

$$\log(ab) = \log(a) + \log(b), \;\; \log(1/q) = -\log(q) \quad \text{und} \quad \log(c^d) = d\log(c).$$

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.

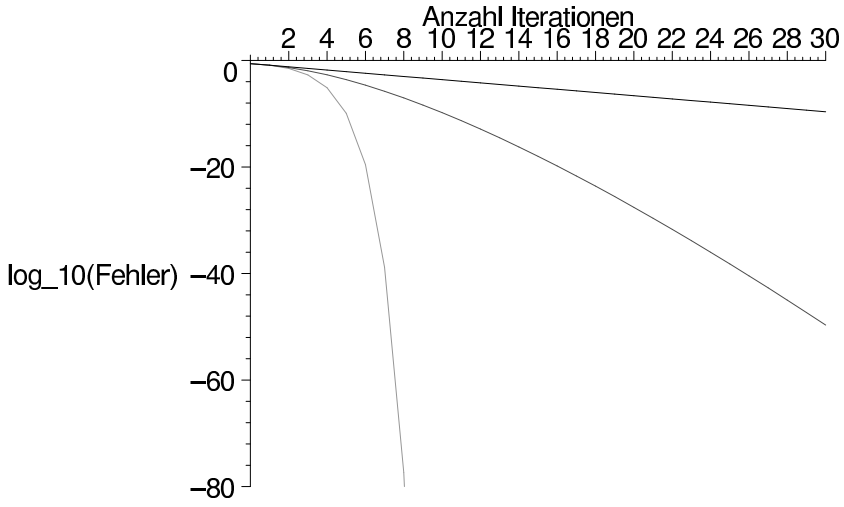

Abbildung 6.3: Konvergenzordnungen: linear, superlinear und quadratisch (von oben nach unten)

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

$$|x^{(i)} - x^*| = 10^d < 10^{-m} = 0.\underbrace{0 \cdots 0}_{(m-1)-\text{mal}}1.$$

Jede Zahl, die kleiner als $10^{-m} =$ ist, ist von der Form

$$0.\underbrace{0 \cdots 0}_{m-\text{mal}} * * * \cdots,$$

wobei der Stern "$*$" beliebige Ziffern symbolisiert. Also ist

$$|x^{(i)} - x^*| \leq 0.\underbrace{0 \cdots 0}_{m-\text{mal}} * * * \cdots,$$

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[1] 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 ξ_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 $q = r/2$

$$\|x^{(i+1)} - x^*\|_\infty = \|g(x^{(i)}) - x^*\|_\infty = \max_{j=1,\ldots,n} |g_j(x^{(i)}) - x_j^*| \leq \max_{j=1,\ldots,n} q(x_j^{(i)} - x_j^*)^2 \leq q\|x^{(i)} - x^*\|_\infty^2,$$

also quadratische Konvergenz.

[1] Wenn $1/(-\log(c)) < 1$ ist, ist dies so zu verstehen, dass die Anzahl korrekter Stellen pro Schritt im Mittel um $-\log(c)$ zunimmt.

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. □

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

$$f(x^{(i)})+f'(x^{(i)})(x^{(i+1)}-x^{(i)}) = 0 \;\Leftrightarrow\; f'(x^{(i)})x^{(i+1)} = f'(x^{(i)})x^{(i)}-f(x^{(i)}) \;\Leftrightarrow\; x^{(i+1)} = x^{(i)}-\frac{f(x^{(i)})}{f'(x^{(i)})}.$$

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 $f : \mathbb{R} \to \mathbb{R}$, ihre Ableitung $f' : \mathbb{R} \to \mathbb{R}$ sowie ein Startwert $x^{(0)} \in \mathbb{R}$ und eine gewünschte Genauigkeit $\varepsilon > 0$. Setze $i = 0$.

(1) Berechne $x^{(i+1)} = x^{(i)} - f(x^{(i)})/f'(x^{(i)})$

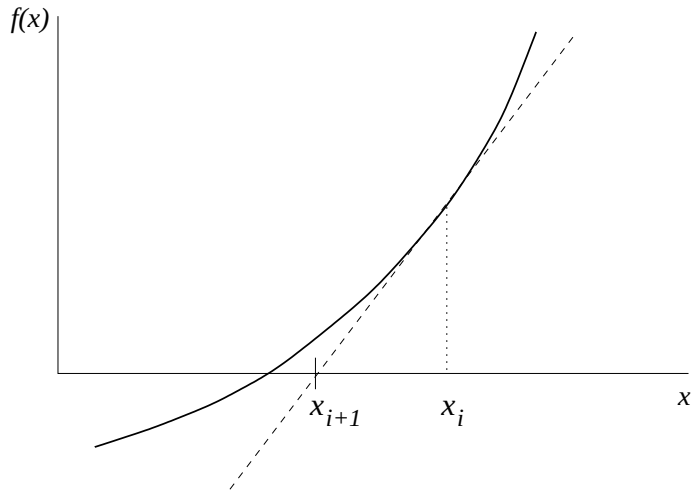

Abbildung 6.4: Newton–Verfahren

(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)

$$\begin{aligned}
x^{(1)} &= \frac{1}{2}2 + \frac{1}{2} &&= \frac{3}{2} &&= 1.5 \\
x^{(2)} &= \frac{1}{2}\frac{3}{2} + \frac{1}{\frac{3}{2}} &&= \frac{17}{12} &&= 1.\underline{41}\overline{6} \\
x^{(3)} &= \frac{1}{2}\frac{17}{12} + \frac{1}{\frac{17}{12}} &&= \frac{577}{408} &&\approx 1.\underline{4142}156862745 \\
x^{(4)} &= \frac{1}{2}\frac{577}{408} + \frac{1}{\frac{577}{408}} &&= \frac{665857}{470832} &&\approx 1.\underline{4142135623}746
\end{aligned}$$

□

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 $f : \mathbb{R} \to \mathbb{R}$

$$x^{(i+1)} = x^{(i)} - \frac{f(x^{(i)})}{f'(x^{(i)})}$$

betrachten, stellt sich die Frage, wie eine geeignete Verallgemeinerung aussehen kann.

Sei dazu $f : \mathbb{R}^n \to \mathbb{R}^n$ 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 den selben 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 $f : \mathbb{R}^n \to \mathbb{R}^n$, ihre Ableitung $Df : \mathbb{R}^n \to \mathbb{R}^{n \times n}$ 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 $f : \mathbb{R}^2 \to \mathbb{R}^2$ 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

$$\frac{\partial f_1}{x_1}(x) = 2x_1, \quad \frac{\partial f_1}{x_2}(x) = 2x_2, \quad \frac{\partial f_2}{x_1}(x) = 1, \quad \frac{\partial f_2}{x_2}(x) = 0$$

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)

$$\begin{aligned}
x^{(1)} &= \begin{pmatrix} 1 \\ 1 \end{pmatrix} - \begin{pmatrix} 0 & 1 \\ \frac{1}{2} & -1 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \end{pmatrix} &&= \begin{pmatrix} 0 \\ \frac{3}{2} \end{pmatrix} &&= \begin{pmatrix} 0 \\ 1.5 \end{pmatrix} \\
x^{(2)} &= \begin{pmatrix} 0 \\ \frac{3}{2} \end{pmatrix} - \begin{pmatrix} 0 & 1 \\ \frac{1}{3} & 0 \end{pmatrix} \begin{pmatrix} \frac{5}{4} \\ 0 \end{pmatrix} &&= \begin{pmatrix} 0 \\ \frac{13}{12} \end{pmatrix} &&= \begin{pmatrix} 0 \\ 1.\underline{0}8\overline{3} \end{pmatrix} \\
x^{(3)} &= \begin{pmatrix} 0 \\ \frac{13}{12} \end{pmatrix} - \begin{pmatrix} 0 & 1 \\ \frac{6}{13} & 0 \end{pmatrix} \begin{pmatrix} \frac{25}{144} \\ 0 \end{pmatrix} &&= \begin{pmatrix} 0 \\ \frac{313}{312} \end{pmatrix} &&\approx \begin{pmatrix} 0 \\ 1.\underline{00}32051282 \end{pmatrix} \\
x^{(4)} &= \begin{pmatrix} 0 \\ \frac{313}{312} \end{pmatrix} - \begin{pmatrix} 0 & 1 \\ \frac{156}{313} & 0 \end{pmatrix} \begin{pmatrix} \frac{625}{97344} \\ 0 \end{pmatrix} &&= \begin{pmatrix} 0 \\ \frac{195313}{195312} \end{pmatrix} &&\approx \begin{pmatrix} 0 \\ 1.\underline{00000}51200 \end{pmatrix}
\end{aligned}$$

□

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 $f : \mathbb{R}^n \to \mathbb{R}^n$, ihre Ableitung $Df : \mathbb{R}^n \to \mathbb{R}^{n\times n}$ 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 $f : D \to \mathbb{R}^n$ 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.

$$\|x^{(i)} - x^*\| < \rho \text{ für alle } i > 0 \quad \text{und} \quad \lim_{i\to\infty} x^{(i)} = x^*.$$

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

$$\|Df(x)^{-1}(f(y) - f(x) - Df(x)(y - x))\| \leq \frac{\omega}{2}\|y - x\|^2 \tag{6.2}$$

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 $\|x^{(0)} - x^*\| \leq \rho$ 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. □

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 Deuflhard/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*. Die numerische Approximation von Ableitungen ist aber im Allgemeinen sehr anfällig gegenüber Rundungsfehlern, weswegen die numerische Stabilität dieser Verfahren — möglichst unter Hinzunahme weiterer Informationen über f — genau überprüft werden sollte.

Im $\mathbb{R}^1$ gibt es eine Alternative zum Newton–Verfahren, die ohne Ableitungen auskommt und trotzdem recht schnell konvergiert. Dieses Verfahren wollen wir im folgenden Abschnitt beschreiben.

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

$$g(x) = f(x^{(i)}) + \frac{(x^{(i)} - x)}{x^{(i)} - x^{(i-1)}} \Big(f(x^{(i-1)}) - f(x^{(i)}) \Big)$$

und wählt $x^{(i+1)}$ als Nullstelle dieser Geraden, also

$$\begin{aligned} 0 &= f(x^{(i)}) + \frac{(x^{(i)} - x^{(i+1)})}{x^{(i)} - x^{(i-1)}} \Big(f(x^{(i-1)}) - f(x^{(i)}) \Big) \\ \Leftrightarrow \quad x^{(i)} - x^{(i+1)} &= -f(x^{(i)}) \frac{x^{(i)} - x^{(i-1)}}{f(x^{(i-1)}) - f(x^{(i)})} \\ \Leftrightarrow \quad x^{(i+1)} &= x^{(i)} - f(x^{(i)}) \frac{x^{(i)} - x^{(i-1)}}{f(x^{(i)}) - f(x^{(i-1)})} \end{aligned}$$

Die Idee ist in Abbildung 6.5 grafisch veranschaulicht.

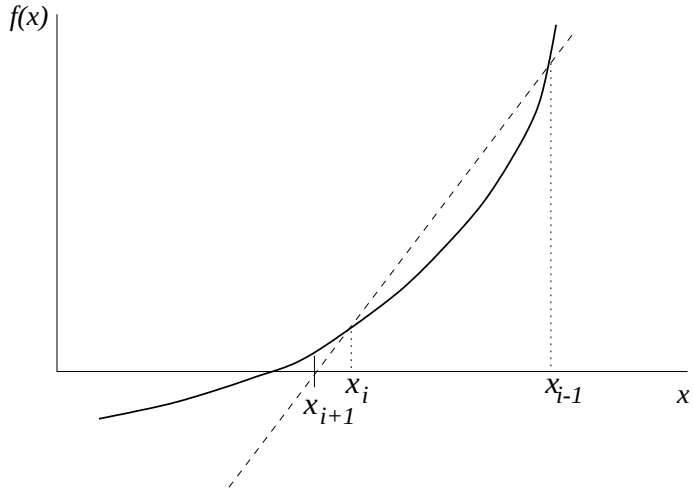

Abbildung 6.5: Sekanten–Verfahren

Formal lässt sich das Verfahren wie folgt beschreiben.

Algorithmus 6.18 (Sekanten–Verfahren) Gegeben sei eine Funktion $f : \mathbb{R} \to \mathbb{R}$ 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 $x^{(i+1)} = x^{(i)} - \dfrac{(x^{(i)} - x^{(i-1)}) f(x^{(i)})}{f(x^{(i)}) - f(x^{(i-1)})}$

(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 $f : \mathbb{R} \to \mathbb{R}$ 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

$$f(x^{(i)}) - f(x^{(i-1)}) \approx f'(x^*)(x^{(i)} - x^{(i-1)}) \tag{6.3}$$

sowie für $j = i - 1$ und $j = i$ die Abschätzungen

$$f(x^{(j)}) \approx f'(x^*)(x^{(j)} - x^*) + \frac{1}{2} f''(x^*)(x^{(j)} - x^*)^2. \tag{6.4}$$

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 Abschätung

$$f(x^{(i)}) - f(x^{(i-1)}) \approx f'(x^*)(x^{(i)} - x^{(i-1)})$$

und mittels (6.4) erhalten wir für den Zähler

$$\begin{aligned}
&(x^{(i-1)} - x^*)f(x^{(i)}) - (x^{(i)} - x^*)f(x^{(i-1)}) \\
&\quad\approx\ (x^{(i-1)} - x^*)\Big(f'(x^*)(x^{(i)} - x^*) + \frac{1}{2} f''(x^*)(x^{(i)} - x^*)^2\Big) \\
&\qquad -\ (x^{(i)} - x^*)\Big(f'(x^*)(x^{(i-1)} - x^*) + \frac{1}{2} f''(x^*)(x^{(i-1)} - x^*)^2\Big) \\
&\quad=\ \frac{1}{2} f''(x^*)(x^{(i-1)} - x^*)(x^{(i)} - x^*)\Big((x^{(i)} - x^*) - (x^{(i-1)} - x^*)\Big) \\
&\quad=\ \frac{1}{2} f''(x^*)(x^{(i-1)} - x^*)(x^{(i)} - x^*)(x^{(i)} - x^{(i-1)}).
\end{aligned}$$

Also folgt

$$|x^{(i+1)} - x^*| \approx \underbrace{\frac{1}{2}\left|\frac{f''(x^*)}{f'(x^*)}\right| |x^{(i-1)} - x^*|}_{\approx c_i} |x^{(i)} - x^*|$$

und damit die behauptete superlineare Konvergenz. □

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

$$x^{(i+1)} = x^{(i)} - \frac{(x^{(i)} - x^{(i-1)})((x^{(i)})^2 - 2)}{(x^{(i)})^2 - (x^{(i-1)})^2} = x^{(i)} - \frac{(x^{(i)} - x^{(i-1)})((x^{(i)})^2 - 2)}{(x^{(i)} - x^{(i-1)})(x^{(i)} + x^{(i-1)})} = x^{(i)} - \frac{(x^{(i)})^2 - 2}{x^{(i)} + x^{(i-1)}}.$$

Mit $x^{(0)} = 2$ und $x^{(1)} = 1$ ergibt sich (korrekte Nachkommastellen sind unterstrichen)

$$\begin{aligned}
x^{(2)} &= 1 - \frac{1^2 - 2}{1 + 2} &&= \frac{4}{3} &&= 1.\overline{3} \\
x^{(3)} &= \frac{4}{3} - \frac{(\frac{4}{3})^2 - 2}{\frac{4}{3} + 1} &&= \frac{10}{7} &&= 1.\underline{4}285714 \\
x^{(4)} &= \frac{10}{7} - \frac{(\frac{10}{7})^2 - 2}{\frac{10}{7} + \frac{4}{3}} &&= \frac{41}{29} &&\approx 1.\underline{41}37931034483 \\
x^{(5)} &= \frac{41}{29} - \frac{(\frac{41}{29})^2 - 2}{\frac{41}{29} + \frac{10}{7}} &&= \frac{816}{577} &&\approx 1.\underline{4142}14384749 \\
x^{(6)} &= \frac{816}{577} - \frac{(\frac{816}{577})^2 - 2}{\frac{816}{577} + \frac{41}{29}} &&= \frac{66922}{47321} &&\approx 1.\underline{414213562}6889
\end{aligned}$$

Das Sekanten–Verfahren konvergiert also deutlich schneller als linear aber langsamer als das Newton–Verfahren. □

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 2.1.1 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 Normalengleichungen

$$A^T A x = A^T z$$

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

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

führt, wobei $\overline{R} \in \mathbb{R}^{n\times n}$ 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 Normalengleichungen, da die Matrix A^TA schlecht konditioniert sein kann.

Die Normalengleichungen 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

$$A^+ = (A^TA)^{-1}A^T$$

□

Man sieht leicht, dass die Lösung des linearen Ausgleichsproblems $\min \|Ax - z\|_2^2$ gerade durch

$$x = A^+z$$

gegeben ist, denn

$$x = A^+z \Leftrightarrow x = (A^TA)^{-1}A^Tz \Leftrightarrow A^TAx = A^Tz,$$

d.h. $x = A^+z$ löst gerade die Normalengleichungen.

Bemerkung 6.22 Man rechnet leicht nach, dass $A^+ = (A^TA)^{-1}A^T \in \mathbb{R}^{n\times m}$ die Eigenschaften

$$\begin{array}{llll} \text{(i)} & (A^+A)^T = A^+A & \text{(ii)} & (AA^+)^T = AA^+ \\ \text{(iii)} & A^+AA^+ = A^+ & \text{(iv)} & AA^+A = A \end{array}$$

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 $\overline{P} := AA^+$ orthogonale Projektionen, d.h. sie erfüllen $P^T = P = P^2$ und $\overline{P}^T = \overline{P} = \overline{P}^2$. Darüberhinaus ist A^+ injektiv, denn aus (iv) folgt

$$n = \dim \operatorname{im} A = \dim \operatorname{im} AA^+A \leq \dim \operatorname{im} A^+,$$

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 $\overline{P}$ gerade gleich dem Bild von A ist. Damit sind P und $\overline{P}$ 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 $AA_1^+ = \overline{P} = AA_2^+$ gelten. Daraus folgt

$$A_1^+ \overset{\text{(iii)}}{=} \underbrace{A_1^+A}_{=A_2^+A} A_1^+ = A_2^+ \underbrace{AA_1^+}_{=AA_2^+} = A_2^+AA_2^+ \overset{\text{(iii)}}{=} A_2^+.$$

□

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

$$f : D \to \mathbb{R}^m$$

und wollen für diese Abbildung das Problem

$$\text{minimiere } g(x) := \|f(x)\|_2^2 \text{ über } x \in D \tag{6.5}$$

lösen.

Wie beim linearen Ausgleichsproblem ist die Interpretation und Anwendung dieses Problems wie folgt: Seien z_i Messwerte zu Parametern t_i für $i = 1, \ldots, m$. Auf Grund theoretischer Überlegungen (z.B. eines zu Grunde liegenden physikalischen Gesetzes) weiß man, dass eine Funktion $h : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^m$, $h(t, x) = z$ 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 ∂D nicht berücksichtigen. Darüberhinaus wollen wir uns auf lokale Minimalstellen $x^* \in D$ von $g : \mathbb{R}^n \to \mathbb{R}$ beschränken, die die hinreichenden Bedingungen

$$Dg(x^*) = 0 \quad \text{und} \quad D^2 g(x^*) \text{ ist positiv definit} \tag{6.6}$$

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

$$G(x) := Df(x)^T f(x) = 0 \tag{6.7}$$

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

$$DG(x^{(i)})\Delta x^{(i)} = G(x^{(i)}), \quad i = 0, 1, 2, \ldots, \tag{6.8}$$

Falls ein lokales Minimum x^* mit (6.6) existiert, so ist

$$DG(x) = D^2 g(x) = Df(x)^T Df(x) + D^2 f(x)^T f(x)$$

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

$$Df(x^{(i)})^T Df(x^{(i)})\Delta x^{(i)} = G(x^{(i)}) = Df(x^{(i)})^T f(x^{(i)}), \quad i = 0, 1, 2, \ldots \tag{6.9}$$

Dies sind gerade die Normalengleichungen 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 $f : \mathbb{R}^n \to \mathbb{R}^m$, ihre Ableitung $Df : \mathbb{R}^n \to \mathbb{R}^{m\times n}$ 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. □

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 $f : D \to \mathbb{R}^m$ 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

$$\|Df(x)^+ f(x^*)\| \leq \kappa^* \|x - x^*\|$$

für alle $x \in D$.

Dann gilt: Für alle Anfangswerte $x^{(0)} \in \mathbb{R}^n$ mit

$$\rho := \|x^* - x^{(0)}\| \leq \frac{2(1-\kappa^*)}{\omega} \quad \text{und} \quad B_\rho(x^*) \subseteq D$$

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.

$$\|x^{(i)} - x^*\| < \rho \text{ für alle } i > 0 \quad \text{und} \quad \lim_{i\to\infty} x^{(i)} = x^*.$$

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 $x, y \in D$.

Für die Lösung x^* des Ausgleichsproblems gilt $Df(x^*)^+ f(x^*) = 0$. Da die Jacobi–Matrix Df auf D nach Annahme vollen Rang besitzt, ist die Pseudoinverse $Df(x)^+$ wohldefiniert und es folgt

$$Df(x)^+ Df(x) = (Df(x)^T Df(x))^{-1} Df(x)^T Df(x) = \mathrm{Id}_{\mathbb{R}^n}.$$

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

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

Mittels Induktion ergibt sich so

$$\|x^{(i+1)} - x^*\| < \|x^{(i)} - x^*\| \leq \rho$$

weswegen die Folge in $B_\rho(x^*)$ bleibt und 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. □

Bemerkung 6.26 Der Parameter κ^* kann als Maß für die "Nichtkompatibilität" des nichtlinearen Ausgleichsproblems aufgefasst werden. Beweistechnisch ist leicht einzusehen, dass $\kappa^* < 1$ notwendig für die Konvergenz ist, da dieses κ^* 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. □

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/`

[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