NUMERISCHE MATHEMATIK FÜR DAS LEHRAMT

VORLESUNGSSKRIPT, WINTERSEMESTER 2016/17

Christian Clason

Stand vom 9. Februar 2017

Fakultät für Mathematik
Universität Duisburg-Essen

INHALTSVERZEICHNIS

ANHANG

VERZEICHNIS DER ALGORITHMEN

Teil I

ALGORITHMEN UND IHRE GRENZEN

GRUNDLAGEN

1

In diesem Kapitel betrachten wir die grundlegenden Fragestellungen der numerischen Mathematik und sammeln die wichtigsten allgemeinen Werkzeuge zu ihrer Untersuchung. Begriffe, die nur für spezifische Probleme relevant sind, sollen grundsätzlich erst dort erläutert werden, wo sie auftreten.

1.1 GRUNDBEGRIFFE: PROBLEM UND ALGORITHMUS

Die numerische Mathematik beschäftigt sich mit der Entwicklung und Analyse konstruktiver Verfahren zur Lösung von Problemen, die mathematisch formulierbar sind. Üblicherweise sind dabei ein *Modell* und *Eingangsdaten* gegeben, aus denen eine (numerische) Antwort (die *Ausgabedaten*) zu ermitteln ist. Mathematisch wird das Modell als eine Funktion $f : X \to Y$ beschrieben, wobei X die Menge der möglichen Eingabe-, Y die Menge der möglichen Ausgabedaten ist. Ein konkretes *Problem* ist dann, für gegebenes $x \in X$ die Ausgabe $f(x) \in Y$ zu berechnen. Das Problem, zwei Zahlen zu addieren, wäre dann beschrieben durch

$$X = \mathbb{R}^2,\ Y = \mathbb{R}, \quad f : (x_1, x_2) \mapsto x_1 + x_2.$$

Ein komplizierteres Problem wäre gegeben durch die genaue Beschreibung eines Motorblocks x, das System partieller Differentialgleichungen zur Beschreibung der Wärmeentwicklung im laufenden Betrieb f und die Frage nach der Temperaturverteilung nach einer gewissen Laufzeit $f(x)$.

Unter einem *Algorithmus* verstehen wir eine eindeutig formulierte Vorschrift zur Lösung eines Problems.[1] Ein Algorithmus wird spezifiziert durch

- die *Eingabedaten*, die zur Durchführung benötigt werden,
- die *Ausgabedaten*, die das Ergebnis des Algorithmus darstellen,

[1] Das bekannteste nicht-mathematische Beispiel ist sicher das Kochrezept.

- sowie die *Beschreibung der durchzuführenden Schritte* inklusive der benötigten *Hilfsgrößen*.

Von einem Algorithmus erwartet man in der Regel neben der Korrektheit (d. h. er liefert tatsächlich eine Lösung der Aufgabe), dass er

- *durchführbar* ist (zum Beispiel, indem nur elementare Schritte[2] durchgeführt werden),
- *terminiert* (d. h. garantiert nach einer endlichen Anzahl von Schritten abgeschlossen ist) und
- *deterministisch* ist (d. h. für gleiche Eingabedaten stets das gleiche Resultat liefert).[3]

Ein guter Algorithmus sollte darüber hinaus die folgenden Eigenschaften erfüllen:

1. *Stabilität*: Kleine Fehler in den Eingabedaten sollen nur kleine Fehler in den Ausgabedaten verursachen.
2. *Genauigkeit*: Die Ausgabedaten sollen eine möglichst gute Lösung der gestellten Aufgabe sein.
3. *Flexibilität*: Der Algorithmus soll für eine große Klasse von Eingabedaten anwendbar sein.
4. *Effizienz*: Der Algorithmus soll auch für umfangreiche Probleme praktisch durchführbar sein.

Die Stabilität ist ein so fundamentales Problem, dass ihr ein eigenes Kapitel 3 gewidmet ist. Bei der Effizienz betrachtet man die Komplexität des Algorithmus, auf die wir weiter unten eingehen werden. Bei der Beurteilung der Genauigkeit wird man in der Regel eine Abschätzung des zu erwartenden Fehlers in den Ausgabedaten erwarten. Dafür benötigen wir einige wichtige Begriffe aus der Analysis.

1.2 MESSEN UND SCHÄTZEN: NORMEN

Der zentrale Begriff für jede Art von Messung ist die *Norm*. Diese ist die fundamentale Abbildung, mit deren Hilfe Fragestellungen in abstrakten Räumen (Vektorräume, Funktionenräume) in die reellen Zahlen übertragen werden können, wo man sich der bekannten Hilfsmittel bedienen kann (etwa der Ordnung).

Definition 1.1. Sei V ein Vektorraum über $\mathbb{K} \in \{\mathbb{R}, \mathbb{C}\}$. Die Abbildung $\|\cdot\| : V \to \mathbb{K}$ heißt *Norm auf* V, falls gilt

[2] Was "elementar" bedeutet, hängt natürlich von dem jeweiligen Kontext ab. Eine Beispiel ist die Anwendung eines existierenden Algorithmus, dessen Korrektheit bekannt ist.

[3] In letzter Zeit sind allerdings *randomisierte* Algorithmen zunehmend populär geworden, die zwar nicht in jedem Fall, aber doch mit hoher Wahrscheinlichkeit, erheblich schneller als deterministische sein können.

(N1) $\|v\| \geqslant 0$ für alle $v \in V$, und $\|v\| = 0$ dann und nur dann, wenn $v = 0$;

(N2) Für alle $a \in \mathbb{K}$ und $v \in V$ gilt $\|av\| = |a|\|v\|$;

(N3) Für alle $v, w \in V$ gilt die *Dreiecksungleichung*

$$\|v + w\| \leqslant \|v\| + \|w\|.$$

Das Paar $(V, \|\cdot\|)$ heißt dann *normierter Raum*.

Beispiel 1.2 (p-Normen auf $\mathbb{R}^n$)**.** Wir betrachten $V = \mathbb{R}^n$ und $x = (x_1, \ldots, x_n)^T \in \mathbb{R}^n$ und definieren für $1 \leqslant p < \infty$

$$\|x\|_p := \left(\sum_{i=1}^{n} |x_i|^p \right)^{\frac{1}{p}}$$

Häufige Spezialfälle sind:

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

(ii) $\|x\|_2 = \sqrt{\sum_{i=1}^{n} |x_i|^2}$ (*Euklidische Norm*),

(iii) $\|x\|_\infty = \max_{i=1,\ldots,n} |x_i|$ (*Maximumsnorm*).

Spezielle Normen für Matrizen werden wir in Kapitel 4 kennen lernen. Wichtige Beispiele für Normen auf unendlichdimensionale Vektorräumen sind die folgenden.

Beispiel 1.3 (Normen auf Funktionenräumen)**.** Auf dem Vektorraum $C([a, b])$ aller stetigen Funktionen vom Intervall $[a, b] \subset \mathbb{R}$ nach $\mathbb{R}$ stellen die folgenden Abbildungen eine Norm dar:

(i) $\|f\|_2 = \left(\int_a^b f(x)^2 \, dx \right)^{\frac{1}{2}}$,

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

In einem normierten Raum definiert $\|v - w\|$ auf kanonische Weise einen Abstand zwischen zwei Vektoren v und w.[4] Damit haben wir schon ein Mittel zur Hand, um Fehler (den Abstand zwischen berechneter und gesuchter Größe) zu berechnen. Allerdings wird üblicherweise die gesuchte Größe nicht bekannt sein; eine wichtige Aufgabe wird also sein, obere Schranken für diesen Fehler anzugeben. Neben der Dreiecksungleichung ist dafür die folgende Ungleichung unerlässlich:

[4] Insbesondere haben wir dadurch einen Konvergenzbegriff: Wir sagen, $x_n \to x^*$, falls $\|x_n - x^*\| \to 0$.

Satz 1.4 (Höldersche Ungleichung). *Für* $x, y \in \mathbb{R}^n$ *und* $1 \leqslant p, q \leqslant \infty$, $1/p + 1/q = 1$ *(bzw.* $p = 1, q = \infty$*) gilt*

$$|x^T y| \leqslant \|x\|_p \|y\|_q.$$

Ein wichtiger Spezialfall ist die Cauchy-Schwarzsche Ungleichung *für* $p = q = 2$*,*

$$|x^T y| \leqslant \|x\|_2 \|y\|_2.$$

Dabei ist $\langle x, y \rangle := x^T y = \sum_{i=1}^{n} x_i y_i$ das *Euklidische Skalarprodukt*[5] im $\mathbb{R}^n$.

Mit Hilfe der Hölderschen Ungleichung sieht man zum Beispiel, dass man eine p-Norm durch eine andere abschätzen kann. Dies lässt sich präzisieren.

Definition 1.5. Zwei Normen $\|\cdot\|$, $\|\cdot\|'$ auf einem Vektorraum V heißen *äquivalent*, falls zwei Konstanten $c_1, c_2 > 0$ existieren, so dass für alle $x \in V$ gilt

$$c_1 \|x\| \leqslant \|x\|' \leqslant c_2 \|x\|.$$

Satz 1.6 (Äquivalenz der Normen). *Auf einem endlichdimensionalen Vektorraum sind alle Normen äquivalent.*

Beispiel 1.7. Für $x \in \mathbb{R}^n$ gilt

(i) $\|x\|_2 \leqslant \|x\|_1 \leqslant \sqrt{n}\|x\|_2$,

(ii) $\|x\|_\infty \leqslant \|x\|_2 \leqslant \sqrt{n}\|x\|_\infty$,

(iii) $\|x\|_\infty \leqslant \|x\|_1 \leqslant n\|x\|_\infty$.

Wozu benötigt man dann noch andere Normen, wenn man immer die Euklidische Norm für Abschätzungen nehmen könnte? Zum einen ist es eine der wesentlichen Aufgaben in der numerischen Mathematik, kontinuierliche (und damit unendlichdimensionale) Probleme durch endlichdimensionale Näherungen zu lösen. Im Unendlichdimensionalen gilt die Äquivalenz der Normen aber in der Regel nicht, und dieser Unterschied sollte berücksichtigt werden. Dies zeigt sich dadurch, dass die Normen mit wachsender Dimension "auseinander laufen", d. h. das Verhältnis c_2/c_1 immer größer wird, wie an den Beispielen ersichtlich wird. Für große Probleme (d. h. großes n) kann dies durchaus bemerkbare Auswirkungen haben.

Zum anderen haben die unterschiedlichen Normen verschiedene "Sichtweisen". So stellt die Maximumsnorm eine harte Schranke dar; alle Komponenten eines Vektors x müssen vom

[5] Die Tatsache, dass $\langle x, x \rangle = \|x\|^2$ gilt, zeichnet die Euklidische Norm unter allen p-Normen aus. Dies gilt auch für die $p = 2$-Norm in Beispiel 1.3, die durch das Skalarprodukt $\langle f, g \rangle = \int_a^b f(x)g(x)\,dx$ induziert wird. Physikalisch kann diese Norm als eine Gesamtenergie interpretiert werden; durch die Energieerhaltungssätze der Physik bekommt sie deshalb zusätzliche Bedeutung. Dies überträgt sich auch auf die Euklidische Vektornorm, da numerische Verfahren häufig auf physikalische Probleme angewendet werden.

Betrag kleiner als $\|x\|_\infty$ sein. Die Euklidische Norm ist dagegen eine Schranke im Mittel (was man leicht sieht, indem man durch die Anzahl der Komponenten dividiert); einzelne Komponenten dürfen durchaus größer sein, solange sie nicht zu groß oder es zu viele sind. Dies gilt auch für die anderen p-Normen, wobei $p = 1$ eine Sonderstellung einnimmt: Die zugehörige Norm bestraft auch bei sehr kleinen Schranken Ausreißer stark. Dies wird in dem aktuellen Forschungsgebiet des *compressed sensing*[6] ausgenutzt.

1.3 DAS SCHWEIZER TASCHENMESSER DER APPROXIMATION: DIE TAYLORENTWICKLUNG

Ein wesentlicher Schritt in der Konstruktion und Untersuchung von Algorithmen zur Lösung kontinuierlicher Probleme ist deren Approximation durch endliche Darstellungen.

Ein einfaches Beispiel liefert die Berechnung der Exponentialfunktion, definiert durch die Reihe

$$e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!} = 1 + x + \frac{1}{2}x^2 + \frac{1}{6}x^3 + \dots.$$

Bricht man die Summation nach einem festen n ab, so erhält man einen endlichen Algorithmus, der eine Näherung der Exponentialfunktion liefert. Der Fehler, der dabei entsteht, ist ein *Approximationsfehler*. Diesen abzuschätzen ist eine wesentliche Aufgabe der numerischen Mathematik.

Ein Werkzeug dafür stellt die Taylorentwicklung dar, die auch die Approximation anderer Funktionen durch Reihen ermöglicht.

Satz 1.8 (Satz von Taylor)**.** *Sei* $I \subset \mathbb{R}$ *ein offenes Intervall,* f *eine* $n+1$ *mal stetig differenzierbare Funktion (d. h. die* $(n+1)$*-te Ableitung* $f^{(n+1)}$ *ist stetig), und* $x_0 \in I$*. Dann gilt für alle* $x \in I$

$$f(x) = \sum_{k=0}^{n} \frac{f^{(k)}(x_0)}{k!}(x - x_0)^k + R_{n+1}(x)$$

mit Restglied

$$R_{n+1}(x) = \frac{(x - x_0)^{n+1}}{(n+1)!} f^{(n+1)}(\xi)$$

für ein $\xi \in (x_0, x)$ *(bzw.* $\xi \in (x, x_0)$*).*

[6] Siehe http://statweb.stanford.edu/~candes/l1magic/examples.html. Dieser Ansatz erlaubt es theoretisch sogar, eine Digitalkamera zu bauen, die mit einem Bildsensor, der nur aus einem einzigen Pixel besteht, Bilder aufnehmen kann: http://terrytao.wordpress.com/2007/04/13/compressed-sensing-and-single-pixel-cameras/

Mit Hilfe des Restglieds haben wir auch sofort eine Abschätzung des Approximationsfehlers zur Hand.

Bemerkung (Taylorentwicklung im $\mathbb{R}^n$)**.** Analog lässt sich auch für eine Funktion $f : \mathbb{R}^n \to \mathbb{R}$ die Taylorentwicklung definieren, wenn man Ableitung durch den Vektor (die Matrix etc.) der partiellen Ableitungen ersetzt. Für den Fall der Taylorentwicklung zweiter Ordnung sei dies exemplarisch dargestellt. Sei $\Omega \subset \mathbb{R}^n$ offen, $f \in C^3(\Omega)$, $x_0 \in \Omega$. Dann existiert ein $\xi \in \Omega$, so dass gilt

$$f(x) = f(x_0) + \sum_{i=1}^{n} \frac{\partial f(x_0)}{\partial x_i}(x_i - (x_0)_i) + \frac{1}{2}\sum_{i,j=1}^{n} \frac{\partial^2 f(x_0)}{\partial x_j \partial x_i}(x_i - (x_0)_i)(x_i - (x_0)_j) + \frac{1}{6}\sum_{i,j,k=1}^{n} \frac{\partial^3 f(\xi)}{\partial x_k \partial x_j \partial x_i}(x_i - (x_0)_i)(x_i - (x_0)_j)(x_k - (x_0)_k).$$

Indem man den *Gradienten*

$$\nabla f(x) := \left(\frac{\partial f}{\partial x_1}(x), \ldots, \frac{\partial f}{\partial x_n}(x)\right)^T$$

und die *Hessematrix*

$$\nabla^2 f(x) := \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2}(x) & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n}(x) \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1}(x) & \cdots & \frac{\partial^2 f(x)}{\partial x_n^2}(x) \end{pmatrix}$$

einführt, lässt sich die Taylorentwicklung einfacher schreiben als

$$f(x) = f(x_0) + \langle \nabla f(x_0), x - x_0 \rangle + \frac{1}{2}\langle x - x_0, \nabla^2 f(x_0)(x - x_0) \rangle + R_3(x). \tag{1.1}$$

Dabei sind $(x - x_0)$, $\nabla f(x_0)$ und $\nabla^2 f(x_0)(x - x_0)$ Vektoren im $\mathbb{R}^n$.

Bemerkung. Die Taylorentwicklung ist natürlich kein Zaubermittel. Zwei wichtige Fallstricke:

(i) Die Differenzierbarkeit von f ist eine zwingende Voraussetzung; insbesondere muss die $n+1$-te Ableitung stetig sein.

(ii) Die Entwicklung wird in der Regel nur für x sehr nahe bei x_0 vernünftige Ergebnisse liefern. (Dies kann man sich anhand der Restglieddarstellung leicht verdeutlichen.)

1.4 LANDAU-SYMBOLE

Oft ist man nicht an dem Fehler für eine feste Größe n interessiert, sondern an dem asymptotischen Verhalten für große n interessiert. Eine kompakte Notation erlauben die Landau-Symbole,[7] die den Begriff der Größenordnung präzisieren:

[7] Die "Groß-O"-Notation geht im Wesentlichen auf Paul Bachmann im Jahre 1894 zurück, Lev Landau präzisierte den Begriff und fügte 1909 "klein-O" hinzu.

Definition 1.9 (Folgen)**.** Seien $\{f_n\}_{n\in\mathbb{N}}, \{g_n\}_{n\in\mathbb{N}} \subset \mathbb{R}$ Folgen. Dann sagen wir,

- $f_n = \mathcal{O}(g_n)$ für $n \to \infty$, falls *ein* $C > 0$ und ein n_0 existieren, so dass gilt

$$|f_n| \leqslant C|g_n| \qquad \text{für alle } n > n_0,$$

- $f_n = o(g_n)$ für $n \to \infty$, falls *für alle* $\varepsilon > 0$ ein n_0 existiert, so dass gilt

$$|f_n| \leqslant \varepsilon|g_n| \qquad \text{für alle } n > n_0.$$

Die zweite Definition ist äquivalent dazu, dass $|f_n| \leqslant c_n|g_n|$ für eine (positive) Nullfolge $\{c_n\}_{n\in\mathbb{N}}$ gilt.

Definition 1.10 (Funktionen)**.** Sei $I \subset \mathbb{R}$ ein offenes Intervall, $f, g : I \to \mathbb{R}$ und $x_0 \in \mathbb{R}$. Dann sagen wir,

- $f(x) = \mathcal{O}(g(x))$ für $x \to x_0$, falls *ein* $C > 0$ und ein $\delta > 0$ existieren, so dass gilt

$$|f(x)| \leqslant C|g(x)| \qquad \text{für alle } |x - x_0| < \delta,$$

- $f(x) = o(g(x))$ für $x \to x_0$, falls *für alle* $\varepsilon > 0$ ein $\delta > 0$ existieren, so dass gilt

$$|f(x)| \leqslant \varepsilon|g(x)| \qquad \text{für alle } |x - x_0| < \delta.$$

Eine Funktion $\mathcal{O}(f(x))$ verhält sich also für $x \to x_0$ im Wesentlichen wie $f(x)$, und eine Funktion $o(f(x))$ ist für x nahe bei x_0 gegenüber $f(x)$ vernachlässigbar. Analog definiert man die Landau-Symbole für $x \to \pm\infty$.

Bemerkung. Auch hier gibt es zwei wichtige Dinge zu beachten:

(i) Die Landau-Symbole sind nur sinnvoll in Verbindung mit der Angabe, für welche Umgebung sie gelten. Für eine andere Richtung als die angegebenen kann die Aussage falsch sein, wie das dritte Beispiel zeigt. Wenn dies aus dem Kontext ersichtlich ist, wird diese Angabe jedoch häufig weggelassen. So ist es Konvention, dass $n \to \infty$ und $\varepsilon, h \to 0$ verstanden wird.

(ii) Die Landau-Symbole stehen immer auf der rechten Seite einer Gleichung; sobald Terme "unter den Teppich gekehrt" wurden, wird das Gleichheitszeichen zur Einbahnstraße.[8]

[8] Formal steht $\mathcal{O}(f(x))$ für die Menge aller Funktionen, für die $C|f(x)|$ eine obere Schranke ist. Das Gleichheitszeichen steht also in Wirklichkeit – aus rein historischen Gründen – für eine Mengeninklusion, die bekanntlich nicht symmetrisch ist.

Für das Rechnen mit den Landau-Symbolen gibt es einige nützliche Regeln:

Lemma 1.11. *Es gilt jeweils für* $x \to x_0$*:*

(i) $f(x) = \mathcal{O}(f(x))$,

(ii) $f(x) = o(g(x)) \Rightarrow f(x) = \mathcal{O}(g(x))$,

(iii) $f(x) = \mathcal{O}(g(x)) \Rightarrow Kf(x) = \mathcal{O}(g(x))$ *für alle* $K \in \mathbb{R}$,

(iv) $f(x) = \mathcal{O}(g_1(x))$ *und* $g_1(x) = \mathcal{O}(g_2(x)) \Rightarrow f(x) = \mathcal{O}(g_2(x))$,

(v) $f_1(x) = \mathcal{O}(g_1(x))$ *und* $f_2(x) = \mathcal{O}(g_2(x)) \Rightarrow f_1(x)f_2(x) = \mathcal{O}(g_1(x)g_2(x))$,

(vi) $f(x) = \mathcal{O}(g(x) + h(x))$ *und* $h(x) = \mathcal{O}(g(x)) \Rightarrow f(x) = \mathcal{O}(g(x))$.

Beispiel 1.12. Es sind

(i) $3n + 5n^2 = 5n^2 + \mathcal{O}(n) = \mathcal{O}(n^2)$ für $n \to \infty$,

(ii) $\sin(x) = x - \frac{1}{6}x^3 + \mathcal{O}(x^5)$ für $x \to 0$,

(iii) $x^n = o(e^x)$ für $x \to \infty$.

Für das Restglied der Taylorentwicklung (und damit den Approximationsfehler) haben wir daher sofort, dass gilt

$$\|R_{n+1}(x)\| = \mathcal{O}(\|x - x_0\|^{n+1}) \qquad \text{für } x \to x_0.$$

Allgemein können wir mit Hilfe der Landau-Symbole die Geschwindigkeit, mit der eine Folge konvergiert, mathematisch sinnvoll definieren.

Definition 1.13. Sei $(V, \|\cdot\|)$ ein normierter Raum, $x^* \in V$, und $\{x_n\}_{n\in\mathbb{N}} \subset V$ eine Folge mit $x_n \to x^*$. Wir sagen,

(i) x_n konvergiert (mindestens) *linear*, wenn es ein n_0 und ein $C < 1$ gibt, so dass gilt

$$\|x_{n+1} - x^*\| \leqslant C\|x_n - x^*\|) \quad \text{für alle } n > n_0.$$

(Beachte, dass dies eine stärkere Forderung als $\|x_{n+1} - x^*\| = \mathcal{O}(\|x_n - x^*\|)$ ist.)

(ii) x_n konvergiert (mindestens) *superlinear*, wenn es ein n_0 gibt, so dass gilt

$$\|x_{n+1} - x^*\| = o(\|x_n - x^*\|) \quad \text{für alle } n > n_0,$$

(iii) x_n besitzt (mindestens) *Konvergenzordnung* k, wenn es ein $C > 0$ und ein n_0 gibt, so dass gilt

$$\|x_{n+1} - x^*\| = \mathcal{O}(\|x_n - x^*\|^k) \quad \text{für alle } n > n_0.$$

Für den Fall $k = 2$ spricht man von *quadratischer Konvergenz*.

Bemerkung. (i) Bei linearer Konvergenz hängt die tatsächliche Konvergenzgeschwindigkeit stark von der Konstanten C ab: Für $C = 0.1$ gewinnt man pro Schritt eine Stelle Genauigkeit, für $C = 0.9$ sind dafür schon 10 bis 11 Schritte nötig. Ist C sehr nahe bei 1, kann die lineare Konvergenz sehr langsam sein.

(ii) Quadratische Konvergenz ist sehr schnell, wenn man schon hinreichend nahe der Lösung ist. Ist man jedoch weit von der Lösung entfernt, sagt die Abschätzung sehr wenig aus; tatsächlich ist sehr langsame Konvergenz möglich.

(iii) Man muss auch den Aufwand für einen einzelnen Iterationsschritt berücksichtigen. Dieser ist für ein quadratisch konvergierendes Verfahren in der Regel höher als für ein linear konvergierendes Verfahren, so dass, am Gesamtaufwand gemessen, ein linear konvergierendes Verfahren günstiger sein kann.

1.5 KOMPLEXITÄT VON ALGORITHMEN

Die Landau-Symbole erlauben es auch, bequem den Aufwand zu beschreiben, den man für die Ausführung eines Algorithmus in Abhängigkeit der Problemgröße n aufwenden muss. Dabei unterscheidet man üblicherweise

1. *Laufzeitkomplexität* (gemessen z. B. durch die Anzahl der elementaren Schritte) und
2. *Speicherkomplexität* (gemessen z. B. durch die Anzahl und Umfang der benötigten Hilfsgrößen).

Oft wird man eine niedrigere Laufzeitkomplexität gegen eine höhere Speicherkomplexität oder umgekehrt abwägen müssen.

Beispiel 1.14 (Komplexität der Polynomauswertung)**.** Wir betrachten ein Polynom $P_n(x) = \sum_{i=0}^{n} a_i x^i$ mit $a_i \in \mathbb{R}$. Für gegebenes $x_0 \in \mathbb{R}$ bestimmen wir jetzt den Wert $P_n(x_0)$ des Polynoms an der Stelle x_0 auf zwei verschiedene Arten, wobei wir als elementare Schritte nur Addition und Multiplikation zulassen. Die naive Variante rechnet die das Polynom Summand für Summand aus, wobei die Potenzen als entsprechende Anzahl von Multiplikationen berechnet werden.

Algorithmus 1.1 : Polynomauswertung (naiv)

Input : $x_0, a_0, \ldots, a_n$
Output : p_0

```
1 p_0 ← a_0
2 for k = 1, ..., n do                          // Berechne a_k x_0^k
3 |   M ← a_k
4 |   for m = 1, ..., k do                      // Berechne x_0^k = x_0 ··· x_0
5 |   |   M ← M · x_0
6 |   p_0 ← p_0 + M
```

Zählen wir die Operationen pro Schritt, werden n mal je eine Addition und k Multiplikationen ausgeführt. Setzen wir für jede Addition und Multiplikation eine konstante Zeit C_1 bzw. C_2 an (und vernachlässigen die Zeit für die Zuweisungen), so erhalten wir als Laufzeitkomplexität

$$\sum_{k=1}^{n} (C_1 + C_2 k) = C_1 n + C_2 \frac{1}{2} n(n+1) = \mathcal{O}(n^2).$$

Die Speicherkomplexität ist $2 = \mathcal{O}(1)$, da – unabhängig von n – nur zwei Speicherplätze, p_0 und M, benötigt werden.

Dagegen basiert das *Horner-Schema* auf der Umformung

$$P_n(x) = a_0 + x\,(a_1 + x\,(\cdots + x(a_{n-1} + x a_n)))\,.$$

Rechnen wir die Klammern von Innen nach außen durch, erhalten wir den folgenden Algorithmus.

Algorithmus 1.2 : Polynomauswertung (Horner-Schema)

Input : $x_0, a_0, \ldots, a_n$
Output : p_0

```
1 p_0 ← a_n
2 for k = n − 1, ..., 0 do
3     p_0 ← p_0 · x_0 + a_k
```

Die Speicherkomplexität ist zwar immer noch $1 = \mathcal{O}(1)$, aber die Berechnung erfordert nur noch pro Schritt je eine Addition und Multiplikation: Die Laufzeitkomplexität ist hier also $2n = \mathcal{O}(n)$. Der Unterschied zwischen $\mathcal{O}(n)$ und $\mathcal{O}(n^2)$ kann für große n beträchtlich sein. Dies wird uns im Kapitel über lineare Gleichungssysteme wieder begegnen.

WEITERFÜHRENDE LITERATUR

R. L. Graham, D. E. Knuth und O. Patashnik (1994). *Concrete mathematics*. 2. Aufl. A foundation for computer science. Addison-Wesley Publishing Company, Reading, MA. URL: http://www-cs-faculty.stanford.edu/~uno/gkp.html.

NUMERISCHE ARITHMETIK

2

Aus der Algebra wissen wir, dass wir für die Lösung bestimmter Aufgaben (zum Beispiel die Nullstellensuche von Polynomen) nicht mit einem diskreten Zahlbegriff auskommen. Für eine numerische Behandlung müssen wir jedoch eine endliche Näherung verwenden. Es ist klar, dass dies nicht ohne Fehler geschehen kann, die aber bei sorgfältiger Behandlung beherrschbar sind. Diese können aber nie ganz vermieden werden: Bereits bei der Eingabe entsteht ein Fehler.

Da sich komplexe Zahlen als Paar reeller Zahlen auffassen lassen, betrachten wir nur das Problem, reelle Zahlen auf endlichen Maschinen, insbesondere dem Computer, darzustellen.

2.1 ZAHLDARSTELLUNG

Grundlage der Maschinendarstellung von Zahlen ist die Dezimalbruchentwicklung, allgemeiner die Basisdarstellung $x = \pm \sum_{i=-\infty}^{\infty} a_i b^i$ mit $b \in \mathbb{N} \setminus \{1\}$ und $a_i \in \{0, \dots, b-1\}$ und $b > 1$. Die Basisdarstellung einfach nach einer festen Anzahl von Stellen abzuschneiden, stellt keine befriedigende Lösung dar, da in einer Rechnung oft Zahlen sehr verschiedener Größenordnungen auftauchen.[1] Deshalb geht man aus von der *Gleitkommadarstellung*[2]

$$x = \sigma b^e \sum_{i=1}^{m} a_i b^{-i}.$$

Diese Darstellung enthält

[1] Dies spricht auch gegen eine Approximation durch rationale Zahlen, da diese für sehr kleine oder sehr große Zahlen unhandlich werden. Auch verhindert die für eine eindeutige Darstellung notwendige Kürzung eine effiziente Nutzung.

[2] Die Gleitkommadarstellung wurde bekanntlich schon im Babylon des 19. Jahrhunderts v. Chr. mit der Basis 60 verwendet. Leider kannte man keine Notation für den Exponenten, so dass man sich für korrekte Rechnung auf das Verständnis des Rechners verlassen musste. Es sind auch (allerdings sehr wenige) Fälle bekannt, in denen falscher Stellenabgleich zu fehlerhafter Addition führte. Die moderne Nutzung des Stellensystems setzt mit der Erfindung des Logarithmus um 1600 ein.

- das *Vorzeichen* $\sigma \in \{-1, +1\}$,
- die *Basis* $b \in \mathbb{N}, b > 1$,
- die *Mantisse* $a_1 \dots a_m$ mit $a_i \in \{0, 1, \dots, b-1\}$ und der *Mantissenlänge* m,
- den *Exponent* $e \in \mathbb{Z}$ mit $e_{\min} \leqslant e \leqslant e_{\max}$.

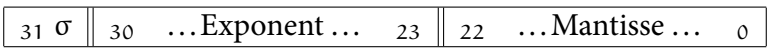
Eindeutigkeit der Darstellung lässt sich über zusätzliche Forderungen an die a_i erreichen. Die üblichste, sogenannte *normalisierte* Darstellung erhält man, indem für $x \neq 0$ gefordert wird, dass $a_1 \neq 0$ gilt. Ein bekanntes Beispiel ist die wissenschaftliche Notation, etwa $0.14 \cdot 10^{-3}$ mit $b = 10$ und $e = -3$. Für die Basis $b = 2$ führt das zu der gebräuchlichsten Gleitkommadarstellung nach dem Standard IEEE 754, die (seit 1985) Grundlage aller Rechnerarchitekturen ist.

2.1.1 GLEITKOMMADARSTELLUNG NACH IEEE 754

Der Standard[3] definiert drei Zahltypen: *single*, *double*, und *extended precision*, die sich in der Bitlänge (32, 64, bzw. 80 Bit; ein Bit entspricht dem Speicherplatz für eine Binärziffer, 0 oder 1) unterscheiden. Ein Bit ist jeweils für das Vorzeichen reserviert, die restlichen Bits werden entsprechend auf Exponent und Mantisse verteilt:

	single	double	extended
Exponent	8 Bit	11 Bit	15 Bit
Mantisse	23 Bit	52 Bit	64 Bit

Die Kodierung einer Zahl in IEEE 754 ist wie folgt definiert (hier am Beispiel single precision):



- Das Vorzeichenbit $\sigma = 0$ bezeichnet eine positive Zahl, $\sigma = 1$ eine negative Zahl.
- Der Exponent ist als verschobene Binärzahl gespeichert; die Verschiebung ist so gewählt, dass nur positive Exponenten gespeichert werden müssen. Das Bitmuster $01 \dots 1$ steht für den Exponenten 0, größere Binärzahlen für positive und kleinere für negative Exponenten. Bei single precision etwa muss also von der gespeicherten Zahl 127 abgezogen werden, so dass der Wertebereich des Exponenten $[-126, 127]$ ist.
- Die Mantisse speichert die Nachkommastellen der Binärzahl, wobei die Normalisierung so erfolgt, dass die Vorkommastelle immer 1 ist (und deshalb nicht gespeichert werden muss).

[3] Der im wesentlichen Konrad Zuses Patent von 1936 für die "selbständige Durchführung von Rechnungen mit Hilfe von Rechenmaschinen" folgt; allerdings ohne es zu zitieren.

- Zusätzlich existieren spezielle Bitmuster, um Sonderfälle anzuzeigen. Sind im Exponenten nur Einsen gesetzt, und ist die Mantisse gleich Null, so wird dies als $\pm\infty$ interpretiert. Ist mindestens ein Mantissenbit gesetzt, so steht dies für NaN – "Not a Number". Dies kann etwa durch eine nicht definierte arithmetische Operation wie $0 \cdot \infty$ ausgelöst werden. Die (positive oder negative) 0 wird durch Nullen in Exponent und Mantisse ausgedrückt.

Beispiel 2.1. (i) Wir betrachten die Darstellung nach IEEE 754 der Dezimalzahl 12.375. Zunächst müssen wir sie in Binärdarstellung umrechnen:

$$\begin{aligned}(12.375)_{10} &= (12)_{10} + (0.375)_{10} = (8+4)_{10} + (\tfrac{1}{4} + \tfrac{1}{8})_{10} \\ &= (1100)_2 + (0.011)_2 = (1100.011)_2.\end{aligned}$$

Die normalisierte Darstellung dieser Binärzahl ist (Kommaverschiebung entspricht Multiplikation mit Zweierpotenz)

$$(1100.011)_2 = (1.100011)_2 \cdot 2^3.$$

Die Nachkommastellen (von rechts mit Null aufgefüllt) ergeben genau die Mantisse. Durch die Verschiebung muss der Exponent nun wegen $3 = 130 - 127$ als 130 ebenfalls in Binärdarstellung gespeichert werden, d. h.

$$(130)_{10} = (128+2)_{10} = 10000010_2.$$

Schließlich ist die Zahl positiv ($\sigma = 0$); das gesamte zu speichernde Bitmuster ist also

0	10000010	1000110…0

(ii) Sei umgekehrt das Bitmuster

1	01111100	010…0

gegeben. Wegen $\sigma = 1$ ist die entsprechende Zahl negativ. Der Exponent ist gespeichert als

$$(01111100)_2 = \left(\sum_{k=2}^{6} 2^k\right)_{10} = (124)_{10} = (-3+127)_{10},$$

d. h. $e = -3$. Da in der Mantisse nur ein Bit, nämlich a_{21}, gesetzt ist, ist die Nachkommastelle

$$(0.01)_2 = (2^{-2})_{10} = (\tfrac{1}{4})_{10}.$$

Nach IEEE 754 entspricht dies also der Zahl

$$x = (-1)^\sigma \left(1 + \sum_{i=1}^{23} a_{23-i} 2^{-i}\right) \cdot 2^e = (-1)\left(1 + \frac{1}{8}\right) \cdot 2^{-3} = -0.15625.$$

Die größte und kleinste darstellbare Zahl wird im Wesentlichen durch $e_{\min}$ und $e_{\max}$ bestimmt:

	kleinste positive Zahl	größte positive Zahl
single prec.	$2^{-126} \approx 1.4 \cdot 10^{-38}$	$2^{128} - 1 \approx 3.4 \cdot 10^{38}$
double prec.	$2^{-1022} \approx 2.2 \cdot 10^{-308}$	$2^{1024} - 1 \approx 1.8 \cdot 10^{308}$
extended prec.	$2^{-16382} \approx 3.3 \cdot 10^{-4962}$	$2^{16384} - 1 \approx 1.2 \cdot 10^{4962}$

Während die Exponentenlänge den maximalen Wertebereich angibt, bestimmt die Mantissenlänge die *relative* Genauigkeit. Ein Nachteil der halb-logarithmischen Darstellung ist nämlich, dass die so darstellbaren Zahlen nicht gleichmäßig im überdeckten Zahlenbereich verteilt sind: So haben die für $b = 10$, $m = 1$ verfügbaren zehn Zahlen $0.1, \ldots, 0.9$ bei $e = -1$ den Abstand 0.01, für $e = 2$ jedoch bereits den Abstand 10.

Dem extended precision Format kommt eine besondere Bedeutung zu, da es Grundlage für die interne Berechnung (in den sogenannten *Registern*) von Computern ist. Auch dies geht auf den IEEE Standard zurück, der fordert, dass genauer gerechnet wird als dargestellt. Deshalb muss nach jeder Elementaroperation das Ergebnis mit geringerer Genauigkeit abgespeichert werden: man spricht von *Rundung*.

2.2 GLEITKOMMAARITHMETIK UND RUNDUNGSFEHLER

Durch die Exponenten- und die Mantissenlänge wird eine endliche Menge darstellbarer Zahlen definiert. Diese stellen in der Regel keinen abgeschlossenen Körper dar, da das Produkt zweier solcher Zahlen den darstellbaren Wertebereich verlassen kann. Um dem zu begegnen, fordert der IEEE Standard, dass jede arithmetische Operation

- intern immer mit höherer Genauigkeit durchgeführt[4] und
- danach korrekt gerundet werden soll.

Damit und mit den zusätzlichen "Sonderzahlen" ∞ und NaN wird sichergestellt, dass das Resultat jeder beliebigen arithmetischen Operation wieder als Gleitkommazahl darstellbar ist.

In den allermeisten Fällen wird die Rundung zur nächsten darstellbaren Zahl durchgeführt. Durch Einsetzen der Darstellung und Fallunterscheidung zeigt man die folgende Fehlerabschätzung.

[4] Dies kann zu einem arithmetischen Heisenberg-Effekt führen: Lässt man sich Zwischenergebnisse einer Rechnung, die eigentlich komplett in den Registern ablaufen würde, ausgeben, werden diese dafür mit geringerer Genauigkeit gespeichert, was zu Rundungsfehlern führen wird. Das Beobachten kann also die Ergebnisse der Rechnung verändern!

Satz 2.2. *Sei* $x \in [-b^{e_{min}}, b^{e_{min}}] \setminus \{0\} \subset \mathbb{R}$*, und sei* $fl(x)$ *die durch Rundung erhaltene Gleitkommazahl. Dann gilt für den* relativen Fehler

$$\frac{|x - fl(x)|}{|x|} \leqslant \frac{1}{2} b^{1-m} =: u.$$

Die rechte Seite gibt eine fundamentale Konstante für die Gleitkommazahldarstellung an, den *Rundungsfehler*. Sie entspricht auch der kleinsten Zahl, für die im Rechner $1 + u$ nicht gleich 1 ergibt. Eine verwandte häufig verwendete Konstante ist die *Maschinengenauigkeit* eps, die den Abstand zweier benachbarter Gleitkommazahlen zwischen 1 und 2 angibt; es gilt $eps = 2u$. Für single precision gilt etwa $eps = 2^{-23} \approx 1.192 \cdot 10^{-7}$, für double precision $eps = 2^{-52} \approx 4.220 \cdot 10^{-16}$.

Bemerkung. Man kann sich merken: single precision erlaubt etwa 6, double precision etwa 15, und extended precision etwa 19 verlässliche Nachkommastellen.

Wenn man dem IEEE Standard folgt, ergibt sich daraus eine Fehlerabschätzung für Gleitkommaoperationen.

Satz 2.3. *Sei* $\circ \in \{+, -, \times, /\}$ *eine arithmetische Operation, und* $\tilde{\circ}$ *ihre Realisierung in Gleitkommaarithmetik. Dann gilt*

$$x \,\tilde{\circ}\, y = fl(x \circ y) \leqslant (x \circ y)(1 + eps).$$

Dadurch, dass der entstandene Fehler von den Eingangsdaten abhängt, ist die Gleitkommaarithmetik in der Regel *weder assoziativ noch distributiv*!

Beispiel 2.4. Wir wählen der Einfachheit halber $b = 10, m = 3$. Eine arithmetische Gleitkomma-Addition wird durchgeführt, indem zuerst die Exponenten (auf den größeren) angeglichen werden, die Mantissen addiert werden, das Ergebnis normalisiert und zuletzt gerundet wird (und ähnlich für die Multiplikation).

(i) Wir wählen $x = 6590, y = 1, z = 4$, d. h. $x + y + z = 6595$. In Gleitkommadarstellung haben wir $fl(x) = 0.659 \cdot 10^4$, $fl(y) = 0.100 \cdot 10^1$, $fl(z) = 0.4 \cdot 10^1$, und die Gleitkomma-Summe ist

$$(x \,\tilde{+}\, y) \,\tilde{+}\, z = 0.659 \cdot 10^4 \,\tilde{+}\, z = 0.659 \cdot 10^4,$$
$$x \,\tilde{+}\, (y \,\tilde{+}\, z) = x \,\tilde{+}\, 0.500 \cdot 10^1 = 0.660 \cdot 10^4,$$

wobei das zweite Ergebnis der Rundung nach exakter Rechnung entspricht.

(ii) Wir wählen jetzt $x = 0.156 \cdot 10^2, y = 0.157 \cdot 10^2$ und berechnen $(x-y) \times (x-y) = 0.01$. In Gleitkommaarithmetik ergibt sich jedoch:

$$(x \,\tilde{-}\, y) \,\tilde{\times}\, (x \,\tilde{-}\, y) = 0.100 \cdot 10^{-1},$$
$$(x \,\tilde{\times}\, x) \,\tilde{-}\, (x \,\tilde{\times}\, y) \,\tilde{-}\, (y \,\tilde{\times}\, x) \,\tilde{+}\, (y \,\tilde{\times}\, y) = -0.100 \cdot 10^{-1}.$$

(iii) Wir wählen $x = 0.73563$, $y = 0.73441$. Dann ist $x - y = 0.00122$, aber

$$\mathrm{fl}(x) \tilde{-} \mathrm{fl}(y) = 0.200 \cdot 10^{-2}.$$

Das letzte Phänomen wird *Auslöschung signifikanter Stellen* genannt. Hier wurde der relative Fehler in den Eingangsdaten von etwa 10^{-3} um nahezu vier Größenordnungen verstärkt. Die Subtraktion beinahe gleich großer Zahlen sollte also nach Möglichkeit vermieden werden, insbesondere, wenn danach durch eine sehr kleine Zahl geteilt wird.[5]

Offensichtlich spielt die Reihenfolge der Operationen also eine wesentliche Rolle; eine ungünstige Wahl kann die unvermeidlichen Fehler verstärken, bis das Ergebnis wertlos ist, eine günstige Wahl die Fehler hingegen dämpfen. Damit werden wir uns im nächsten Kapitel genauer beschäftigen.

WEITERFÜHRENDE LITERATUR

D. Goldberg (1991). *What every computer scientist should know about floating-point arithmetic*. ACM Comput. Surv. 23.1, S. 5–48. DOI: 10.1145/103162.103163.

J. H. Wilkinson (1994). *Rounding errors in algebraic processes*. Reprint of the 1963 original. Dover Publications, Inc., New York.

[5] Was in der Praxis auch unbedingt vermieden werden sollte, ist der Vergleich von zwei Gleitkommazahlen x und y auf Gleichheit, da eine unterschiedliche Reihenfolge in der Berechnung zweier an sich gleicher Zahlen in Gleitkommaarithmetik verschiedene Ergebnisse produzieren kann. Besser ist der Vergleich $|x - y| < eps \max(|x|, |y|)$.

FEHLERANALYSE

3

Wir haben gesehen, dass bei der konkreten Durchführung von mathematischen Operationen unweigerlich Fehler entstehen. Diese lassen sich unterteilen in

- *Datenfehler*: Die Eingaben vieler Berechnungen sind physikalische Messwerte oder empirische Untersuchungen, die durch Messungenauigkeiten immer fehlerbehaftet sind. Sie treten aber auch bei rein mathematischen Berechnungen wie $r^2\pi$ durch die Darstellung reeller Zahlen als Gleitkommazahlen auf.
- *Verfahrensfehler*: Häufig kann eine Aufgabe numerisch nur näherungsweise gelöst werden (etwa die Berechnung von e^x durch endlich Summation). Dabei entstehen Approximations- bzw. Diskretisierungsfehler (wozu auch Abbruchfehler bei iterativen Verfahren zählen).
- *Rundungsfehler*: Sind wie beschrieben die Folge der arithmetischen Grundoperationen mit Gleitkommazahlen, die die elementarsten Schritte jedes numerischen Algorithmus darstellen.

Hinzu kommen in der Praxis noch *Modellfehler*, die darauf beruhen, dass bei der Aufstellung des mathematischen Modells idealisierte Annahmen getroffen werden, die in der Realität nicht zutreffen (siehe Sleipner A), und *Darstellungsfehler*, die auf einer unzureichenden Interpretation des Ergebnisses beruhen (etwa durch Visualisierung komplexer dreidimensionaler Strömungen, in der wichtige Details zu Gunsten einer ansprechenden Darstellung vernachlässigt werden[1]). Nicht zu vernachlässigen sind auch *semantische Fehler*, die bei der Implementierung der Algorithmen entstehen (etwa Programmierfehler).

Da der Datenfehler außerhalb unserer Kontrolle liegt, müssen wir uns zuerst fragen, ob die gestellte Aufgabe prinzipiell unter dieser Voraussetzung sinnvoll lösbar ist. Erst dann können wir fragen, ob sich ein spezieller Algorithmus für deren Lösung eignet.

[1] Dieses oft in der numerischen Strömungsmechanik (engl. "computational fluid dynamics", CFD) anzutreffende Phänomen hat den Spitznamen "colorful fluid dynamics" für diese Disziplin geprägt.

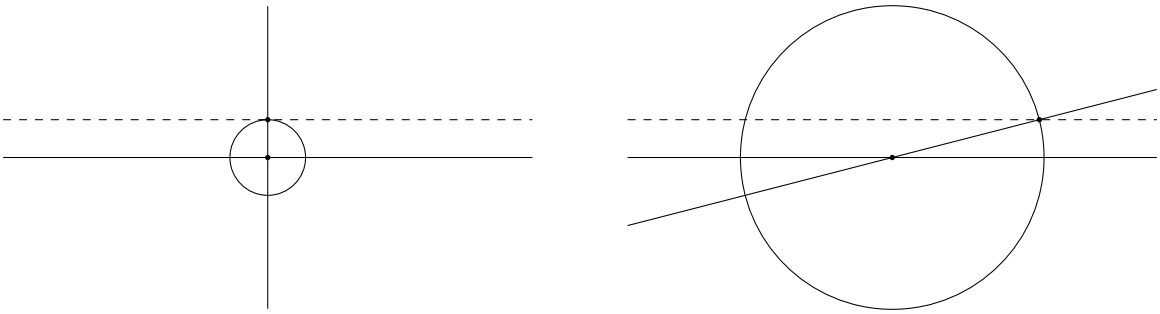


Abbildung 3.1: Schnittpunktbestimmung zweier Geraden. Links: gut konditioniert; rechts: schlecht konditioniert. (Kreisradius entspricht Fehler).

3.1 KONDITION EINES PROBLEMS

Die erste Frage beinhaltet, wie sensibel die Lösung eines Problem (unabhängig von dem verwendeten Verfahren) auf Störungen der Eingangsdaten reagiert. Führen kleine Änderungen der Eingabe auch nur zu kleinen Änderungen der Ausgabe, so nennt man ein Problem *gut konditioniert*. Können sich hingegen die Ausgaben nur leicht unterschiedlicher Daten stark unterscheiden, so spricht man von einem *schlecht konditionierten* Problem.

Ein einfaches Beispiel soll dies verdeutlichen:

Beispiel 3.1. Möchte man den Schnittpunkt zweier Geraden bestimmen (siehe Abb. 3.1), die annähernd senkrecht aufeinander stehen, so wird eine kleine Verschiebung einer Geraden (gestrichelte Linie) den Schnittpunkt nicht viel bewegen. Sind die Geraden hingegen fast parallel, ändert sich der Schnittpunkt stark, wenn eine Gerade verschoben wird. Das Problem ist also im Allgemeinen schlecht konditioniert. Wir sehen auch, dass die Kondition von den Eingabedaten abhängen kann.

Hinter der Kondition erkennt man sofort eine Stetigkeitsaussage; es liegt also nahe, den Begriff mathematisch zu präzisieren. Wir wollen dazu wieder ein Problem auffassen als eine Abbildung $f : X \to Y$, die für eine Eingabe x aus der Menge der zulässigen Eingabedaten X ein Resultat $y = f(x)$ aus der Menge der möglichen Resultate Y liefert, zusammen mit der gewünschten Eingabe x und einem maximalen Datenfehler[2] δ. Die fehlerbehaftete Eingabe bezeichnen wir mit $\tilde{x}$. Die Kondition drückt dann aus, wie stark sich die Eingabefehler auf das Ergebnis auswirken:

Definition 3.2. Für ein Problem (f, x, δ) ist die *absolute Kondition* die kleinste Zahl κ_{abs}, für die gilt

$$\|f(\tilde{x}) - f(x)\| \leqslant \kappa_{abs}\|\tilde{x} - x\| \qquad \text{für alle } \|\tilde{x} - x\| \leqslant \delta.$$

[2]Dieser wird für uns in der Regel der Maschinengenauigkeit eps entsprechen.

Die *relative Kondition* ist die kleinste Zahl κ_{rel}, für die gilt

$$\frac{\|f(\tilde{x}) - f(x)\|}{\|f(x)\|} \leqslant \kappa_{\text{rel}} \frac{\|\tilde{x} - x\|}{\|x\|} \qquad \text{für alle } \|\tilde{x} - x\| \leqslant \delta.$$

Ein gut konditioniertes Problem liegt dann vor, wenn $\kappa_{\text{rel}} \approx 1$ ist, der relative Fehler also in der Größenordnung des unvermeidlichen relativen Eingangsfehlers liegt. Da diese Definition unabhängig von Problem und Eingabedaten ist, wird man sich in der Regel nur für die relative Kondition interessieren.

Über die Taylorentwicklung von $f(x)$ erhält man sofort eine Abschätzung der Konditionen über die Ableitung:

Satz 3.3. *Ist* f *stetig differenzierbar, so gilt für das Problem* (f, x, δ)

$$\kappa_{\text{abs}} = \|f'(x)\| + \mathcal{O}(\delta^2), \qquad \kappa_{\text{rel}} = \|f'(x)\| \frac{\|x\|}{\|f(x)\|} + \mathcal{O}(\delta^2).$$

Analog lässt sich die Kondition eines Problems mit mehreren Eingabedaten $x_1, \ldots, x_n$ über die partiellen Ableitungen darstellen, z. B. durch

$$\kappa_{\text{abs}} = \max_{1 \leqslant j \leqslant n} \left\| \frac{\partial f}{\partial x_j}(x) \right\| + \mathcal{O}(\delta^2), \quad \kappa_{\text{rel}} = \max_{1 \leqslant j \leqslant n} \left\| \frac{\partial f}{\partial x_j}(x) \cdot \frac{x_j}{f(x)} \right\| + \mathcal{O}(\delta^2).$$

Für Matrizen werden wir noch eine spezielle Definition kennen lernen.

Beispiel 3.4. Für die Kondition der Addition (bzw. Subtraktion) betrachten wir für festes $y \neq 0$ die Funktion $f(x) := x + y$. Dann ist $f'(x) = 1$, es gilt also:

$$\kappa_{\text{rel}} = \frac{|x|}{|x + y|}.$$

Wir erkennen das Phänomen der Auslöschung wieder: ist $x \approx -y$, so wird der Nenner beliebig klein, und das Problem ist schlecht gestellt. Die Addition positiver Zahlen ist hingegen gut konditioniert, da dann immer $\frac{|x|}{|x+y|} \leqslant 1$ gilt.

Wir halten fest: Die Kondition ist eine Eigenschaft des Problems, nicht des Lösungsverfahrens. Es kann aber durchaus möglich sein, eine konkrete Fragestellung durch ein anderes (mathematisches) Problem auszudrücken, das besser konditioniert ist!

3.2 STABILITÄT EINES ALGORITHMUS

Für ein gut konditioniertes Problem[3] müssen wir nun entscheiden, ob das eingesetzte Verfahren *numerisch stabil* ist, d. h. neben den Eingabefehlern auch die Verfahrens- und Rundungsfehler nur geringen Einfluss auf das Resultat haben. Wir betrachten also ein numerisches Verfahren als Näherung $\tilde{f}$ eines Problems f, und fragen uns, wie sich $\tilde{f}(x)$ von $f(x)$ unterscheidet. Da Eingabefehler unvermeidlich sind, sind wir zufrieden, wenn $\|\tilde{f}(\tilde{x}) - f(x)\|$ in der Größenordnung von $\|f(\tilde{x}) - f(x)\|$ liegt. Dies lässt sich auf zwei Arten untersuchen:

3.2.1 VORWÄRTSANALYSE

Hierbei wird betrachtet, wie das Verfahren den durch die Kondition bestimmten, unausweichlichen, Fehler verstärkt oder dämpft.

Definition 3.5 (Vorwärtsstabilität)**.** Sei $\tilde{f}$ ein numerisches Verfahren zur Lösung des Problems (f, x, δ). Der *Stabilitätsindikator* (der Vorwärtsanalyse) ist die kleinste Zahl $\sigma_v \geqslant 0$, für die gilt

$$\frac{\|\tilde{f}(\tilde{x}) - f(\tilde{x})\|}{\|f(\tilde{x})\|} \leqslant \sigma_v \kappa_{\text{rel}} \delta \qquad \text{für alle } \tilde{x} \text{ mit } \frac{\|\tilde{x} - x\|}{\|x\|} \leqslant \delta.$$

Ein Verfahren heißt *(vorwärts)stabil*, falls $\sigma_v \kappa_{\text{rel}}$ kleiner als die Anzahl der durchgeführten Elementarschritte ist.

Beispiel 3.6. Für die arithmetischen Operationen $+, -, \times, /$ und ihre Gleitkommarealisierungen $\tilde{+}, \tilde{-}, \tilde{\times}, \tilde{/}$ ist $\sigma_v \kappa_{\text{rel}} \leqslant 1$, denn nach Satz 2.3 gilt für die Gleitkommaarithmetik $x \tilde{\circ} y \leqslant (x \circ y)(1 + \text{eps})$ und damit

$$\frac{|x \tilde{\circ} y - x \circ y|}{|x \circ y|} \leqslant \frac{|(x \circ y)(1 + \text{eps}) - x \circ y|}{|x \circ y|} = \text{eps}.$$

Da für Gleitkommazahlen $\delta \geqslant \text{eps}$ gilt (wir müssen mindestens mit Rundungsfehlern rechnen), folgt die Aussage.

In der Praxis ist es jedoch umständlich, die Konditionszahl eines Problems und die tatsächliche Anzahl der Schritte zu bestimmen. Für komplexere Verfahren bedient man sich deshalb oft einer anderen Methode.

[3] und nur für ein solches!

3.2.2 RÜCKWÄRTSANALYSE

Die Idee der Rückwärtsanalyse[4] ist es, den Verfahrensfehler auch wie einen Eingangsfehler zu behandeln: Das Ergebnis $\tilde{y} = \tilde{f}(\tilde{x})$ wird als das exakte Resultat $\tilde{y} = f(\hat{x})$ für gestörte Daten $\hat{x}$ angesehen. Es ist klar, dass das nur möglich ist, falls $\tilde{y}$ überhaupt eine mögliche Lösung des Problems f ist. Ist dies nicht der Fall, kann das Verfahren also sinnlose Ergebnisse liefern, betrachten wir den Algorithmus von vornherein als instabil.

Definition 3.7 (Rückwärtsstabilität)**.** Sei $\tilde{f}$ ein numerisches Verfahren zur Lösung des Problems (f, x, δ). Der *Stabilitätsindikator* (der Rückwärtsanalyse) ist die kleinste Zahl $\sigma_r \geqslant 0$, für die für alle $\tilde{x}$ mit $\|\tilde{x} - x\|/\|x\| \leqslant \delta$ ein $\hat{x}$ existiert mit $\tilde{f}(\tilde{x}) = f(\hat{x})$ und

$$\frac{\|\hat{x} - \tilde{x}\|}{\|\tilde{x}\|} \leqslant \sigma_r \delta.$$

Ein Verfahren heißt *(rückwärts)stabil*, falls $\sigma_r \approx 1$ ist.

Während die Vorwärtsanalyse untersucht, wie gut das Ergebnis des Verfahrens mit der exakten Lösung übereinstimmt, überprüft die Rückwärtsanalyse, wie gut das Ergebnis das exakte Problem löst. Der Vorteil der Rückwärtsanalyse ist dabei, dass der Indikator direkt die Stabilität des Verfahrens angibt. Darüber hinaus gilt:

Lemma 3.8. *Für ein gut konditioniertes Problem* (f, x, δ) *ist jedes rückwärtsstabile Verfahren* $\tilde{f}$ *auch vorwärtsstabil.*

Beweis. Nach Definition existiert für ein rückwärtsstabiles Verfahren ein $\hat{x}$, so dass $\tilde{f}(\tilde{x}) = f(\hat{x})$ und

$$\frac{\|\hat{x} - \tilde{x}\|}{\|\tilde{x}\|} \leqslant \sigma_r \delta$$

gilt. Hat f die Kondition κ_{rel}, dann erfüllt der relative Fehler im Ergebnis daher.

$$\frac{\|\tilde{f}(\tilde{x}) - f(\tilde{x})\|}{\|f(\tilde{x})\|} = \frac{\|f(\hat{x}) - f(\tilde{x})\|}{\|f(\tilde{x})\|} \leqslant \kappa_{\text{rel}} \frac{\|\hat{x} - \tilde{x}\|}{\|\tilde{x}\|} \leqslant \kappa_{\text{rel}} \sigma_r \delta.$$

Nach Annahme ist nun das Problem gut konditioniert und daher $\kappa_{\text{rel}} \approx 1$. Wegen der Rückwärtsstabilität ist weiter $\sigma_v := \sigma_r \approx 1$ und damit auch $\sigma_v \kappa_{\text{rel}} \approx 1$. Das Verfahren ist also auch vorwärtsstabil. □

[4] Für die Fehleranalyse wurde sie von James Hardy Wilkinson eingeführt; die Idee geht jedoch zurück auf Cornelius Lanczos.

Beispiel 3.9. Wir betrachten die Addition von drei Gleitkommazahlen x_1, x_2, x_3. Da für die Gleitkommaaddition die Reihenfolge eine Rolle spielt, legen wir diese als $(x_1 + x_2) + x_3$ fest. Wir haben also

$$\begin{aligned}(x_1 \tilde{+} x_2) \tilde{+} x_3 &= ((x_1 + x_2)(1 + \delta_2) + x_3)(1 + \delta_3) \\ &= x_1(1 + \delta_2 + \delta_3 + \delta_2\delta_3) + x_2(1 + \delta_2 + \delta_3 + \delta_2\delta_3) + x_3(1 + \delta_3) \\ &=: \hat{x}_1 + \hat{x}_2 + \hat{x}_3\end{aligned}$$

für je ein $\delta_2, \delta_3 < \mathrm{eps}$. Nun ist für $\mathrm{eps} < 1$:

$$\begin{aligned}\frac{|\hat{x}_1 - x_1|}{|x_1|} &= \delta_2 + \delta_3 + \delta_2\delta_3 < 3\mathrm{eps}, \\ \frac{|\hat{x}_2 - x_2|}{|x_2|} &= \delta_2 + \delta_3 + \delta_2\delta_3 < 3\mathrm{eps}, \\ \frac{|\hat{x}_3 - x_3|}{|x_3|} &= \delta_3 < \mathrm{eps}.\end{aligned}$$

Setzen wir $x := (x_1, x_2, x_3) \in \mathbb{R}^3$ und analog für $\hat{x}$, so folgt z. B.

$$\begin{aligned}\frac{\|\hat{x} - x\|_1}{\|x\|_1} &= \frac{|\hat{x}_1 - x_1|}{\|x\|_1} + \frac{|\hat{x}_2 - x_2|}{\|x\|_1} + \frac{|\hat{x}_3 - x_3|}{\|x\|_1} \\ &\leqslant \frac{|\hat{x}_1 - x_1|}{|x_1|} + \frac{|\hat{x}_2 - x_2|}{|x_2|} + \frac{|\hat{x}_3 - x_3|}{|x_3|} \\ &< 7\mathrm{eps}.\end{aligned}$$

Wegen $\delta \geqslant \varepsilon$ ist die Gleitkommaaddition also auch rückwärtsstabil. Man sieht außerdem, dass die zuerst addierten Terme eine größere Fehlerverstärkung erfahren. Es ist also numerisch vorteilhaft, bei der Addition mehrerer Terme mit dem betragskleinsten zu beginnen.

WEITERFÜHRENDE LITERATUR

N. J. Higham (2002). *Accuracy and stability of numerical algorithms*. 2. Aufl. Society for Industrial und Applied Mathematics (SIAM), Philadelphia, PA.

Teil II

NUMERISCHE LINEARE ALGEBRA

GRUNDLAGEN

4

In diesem Kapitel werden wichtige Begriffe und Werkzeuge behandelt, die speziell für die numerische Lösung von Problemen aus der Linearen Algebra eine Rolle spielen. Dabei werden im Wesentlichen die allgemeinen Betrachtungen des ersten Teils konkretisiert.

4.1 MATRIXNORMEN

Jede Matrix $A \in \mathbb{R}^{n\times n}$ lässt sich als Vektor im Raum $\mathbb{R}^{(n^2)}$ auffassen, womit sich jede der bereits bekannten Vektornormen zur Untersuchung einer Matrix verwenden ließe. Jedoch haben Matrizen ein zweites Gesicht: Sie sind auch lineare Operatoren, die auf Vektoren wirken – und diese Sichtweise wird für uns weitaus relevanter sein. Wir verlangen von einer Matrixnorm daher über die Normaxiome in Definition 1.1 hinaus zusätzliche Eigenschaften.

Definition 4.1. Eine Matrixnorm $\|\cdot\|$ auf $\mathbb{R}^{n\times n}$ heißt *submultiplikativ*, wenn für alle $A, B \in \mathbb{R}^{n\times n}$ gilt

$$\|AB\| \leqslant \|A\|\|B\|.$$

Definition 4.2. Eine Matrixnorm $\|\cdot\|_M$ auf $\mathbb{R}^{n\times n}$ heißt *verträglich* mit einer Vektornorm $\|\cdot\|_V$, wenn für alle $A \in \mathbb{R}^{n\times n}$ und $x \in \mathbb{R}^n$ gilt

$$\|Ax\|_V \leqslant \|A\|_M \|x\|_V.$$

Die wichtigsten Beispiele für Matrixnormen sind die *Operatornormen*.

Definition 4.3. Für eine Vektornorm $\|\cdot\|_V$ auf $\mathbb{R}^n$ heißt die durch

$$\|A\|_M := \sup_{\|x\|\neq 0} \frac{\|Ax\|_V}{\|x\|_V}$$

definierte Norm die *natürliche* oder die (durch $\|\cdot\|_V$) *induzierte* Norm.

Satz 4.4. *Jede natürliche Norm ist submultiplikativ und mit der sie induzierenden Vektornorm verträglich.*

Die am häufigsten gebrauchten Normen sind auch hier die p-Normen, wobei die Euklidische Norm wieder hervorgehoben ist.

Definition 4.5. Der *Spektralradius* einer Matrix $A \in \mathbb{R}^{n\times n}$ ist definiert als

$$\rho(A) = \max_{1\leqslant i\leqslant n} |\lambda_i|,$$

wobei λ_i die Eigenwerte von A sind.

Beispiel 4.6 (Induzierte p-Normen auf $\mathbb{R}^{n\times n}$)**.**

(i) $\|A\|_1 = \max_{1\leqslant j\leqslant n} \sum_{i=1}^{n} |a_{ij}|$ (*Spaltensummennorm*)

(ii) $\|A\|_\infty = \max_{1\leqslant i\leqslant n} \sum_{j=1}^{n} |a_{ij}|$ (*Zeilensummennorm*)

(iii) $\|A\|_2 = \sqrt{\rho(A^T A)}$ (*Spektralnorm*).

Nicht alle Matrixnormen sind induzierte Normen. Auch die folgende Norm, genannt *Frobeniusnorm*, ist submultiplikativ und verträglich mit der Euklidischen Norm:[1]

$$\|A\|_F = \sqrt{\sum_{i,j=1}^{n} |a_{ij}|^2}.$$

Der Wert der Frobeniusnorm besteht darin, dass sie eine scharfe obere Schranke für die – schwer zu berechnende – Spektralnorm darstellt. Wie Vektornormen sind auch alle Matrixnormen äquivalent; insbesondere gelten die folgenden (optimalen) Abschätzungen:

Lemma 4.7. *Für $A \in \mathbb{R}^{n\times n}$ gilt*

(i) $\|A\|_2 \leqslant \|A\|_F \leqslant \sqrt{n}\|A\|_2$,

(ii) $\frac{1}{\sqrt{n}}\|A\|_\infty \leqslant \|A\|_2 \leqslant \sqrt{n}\|A\|_\infty$,

(iii) $\frac{1}{\sqrt{n}}\|A\|_1 \leqslant \|A\|_2 \leqslant \sqrt{n}\|A\|_1$.

Lemma 4.8. *Sei $A \in \mathbb{R}^{n\times n}$ und $\|\cdot\|$ eine induzierte Norm. Dann gilt:*

$$\rho(A) \leqslant \|A\|$$

Beweis. Sei λ der betragsgrößte Eigenwert von A und v der zugehörige Eigenvektor. Dann gilt mit der induzierenden Vektornorm $\|\cdot\|_V$

$$|\lambda|\|v\|_V = \|\lambda v\|_V = \|Av\|_V \leqslant \|A\|\|v\|_V.$$

Da Eigenvektoren nach Definition ungleich Null sind, können wir durch $\|x\|_V$ dividieren und erhalten die Aussage. □

[1] Sie entspricht der Euklidischen Norm von $A \in \mathbb{R}^{n\times n}$, aufgefasst als Vektor in $\mathbb{R}^{n^2}$.

4.2 SPEZIELLE MATRIZEN

Wir werden im Laufe der nächsten Kapitel erkennen, dass die Stabilität und Effizienz von Verfahren zur Lösung von Problemen der Linearen Algebra wesentlich von der Struktur der gegebenen Matrix abhängt. Hier betrachten wir zwei der wichtigsten Beispiele von Matrizen mit günstiger Struktur.

Definition 4.9. Eine Matrix $A \in \mathbb{R}^{n\times n}$ heißt *positiv definit*, falls gilt[2]

$$x^T Ax > 0 \qquad \text{für alle } x \in \mathbb{R}^n \setminus \{0\}.$$

Durch Einsetzen geeigneter Vektoren x zeigt man folgende Eigenschaften positiv definiter Matrizen.

Satz 4.10. *Sei* $A = (a_{ij})_{i,j=1}^n \in \mathbb{R}^{n\times n}$ *positiv definit. Dann gilt:*

(i) $a_{ii} > 0$ *für alle* $1 \leqslant i \leqslant n$*;*

(ii) $a_{ij}^2 < a_{ii}a_{jj}$*, für alle* $i \neq j$*,* $1 \leqslant i, j \leqslant n$*;*

(iii) es existiert ein $k \in \{1, \dots, n\}$*, so dass* $\max_{1\leqslant i,j\leqslant n} |a_{ij}| = a_{kk}$.

Eine einfache Charakterisierung von positiver Definitheit gelingt für symmetrische Matrizen.

Satz 4.11. *Sei* $A \in \mathbb{R}^{n\times n}$ *symmetrisch. Dann ist* A *positiv definit genau dann, wenn* $\lambda_i > 0$ *gilt für alle Eigenwerte* λ_i*,* $1 \leqslant i \leqslant n$*, von* A.

Für eine wichtige Unterklasse lässt sich die Definitheit direkt an den Matrixelementen erkennen.

Definition 4.12. $A \in \mathbb{R}^{n\times n}$ heißt *streng diagonal dominant*, falls gilt

$$|a_{ii}| > \sum_{\substack{j=1\\ j\neq i}}^{n} |a_{ij}| \qquad \text{für alle } 1 \leqslant i \leqslant n.$$

Satz 4.13. *Sei* $A \in \mathbb{R}^{n\times n}$ *symmetrisch und streng diagonal dominant. Dann ist* A *positiv definit.*

[2] Ist auch Gleichheit möglich, so spricht man von semidefiniten Matrizen; die umgekehrte Ungleichung charakterisiert negativ definite Matrizen. Ist keine dieser Aussage möglich, nennt man die Matrix indefinit.

4.3 DIE KONDITIONSZAHL EINER MATRIX

Der zentrale Begriff für die Untersuchung der Kondition von Problemen der linearen Algebra ist die Konditionszahl einer Matrix. Wir werden diese für die Lösung linearer Gleichungssysteme motivieren; sie ist aber auch für viele andere Problemstellungen, die wir im Weiteren betrachten werden, relevant.

Sei also $A \in \mathbb{R}^{n\times n}$ und $b \in \mathbb{R}^n$ gegeben, und derjenige Vektor $x \in \mathbb{R}^n$ gesucht, der $Ax = b$ erfüllt. Existiert überhaupt solch ein x, so lässt es sich über die inverse Matrix A^{-1} berechnen; es ist $x = A^{-1}b$. Wie in Kapitel 3.1 schreiben wir dies als $f : b \mapsto A^{-1}b$. Dies ist eine lineare Funktion, es gilt also $f'(b) = A^{-1}$. Nach Satz 3.3 ist deshalb $\kappa_{abs} = \|A^{-1}\|$ und

$$\kappa_{rel} = \frac{\|b\|}{\|A^{-1}b\|}\|A^{-1}\| = \frac{\|Ax\|}{\|x\|}\|A^{-1}\| \leqslant \|A\|\|A^{-1}\|.$$

Dies motiviert die folgende Definition:

Definition 4.14. Für eine reguläre (d. h. invertierbare) Matrix $A \in \mathbb{R}^{n\times n}$ und eine Matrixnorm $\|\cdot\|$ heißt

$$\kappa(A) := \|A\|\|A^{-1}\|$$

(relative) *Konditionszahl* von A bezüglich $\|\cdot\|$.

Dabei werden wir (falls nicht anders angegeben) im Weiteren annehmen, dass wir für die Kondition eine induzierte Matrixnorm zugrunde legen. Speziell für die Spektralnorm erhält man eine nützliche Darstellung:

Lemma 4.15. *Die Konditionszahl einer regulären Matrix $A \in \mathbb{R}^{n\times n}$ bezüglich einer induzierten Matrixnorm $\|\cdot\|_p$ ist*

$$\kappa_p(A) = \frac{\max\|Ax\|_p}{\min\|Ax\|_p},$$

wobei das Maximum und Minimum über alle $x \in \mathbb{R}^n$ mit $\|x\|_p = 1$ genommen wird.

Ist A symmetrisch und positiv definit, so ist

$$\kappa_2(A) = \frac{\lambda_{max}}{\lambda_{min}},$$

d. h. gleich dem Verhältnis zwischen betragsgrößtem und betragskleinstem Eigenwert von A.

Wir haben bislang noch nicht berücksichtigt, dass in der Regel auch die Matrixeinträge von A, etwa durch Darstellung als Maschinenzahlen, mit Fehlern behaftet sind. Wir müssen also untersuchen, wie sich die Lösung x verhält, wenn wir statt $Ax = b$ das System $\tilde{A}x :=$

$(A + \delta A)x = b$ lösen. Der folgende Satz[3] besagt insbesondere, dass eine kleine Störung die Invertierbarkeit einer Matrix nicht ändert.

Satz 4.16. *Seien* $A, \tilde{A} \in \mathbb{R}^{n\times n}$ *und* $b \in \mathbb{R}^n$. *Sei weiterhin* A *invertierbar und* $x \in \mathbb{R}^n$ *Lösung von* $Ax = b$.

Gilt $\|A^{-1}\|\|\tilde{A} - A\| < 1$, *so ist auch* $\tilde{A}$ *invertierbar, und für die Lösung* $\tilde{x} \in \mathbb{R}^n$ *von* $\tilde{A}\tilde{x} = b$ *gilt für eine beliebige Vektornorm und der durch sie induzierten Matrixnorm:*

$$\frac{\|\tilde{x} - x\|}{\|x\|} \leqslant \frac{\kappa(A)}{1 - \kappa(A)\frac{\|\tilde{A}-A\|}{\|A\|}} \frac{\|\tilde{A} - A\|}{\|A\|}$$

Die Konditionszahl bestimmt also die Kondition der Lösung von linearen Gleichungssystemen, sowohl gegenüber Störungen der rechten Seite als auch der Matrix selber. Eine Möglichkeit, die Konditionszahl einer Matrix zu verbessern, liegt in der Skalierung der einzelnen Zeilen. Zwar ist keine allgemeine Methode bekannt, um dies für beliebige Normen zu erreichen, aber für die Maximumsnorm können wir eine solche angeben:

Sei $D \in \mathbb{R}^{n\times n}$ eine Diagonalmatrix mit Diagonaleinträgen

$$d_i = \left(\sum_{j=1}^{n} |a_{ij}|\right)^{-1}.$$

Die *zeilenäquilibrierte* Matrix DA hat dann die Maximumsnorm

$$\|DA\|_\infty = \max_{1\leqslant i\leqslant n} \sum_{j=1}^{n} \left(\sum_{k=1}^{n} |a_{ik}|\right)^{-1} |a_{ij}| = \max_{1\leqslant i\leqslant n} \left(\sum_{j=1}^{n} |a_{ij}|\right)^{-1} \sum_{j=1}^{n} |a_{ij}| = 1,$$

und man kann zeigen, dass $\kappa_\infty(DA) < \kappa_\infty(\tilde{D}A)$ für jede andere Diagonalmatrix $\tilde{D}$ (inklusive der Identitätsmatrix) gilt. Analog lässt sich (durch Multiplikation von rechts mit einer ähnlichen Diagonalmatrix) eine *Spaltenäquilibrierung* durchführen, die die Konditionszahl bezüglich der Spaltensummennorm minimiert.

[3] siehe z. B. G. Hämmerlin und K.-H. Hoffmann (1994). *Numerische Mathematik*. 4. Aufl. Springer-Verlag, Berlin, Seite 79–80

LÖSUNG LINEARER GLEICHUNGSSYSTEME

5

Das Lösen linearer Gleichungssysteme gehört zu den wichtigsten Aufgaben der numerischen Mathematik. Zum einen sind viele in der Anwendung erfolgreiche mathematische Modelle linearer Natur (etwa das Leontief-Modell in den Wirtschaftswissenschaften, die Kirchhoffschen Gesetze in der Elektrotechnik oder Bilanzgleichungen in der Chemie). Zum anderen basieren fast alle numerischen Verfahren der höheren Mathematik (etwa die Lösung partieller Differentialgleichungen) letztlich auf der Reduktion auf lineare Gleichungssysteme. Es ist also von zentralem Interesse, stabile und effiziente Verfahren für dieses Problem zu haben.

Wir beschäftigen uns daher in diesem Kapitel mit folgendem Problem:

- Gegeben:

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ a_{n1} & \dots & \dots & a_{nn} \end{pmatrix} \in \mathbb{R}^{n\times n}, \quad b = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} \in \mathbb{R}^n,$$

- Gesucht: $x \in \mathbb{R}^n$, so dass $Ax = b$.

Aus der linearen Algebra wissen wir, wann solch ein Gleichungssystem lösbar ist.

Satz 5.1. *Sei* $A \in \mathbb{R}^{n\times n}$ *und* $b \in \mathbb{R}^n$. *Ist* $\det(A) \neq 0$, *so existiert genau ein* $x \in \mathbb{R}^n$ *mit* $Ax = b$.

Über die Determinante lässt sich theoretisch auch die Lösung des Gleichungssystems berechnen; die Cramersche Regel besagt, dass $x_j = \frac{\det(A_j)}{\det(A)}$ ist, wobei A_j die Matrix ist, die durch Ersetzen der j-ten Spalte von A durch b entsteht. Dies ist allerdings ein denkbar unpraktikables Verfahren: Die notwendige Berechnung der $n+1$ Determinanten erfordert $\mathcal{O}((n+1)!)$ Operationen; selbst moderne Graphikprozessoren, die 1 Teraflop – also eine Billion Gleitkommaoperationen pro Sekunde – leisten können, würden für die Lösung eines Gleichungssystems mit $n = 20$ Zeilen 1.6 Jahre benötigen; für $n = 22$ wären bereits 820 Jahre nötig; und $n = 30$ wäre in 266 Billionen Jahren noch nicht fertig berechnet.

Die in der Praxis auftretenden Matrizen haben aber üblicherweise Größen von mehreren Hundert bis zu mehreren Millionen Zeilen bzw. Spalten. Die in Frage kommenden Verfahren müssen also auch solche Probleme behandeln können. Die in diesem Kapitel behandelten Methoden fallen in zwei Klassen:

1. *Direkte Verfahren* liefern nach endlich vielen Schritten die Lösung (wobei evtl. Rundungsfehler zu berücksichtigen sind).
2. *Iterative Verfahren* liefern ausgehend von einer Anfangsnäherung nach jedem Schritt eine verbesserte Näherung. Zu den Rundungsfehlern treten hier also noch Verfahrensfehler; dafür weisen diese Methoden eine günstigere Komplexität auf.

5.1 DIREKTE VERFAHREN

Zunächst betrachten wir eine Klasse von Matrizen, für die das zugehörige Gleichungssystem sehr leicht lösbar ist. Im nächsten Schritt führen wir dann den allgemeinen Fall auf dieses Problem zurück.

5.1.1 AUFLÖSUNG VON DREIECKSMATRIZEN

IDEE: Haben wir eine Matrix der Form $A = \begin{pmatrix} a_{11} & a_{12} \\ 0 & a_{22} \end{pmatrix}$ gegeben, so können wir die Lösung von $Ax = b$ einfach durch sukzessives Einsetzen bestimmen: Zuerst finden wir $x_2 = b_2/a_{22}$, und setzen $x_1 = (b_1 - a_{12}x_2)/a_{11}$. Für den allgemeinen Fall $A \in \mathbb{R}^{n\times n}$ führt das auf die folgende Definition.

Definition 5.2. Eine Matrix $A \in \mathbb{R}^{n\times n}$ heißt

- *obere Dreiecksmatrix*, falls $a_{ij} = 0$ gilt für alle $i > j$,
- *untere Dreiecksmatrix*, falls $a_{ij} = 0$ gilt für alle $i < j$.

Ist A eine obere Dreiecksmatrix, so kann man das Gleichungssystem $Ax = b$ durch *Rückwärtssubstitution* bestimmen.

Algorithmus 5.1 : Rückwärtssubstitution

Input : a_{ij}, b_j

1 **for** $j = n, \ldots, 1$ **do**

2 $\quad x_j \leftarrow \left(b_j - \sum_{k=j+1}^{n} a_{jk}x_k\right)/a_{jj}$

Output : x_j

Dabei müssen $\frac{1}{2}n(n-1)$ Additionen und $\frac{1}{2}n(n+1)$ Multiplikationen durchgeführt werden, die Komplexität ist also $\mathcal{O}(n^2)$. Sind alle Diagonalelemente $a_{ii} \neq 0$, so ist der Algorithmus durchführbar; dies ist (wegen der Dreiecksform) gleichbedeutend mit $\det(A) \neq 0$.

Analog führt man für eine untere Dreiecksmatrix eine *Vorwärtssubstitution* durch.

Algorithmus 5.2 : Vorwärtssubstitution

Input : a_{ij}, b_j

1 **for** $j = 1, \ldots, n$ **do**

2 $\quad x_j \leftarrow \left(b_j - \sum_{k=1}^{j-1} a_{jk} x_k\right) / a_{jj}$

Output : x_j

5.1.2 GAUSS-ELIMINATION

IDEE: Eine allgemeine Matrix wird durch Zeilenumformungen auf obere Dreiecksform gebracht, so dass sich Rückwärtssubstitution zur Lösung anwenden lässt.[1]

Zur Herleitung des Verfahrens nehmen wir zunächst an, dass $a_{11} \neq 0$ gilt. Dann können wir geeignete Vielfache der ersten Zeile der Matrix A und der rechten Seite b von allen weiteren subtrahieren:

1 **for** $i = 2, \ldots, n$ **do**

2 $\quad l_i^{(1)} = \frac{a_{i1}}{a_{11}}$

3 $\quad$ **for** $j = 1, \ldots, n$ **do**

4 $\quad\quad a_{ij}^{(2)} = a_{ij} - l_i^{(1)} a_{1j}$

5 $\quad b_i^{(2)} = b_i - l_i^{(1)} b_1$

und wir erhalten eine neue Matrix $A^{(2)}$ und rechte Seite $b^{(2)}$:

$$A^{(2)} = \begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ 0 & a_{22}^{(2)} & \ldots & a_{2n}^{(2)} \\ \vdots & \vdots & & \vdots \\ 0 & a_{n2}^{(2)} & \ldots & a_{nn}^{(2)} \end{pmatrix}, \qquad b^{(2)} = \begin{pmatrix} b_1 \\ b_2^{(2)} \\ \vdots \\ b_n^{(2)} \end{pmatrix}$$

Ist nun $a_{22}^{(2)} \neq 0$, so wiederholen wir diesen Schritt mit der Teilmatrix $\bar{A}^{(2)} = (a_{ij}^{(2)})_{i,j=2}^{n}$ und dem Vektor $\bar{b}^{(2)} = (b_i^{(2)})_{i=2}^{n}$. Da wir in jedem Schritt k nur die Matrixelemente mit $i, j \geqslant k$ betrachten, erhalten wir also garantiert nach $n-1$ Schritten eine obere Dreiecksmatrix

[1] Carl Friedrich Gauß hat dieses Verfahren 1810 in einer Abhandlung zur Untersuchung des Orbits des Planetoiden Pallas verwendet (zur Lösung eines Gleichungssystems für sechs Unbekannte, das er aus einer Reihe von Beobachtungen ableitete). Das allgemeine Verfahren ist allerdings bereits 200 v.Chr. in dem chinesischen Text *Jiŭzhāng Suànshù* ("Neun Kapitel über die mathematische Kunst", dem chinesischen Äquivalent von Euklids *Elementen*) unter dem Begriff *fāngchéng shù* ("Vorschriften über quadratische Felder") beschrieben.

$A^{(n)}$, die wir per Rückwärtssubstitution auflösen können. Dabei sind im Schritt k insgesamt $(n-k)((n-k+1)+1)$ Multiplikationen und Subtraktionen sowie $(n-k)$ Divisionen nötig; der komplette Aufwand für die Gauß-Elimination beträgt also $\mathcal{O}(n^3)$. Der Algorithmus scheitert, wenn wir in einem Schritt durch Null dividieren müssen. Für streng diagonal dominante Matrizen können wir das im ersten Schritt ausschliessen; per Induktion zeigt man, dass diese Eigenschaft auch für alle modifizierten Matrizen $A^{(k)}$ erhalten bleibt. Wir halten fest:

Satz 5.3. *Für eine streng diagonal dominante Matrix $A \in \mathbb{R}^{n\times n}$ ist die Gauß-Elimination durchführbar.*

Ist A nicht diagonal dominant, können selbst für invertierbare Matrizen[2] Nullelemente auf der Diagonalen auftreten. Dies kann durch einen Zeilentausch (entspricht Umsortieren der Gleichungen) umgangen werden: Taucht im Schritt k ein Diagonalelement $a_{kk}^{(k)} = 0$ auf, so sucht man ein $a_{mk}^{(k)} \neq 0$ mit $m > k$ und vertauscht die Zeilen m und k. Dieses Verfahren heißt *Pivotsuche*, $a_{mk}^{(k)}$ *Pivotelement*. Solches Vorgehen ist auch sinnvoll, wenn $a_{kk}^{(k)}$ sehr klein ist, um Fehlerverstärkung bei der Berechnung von $l^{(k)}$ zu vermeiden. Daher wählt man sinnvollerweise als Pivotelement das betragsgrößte $a_{mk}^{(k)}$.[3]

Müssen wir mehrere Gleichungssysteme lösen, bei denen sich nur die rechte Seite ändert, sollten die $\mathcal{O}(n^3)$ Operationen nur einmal durchgeführt werden. Dies lässt sich erreichen, indem man die nötigen Zeilenumformungen in der Form von Matrixmultiplikationen speichert.

5.1.3 LR-ZERLEGUNG

Wir betrachten wieder zuerst den Fall ohne Pivotisierung. Im Schritt k der Gauß-Elimination wird von allen Zeilen mit Index $i > k$ ein Vielfaches der k-ten Zeile abgezogen, was sich als Multiplikation mit einer geeigneten Matrix darstellen lässt. Durch explizites Ausschreiben der Multiplikationen vergewissert man sich, dass gilt

$$A^{(k+1)} = L_k A^{(k)}, \qquad b^{(k+1)} = L_k b^{(k)},$$

mit $A^{(1)} = A$ und

$$L_k = \begin{pmatrix} 1 & 0 & \dots & \dots & & 0 \\ 0 & \ddots & & & & 0 \\ \vdots & & 1 & & & \vdots \\ \vdots & & -l_{k+1}^{(k)} & \ddots & & \vdots \\ \vdots & & \vdots & & \ddots & 0 \\ 0 & \dots & -l_n^{(k)} & \dots & \dots & 1 \end{pmatrix} = I - \underbrace{(0,\dots,l_{k+1}^{(k)},\dots,l_n^{(k)})^T}_{=:l^{(k)}} \underbrace{(0,\dots,0,1,0,\dots,0)}_{=:e_k^T},$$

[2] z. B. $\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$

[3] Merke: Die Pivotsuche verbessert nur die Stabilität des *Verfahrens*, eine Zeilenäquilibrierung dagegen nur die Kondition des *Problems*. Man wird also beide Techniken anwenden.

wobei $l_i^{(k)} = \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}$ genau die in der Gauß-Elimination berechneten Faktoren sind.

Solche Matrizen, die sich nur in einer Spalte von der Einheitsmatrix unterscheiden, heißen *Frobeniusmatrizen*. Wir haben also nach $n-1$ Schritten eine obere Dreiecksmatrix

$$A^{(n)} = L_{n-1} \cdots L_1 A.$$

Durch Nachrechnen sieht man die folgenden Eigenschaften.

Lemma 5.4. *Es sei* $L_k = I - l^{(k)} e_k^T$ *eine Frobeniusmatrix. Dann gilt:*

(i) $L_k^{-1} = I + l^{(k)} e_k^T$,

(ii) $L := \prod_{k=1}^{n-1} L_k^{-1} = I + \sum_{k=1}^{n-1} l^{(k)} e_k^T$.

Damit ist L also eine untere Dreiecksmatrix mit nicht verschwindenden Diagonaleinträgen ($= 1$), und wir können A als Produkt von unteren und oberen Dreiecksmatrizen schreiben

$$A = LR \quad \text{mit} \quad R := A^{(n)}, \quad L := (l_{ij})_{i,j=1}^n, \quad l_{ij} := \begin{cases} \frac{a_{ij}^{(j)}}{a_{jj}^{(j)}} & i < j, \\ 1 & i = j, \\ 0 & i > j. \end{cases}$$

Diese Faktorisierung nennt man *LR-Zerlegung*; sie hat offensichtlich die gleiche Komplexität $\mathcal{O}(n^3)$ wie die Gauß-Elimination. Wollen wir jetzt die Gleichung $Ax = b$ lösen, so lässt sich das schrittweise durchführen:

1. Berechne L, R, so dass $A = LR$ gilt,
2. Löse $Ly = b$ durch Vorwärtssubstitution,
3. Löse $Rx = y$ durch Rückwärtssubstitution.

Dabei ist der erste (und aufwendigste) Schritt unabhängig von b, was eine Ersparnis ermöglicht, falls man ein Gleichungssystem für mehrere rechte Seiten lösen möchte.

Nun soll auch die Pivotisierung berücksichtigt werden. Dazu müssen wir die Vertauschung von zwei Zeilen k und m auch als Matrixmultiplikation schreiben. Es gilt $\tilde{A} = P_{km} A$ für

$$P_{km} = \begin{pmatrix} 1 & 0 & \dots & & & & & & 0 \\ 0 & \ddots & & & & & & & \\ \vdots & & 1 & & & & & & \\ & & & 0 & \dots & \dots & 1 & & \\ & & & \vdots & \ddots & & \vdots & & \\ & & & \vdots & & 1 & \vdots & & \\ & & & \vdots & & & \ddots & \vdots & \\ & & & 1 & \dots & \dots & 0 & & \\ & & & & & & & 1 & \vdots \\ & & & & & & & \ddots & 0 \\ 0 & & & & & & & \dots & 0 & 1 \end{pmatrix} \begin{matrix} \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \end{matrix} \begin{matrix} \\ \\ \\ \leftarrow \text{k-te Zeile} \\ \\ \\ \\ \leftarrow \text{m-te Zeile} \\ \\ \\ \end{matrix}.$$

Wird also im Schritt k die notwendige Zeilenvertauschung durch die *Permutationsmatrix* $P_k := P_{km}$ für ein $m \geqslant k$ (d. h. $P_k = I$ für k_m) durchgeführt, so ist $A^{(k+1)} = L_k P_k A^{(k)}$, und schließlich

$$A^{(n)} = L_{n-1} P_{n-1} \cdots L_1 P_1 A.$$

Wir zeigen jetzt, dass wir dies wieder als LR-Zerlegung auffassen können. Dazu verwenden wir, dass

(i) $P_k^{-1} = P_k$ für alle k gilt, und

(ii) P_k nur Zeilen mit Index $m \geqslant k$ vertauscht.

Damit können wir die P_j "an den L vorbeiziehen": Man erhält für alle $m < k$

$$P_k L_m P_k = P_k I P_k - P_k (l^{(m)} e_m^T) P_k = I - (P_k l^{(m)})(P_k^T e_m)^T = I - \tilde{l}^{(m)} e_m^T =: \tilde{L}_m, \tag{5.1}$$

und somit $P_k L_m = \tilde{L}_m P_k$ für $m < k$. Also ist

$$A^{(n)} = L_{n-1} P_{n-1} \cdots L_2 P_2 L_1 P_1 A = L_{n-1} P_{n-1} \cdots L_2 \tilde{L}_1 P_2 P_1 A, \quad \tilde{L}_1 := P_2 L_1. \tag{5.2}$$

Dies lässt sich wiederholen für $L_2, \ldots, L_{n-1}$. Multipliziert man wieder beide Seiten mit den Inversen der $\tilde{L}_k$, $k = 1, \ldots, n-1$, so erhält man:

$$\tilde{L} A^{(n)} := \tilde{L}_1^{-1} \cdots \tilde{L}_{n-1}^{-1} A^{(n)} = P_{n-1} \cdots P_1 A =: PA$$

Da $\tilde{L}$ nach (5.1) und Lemma 5.4 wieder eine Frobeniusmatrix ist, haben wir bewiesen:

Satz 5.5. *Für jede reguläre Matrix $A \in \mathbb{R}^{n \times n}$ existiert eine Permutationsmatrix P, eine untere Dreiecksmatrix L und eine obere Dreiecksmatrix R, so dass gilt*

$$PA = LR.$$

Die Stabilität der LR-Zerlegung lässt sich mit Hilfe der Rückwärtsanalyse und Satz 4.16 beweisen.

Der folgende Algorithmus berechnet die LR-Zerlegung einer Matrix A, die dabei überschrieben wird. Dadurch wird gewährleistet, dass die Permutationen auch auf die Matrixelemente von L angewendet werden (siehe (5.2)). Die Permutation P speichert man am einfachsten dadurch, dass jede Zeilenvertauschung auch auf einen Permutationsvektor $p = (1, \ldots, n)^T$ angewendet wird; nach Ende des Algorithmus geben die Einträge von p die Reihenfolge der Einträge von Pb an (z. B. ist für $p = (3, 2, 1)$ dann $Pb = (b_3, b_2, b_1)$).

Algorithmus 5.3 : LR-Zerlegung mit Pivotisierung

Input : a_{ij}, p_j

```
p ← (1, ..., n)                                   // Initialisiere Permutationsvektor
for j = 1, ..., n − 1 do
    Finde s ⩾ j so dass |a_sj| = max_{j⩽i⩽n} |a_ij|          // Pivotsuche
    Vertausche a_sk und a_jk für alle 1 ⩽ k ⩽ n              // Pivotisierung
    Vertausche p_s und p_j                          // Aktualisiere Permutationsvektor
    for i = j + 1, ..., n do
        a_ij ← a_ij / a_jj                  // Matrix L ( Diagonalelemente sind immer 1)
        for k = j + 1, ..., n do
            a_ik ← a_ik − a_ij a_jk                                  // Matrix R
```

Output : $l_{ij} = a_{ij}$ $(i > j)$, $r_{ij} = a_{ij}$ $(i \leqslant j)$, p_j

Zur Lösung von $Ax = b$ geht man nun wie folgt vor:

1. Berechne LR-Zerlegung mit Pivotsuche: $PA = LR$
2. Löse $Ly = Pb$ mit Vorwärtssubstitution
3. Löse $Rx = y$ mit Rückwärtssubstitution

5.1.4 CHOLESKY-ZERLEGUNG

Für eine symmetrisch positiv definite Matrix A kann die LR-Zerlegung vereinfacht werden. Natürlich ließe sich L mit Hilfe der LR-Zerlegung berechnen, aber man kann durch Ausnutzen der Symmetrie ein günstigeres Verfahren finden. Die Beziehung $A = LL^T$, betrachtet für die einzelnen Matrixeinträge a_{ij}, liefert $\frac{1}{2}n(n+1)$ unabhängige Gleichungen für die $\frac{1}{2}n(n+1)$ Unbekannten l_{ij}, $i > j$. Durch eine geeignete Anordnung kann man diese Gleichungen durch einfaches Einsetzen auflösen. Dies sieht man am einfachsten für $n = 2$, d. h.

$$A = \begin{pmatrix} a_{11} & a_{21} \\ a_{21} & a_{22} \end{pmatrix}, \qquad L = \begin{pmatrix} l_{11} & 0 \\ l_{21} & l_{22} \end{pmatrix}, \qquad L^T = \begin{pmatrix} l_{11} & l_{21} \\ 0 & l_{22} \end{pmatrix}.$$

Ausmultiplizieren von $A = LL^T$ liefert dann die Gleichungen

$$a_{11} = (l_{11})^2, \qquad a_{21} = l_{11}l_{21}, \qquad a_{22} = (l_{21})^2 + (l_{22})^2.$$

Da A symmetrisch positiv definit ist, sind die Diagonaleinträge strikt positiv. Wir können also durch spaltenweises Vorgehen sämtliche Unbekannten in stabiler Weise der Reihe nach berechnen; eine Pivotsuche ist nicht notwendig. Es bleibt zu zeigen, dass das Verfahren allgemein für symmetrisch positiv definite Matrizen durchführbar ist.

Satz 5.6. *Jede symmetrische positiv definite Matrix A hat eine Zerlegung der Form $A = LL^T$, wobei L eine reguläre untere Dreiecksmatrix ist.*

Beweis. Wir beweisen dies durch Induktion nach k, indem wir – analog zur LR-Zerlegung – sukzessive Matrixmultiplikationen betrachten.

- $k = 1$: Wir definieren

$$L^{(1)} := \begin{pmatrix} \sqrt{a_{11}} & 0 \\ \frac{1}{\sqrt{a_{11}}} b_1 & I_{n-1} \end{pmatrix}, \quad A^{(1)} := \begin{pmatrix} 1 & 0 \\ 0 & B^{(1)} \end{pmatrix},$$

  wobei $b_1 = (a_{21}, \ldots, a_{n1})^T$, I_{n-1} die $(n-1) \times (n-1)$—Einheitsmatrix ist, und $B^{(1)} = (a_{ij})_{i,j=2}^{n}$ die restlichen Einträge von A enthält. Da A symmetrisch und positiv definit ist, ist $a_{11} > 0$, und deshalb $L^{(1)}$ wohldefiniert. Man rechnet nun leicht nach, dass gilt

$$A = L^{(1)} A^{(1)} (L^{(1)})^T.$$

  Ausserdem ist $a_{22}^{(1)} = a_{22} > 0$.

- $k - 1 \to k$ für $k \geqslant 2$: Wir nehmen an, dass $A^{(k-1)}$ in der Form

$$A^{(k-1)} := \begin{pmatrix} I_{k-1} & 0 & 0 \\ 0 & a_{kk}^{(k-1)} & b_{k-1}^T \\ 0 & b_{k-1} & B^{(k-1)} \end{pmatrix}$$

  vorliegt (mit b_k, $B^{(k)}$ analog zu $k = 1$), wobei $a_{kk}^{(k-1)} > 0$ gilt (Induktionsannahme). Wieder rechnet man nach, dass

$$A^{(k-1)} = L^{(k)} A^{(k)} {L^{(k)}}^T$$

  gilt, wenn wir

$$L^{(k)} := \begin{pmatrix} I_{k-1} & 0 & 0 \\ 0 & \sqrt{a_{kk}^{(k-1)}} & 0 \\ 0 & \frac{1}{\sqrt{a_{kk}^{(k-1)}}} b_{k-1} & I_{n-k} \end{pmatrix}, \quad A^{(k)} := \begin{pmatrix} I_k & 0 \\ 0 & B^{(k-1)} - \frac{1}{a_{kk}^{(k-1)}} b_{k-1} b_{k-1}^T \end{pmatrix},$$

  setzen (mit b_k, $B^{(k)}$ analog zu $k = 1$). Wir müssen nun nachweisen, dass $a_{k+1,k+1}^{(k)} > 0$ gilt. Da A (und damit $A^{(k-1)}$)[4] symmetrisch positiv definit ist, gilt nach Lemma 4.10

$$a_{k+1,k+1}^{(k-1)} a_{k,k}^{(k-1)} > (a_{k+1,k-1}^{(k-1)})^2$$

  und damit

$$a_{k+1,k+1}^{(k)} = a_{k+1,k+1}^{(k-1)} - \frac{1}{a_{k,k}^{(k-1)}} (a_{k+1,k-1}^{(k-1)})^2 > 0,$$

  womit der Induktionsschritt komplett ist.

[4] Dies sieht man durch geeignete Wahl von x für $x^T A x > 0$.

Man erhält also nach n Schritten $A^{(n)} = I$ und damit

$$A = L^{(1)} \cdots L^{(n)}\, I\, L^{(n)^T} \cdots L^{(1)^T} =: LL^T.$$

Nach Konstruktion sind die $L^{(k)}$ nun Frobeniusmatrizen, und wie bei der LR-Zerlegung weist man nach, dass ihr Produkt L eine untere Dreiecksmatrix ist. □

Umgekehrt folgt aus $A = LL^T$ mit L invertierbar, dass A symmetrisch ist und es gilt

$$x^T Ax = x^T LL^T x = (L^T x)^T (L^T x) = \|L^T x\|_2^2 > 0 \qquad \text{für alle } x \neq 0,$$

d. h. A ist positiv definit.

Dieser Beweis lässt sich konstruktiv umsetzen; man erhält dadurch die *Cholesky-Zerlegung*.[5]

Algorithmus 5.4 : Cholesky-Zerlegung

Input : a_{ij}

1 **for** $k = 1, \ldots, n$ **do**

2 $\quad l_{kk} \leftarrow \sqrt{a_{kk} - \sum_{j=1}^{k-1} (l_{kj})^2}$

3 $\quad$ **for** $i = k+1, \ldots, n$ **do**

4 $\quad\quad l_{ik} \leftarrow \left(a_{ik} - \sum_{j=1}^{k-1} l_{ij} l_{kj}\right) / l_{kk}$

Output : $l_{ij}, i \geqslant j$

Der Aufwand für die Cholesky-Zerlegung beträgt zwar auch $\mathcal{O}(n^3)$, bei genauerer Betrachtung sind aber nur halb so viele Operationen notwendig wie für eine LR-Zerlegung. Außerdem ist keine Pivotsuche notwendig.

5.2 ITERATIVE VERFAHREN

Wir sind nun an Verfahren interessiert, die auch für allgemeinere große Matrizen mit wenigen Einträgen[6] den Vorteil besitzen, für die Lösung eines linearen Gleichungssystems denselben Speicheraufwand wie für ihre Speicherung zu besitzen. Dies wird dadurch erreicht, dass diese Verfahren im Wesentlichen auf der wiederholten Matrixmultiplikation beruhen. Dadurch wird man aber in der Regel nie in endlicher Zeit auf die exakte Lösung kommen, sondern Schritt für Schritt immer bessere Näherungen erhalten. Die vorrangige Aufgabe wird sein, die Konvergenz dieser Näherungen gegen die exakte Lösung zu garantieren.

Wir betrachten hier zwei Klassen von Verfahren.

[5] Von André-Louis Cholesky um 1905 für lineare Ausgleichsprobleme bei der Land-Neuvermessung Frankreichs entwickelt; die Methode wurde erst 1924, nach seinem Tod im ersten Weltkrieg, publiziert.

[6] Solche Matrizen nennt man *dünn besetzt* (engl. "*sparse*"), falls die Anzahl der von Null verschiedenen Einträge nur $\mathcal{O}(n)$ (statt $\mathcal{O}(n^2)$) ist.

5.2.1 LINEARE ITERATIONSVERFAHREN

Der zentrale Ansatz für diese Verfahrensklasse ist, das lineare Gleichungssystem als eine Fixpunktgleichung zu schreiben. Es ist $Ax = b$ genau dann, wenn gilt

$$x = x + b - Ax = b + (I - A)x.$$

Dies führt auf die *Fixpunktiteration*

$$x^{(k+1)} = b + (I - A)x^{(k)}.$$

Dieses Verfahren wird *Richardson-Iteration*[7] genannt; zu seiner Durchführung ist nur die Multiplikation mit der Matrix A notwendig.[8] Es ist natürlich nur dann sinnvoll, wenn $x^{(k)} \to x^* := A^{-1}b$ für $k \to \infty$ gilt. Um das zu prüfen, betrachten wir den Fehler im $k+1$-ten Schritt. Durch Einsetzen der Iterationsvorschrift erhalten wir

$$\begin{aligned} x^* - x^{(k+1)} &= x^* - b - (I - A)x^{(k)} = x^* - x^{(k)} - (b - Ax^{(k)}) \\ &= x^* - x^{(k)} - (Ax^* - Ax^{(k)}) = (I - A)(x^* - x^{(k)}), \end{aligned}$$

und damit

$$\|x^* - x^{(k+1)}\| = \|(I - A)(x^* - x^{(k)})\| \leqslant \|I - A\|\|x^* - x^{(k)}\|.$$

Es gilt also

$$\|x^* - x^{(k)}\| \leqslant \|I - A\|^k \|x^* - x^{(0)}\|.$$

Die Richardson-Iteration konvergiert also dann und nur dann für alle Startwerte $x^{(0)}$, wenn $\|I - A\| < 1$ gilt.[9] Dies ist aber in der Regel nicht der Fall. Die Idee führt aber zu brauchbaren Verfahren, indem wir die Iteration mit einer "besseren" Matrix $\tilde{A} = CA$ (und rechter Seite Cb) für eine geeignete Matrix C durchführen:

Definition 5.7. Sei $C \in \mathbb{R}^{n\times n}$ regulär. Dann heißt die Fixpunktiteration

$$x^{(k+1)} = (I - CA)x^{(k)} + Cb$$

lineares Iterationsverfahren, $(I - CA)$ nennt man *Iterationsmatrix*.

Wir wollen nun die Konvergenz dieser Verfahren sorgfältiger untersuchen:

[7] Diese Methode geht auf Lewis Fry Richardson zurück. Er propagierte auch 1922 die heutige Methode der Wettervorhersage auf Basis von numerischer Simulation. (Ein eigener erster Versuch von 1910 – durchgeführt von Hand! – war grundsätzlich korrekt, lieferte aber wegen gestörter Eingabedaten ein falsches Ergebnis.)

[8] Die Matrix muss nicht explizit gegeben sein; es genügt, eine Prozedur zu kennen, die für einen gegebenen Vektor v den Vektor Av berechnet.

[9] Dies ist eine sehr einfache Form des Banachschen Fixpunktsatzes 10.1.

Satz 5.8. *Ein lineares Iterationsverfahren mit der Iterationsmatrix* $(I - CA)$ *konvergiert für jeden Startwert* $x^{(0)}$ *genau dann, wenn gilt*

$$\rho(I - CA) < 1.$$

Beweis. Da das Verfahren linear ist, gilt analog zur Richardson-Iteration für den Fehler $e^{(k)} := x^* - x^{(k)}$

$$e^{(k)} = (I - CA)^k e^{(0)}.$$

Wir nehmen nun der Einfachheit halber an, dass $I - CA$ diagonalisierbar ist, d. h. es existiert ein $T \in \mathbb{R}^{n \times n}$ mit

$$T^{-1}(I - CA)T = D = \begin{pmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{pmatrix},$$

wobei λ_i die Eigenwerte von $(I - CA)$ sind. Damit ist

$$e^{(k)} = (I - CA)^k e^{(0)} = TD^k T^{-1} e^{(0)},$$

und wir haben

$$\|e^{(k)}\|_2 = \|TD^k T^{-1} e^{(0)}\|_2 \leqslant \|T\|_2 \|D\|_2^k \|T^{-1} e^{(0)}\|_2 = \left(\max_{1 \leqslant i \leqslant n} |\lambda_i| \right)^k \|T\|_2 \|T^{-1} e^{(0)}\|_2 \to 0$$

falls $\max_{1 \leqslant i \leqslant n} |\lambda_i| = \rho(I - CA) < 1$ gilt.

Andererseits sei angenommen, dass $\lambda_j \geqslant 1$ für ein $1 \leqslant j \leqslant n$. Nun wähle man als Startwert $x^{(0)} = x^* - v_j$ mit v_j Eigenvektor zu λ_j. Dann gilt aber $e^{(0)} = v_j$ und somit für alle $k > 0$

$$\|e^{(k)}\|_2 = \|(I - CA)^k v_j\| = |\lambda_j|^k \|v_j\| \geqslant \|e^{(0)}\|.$$

Das Verfahren konvergiert also nicht für diesen Startwert. □

Da jede induzierte Matrixnorm eine obere Schranke für den Spektralradius ist, erhält man

Folgerung 5.9. *Gilt* $\|I - CA\| < 1$ *für eine induzierte Matrixnorm, so konvergiert das lineare Iterationsverfahren für jeden Startwert.*

Bemerkung. Die Konvergenz des Iterationsverfahrens kann beliebig langsam sein. Ein wichtiges Maß für die Konvergenzgeschwindigkeit ist, wie stark der Fehler in jedem Schritt reduziert wird, und damit wie viele Iterationen notwendig sind, um den Fehler unter eine vorgegebene Schranke zu bringen. Dieser Faktor hängt für gegebenes C jedoch sehr von der Struktur und Größe der Matrix A ab (nicht aber von der rechten Seite). Üblicherweise erwartet man, dass zwischen $\mathcal{O}(1)$ und $\mathcal{O}(n)$ Iteration erforderlich sind.

Wir betrachten nun konkrete lineare Iterationsverfahren, d. h. konkrete Beispiel für die Wahl der Matrix C. Dabei soll nach Satz 5.8 möglichst $CA \approx I$ sein, idealerweise also $C = A^{-1}$. Dann wäre das Verfahren zwar in einem Schritt fertig, man hätte aber nichts an Effizienz gewonnen. Wir brauchen deshalb eine Matrix, die "nahe bei" A^{-1} ist, aber leicht berechnet werden kann.

Die klassischen Methoden gehen dafür alle von der Zerlegung $A = L + D + R$ mit einer unteren Dreiecksmatrix L, einer Diagonalmatrix D und einer oberen Dreiecksmatrix R aus.

JACOBI-ITERATION: $C = D^{-1}$

Diese Wahl[10] führt zu der Iterationsvorschrift

$$x^{(k+1)} = (I - D^{-1}A)x^{(k)} + D^{-1}b,$$

die durchführbar ist, solange $a_{ii} \neq 0$ für alle $1 \leqslant i \leqslant n$. Nutzt man die Schreibweise $A = L + D + R$, so kann man das Verfahren effizienter schreiben:

$$Dx^{(k+1)} = b - (L + R)x^{(k)}.$$

Algorithmus 5.5 : Jacobi-Iteration

Input : a_{ij}, b_i, x_i^0, k^*

1 **for** $k = 1, \dots, k^*$ **do**

2 **for** $i = 1, \dots, n$ **do**

3 $x_i^k \leftarrow \left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{k-1} - \sum_{j=i+1}^{n} a_{ij}x_j^{k-1}\right)/a_{ii}$

Output : x_i

Die Komplexität dieses Verfahren ist pro Schritt $\mathcal{O}(n)$, falls A eine dünn besetzte Matrix ist (und damit die Summen nicht über die vollen n Summanden zu berechnen sind); die gesamte Komplexität hängt von der Anzahl der Iterationen ab. Dabei erfolgt die Berechnung der x_i in jedem Schritt unabhängig voneinander, das Verfahren ist also sehr gut parallelisierbar. Für die Konvergenz zeigt man leicht:

Satz 5.10. *Sei $A \in \mathbb{R}^{n\times n}$ streng diagonal dominant. Dann konvergiert die Jacobi-Iteration.*

Beweis. Es gilt

$$\rho(I - D^{-1}A) = \rho(D^{-1}(L + R)) \leqslant \|D^{-1}(L + R)\|_\infty = \max_{1\leqslant i\leqslant j} \sum_{j\neq i} \frac{|a_{ij}|}{|a_{ii}|} < 1. \qquad \square$$

[10] Dies ist eine stark vereinfachte Fassung des Jacobi-Verfahrens für die Eigenwertberechnung, welches Carl Gustav Jacobi 1846 für die Berechnung von Störungen von Planetenbahnen entwickelte.

GAUSS–SEIDEL-ITERATION: $C = (D + L)^{-1}$

Der Iterationsschritt[11] dazu lautet also:

$$x^{(k+1)} = (I - (D + L)^{-1}A)x^{(k)} + (D + L)^{-1}b.$$

Dabei berechnet man $(D + L)^{-1}$ nicht explizit; da $D + L$ eine untere Dreiecksmatrix ist, lässt sich das zugehörige Gleichungssystem leicht durch Vorwärtssubstitution lösen:

$$(D + L)x^{(k+1)} = b - Rx^{(k)}.$$

Algorithmus 5.6 : Gauß–Seidel-Iteration

Input : a_{ij}, b_i, x_i^0, k^*

1 **for** $k = 1, \dots, k^*$ **do**

2 **for** $i = 1, \dots, n$ **do**

3 $x_i^k \leftarrow \left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^k - \sum_{j=i+1}^{n} a_{ij}x_j^{k-1}\right)/a_{ii}$

Output : x_i

Dies entspricht der Jacobi-Iteration, wenn man bereits berechnete x_i in der Gleichung für x_j einsetzt. Damit ist das Verfahren nicht mehr ohne weiteres parallelisierbar. Außerdem hängt die Konvergenzgeschwindigkeit von der Reihenfolge der Berechnung der x_i ab, ist aber für wichtige Klassen von Matrizen höher als bei der Jacobi-Iteration. Für die Konvergenz halten wir fest:

Satz 5.11. *Sei $A \in \mathbb{R}^{n\times n}$ streng diagonal dominant oder symmetrisch positiv definit. Dann konvergiert die Gauß–Seidel-Iteration.*

SOR-VERFAHREN ("*successive over-relaxation*")

Das SOR-Verfahren[12] basiert auf einer anderen Schreibweise der Gauß–Seidel-Iteration: als

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

Wir addieren also in jedem Schritt eine "Korrektur" zu $x_i^{(k)}$. Konvergiert das Verfahren nun zu langsam, können wir womöglich durch eine stärkere Korrektur die Konvergenz beschleunigen.

[11] publiziert 1874 von Philipp Ludwig von Seidel

[12] entwickelt von David Young für seine 1950 verfasste Doktorarbeit über die numerische Lösung von partiellen Differentialgleichungen

(Konvergiert das Verfahren nicht, könnte eine kleinere Korrektur die Konvergenz wiederherstellen.) Wir multiplizieren den Korrekturterm deshalb mit einem *Relaxationsparameter* ω, und erhalten

Algorithmus 5.7 : SOR-Iteration

Input : $a_{ij}, b_i, x_i^0, k^*, \omega$

for $k = 1, \ldots, k^*$ **do**

  for $i = 1, \ldots, n$ **do**

    $x_i^k = x_i^{k-1} - \omega \left(\sum_{j=1}^{i-1} a_{ij} x_j^k + \sum_{j=i}^{n} a_{ij} x_j^{k-1} - b_i \right) / a_{ii}$

Output : x_i

Durch Wahl des richtigen Parameters lässt sich die Konvergenz des Verfahrens entscheidend beschleunigen. Leider ist der optimale Parameter in der Regel nicht bekannt; man kann aber zumindest beweisen, welche Parameter überhaupt zu einem konvergenten Verfahren führen:

Satz 5.12. *Sei $A \in \mathbb{R}^{n \times n}$. Dann konvergiert das SOR-Verfahren genau dann, wenn*

- $\omega \in (0, 2)$ *für symmetrisch positiv definite* A
- $\omega \in (0, 1]$ *für streng diagonal dominante* A

5.2.2 GRADIENTENVERFAHREN

Der exponentiellen Anstieg der Möglichkeiten der Computersimulation in der zweiten Hälfte dieses Jahrhunderts übertrifft deutlich das reine Anwachsen der Leistungsfähigkeit der verwendeten Computer. Entscheidend war auch die Entwicklung von iterativen Verfahren zur Lösung linearer Gleichungssysteme, die schneller als linear konvergieren können. Stellvertretend für diese Klasse von Verfahren wollen wir uns hier mit dem Gradientenverfahren beschäftigen.

Wir werden uns von nun an auf symmetrisch positiv definite Matrizen konzentrieren. Für diese gelten nämlich folgende zentrale Aussagen:

Lemma 5.13. *Sei $A \in \mathbb{R}^{n \times n}$ symmetrisch positiv definit. Dann gilt:*

a) $\langle x, y \rangle_A := x^T A y$ *definiert ein Skalarprodukt auf* $\mathbb{R}^n$*, mit induzierter Norm* $\|x\|_A^2 := x^T A x$ *(*Energienorm*).*

b) Für $b \in \mathbb{R}^n$ *hat die Funktion* $f : \mathbb{R}^n \to \mathbb{R}$, $f(x) := \frac{1}{2} x^T A x - b^T x$ *ein eindeutiges Minimum in* x^* *genau dann, wenn* $A x^* = b$.

Beweis. a) Man prüft leicht die Axiome für das Skalarprodukt.

b) Es gelte $Ax^* = b$, und sei $e \neq 0$ ein beliebiger Vektor in $\mathbb{R}^n$. Dann gilt (wegen der Symmetrie von A)

$$\begin{aligned} f(x^* + e) &= \frac{1}{2}(x^* + e)^T A(x^* + e) - b^T(x^* + e) \\ &= \frac{1}{2}x^{*T}Ax^* - b^Tx^* + \frac{1}{2}x^{*T}Ae + \frac{1}{2}e^TAx^* - b^Te + \frac{1}{2}e^TAe \\ &= f(x^*) + x^{*T}Ae - (Ax^*)^Te + \frac{1}{2}e^TAe \\ &= f(x^*) + \frac{1}{2}\|e\|_A^2 > 0. \end{aligned}$$

Jede Veränderung des Arguments vergrößert also den Wert von f, weshalb $f(x^*)$ minimal sein muss.

Umgekehrt ist eine notwendige Bedingung dafür, dass $f(x)$ in x^* ein Minimum hat, dass gilt

$$0 = \nabla f(x^*) = ((\partial_{x_1} f)(x^*), \ldots, (\partial_{x_n} f)(x^*))^T = Ax^* - b. \qquad \square$$

IDEE: Statt $Ax = b$ zu lösen, wird ein Minimum von $f(x)$ gesucht, und zwar mit folgender Iteration:

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

Wie wählt man nun die *Abstiegsrichtung* $d^{(k)}$ und die *Schrittweite* $\alpha^{(k)}$?

1. Aus der Analysis ist bekannt, dass der Gradient ∇f senkrecht auf den Höhenlinien von f steht. Der *steilste Abstieg* ist also entlang des negativen Gradienten:

   $$d^{(k)} = -\nabla f(x^{(k)}) = b - Ax^{(k)} =: r^{(k)},$$

   d. h. entlang des *Residuums* $r^{(k)}$.

2. Die optimale Schrittweite für eine vorgegebene Richtung $d^{(k)}$ ist das Minimum von

   $$g(\alpha) := f(x^{(k)} + \alpha d^{(k)}) = \frac{1}{2}(x^{(k)} + \alpha d^{(k)})^T A(x^{(k)} + \alpha d^{(k)}) - b^T(x^{(k)} + \alpha d^{(k)}).$$

   Wir setzen also für $d^{(k)} = r^{(k)}$

   $$0 = g'(\alpha) = r^{(k)T}Ax^{(k)} + \alpha r^{(k)T}Ar^{(k)} - b^Tr^{(k)} = -r^{(k)T}r^{(k)} + \alpha r^{(k)T}Ar^{(k)}.$$

   Da A positiv definit ist, ist der letzte Term ungleich Null, solange das Residuum nicht Null ist (und dann wären wir ja bereits fertig). Wir können also nach α auflösen und erhalten

   $$\alpha^{(k)} = \frac{r^{(k)T}r^{(k)}}{r^{(k)T}Ar^{(k)}}.$$

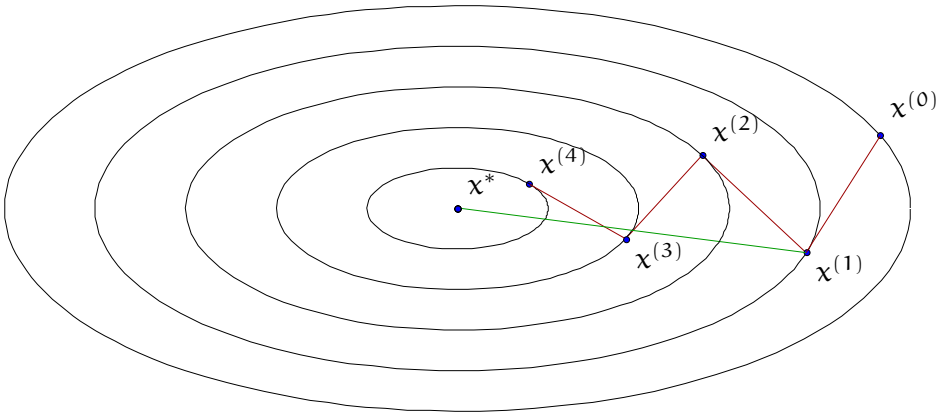


Abbildung 5.1: Konvergenz der Iterierten $x^{(k)}$ gegen die exakte Lösung x^* für das Gradientenverfahren (rot) bzw. CG-Verfahren (grün). Die schwarz gezeichneten Ellipsen zeigen Niveaulinien der Funktion $f(x) = \frac{1}{2}\|x - x^*\|_A^2$.

Damit erhalten wir das *Gradientenverfahren* (oder *steepest-descent-Verfahren*)

Algorithmus 5.8 : Gradientenverfahren

Input : A, b, x^0, k^*

1 **for** $k = 1, \ldots, k^*$ **do**

2 $\quad r^k \leftarrow b - Ax^{k-1}$

3 $\quad \alpha^k \leftarrow ({r^k}^T r^k)/({r^k}^T A r^k)$

4 $\quad x^k \leftarrow x^{k-1} + \alpha^k r^k$

Output : x

Für festes $\alpha = 1$ entspricht dies der Richardson-Iteration. Und tatsächlich kann auch dieses Verfahren unter Umständen sehr langsam konvergieren: Wenn die Höhenlinien von f (die Ellipsen darstellen, deren Hauptachsen durch die Eigenwerte von A gegeben sind) stark verzerrt sind (was einer schlechten Kondition von A entspricht), dann wird der Abstieg in immer kleineren Schritten erfolgen, die jeweils paarweise (annähernd) senkrecht stehen (siehe Abbildung 5.1).

Ein deutlich besseres Verfahren erhalten wir, wenn wir nicht stur den Gradienten im Punkt $x^{(k)}$ verwenden, sondern auch die bereits gemachten Schritte berücksichtigen. Wir wollen also in jedem Schritt $f(x)$ minimieren, und zwar entlang aller zuvor berechneten Richtungen $d^{(i)}$ für $i = 1, \ldots, k-1$. Dazu schreiben wir $f(x)$ als

$$f(x) = \frac{1}{2}(x - x^*)^T A(x - x^*) - \frac{1}{2}{x^*}^T A x^* = \frac{1}{2}\|x - x^*\|_A^2 - c,$$

wobei c nicht von x abhängt, wir also nur den ersten Term minimieren müssen:

$$\text{Minimiere} \quad \frac{1}{2}(x - x^*)^T A(x - x^*) = \frac{1}{2}\|x - x^*\|_A^2 \quad \text{über alle } x \in \operatorname{span}\{d^{(i)}\}.$$

Dieses Problem kann durch (A-)*orthogonale Projektion* gelöst werden:

Lemma 5.14 (Orthogonale Projektion)**.** *Sei K_m ein m-dimensionaler Unterraum von $\mathbb{R}^n$, $m \leqslant n$, mit A-orthogonaler Basis $\{p_0, \dots, p_{m-1}\}$ (d. h. $\langle p_i, p_j \rangle_A = 0$ für alle $i \neq j$). Für $v \in \mathbb{R}^n$ gilt*

$$\|\tilde{u} - v\|_A = \min_{u \in K_m} \|u - v\|_A$$

genau dann, wenn gilt $\tilde{u} \in K_m$ und

$$\langle \tilde{u} - v, w \rangle_A = 0 \quad \text{für alle } w \in K_m.$$

Die eindeutige Lösung dieser Gleichungen ist gegeben durch

$$\tilde{u} = \sum_{j=0}^{m-1} \frac{\langle v, p_j \rangle_A}{\langle p_j, p_j \rangle_A} p_j.$$

Für $A = I$ entspricht dies dem aus der linearen Algebra bekannten Resultaten, und in der Tat ist der Beweis wörtlich identisch.

Nehmen wir $K_m = \operatorname{span}\{r^{(0)}, \dots, r^{(m-1)}\}$, so erhalten wir den folgenden Algorithmus:

1 Beginne mit $x^{(0)} = 0$, $r^{(0)} = b$, $K_1 = \operatorname{span}\{r^{(0)}\}$.
2 **for** $k = 1, \dots$ **do**
3 a) Bestimme $x^{(k)}$ durch A-orthogonale Projektion auf K_k
4 b) Erweitere den Unterraum $K_{k+1} = \operatorname{span}(K_k \cup \{r^{(k)}\})$
5 c) Bestimme A-orthogonale Basis $\{p_0, \dots, p_k\}$ von K_{k+1}

Statt der Gradienten nehmen wir also A-orthogonalisierte bzw. A-*konjugierte* Gradienten. Deshalb heißt dieser Algorithmus auch *Methode der konjugierten Gradienten* oder *CG-Verfahren* (nach engl.: *"conjugate gradients"*). Dass dieses Verfahren überhaupt konvergiert, liegt an der Orthogonalität[13] der Residuen:

Lemma 5.15. *Die Residuen $\{r^{(0)}, \dots, r^{(m-1)}\}$ im CG-Verfahren bilden eine Orthogonalbasis von K_m, falls $r^{(i)} \neq 0$ für $i = 0, \dots, m-1$.*

Beweis. Durch Induktion nach m: Für $m = 1$ ist $K_1 = r^{(0)}$. Sei jetzt $\{r^{(0)}, \dots, r^{(m-1)}\}$ eine Orthogonalbasis von K_m. Dann ist für $i < m$

$$r^{(i)^T} r^{(m)} = r^{(i)^T}(b - Ax^{(m)}) = -r^{(i)^T} A(x^* - x^{(m)}) = -\langle r^{(i)}, x^* - x^{(m)} \rangle_A = 0,$$

da $x^{(m)}$ die A-orthogonale Projektion auf x^* in K_m ist. □

[13] aber nicht A-Orthogonalität!

Satz 5.16. *Sei* $A \in \mathbb{R}^{n\times n}$ *symmetrisch positiv definit und* $b \in \mathbb{R}^n$. *Dann gilt für die Iterierte* $x^{(n)}$ *im CG-Verfahren* $Ax^{(n)} = b$.

Beweis. Aus Lemma 5.15 folgt, dass die Residuen $r^{(0)}, \dots, r^{(n-1)}$ orthogonale Vektoren sind, ihr Spann K_n hat also Dimension n und ist daher gleich $\mathbb{R}^n$. Damit ist $x^{(n)}$ nach Konstruktion der Minimierer von $\|x^* - x\|_A$ über ganz $\mathbb{R}^n$, d. h. es muss $x^{(n)} = x^* = A^{-1}b$ sein. □

Damit könnte man das CG-Verfahren auch zu den direkten Verfahren zählen, da es nach n Schritten die exakte Lösung liefert. Freilich gilt dieses Resultat nur in exakter Arithmetik, und eine hinreichend gute Näherungslösung erhält man in der Regel in deutlich weniger Schritten.

Dass diese Methode zu den berühmtesten Algorithmen gehört, liegt daran, dass die oben genannten Schritte effizient rekursiv berechenbar sind – auch wenn es auf den ersten Blick nicht so aussieht.

A) A-ORTHOGONALE PROJEKTION Sei $\{p_0, \dots, p_{k-1}\}$ eine A-orthogonale Basis von K_k. Dann haben wir

$$x^{(k)} = \sum_{j=0}^{k-1} \frac{x^{*T}Ap_j}{p_j{}^T Ap_j} p_j = \sum_{j=0}^{k-2} \frac{x^{*T}Ap_j}{p_j{}^T Ap_j} p_j + \frac{x^{*T}Ap_{k-1}}{p_{k-1}{}^T Ap_{k-1}} p_{k-1}.$$

Da einerseits $\{p_0, \dots, p_{k-2}\}$ eine A-orthogonale Basis von K_{k-1} ist, ist die erste Summe gerade die Projektion von x^* auf K_{k-1}, also $x^{(k-1)}$. Andererseits folgt aus der Symmetrie von A, dass $x^{*T}A = (Ax^*)^T = b^T$. Wir erhalten also die Rekursionsformeln

$$\alpha^{(k)} = \frac{b^T p_{k-1}}{p_{k-1}{}^T Ap_{k-1}}, \tag{5.3}$$

$$x^{(k)} = x^{(k-1)} + \alpha^{(k)} p_{k-1}. \tag{5.4}$$

B) ERWEITERUNG DES SUCHRAUMS Auch das neue Residuum $r^{(k)}$ kann rekursiv berechnet werden: Indem (5.4) auf beiden Seiten mit A multipliziert und von b subtrahiert wird, erhalten wir

$$r^{(k)} = r^{(k-1)} - \alpha^{(k)} Ap_{k-1}. \tag{5.5}$$

C) A-ORTHOGONALE BASIS Da die $p_0, \dots, p_{k-1}$ bereits A-orthogonal sind, müssen wir nur noch einen A-orthogonalen Basisvektor p_k berechnen. Dazu wenden wir das Gram–Schmidt-Verfahren auf den Vektor $r^{(k)}$ an (der ja senkrecht auf K_k steht, also linear unabhängig zu allen bisherigen p_j ist) und erhalten

$$p_k = r^{(k)} - \sum_{j=0}^{k-1} \frac{\langle r^{(k)}, p_j \rangle_A}{\langle p_j, p_j \rangle_A} p_j.$$

Der Clou des CG-Verfahrens ist nun, dass wir gar nicht die komplette Summe berechnen müssen. Für $j < k-1$ gilt nämlich wegen (5.5)

$$\begin{aligned} \langle r^{(k)}, p_j \rangle_A &= \langle r^{(k)}, Ap_j \rangle = \langle r^{(k)}, \frac{1}{\alpha^{(j+1)}}(r^{(j)} - r^{(j+1)}) \rangle \\ &= \frac{1}{\alpha^{(j+1)}} \langle r^{(k)}, r^{(j)} \rangle - \frac{1}{\alpha^{(j+1)}} \langle r^{(k)}, r^{(j+1)} \rangle = 0 \end{aligned}$$

nach Lemma 5.15. Es ist also nur der letzte Summand von Null verschieden, und wir erhalten als letzte Rekursion

$$\beta^{(k)} = -\frac{{r^{(k)}}^T A p_{k-1}}{{p_{k-1}}^T A p_{k-1}}, \tag{5.6}$$

$$p_k = r^{(k)} + \beta^{(k)} p_{k-1}. \tag{5.7}$$

Die Gleichungen (5.3), (5.4), (5.5), (5.6) und (5.7) ergeben zusammen den berühmten CG-Algorithmus:[14]

Algorithmus 5.9 : CG-Verfahren

Input : A, b, k^*

1 $x^0 = 0, p^0 = r^0 = b$
2 **for** $k = 1, \dots, k^*$ **do**
3 $\quad \alpha^k \leftarrow ({r^{k-1}}^T r^{k-1})/({p^{k-1}}^T A p^{k-1})$
4 $\quad x^k \leftarrow x^{k-1} + \alpha^k p^{k-1}$
5 $\quad r^k \leftarrow r^{k-1} - \alpha^k A p^{k-1}$
6 $\quad \beta^k \leftarrow ({r^k}^T r^k)/({r^{k-1}}^T r^{k-1})$
7 $\quad p^k \leftarrow r^k + \beta^k p^{k-1}$

Output : x

Dabei wurden die Gleichungen für α^k und β^k mit Hilfe von (5.4) etwas umgeformt, um die Anzahl der zu berechnenden Skalarprodukte zu minimieren. In jedem Schritt ist nur eine Matrixmultiplikation nötig, der Aufwand pro Schritt entspricht also dem eines linearen Iterationsverfahrens.

Für die Konvergenzgeschwindigkeit kann man zeigen:

[14] Durch Magnus Hestenes und Eduard Stiefel unabhängig voneinander entdeckt und 1952 gemeinsam publiziert.

Satz 5.17. *Sei* $A \in \mathbb{R}^{n\times n}$ *symmetrisch positiv definit und* $b \in \mathbb{R}^n$. *Dann gilt für die Iterierte* $x^{(k)}$ *im CG-Verfahren und* $x^* := A^{-1}b$

$$\|x^{(k)} - x^*\|_A \leqslant 2\left(\frac{\sqrt{\kappa_2(A)}-1}{\sqrt{\kappa_2(A)}+1}\right)^k \|x^{(0)} - x^*\|_A.$$

Da die Klammer auf der rechten Seite kleiner als 1 ist, folgt die Konvergenz des CG-Verfahrens für symmetrisch positiv definite Matrizen. Zusätzlich kann man Satz 5.17 entnehmen, dass das CG-Verfahren desto schneller konvergiert, je näher die Konditionszahl $\kappa_2(A)$ an 1 liegt, d. h. je kleiner das Verhältnis zwischen dem betragsgrößten und -kleinsten Eigenwert ist. (Zum Vergleich: Die Konvergenzgeschwindigkeit von stationären Iterationsverfahren hängt dagegen direkt von der Größe der Eigenwerte ab.) Ganz analog zu den linearen Iterationsverfahren lässt sich also die Konvergenz beschleunigen, indem man für eine geeignete Matrix M das Gleichungssystem $MAx = Mb$ betrachtet. Dies bezeichnet man als *Präkonditionierung*, M als *Präkonditionierer*.

Bemerkung. Verfahren, die in jedem Schritt den Fehler in solch einer Folge von wachsenden Unterraum minimieren, heißen *Krylov-Raum-Verfahren* (die K_m sind *Krylov-Räume*). Von diesen ist für symmetrisch positiv definite Matrizen das CG-Verfahren das beste. Weitere wichtige Verfahren, die nicht nur für symmetrisch positiv definite Matrizen konvergieren, sind

- MINRES für indefinite, aber symmetrische Matrizen (auch mit garantierter Konvergenz),
- GMRES für unsymmetrische, indefinite Matrizen,
- BICGSTAB für unsymmetrische, indefinite Matrizen.

WEITERFÜHRENDE LITERATUR

C. Clason (2016). „Iterative Verfahren für lineare Gleichungssysteme und Eigenwertprobleme“. Vorlesungsskript, Fakultät für Mathematik, Universität Duisburg-Essen. URL: https://www.uni-due.de/~adf040p/skripte/IterativSkript16.pdf.

G. H. Golub und C. F. Van Loan (2013). *Matrix Computations*. 4. Aufl. Johns Hopkins University Press, Baltimore, MD.

Y. Saad (2003). *Iterative Methods For Sparse Linear Systems*. 2. Aufl. SIAM, Philadelphia, PA. URL: http://www-users.cs.umn.edu/~saad/IterMethBook_2ndEd.pdf.

J. R. Shewchuk (1994). *An Introduction to the Conjugate Gradient Method Without the Agonizing Pain*. Techn. Ber. Pittsburgh, PA, USA. URL: http://www.cs.cmu.edu/~quake-papers/painless-conjugate-gradient.pdf.

LINEARE AUSGLEICHSRECHNUNG

6

Die bislang betrachteten Verfahren setzen voraus, dass die Matrix A quadratisch ist. In zahlreichen Anwendungen ist dies jedoch nicht der Fall, so dass wir in diesem Kapitel Methoden zur Behandlung von linearen Gleichungssystemen mit nichtquadratischen Matrizen $A \in \mathbb{R}^{m\times n}$ untersuchen wollen.

Wir betrachten hier nur den Fall $m > n$. Das Gleichungssystem $Ax = b$, $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$ ist dann überbestimmt, so dass wir in der Regel keine Lösung erwarten dürfen. Wir müssen also unseren Begriff von "Lösung" erweitern: Statt exakter Gleichheit erwarten wir nur noch, dass Ax "so nahe wie möglich" bei b liegt. Ähnlich wie in Abschnitt 5.2.2 führt dies auf ein Minimierungsproblem: Wir nennen $x^* \in \mathbb{R}^n$ *Lösung des linearen Ausgleichsproblems*, falls

$$\|Ax^* - b\|_2^2 = \min_{x\in\mathbb{R}^n} \|Ax - b\|_2^2$$

gilt. Dass hierbei die Euklidische Norm $\|\cdot\|_2$ verwendet wird, ist fundamental, genau wie die Minimierung des Quadrats der Norm. Aus diesem Grunde wird dieser Ansatz auch "Methode der kleinsten (Fehler-)Quadrate", engl. "least squares", genannt.

Aus der Fülle der Anwendungen in Mathematik und Praxis seien hier zwei genannt.

Beispiel 6.1. (i) In der Anwendung müssen häufig Größen bestimmt werden, die nicht direkt messbar sind. Gesucht sind etwa die Werte $x_1, \dots, x_n$ (z. B. Temperatur in gleichmäßig verteilten Punkten eines Stabes, der zum Teil in einer Wand versenkt ist), verfügbar ist jedoch nur ein Messwert φ (z. B. die Temperatur am frei liegenden Stabende), der abhängig ist von den Unbekannten x_i sowie einem kontrollierbaren Parameter t (z. B. die Umgebungstemperatur). Letzterer erlaubt es, mehrere unabhängige Messungen φ_j, $1 \leqslant j \leqslant m$, durchzuführen. Ist nun die Abhängigkeit $\varphi_j = \varphi(t_j; x_1, \dots, x_2)$ bekannt, so kann man hoffen, eine Schätzung der x_i zu bekommen, indem man diejenigen x_i^* bestimmt, für die

$$\sum_{j=1}^{m} |\varphi(t_j; x_1, \dots, x_n) - \varphi_j|^2$$

minimal ist. Im einfachsten Fall einer linearen Abhängigkeit

$$\varphi(t_j; x_1, \dots, x_2) = \sum_{i=1}^{n} a_i(t_j) x_i = \sum_{i=1}^{n} a_{ij} x_i$$

mit bekannten Koeffizienten a_{ij} führt dies auf das lineare Ausgleichsproblem $\|Ax - \varphi\|_2^2 \to \min$.

(ii) Eine weiteres Beispiel ist die *Funktionsapproximation*, deren einfachster Spezialfall die Suche nach der *Regressionsgerade* ist. Hierbei handelt es sich um das Problem, zu vorgegebenen Werten $x_1, \dots, x_m$ und $y_1, \dots, y_m$ eine Gerade $y = ax + b$ so zu finden, dass die Punkte (x_i, y_i) möglichst nahe bei ihr liegen. Mit der Schreibweise $\varphi(x_j; a, b) = ax_j + b$, $x = (x_1, \dots, x_m)^T$, $y = (y_1, \dots, y_m)^T$, $\mathbb{1} = (1, \dots, 1)^T \in \mathbb{R}^m$ führt dies auf das lineare Ausgleichsproblem

$$\left\| (\mathbb{1}, x) \begin{pmatrix} a \\ b \end{pmatrix} - y \right\|_2^2 = \sum_{j=1}^{n} |\varphi(x_j; a, b) - y_j|^2 \to \min .$$

6.1 DIE NORMALENGLEICHUNGEN

Wie schon bei der Herleitung des Gradientenverfahrens müssen wir eine quadratische Funktion, hier $f(x) = \|Ax - b\|_2^2$, minimieren, und betrachten dafür die notwendige Bedingung

$$\begin{aligned} 0 = \nabla f(x) &= \nabla \left[\langle Ax - b, Ax - b \rangle \right] = \nabla \left[\langle Ax, Ax \rangle - 2\langle Ax, b \rangle - \langle b, b \rangle \right] \\ &= 2A^T Ax - 2A^T b. \end{aligned}$$

Jedes Minimum x^* von $f(x)$ erfüllt also die *Normalengleichungen*

$$A^T Ax = A^T b.$$

Geometrisch bedeutet dies, dass das Residuum $b - Ax^*$ senkrecht steht auf dem Bild von A (siehe Abbildung 6.1).

Wir erhalten das folgende Resultat über die Lösbarkeit des Ausgleichsproblems

Satz 6.2. *Sei $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$. Dann gilt*

(i) x^* ist Minimum von $\|Ax - b\|_2^2$ genau dann, wenn $A^T Ax^* = A^T b$ gilt.

(ii) Die Normalengleichungen $A^T Ax^* = A^T b$ haben immer eine Lösung. Ist der Rang von A gleich n, so ist diese Lösung eindeutig.

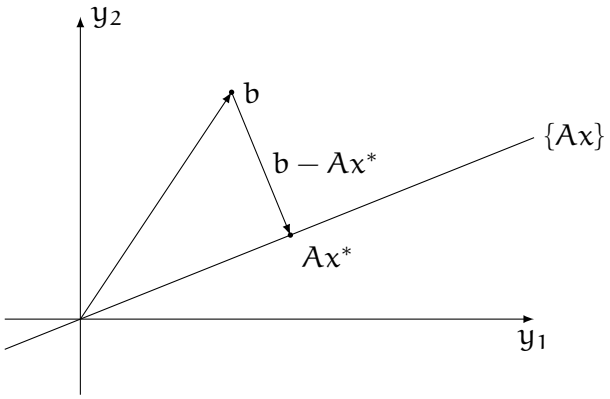


Abbildung 6.1: Die Minimierung von $\|Ax - b\|_2^2$ ist äquivalent dazu, dass $b - Ax^*$ senkrecht auf dem Bild von A (der Menge $\{Ax : x \in \mathbb{R}^n\}$) steht. (Hier: $m = 2$, $n = 1$.)

Beweis. Die Aussage (i) zeigt man völlig analog zu Lemma 5.14).

Zu (ii). Die Existenz folgt aus der Tatsache, dass $f(x) = \|Ax - b\|_2^2$ als positive quadratische Funktion stets ein Minimum hat, welches nach (i) identisch mit der Lösung der Normalengleichungen ist. Die Eindeutigkeit zeigen wir durch einen Widerspruch: Angenommen, $A^TAx = A^Tb$ hätte keine eindeutige Lösung, A^TA ist also nicht injektiv. Dann müsste es ein $x \neq 0$ geben, für das $A^TAx = 0$ gälte. Für dieses x würde dann aber auch gelten

$$0 = x^TA^TAx = \|Ax\|_2^2,$$

was nur für $Ax = 0$ möglich ist. Hat A aber vollen Spaltenrang (d. h. $\mathrm{Rang}(A) = n$), so ist A injektiv, d. h. es gilt $Ax \neq 0$ für alle $x \neq 0$. Damit hat unsere Annahme zu einem Widerspruch geführt, weshalb $A^TA \in \mathbb{R}^{n\times n}$ also invertierbar sein muss. □

Insbesondere folgt aus dem Beweis, dass bei vollem Spaltenrang $x^TA^TAx > 0$ für $x \neq 0$ gelten muss, A^TA also (symmetrisch und) positiv definit ist.

Wir betrachten jetzt die Kondition des linearen Ausgleichsproblems. Dazu müssen wir uns zuerst darauf einigen, was mit der Konditionszahl einer nichtquadratischen Matrix gemeint sein soll: Diese war ja über die Norm der inversen Matrix definiert, welche nur für quadratische Matrizen existiert. Wir haben allerdings in Lemma 4.15 eine äquivalente Darstellung kennengelernt, die auch für nichtquadratische Matrizen einen Sinn ergibt. Für eine Matrix $A \in \mathbb{R}^{m\times n}$ definieren wir die Konditionszahl deshalb als

$$\kappa_2(A) = \frac{\max\|Ax\|_2}{\min\|Ax\|_2},$$

wobei das Maximum und Minimum über alle $x \in \mathbb{R}^n$ mit $\|x\|_2 = 1$ genommen wird.

Damit lässt sich das folgende Resultat über die Kondition des linearen Ausgleichproblems beweisen.

Satz 6.3. *Sei* $A \in \mathbb{R}^{m\times n}$, $b, \tilde{b} \in \mathbb{R}^m$. *Für das Minimum* x^* *von* $\|Ax - b\|_2^2$ *und das Minimum* $\tilde{x}$ *von* $\|Ax - \tilde{b}\|_2^2$ *gilt:*

$$\frac{\|x^* - \tilde{x}\|_2}{\|x^*\|_2} \leqslant \kappa_2(A) \frac{\|b\|_2}{\|Ax^*\|_2} \frac{\|\tilde{b} - b\|_2}{\|b\|_2}$$

Im Unterschied zur Lösung eines linearen Gleichungssystem wird für das lineare Ausgleichsproblem im Allgemeinen nicht $\|Ax^*\| = \|b\|$ gelten. Die Kondition $\kappa_2(A)\frac{\|b\|_2}{\|Ax^*\|_2}$ hängt also wesentlich von der Größe des Residuums $b - Ax^*$ ab: Ist dieses klein (gilt also $\|Ax^*\|_2 \approx \|b\|_2$), so verhält sich das Ausgleichsproblem wie ein lineares Gleichungssystem. Ist aber das Residuum groß, so ist $\|Ax^*\|_2$ klein obwohl $\|b\|_2$ dies nicht ist, und das Problem wird schlechter konditioniert.

Aus Satz 6.2 folgt, dass für eine Matrix A mit vollem Spaltenrang die Matrix A^TA invertierbar und sogar symmetrisch und positiv definit ist, so dass die Normalengleichungen stabil mit dem Cholesky- oder CG-Verfahren gelöst werden können. Dieses Vorgehen hat für größere Matrizen allerdings zwei entscheidende Nachteile:

(i) Um die Cholesky-Zerlegung auf die Matrix A^TA anzuwenden, muss diese erst durch eine Matrix-Matrix-Multiplikation berechnet werden. Das ist jedoch eine aufwändige Operation (Komplexität $\mathcal{O}(mn^2)$).

(ii) Es gilt $\kappa_2(A^TA) = \kappa_2(A)^2$; die Normalengleichungen werden also eine deutlich schlechtere Kondition haben, was sich negativ auf die Konvergenzgeschwindigkeit des CG-Verfahrens auswirkt.

6.2 QR-ZERLEGUNG

Eine für das Ausgleichsproblem vorteilhaftere Zerlegung erhalten wir, indem wir fordern, dass einer der beiden Matrixfaktoren orthogonal ist:

Definition 6.4. Eine Matrix $Q \in \mathbb{R}^{n\times n}$ heißt orthogonal, falls $Q^T = Q^{-1}$ gilt.

Lemma 6.5. *Sei* $Q \in \mathbb{R}^{n\times n}$ *orthogonal. Dann gilt:*

(i) $\|Qx\|_2 = \|x\|_2$,

(ii) $\kappa_2(Q) = 1$,

(iii) ist $\tilde{Q} \in \mathbb{R}^{n\times n}$ *orthogonal, so auch* $\tilde{Q}Q$.

Wir suchen also eine Zerlegung $A = QR$, wobei $Q \in \mathbb{R}^{m\times m}$ eine orthogonale Matrix und $R \in \mathbb{R}^{m\times n}$ eine obere Dreiecksmatrix ist. Hat A vollen Spaltenrang, so hat R die Form

$$R = \begin{pmatrix} R_1 \\ 0 \end{pmatrix}, \tag{6.1}$$

wobei $R_1 \in \mathbb{R}^{n\times n}$ eine obere Dreiecksmatrix ist und 0 für einen $(m-n)\times n$-Block mit Null-Einträgen steht. Nehmen wir an, wir haben solch eine Zerlegung gefunden. Partitionieren wir auch die rechte Seite als

$$Q^T b = \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} \tag{6.2}$$

mit $c_1 \in \mathbb{R}^n$ und $c_2 \in \mathbb{R}^{m-n}$, dann erhalten wir

$$\begin{aligned} \|Ax - b\|_2^2 = \|QRx - b\|_2^2 = \|Q(Rx - Q^T b)\|_2^2 = \|Rx - Q^T b\|_2^2 \\ = \|R_1 x - c_1\|_2^2 + \|c_2\|_2^2 \end{aligned} \tag{6.3}$$

wegen der Orthogonalität von Q und Lemma 6.5 (i). Wenn A vollen Spaltenrang hat, dann ist auch der Rang von R_1 gleich n; die Gleichung $R_1 x = c_1$ hat also eine eindeutige Lösung. Da der zweite Term auf der rechten Seite von (6.3) nicht von x abhängt, ist diese Lösung auch das eindeutige Minimum des Ausgleichsproblems. Wir halten fest:

Satz 6.6. *Sei $A \in \mathbb{R}^{m\times n}$ mit Rang$(A) = n$, $Q \in \mathbb{R}^{m\times m}$ orthogonal und $R \in \mathbb{R}^{m\times n}$ eine obere Dreiecksmatrix, mit $A = QR$. Dann ist das Minimum x^* von $\|Ax - b\|_2^2$ die Lösung von $R_1 x = c_1$, und für das Residuum gilt $\|Ax^* - b\|_2^2 = \|c_2\|_2^2$ (mit R_1 aus (6.1) und c_1, c_2 aus (6.2)).*

Die gewünschte Zerlegung lässt sich analog zu der LR-Zerlegung konstruieren: Wir bringen die Matrix A stufenweise durch Elementarumformungen auf Dreiecksform, wobei wir allerdings nur solche Umformungen zulassen, die durch Multiplikation mit orthogonalen Matrizen repräsentiert werden können. Wir überlegen uns die Idee in $\mathbb{R}^2$. Dann ist die Grundaufgabe, die Matrix A in die folgende Form zu bringen:

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \longrightarrow \begin{pmatrix} a'_{11} & a'_{12} \\ 0 & a'_{22} \end{pmatrix} =: R.$$

Der Vektor $(a_{11}, a_{21})^T$ soll also durch eine *längenerhaltende*[1] Transformation in den Vektor $(a'_{11}, 0)^T$ überführt werden. Dies lässt sich zum Beispiel durch eine Rotation bewerkstelligen (siehe Abbildung 6.2), eine Alternative wäre die Spiegelung an der Winkelhalbierenden. Der erste Ansatz führt auf die *Givens-Rotationen*, der zweite auf die *Householder-Reflexionen*; wir beschränken uns hier auf Erstere.

Algebraisch wird eine Rotation um den Winkel φ im Uhrzeigersinn durch Multiplikation mit der Matrix

$$\begin{pmatrix} \cos(\varphi) & \sin(\varphi) \\ -\sin(\varphi) & \cos(\varphi) \end{pmatrix}$$

[1] vergleiche Lemma 6.5 (i)

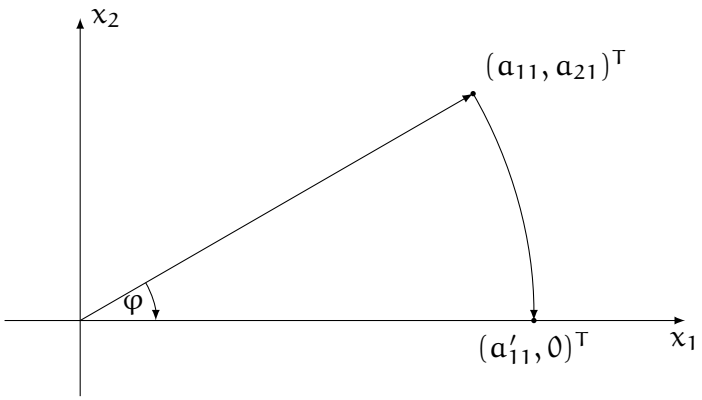


Abbildung 6.2: Der Vektor $(a_{11}, a_{21})^T$ soll durch Rotation im Uhrzeigersinn in den Vektor $(a'_{11}, 0)^T$ transformiert werden.

ausgedrückt (*Jacobi-* oder *Drehmatrix*). Es läßt sich leicht nachrechnen, dass diese Matrix immer orthogonal ist. Da der Winkel selber nicht von Interesse ist, reicht es, folgende Aufgabe zu lösen:

Finde für gegebene a, b ein Paar c, s so, dass

(i) $\begin{pmatrix} c & s \\ -s & c \end{pmatrix}$ orthogonal und

(ii) $\begin{pmatrix} c & s \\ -s & c \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} r \\ 0 \end{pmatrix}$ ist.

Dies führt auf die Bedingungen

$$\begin{cases} c^2 + s^2 = 1, \\ a^2 + b^2 = r^2, \\ -sa + cb = 0, \end{cases}$$

die erfüllt sind für

$$r = \pm\sqrt{a^2 + b^2}, \qquad c = \frac{a}{r}, \qquad s = \frac{b}{r}.$$

Die Rotationsmatrizen sind bis auf das Vorzeichen von r eindeutig (da sowohl eine Drehung um φ im Uhrzeigersinn als auch eine Drehung um $\pi - \varphi$ gegen den Uhrzeigersinn den Vektor in die $(x_1, 0)$-Ebene rotiert). Üblicherweise wird das Vorzeichen so gesetzt, dass $s \geqslant 0$ gilt.

Im $\mathbb{R}^m$ wird die zweidimensionale Rotation so in eine Identitätsmatrix eingebettet, dass die Rotation in der (i, k)-Ebene erfolgt.

Definition 6.7. Für $1 \leqslant i < k \leqslant m$ heißt $G_{ik} \in \mathbb{R}^{m\times m}$ *Givens-Rotation* (von $a \in \mathbb{R}^m$), falls für $r = \pm\sqrt{a_i^2 + a_k^2}$, $c = \frac{a_i}{r}$ und $s = \frac{a_k}{r} \geqslant 0$ gilt

$$G_{ik} = \begin{pmatrix} 1 & 0 & \dots & & & & & & \\ 0 & \ddots & & & & & & & \\ \vdots & & 1 & & & & & & \\ & & & c & \dots & \dots & \dots & s & \\ & & & \vdots & \ddots & & & \vdots & \\ & & & \vdots & & 1 & & \vdots & \\ & & & \vdots & & & \ddots & \vdots & \\ & & & -s & \dots & \dots & \dots & c & \\ & & & & & & & & 1 & & \vdots \\ & & & & & & & & & \ddots & 0 \\ & & & & & & & & \dots & 0 & 1 \end{pmatrix} \begin{matrix} \\ \\ \\ \\ \\ \\ \\ \leftarrow \text{i-te Zeile} \\ \\ \leftarrow \text{k-te Zeile} \\ \\ \\ \\ \end{matrix}, \qquad G_{ik} \begin{pmatrix} a_1 \\ \vdots \\ a_{i-1} \\ a_i \\ a_{i+1} \\ \vdots \\ a_{k-1} \\ a_k \\ a_{k+1} \\ \vdots \\ a_m \end{pmatrix} = \begin{pmatrix} a_1 \\ \vdots \\ a_{i-1} \\ r \\ a_{i+1} \\ \vdots \\ a_{k-1} \\ 0 \\ a_{k+1} \\ \vdots \\ a_m \end{pmatrix}.$$

Givens-Rotationen haben folgende wichtigen Eigenschaften:

Lemma 6.8. *Sei* $G_{ik} \in \mathbb{R}^{m\times m}$. *Dann gilt:*

(i) G_{ik} *ist orthogonal und*

(ii) Multiplikation mit G_{ik} *verändert nur die* i*-te und* k*-te Zeile.*

Die *QR-Zerlegung* von A besteht nun darin, durch Givens-Rotationen $G_{i_1k_1}, \dots, G_{i_lk_l}$ in der richtigen Reihenfolge solange Matrix-Elemente zu annullieren, bis $G_{i_lk_l} \cdots G_{i_1k_1}A$ obere Dreiecksform hat. Dabei können wir ausnutzen, dass dabei jeweils der Vektor a unverändert bleibt, wenn $(a_i, a_k) = (0, 0)$ gilt. Indem wir von unten nach oben und links nach rechts vorgehen, können wir also für jedes zu annullierende Element die zugehörige Givens-Rotation berechnen, wobei bereits erzeugte Nullelemente nicht wieder aufgefüllt werden.

BEISPIEL: $A \in \mathbb{R}^{4\times 3}$ ($*$ Eintrag ungleich Null, $\circledast$ veränderter Eintrag)

$$A = \begin{pmatrix} * & * & * \\ * & * & * \\ * & * & * \\ * & * & * \end{pmatrix} \xrightarrow{G_{34}} \begin{pmatrix} * & * & * \\ * & * & * \\ \circledast & \circledast & \circledast \\ 0 & \circledast & \circledast \end{pmatrix} \xrightarrow{G_{23}} \begin{pmatrix} * & * & * \\ \circledast & \circledast & \circledast \\ 0 & \circledast & \circledast \\ 0 & * & * \end{pmatrix} \xrightarrow{G_{12}} \begin{pmatrix} \circledast & \circledast & \circledast \\ 0 & \circledast & \circledast \\ 0 & * & * \\ 0 & * & * \end{pmatrix}$$

$$\xrightarrow{G'_{34}} \begin{pmatrix} * & * & * \\ 0 & * & * \\ 0 & \circledast & \circledast \\ 0 & 0 & \circledast \end{pmatrix} \xrightarrow{G'_{23}} \begin{pmatrix} * & * & * \\ 0 & \circledast & \circledast \\ 0 & 0 & \circledast \\ 0 & 0 & * \end{pmatrix} \xrightarrow{G''_{34}} \begin{pmatrix} * & * & * \\ 0 & * & * \\ 0 & 0 & \circledast \\ 0 & 0 & 0 \end{pmatrix} =: R$$

Am Ende ergibt sich also

$$R = (G''_{34}G'_{23}G'_{34}G_{12}G_{23}G_{34})A =: Q^TA. \tag{6.4}$$

Dabei ist Q^T und daher auch Q als Produkt orthogonaler Matrizen orthogonal.

Für den allgemeinen Fall geht man analog vor. Wir halten wir fest:

Satz 6.9. *Sei $A \in \mathbb{R}^{m \times n}$. Dann existieren eine orthogonale Matrix $Q \in \mathbb{R}^{m \times m}$ und eine obere Dreiecksmatrix $R \in \mathbb{R}^{m \times n}$, so dass $A = QR$ gilt. Mit obiger Vorzeichenkonvention sind Q, R eindeutig bestimmt.*

Zur Lösung des linearen Ausgleichsproblems wird die QR-Zerlegung nicht explizit gespeichert, sondern die einzelnen Givens-Rotationen gleich auch auf die rechte Seite b angewendet, um $Q^T b$ zu berechnen (vgl. (6.4)). Sowohl Matrix als auch rechte Seite können dabei in jedem Schritt überschrieben werden. In der Praxis werden für die Givens-Rotationen auch nicht die volle $m \times m$-Matrix aufgestellt, sondern nur die entsprechenden 2×2-Matrizen auf die betroffenen Zeilen angewendet.

Algorithmus 6.1 : Ausgleichsrechnung mit QR-Zerlegung

Input : A, b

for $j = 1, \dots, n$ **do**
  for $i = m, m-1, \dots, j+1$ **do**
    Berechne Givens-Rotation $G_{i-1,i}$ so, dass $a_{ij} = 0$
    $A \leftarrow G_{i-1,i} A$ // $Q^T A = R$
    $b \leftarrow G_{i-1,i} b$ // $Q^T b = c$
Setze $A' = (a_{ij})_{i,j=1}^{n}$, $b' = (b_i)_{i=1}^{n}$
Löse $A'x = b'$ durch Rückwärtssubstitution // $R_1 x = c_1$

Output : x

WEITERFÜHRENDE LITERATUR

G. H. Golub und C. F. Van Loan (2013). *Matrix Computations*. 4. Aufl. Johns Hopkins University Press, Baltimore, MD.

EIGENWERTE UND EIGENVEKTOREN

7

In diesem Kapitel betrachten wir das folgende Problem: Gegeben sei eine reelle[1] quadratische Matrix $A \in \mathbb{R}^{n\times n}$. Wir suchen ein $\lambda \in \mathbb{C}$ und ein $v \in \mathbb{C}^n \setminus \{0\}$, die die *Eigenwertgleichung*

$$Av = \lambda v$$

erfüllen. Die Zahl λ heißt *Eigenwert*, der Vektor v *Eigenvektor* zum Eigenwert λ.

Dies ist ein sehr häufiges Problem in Physik und Technik; Schwingungsvorgänge lassen sich zum Beispiel als Eigenwertprobleme auffassen (wobei die Eigenwerte zum Beispiel Resonanzfrequenzen entsprechen). Zwei weitere Beispiele:

Beispiel 7.1. (i) Der für die Konditionsbestimmung wichtige *Spektralradius* einer Matrix wird durch die Bestimmung des betragsgrößten Eigenwerts berechnet.

(ii) Googles Erfolg als Suchmaschine beruht zu einem großen Teil in dem PageRank-Algorithmus, der dafür sorgt, dass aus der – möglicherweise mehrere Millionen Seiten umfassenden – Liste der Treffer zu einem Suchbegriff die (vermutlich) relevantesten Seiten zuerst erscheinen. Die Grundidee ist dabei die folgende: Eine Seite ist wichtig, wenn sie von vielen wichtigen Seiten verlinkt wird. Man ordnet also jeder Seite i eine "Wichtigkeit" x_i zu. Diese Zahl "verteilt" die Seite nun gleichmäßig auf alle Seiten, die von ihr verlinkt werden (sind also 4 Seiten verlinkt, so erhält jede dieser Seiten $\frac{x_i}{4}$). Die Summe dieser Beiträge von allen Seiten, die auf eine Seite j linken, ergibt die Wichtigkeit x_j. Man definiert nun eine Verknüpfungsmatrix A durch $a_{ij} = \frac{1}{m}$, falls Seite j einen von insgesamt m Links auf Seite i hat, und $a_{ij} = 0$, falls Seite j nicht auf Seite i linkt. Damit lässt sich der Vektor x der Wichtigkeiten als Lösung der Gleichung $Ax = x$ berechnen.

Von der inzwischen entwickelten Vielzahl von Verfahren betrachten wir hier nur eine Auswahl der einfachsten. Vorher sollen aber einige wichtige Grundlagen zusammengetragen werden.

[1] Die hier vorgestellten Methoden lassen sich auch auf komplexe Matrizen übertragen.

7.1 ALGEBRAISCHE GRUNDLAGEN

Wir beginnen mit der Charakterisierung von Eigenwerten als Nullstellen des *charakteristischen Polynoms*.

Lemma 7.2. *Die Zahl $\lambda \in \mathbb{C}$ ist Eigenwert von A genau dann, wenn gilt*

$$\det(A - \lambda I) = 0.$$

Diese Charakterisierung ist nur von theoretischem Nutzen, da die Nullstellenberechnung für Polynome sehr schlecht konditioniert ist (die Nullstellen hängen nicht stetig von den Koeffizienten ab). Aus dem Fundamentalsatz der Algebra folgt aber sofort, dass eine Matrix $A \in \mathbb{R}^{n\times n}$ genau n Eigenwerte hat, die komplex sein können und nicht unbedingt verschieden sein müssen. Die Menge aller paarweise verschiedenen Eigenwerte

$$\sigma(A) := \{\lambda \in \mathbb{C} : \det(A - \lambda I) = 0\}$$

heißt *Spektrum* von A. In manchen Fällen kann man die Eigenwerte explizit angeben:

Beispiel 7.3. Sei $A \in \mathbb{R}^{n\times n}$ mit Eigenwerten λ_i, $1 \leqslant i \leqslant n$.

(i) Ist A obere (oder untere) Dreiecksmatrix, so ist $\lambda_i = a_{ii}$. (Dies gilt insbesondere für Diagonalmatrizen.)

(ii) Ist A regulär, so hat A^{-1} die Eigenwerte λ_i^{-1}.

(iii) A^T hat die selben Eigenwerte wie A.

Gilt $B = T^{-1}AT$ für eine invertierbare Matrix, so heißen A und B *ähnlich*. Zwei ähnliche Matrizen haben das selbe Spektrum.

Lemma 7.4. *Sei $A \in \mathbb{R}^{n\times n}$. Dann gilt für jede nichtsinguläre Matrix $T \in \mathbb{R}^{n\times n}$*

$$\sigma(A) = \sigma(T^{-1}AT).$$

Eine Matrix, für die ein T existiert, so dass $T^{-1}AT$ eine Diagonalmatrix ist, heißt *diagonalisierbar*.

Satz 7.5 (Schur-Faktorisierung)**.** *Sei $A \in \mathbb{R}^{n\times n}$. Dann existiert eine orthogonale Matrix $Q \in \mathbb{R}^{n\times n}$, so dass gilt*

$$Q^T A Q = \begin{pmatrix} R_{11} & & * \\ & \ddots & \\ 0 & & R_{mm} \end{pmatrix} =: R, \tag{7.1}$$

wobei alle R_{ii} reell und entweder 1×1- oder 2×2-Matrizen sind. (In letzterem Fall hat R_{ii} ein Paar komplex konjugierter Eigenwerte.) Weiterhin gilt

$$\sigma(A) = \bigcup_{i=1}^{m} \sigma(R_{ii}).$$

Folgerung 7.6. *Ist $A \in \mathbb{R}^{n\times n}$ symmetrisch, so ist R in (7.1) eine Diagonalmatrix. A hat also nur reelle Eigenwerte, und die Spalten der Matrix Q bilden eine orthogonale Basis von $\mathbb{R}^n$ aus Eigenvektoren von A.*

Wir schließen mit einer einfachen, geometrischen, Abschätzung für die Eigenwerte von A.

Satz 7.7 (Gerschgorin-Kreise)**.** *Sei $A \in \mathbb{R}^{n\times n}$. Dann gilt*

$$\sigma(A) \subset \bigcup_{i=1}^{n} K_i, \qquad K_i := \left\{ z \in \mathbb{C} : |z - a_{ii}| \leqslant \sum_{j\neq i} |a_{ij}| \right\}.$$

Beweis. Sei $\lambda \in \sigma(A)$, und v der zu λ gehörende Eigenvektor. Wähle i so, dass $v_i = \|v\|_\infty$, und betrachte die i-te Zeile von $Av = \lambda v$, d. h.

$$\sum_{j=1}^{n} a_{ij} v_j = \lambda v_i.$$

Bringen wir nur v_i auf die linke Seite und bilden den Betrag, erhalten wir:

$$|a_{ii} - \lambda||v_i| \leqslant \sum_{j\neq i} |a_{ij}||v_j| \leqslant \sum_{j\neq i} |a_{ij}||v_i|.$$

Da Eigenvektoren von Null verschieden sind, können wir durch $|v_i| \neq 0$ dividieren, und erhalten die Aussage. □

Bemerkung. Die Eigenwerte von A liegen also in der Vereinigung der *Gerschgorin-Kreise* K_i. Beachten Sie, dass nicht gesagt ist, dass in jedem Kreis ein Eigenwert liegt! Man kann aber zeigen, dass die Vereinigung von k Gerschgorin-Kreisen, die die restlichen Kreise nicht schneidet, genau k Eigenwerte enthält.

Zuletzt betrachten wir die Kondition des Eigenwertproblems:

Satz 7.8. *Sei $A \in \mathbb{R}^{n\times n}$ diagonalisierbar für $T \in \mathbb{R}^{n\times n}$ (d. h. $T^{-1}AT$ ist diagonal) mit Eigenwerten $\lambda_1, \ldots \lambda_n$ und sei $\delta A \in \mathbb{R}^{n\times n}$. Für jeden Eigenwert μ der gestörten Matrix $A + \delta A$ gilt*

$$\min_{1\leqslant i\leqslant n} |\lambda_i - \mu| \leqslant \kappa_2(T)\|\delta A\|_2.$$

Der Abstand eines gestörten zum nächstgelegenen ungestörten Eigenwert ist also nicht durch die Konditionszahl der Matrix A sondern durch die der Eigenvektorbasis T bestimmt. Für symmetrische Matrizen ist diese Basis orthogonal (Folgerung 7.6) und daher $\kappa_2(T) = 1$. Das symmetrische Eigenwertproblem ist also immer gut konditioniert.

Da es für Polynome vom Grad höher als vier keine geschlossene Formel für die Nullstellen geben kann, ist auch kein direktes Verfahren für die Eigenwertberechnung möglich. Alle Verfahren sind also iterativ.

7.2 VEKTORITERATION

Die *Vektoriteration* (oder *Potenzmethode*) ist ein einfaches Verfahren zur Bestimmung des betragsgrößten[2] Eigenwerts λ_1 und des dazugehörigen Eigenvektors v_1. Ausgangspunkt ist die folgende Beobachtung: Unter gewissen Voraussetzungen konvergiert für gegebenes $x \in \mathbb{R}^n \setminus \{0\}$ und eine beliebige Norm

$$\frac{A^k x}{\|A^k x\|} \to c v_1, \text{ für } k \to \infty, c \in \mathbb{R}. \tag{7.2}$$

Um dies zu zeigen, nehmen wir an, dass $\lambda_1 \in \mathbb{R}$ ein einfacher Eigenwert ist, d. h. es gilt

$$|\lambda_1| > |\lambda_2| \geqslant \cdots \geqslant |\lambda_n|. \tag{7.3}$$

(Da komplexe Eigenwerte reeller Matrizen immer in komplex konjugierten Paaren auftreten, muss dann auch $\lambda_1 \in \mathbb{R}$ sein.) Außerdem nehmen wir der Einfachheit an, dass A diagonalisierbar ist. Dann hat $\mathbb{R}^n$ eine Basis $\{v_1, \ldots, v_n\}$ aus Eigenvektoren von A und wir können daher jedes $x \in \mathbb{R}^n$ schreiben als

$$x = \xi_1 v_1 + \xi_2 v_2 + \cdots + \xi_n v_n.$$

Damit ist

$$\begin{aligned} A^k x &= \xi_1 A^k v_1 + \xi_2 A^k v_2 + \cdots + \xi_n A^k v_n \\ &= \xi_1 \lambda_1^k v_1 + \xi_2 \lambda_2^k v_2 + \cdots + \xi_n \lambda_n^k v_n \\ &= \lambda_1^k \left[\xi_1 v_1 + \xi_2 \left(\frac{\lambda_2}{\lambda_1} \right)^k v_2 + \cdots + \xi_n \left(\frac{\lambda_n}{\lambda_1} \right)^k v_n \right]. \end{aligned}$$

Wegen (7.3) gilt

$$1 > \frac{|\lambda_2|}{|\lambda_1|} \geqslant \cdots \geqslant \frac{|\lambda_n|}{|\lambda_1|},$$

die entsprechenden Terme konvergieren also für $k \to \infty$ gegen Null. Falls nun $\xi_1 \neq 0$ ist, so gilt für $\lambda_1 > 0$

$$\frac{A^k x}{\|A^k x\|} = \frac{\lambda_1^k \left[\xi_1 v_1 + \mathcal{O}\left(\left| \frac{\lambda_2}{\lambda_1} \right|^k \right) \right]}{|\lambda_1^k| \left\| \xi_1 v_1 + \mathcal{O}\left(\left| \frac{\lambda_2}{\lambda_1} \right|^k \right) \right\|} \xrightarrow[k \to \infty]{} \tilde{v}_1 := \frac{v_1}{\|v_1\|}. \tag{7.4}$$

(Ist $\lambda_1 < 0$, so existieren zwei Teilfolgen, von denen eine gegen $\tilde{v}_1$, die andere gegen $-\tilde{v}_1$ konvergiert.)

[2] Im folgenden gehen wir immer davon aus, dass die Eigenwerte des Betrags nach abfallend geordnet sind.

Die Vektoriteration liefert also für passendes[3] $x^0 \in \mathbb{R}^n$ eine Approximation $x^{(k)} = \frac{A^k x^{(0)}}{\|A^k x^{(0)}\|}$ von v_1. Um daraus eine Näherung des zugehörigen Eigenwerts λ_1 zu bestimmen, kann man folgenden Ansatz nutzen: zu gegebenem $x^{(k)}$ suchen wir diejenige Zahl λ, die das Residuum der Eigenwertgleichung

$$\|x^{(k)}\lambda - Ax^{(k)}\|_2^2$$

minimiert. Dies ist ein lineares Ausgleichsproblem für λ mit der "Matrix" $x^{(k)} \in \mathbb{R}^{n\times 1}$ und der rechten Seite $Ax^{(k)} \in \mathbb{R}^n$. Die Normalengleichungen dazu sind

$$x^{(k)^T} x^{(k)} \lambda = x^{(k)^T} A x^{(k)}.$$

Da $x^{(k)}$ nach Konstruktion die Norm $\|x^{(k)}\| = 1$ hat, ist $x^{(k)} \neq 0$. Wir können daher nach λ auflösen und erhalten den *Rayleigh-Quotienten*[4]

$$\lambda^{(k)} = \frac{x^{(k)^T} A x^{(k)}}{x^{(k)^T} x^{(k)}}.$$

Wählt man auch in (7.2) die Euklidische Norm, ergibt das die *Vektor-* oder *von-Mises-Iteration*:[5]

Algorithmus 7.1 : Vektoriteration

Input : $A, x^0 \neq 0, \varepsilon$

1 **repeat**
2 $\quad k \leftarrow k + 1$
3 $\quad w \leftarrow Ax^{k-1}$
4 $\quad x^k \leftarrow w/\|w\|_2$
5 $\quad \lambda^k \leftarrow (x^k)^T A x^k$
6 **until** $\|\lambda^k x^k - Ax^k\|_2 < \varepsilon$

Output : x^k, λ^k

Zur Konstruktion haben wir nur die lineare Unabhängigkeit der Eigenvektoren verwendet. Für die Konvergenz des Verfahrens folgt also aus den obigen Überlegungen und der Stetigkeit des Skalarprodukts:

[3] Ein zufällig gewählter Vektor ungleich 0 wird in der Regel eine Komponente in v_1-Richtung haben. Ist dies nicht der Fall, erzeugen Rundungsfehler während der Iteration einen solchen Anteil.

[4] John William Strutt, dritter Baron Rayleigh (1842-1919), ist in der zweiten Hälfte des 19. Jahrhunderts durch seine Beiträge zu fast allen Bereichen der Physik, insbesondere zu Schwingungsphänomenen, berühmt geworden.

[5] Vorgeschlagen 1929 durch Richard von Mises, der in der ersten Hälfte des 20. Jahrhunderts zur angewandte Mathematik (etwa im Bereich der Flugzeugkonstruktion) viel beigetragen hat.

Satz 7.9. *Hat $A \in \mathbb{R}^{n\times n}$ genau n linear unabhängige Eigenvektoren, gilt $|\lambda_1| > |\lambda_2|$ und $\langle x^{(0)}, v_1\rangle \neq 0$, so folgt für die Iterierten der Vektoriteration:*

$$\|\pm x^{(k)} - v_1\|_2 = \mathcal{O}\left(\left|\frac{\lambda_2}{\lambda_1}\right|^k\right),$$
$$\|\lambda^{(k)} - \lambda_1\|_2 = \mathcal{O}\left(\left|\frac{\lambda_2}{\lambda_1}\right|^k\right).$$

Ist A symmetrisch, gilt sogar

$$\|\lambda^{(k)} - \lambda_1\|_2 = \mathcal{O}\left(\left|\frac{\lambda_2}{\lambda_1}\right|^{2k}\right).$$

7.3 INVERSE VEKTORITERATION

Die Vektoriteration kann auch zur Berechnung weiterer Eigenwerte und Eigenvektoren verwendet werden. Ist nämlich $\lambda \in \sigma(A)$ und $\mu \notin \sigma(A)$, so gilt

$$Ax = \lambda x \Leftrightarrow (A-\mu I)x = (\lambda-\mu)x \Leftrightarrow (A-\mu I)^{-1}x = (\lambda-\mu)^{-1}x.$$

Ist $|\lambda-\mu|$ also klein genug, dann ist $|\frac{1}{\lambda-\mu}|$ der betragsgrößte Eigenwert von $(A-\mu I)^{-1}$. Kennen wir also eine gute Schätzung von λ, so können wir die Vektoriteration zur Berechnung des zugehörigen Eigenvektors (und einer verbesserten Näherung des Eigenwerts) einsetzen. Diesen Ansatz nennt man *inverse Vektoriteration mit Spektralverschiebung*.[6] Natürlich wird die Matrix $(A-\mu I)^{-1}$ dabei nicht explizit gebildet, sondern das zugehörige lineare Gleichungssystem gelöst.

Algorithmus 7.2 : Inverse Vektoriteration

Input : $\mu, A, x^0 \neq 0, \varepsilon$

Berechne LR-Faktorisierung von $A-\mu I$
repeat
  $k \leftarrow k+1$
  Löse $(A-\mu I)w = x^{k-1}$ durch Vorwärts-/Rückwärtssubstitution
  $x^k \leftarrow w/\|w\|_2$
  $\lambda^k \leftarrow \left(w^T x^{k-1}\right)^{-1} + \mu$
until $\|\lambda^k x^k - Ax^k\|_2 < \varepsilon$

Output : x^k, λ^k

[6] Helmut Wielandt – eigentlich ein reiner Mathematiker – entwickelte diese Methode 1944 während seiner Arbeit an der Aerodynamischen Versuchsanstalt Göttingen.

Dabei haben wir in der letzten Zeile verwendet, dass der Rayleigh-Quotient

$$(x^k)^T(A-\mu I)^{-1}x^k = (x^k)^T w =: (\lambda^k - \mu)^{-1}$$

eine Näherung für den betragsgrößten Eigenwert $(\lambda-\mu)^{-1}$ darstellt.

Die Konvergenz der Iteration hängt vom Abstand von μ zum nächsten benachbarten Eigenwert ab. Je näher μ am gewünschten Eigenwert λ_i, und je weiter weg vom nächstgelegenen Eigenwert λ_j, desto schneller konvergiert die Iteration. Präziser wird der Fehler in jeder Iteration um den Faktor

$$\max_{j\neq i} \frac{|\lambda_i - \mu|}{|\lambda_j - \mu|}$$

reduziert. Die Konvergenzgeschwindigkeit kann wesentlich gesteigert werden, indem in jedem Iterationsschritt der Rayleigh-Quotient als Verschiebung $\mu = \lambda^{(k-1)}$ verwendet wird (*Rayleigh-Iteration*). Allerdings steigt dadurch der Rechenaufwand stark an, da die LR-Zerlegung von $(A-\lambda^{(k-1)}I)$ in jedem Schritt neu durchgeführt werden muss. Eine geeignete Wahl von μ ist in der Praxis schwierig; jedoch können die Gerschgorin-Kreise aus Satz 7.7 einen ersten Ansatz liefern.

Um weitere Eigenwerte und Eigenvektoren zu bestimmen, kann man eine *Deflation* durchführen. Sind $\lambda_1, \dots, \lambda_n$ die Eigenwerte von A mit zugehörigen orthogonalen Eigenvektoren $v_1, \dots, v_n$, so hat die Matrix

$$\tilde{A} := A - \lambda_1 \frac{v_1 v_1^T}{\|v_1\|_2^2}$$

die Eigenwerte $0, \lambda_2, \dots, \lambda_n$ mit zugehörigen Eigenvektoren $v_1, v_2, \dots, v_n$. Man kann also, beginnend mit dem betragsgrößten (oder betragskleinsten) sämtliche Eigenwerte und Eigenvektoren durch Vektoriteration (bzw. inverse Iteration) der Reihe nach berechnen. In der Praxis ist dieses Verfahren jedoch zu aufwendig, falls mehr als die ersten paar Eigenwerte gesucht sind.

7.4 DAS QR-VERFAHREN

Ein sehr elegantes Verfahren, sämtliche Eigenwerte und Eigenvektoren einer Matrix zu berechnen, ist das *QR-Verfahren*.[7] Dabei wird im Wesentlichen mit Hilfe der Vektoriteration und

[7] In der heute verwendeten Form geht das Verfahren auf die unabhängigen Veröffentlichungen von Wera Nikolajewna Kublanowskaja (1961) und John G. F. Francis (1961/62, inklusive impliziter Shifts) zurück. Der Kerngedanke, eine Matrixzerlegung für die Eigenwertberechnung einzusetzen, stammt von Heinz Rutishauser, der 1954 eine Rekursion für die Eigenwertberechnung basierend auf Nullstellen charakteristischer Polynome (den *qd-Algorithmus*) herleitete, und erkannte, dass diese in Matrixschreibweise genau dem *LR-Verfahren* $L^{(k)}R^{(k)} = A^{(k)}$, $A^{(k+1)} = R^{(k)}L^{(k)}$ entspricht. Die Details sind nachzulesen in M. H. Gutknecht und B. N. Parlett (2011). *From qd to LR, or, how were the qd and LR algorithms discovered?* IMA Journal of Numerical Analysis 31.3, S. 741–754. DOI: 10.1093/imanum/drq003.

der QR-Zerlegung die reelle Schur-Faktorisierung einer Matrix (näherungsweise) konstruiert. Der Algorithmus selber ist äußerst einfach:

Algorithmus 7.3 : QR-Verfahren

Input : A, m^*

1 $A^{(0)} \leftarrow A$
2 **for** $m = 1, \ldots, m^*$ **do**
3 $\quad Q^{(m)}, R^{(m)} \leftarrow A^{(m-1)}$ `// QR-Zerlegung von A`
4 $\quad A^{(m)} \leftarrow R^{(m)} Q^{(m)}$

Output : $A^{(m)}, Q^{(m)}$

Die Iterierten $A^{(k)}$ im QR-Verfahren konvergieren (unter bestimmten Voraussetzungen) gegen eine obere Dreiecksmatrix, deren Diagonalelemente die Eigenwerte von A sind, und die $Q^{(k)}$ konvergieren gegen eine orthogonale Matrix, deren Spalten die zugehörigen Eigenvektoren sind.

Wir nähern uns der Konstruktion dieses Verfahrens in mehreren Schritten. Wir nehmen dabei der Bequemlichkeit halber an, dass alle Eigenwerte von A einfach sind, d. h.

$$|\lambda_1| > |\lambda_2| > \cdots > |\lambda_n|, \tag{7.5}$$

die zugehörigen Eigenvektoren $v_1, v_2, \ldots, v_n$ also eine Basis des $\mathbb{R}^n$ bilden.

SCHRITT 1: (*Unterraumiteration*)

Die Vektoriteration liefert für ein x mit $x^T v_1 \neq 0$ eine Folge $x^{(m)} = A^m x$, die für $m \to \infty$ gegen $c v_1$ für $c \in \mathbb{R}$ konvergiert, d. h. ein Element aus dem von v_1 aufgespannten Unterraum. Ein naheliegender Ansatz für die Berechnung mehrerer Eigenvektoren ist also, die Vektoriteration für einen Unterraum mit einer größeren Dimension $1 \leqslant k \leqslant n$ durchzuführen. Dafür wählen wir k linear unabhängige Vektoren $x_1, \ldots, x_k$ und setzen

$$\begin{aligned} S_k &= \operatorname{span}\{x_1, \ldots, x_k\}, \\ T_k &= \operatorname{span}\{v_1, \ldots, v_k\}, \\ U_k &= \operatorname{span}\{v_{k+1}, \ldots, v_n\}. \end{aligned}$$

Mit $A^m S_k = \{A^m x : x \in S_k\}$ kann man dann zeigen, dass

$$A^m S_k \longrightarrow T_k \quad \text{für } m \to \infty,$$

falls $S_k \cap U_k = \{0\}$ gilt. Sei dafür $x \in S_k$ beliebig, und schreibe

$$x = (\xi_1 v_1 + \cdots + \xi_k v_k) + (\xi_{k+1} v_{k+1} + \cdots + \xi_n v_n).$$

Dann gilt

$$\begin{aligned} A^m x &= (\xi_1 \lambda_1^m v_1 + \cdots + \xi_k \lambda_k^m v_k) + \left(\xi_{k+1} \lambda_{k+1}^m v_{k+1} + \cdots + \xi_n \lambda_n^m v_n\right) \\ &= \lambda_k^m \left[\left(\xi_1 \left(\frac{\lambda_1}{\lambda_k} \right)^m v_1 + \cdots + \xi_k v_k \right) \right. \\ &\qquad \left. + \left(\xi_{k+1} \left(\frac{\lambda_{k+1}}{\lambda_k} \right)^m v_{k+1} + \cdots + \xi_n \left(\frac{\lambda_n}{\lambda_k} \right)^m v_n \right) \right]. \end{aligned}$$

Wegen (7.5) sind sämtliche Brüche in der ersten Klammer größer als 1, und sämtliche Brüche in der zweiten Klammer kleiner als 1. Für $m \to \infty$ konvergiert die zweite Klammer also gegen 0. Es gilt also

$$A^m x \longrightarrow v \in T_k.$$

Da S_k und T_k die gleiche Dimension haben und wegen $x \notin U_k$ die erste Klammer nicht verschwindet, muss also $A^m S_k$ gegen T_k konvergieren.

SCHRITT 2: (*Basisiteration*)

Für die praktische Durchführung der Unterraumiteration wählt man eine Basis $\{x_1, \ldots, x_k\}$ und berechnet $\{A^m x_1, \ldots, A^m x_k\}$. Allerdings gilt wegen (7.4), dass $A^m x_i$ für (fast) alle x_i gegen v_1 konvergiert;[8] diese Basis wird also mit fortschreitender Iteration immer schlechter konditioniert sein. Außerdem ist eine Normalisierung nötig, damit die Iterierten beschränkt bleiben. Deshalb wird in jedem Schritt eine neue Orthonormalbasis $\{q_1^{(m)}, \ldots, q_k^{(m)}\}$ von $A^m S_k$ bestimmt. Der Name des QR-Verfahrens stammt nun daher, dass dies durch eine QR-Zerlegung geschehen kann. Seien $a_1, \ldots, a_n$ die Spalten von A, und $q_1, \ldots, q_n$ die (orthonormalen) Spalten von Q, so folgt nämlich aus $A = QR$:

$$\begin{aligned} a_1 &= q_1 r_{11} && \Rightarrow && \operatorname{span}\{a_1\} = \operatorname{span}\{q_1\}, \\ a_2 &= q_1 r_{12} + q_2 r_{22} && \Rightarrow && \operatorname{span}\{a_1, a_2\} = \operatorname{span}\{q_1, q_2\}, \\ &\vdots && && \vdots \\ a_n &= q_n r_{1n} + \cdots + q_n r_{nn} && \Rightarrow && \operatorname{span}\{a_1, a_2, \ldots, a_n\} = \operatorname{span}\{q_1, q_2, \ldots, q_n\}. \end{aligned}$$

Um alle Eigenvektoren zu bestimmen, führen wir also jeweils einen Basisiterationsschritt auf den n orthonormalen Vektoren $q_1^{(0)}, \ldots, q_n^{(0)}$ durch, und berechnen dann durch QR-Zerlegung eine neue orthonormale Basis. Dann gilt für alle $1 \leqslant k \leqslant n$:

$$S_k^{(m)} := \operatorname{span}\left\{ q_1^{(m)}, \ldots, q_k^{(m)} \right\} = A^m S_k \longrightarrow T_k.$$

Da die Spalten von Q orthogonal sind, ist $q_k^{(m)} \in A^m S_k$ orthogonal zu $A^m S_{k-1}$ für alle $2 \leqslant k \leqslant n$. Somit konvergiert $q_1^{(m)} \in A^m S_1$ gegen den (einzigen) Basisvektor v_1 von T_1;

[8] wenn auch nicht mit der gleichen Geschwindigkeit

damit konvergiert $q_2^{(m)} \in A^m S_2$ gegen den zu v_1 orthogonalen Basisvektor v_2 von T_2; und so weiter. In anderen Worten, die Spalten $q_i^{(m)}$, $1 \leqslant i \leqslant n$, konvergieren simultan gegen die entsprechenden Eigenvektoren von A.

Wählen wir also eine Matrix $Q^{(0)}$ mit orthonormalen Spalten (etwa die Identität), so lassen sich die obigen Überlegungen mit Hilfe der QR-Zerlegung in Matrixform schreiben:[9]

$$\begin{cases} S^{(m)} = AQ^{(m-1)} \\ Q^{(m)}R^{(m)} = S^{(m)} \end{cases} \tag{7.6}$$

Die Spalten der Matrizen $Q^{(m)}$ konvergieren dann gegen eine orthonormale Basis aus Eigenvektoren.

SCHRITT 3: (Schur-Faktorisierung)

Die Spalten der Matrizen $Q^{(m)}$ konvergieren also gegen eine Basis aus Eigenvektoren. Dies liefert uns eine iterative Approximation der reellen Schur-Faktorisierung (7.1):

$$A^{(m)} := {Q^{(m)}}^T A Q^{(m)} \tag{7.7}$$

konvergiert gegen eine obere Dreiecksmatrix, auf deren Diagonalen die Eigenwerte von A stehen. Um das zu zeigen, betrachten wir (7.7) spaltenweise. Bezeichne die j-te Spalte $q_j^{(m)}$ von $Q^{(m)}$. Da $q_j^{(m)}$ im Unterraum $A^m S_j$ liegt, welcher gegen T_j konvergiert, können wir diesen Vektor schreiben als

$$q_j^{(m)} = \sum_{k=1}^{j} c_k v_k + \varepsilon_1^{(m)},$$

mit einer Folge $\varepsilon_1^{(m)} \to 0 \in \mathbb{R}^n$ für $m \to \infty$. Dann gilt:

$$Aq_j^{(m)} = \sum_{k=1}^{j} c_k \lambda_k v_k + \varepsilon_2^{(m)},$$

mit einer neuen Nullfolge $\varepsilon_2^{(m)} \in \mathbb{R}^n$. Da $A^m S_j$ gegen T_j konvergiert, konvergieren die $q_i^{(m)}$, $1 \leqslant i \leqslant j$, gegen eine Basis von T_j. Es lässt sich also umgekehrt jeder Eigenvektor $v_k \in T_j$ darstellen als

$$v_k = \sum_{i=1}^{j} d_{k,i}^{(m)} q_i^{(m)} + \varepsilon_3^{(m)},$$

[9] Die Vektoren $Aq_1^m, \ldots, Aq_n^m$ sind ja gerade die Spalten von AQ^m.

wobei wieder $\varepsilon_3^{(m)} \to 0$ mit $m \to \infty$ konvergiert. Zusammen gilt also

$$Aq_j^{(m)} = \sum_{k=1}^{j} \alpha_k^{(m)} q_k^{(m)} + \varepsilon^{(m)},$$

wobei wir alle auftretenden Nullfolgen in $\varepsilon^{(m)}$ gesammelt haben. Da die $q_k^{(m)}$ nach Konstruktion orthogonal sind, folgt für $i > j$:

$$(q_i^{(m)})^T A q_j^{(m)} = \sum_{k=1}^{j} \alpha_k^{(m)} \lambda_k \underbrace{(q_i^{(m)})^T q_k^{(m)}}_{=0} + (q_i^{(m)})^T \varepsilon^{(m)} = (q_i^{(m)})^T \varepsilon^{(m)} \longrightarrow 0.$$

Die Subdiagonaleinträge von $(Q^{(m)})^* A Q^{(m)}$ konvergieren also gegen Null. Weiterhin sind die Diagonalelemente genau die Rayleigh-Quotienten zu den $q^{(m)}$, die wie in der Vektoriteration gegen die entsprechenden Eigenwerte von A konvergieren.

SCHRITT 4: (QR-Algorithmus)

Der Algorithmus 7.3 konstruiert genau die in (7.7) definierten Matrizen $A^{(m)}$ für $Q^{(0)} = I$. Dies zeigt man durch Induktion nach m: Der Induktionsanfang für $m = 0$ ist klar. Bezeichne nun $\tilde{R}^{(m)}, \tilde{Q}^{(m)}, \tilde{A}^{(m)}$ die Matrizen aus dem QR-Verfahren, und $R^{(m)}, Q^{(m)}, A^{(m)}$ die entsprechenden Matrizen aus (7.6) und (7.7). Einsetzen von (7.6) für $m + 1$ in (7.7) ergibt dann

$$A^{(m)} = (Q^{(m)})^* A Q^{(m)} = \left[(Q^{(m)})^* Q^{(m+1)}\right] R^{(m+1)} =: QR.$$

Da die QR-Zerlegung (nach Festlegen des Vorzeichens) eindeutig ist, muss nach Induktionsvoraussetzung QR auch die QR-Zerlegung von $\tilde{A}^{(m)}$ sein. Es folgt also aus Schritt 4 im QR-Algorithmus 7.3:

$$\begin{aligned} \tilde{A}^{(m+1)} &= RQ = R^{(m+1)} \left[(Q^{(m)})^* Q^{(m+1)}\right] \\ &= ((Q^{(m+1)})^* A Q^{(m)})(Q^{(m)})^* Q^{(m+1)} \\ &= (Q^{(m+1)})^* A Q^{(m+1)} = A^{(m+1)}, \end{aligned}$$

wobei wir in der letzten Zeile die Gleichung $Q^{(m)} R^{(m)} = A Q^{(m-1)}$ aus (7.6), aufgelöst nach $R^{(m)}$, verwendet haben.

Wir fassen zusammen:

Satz 7.10. *Sei $A \in \mathbb{R}^{n \times n}$, gelte für die Eigenwerte von A, dass $|\lambda_1| > |\lambda_2| > \cdots > |\lambda_n|$ und für die Eigenvektoren*

$$\text{span}\{e_1, \ldots, e_k\} \cap \text{span}\{v_{k+1}, \ldots, v_n\} = \{0\} \qquad \text{für alle } 1 \leqslant k < n, \tag{7.8}$$

so konvergieren die Iterierten $A^{(m)}$ im QR-Verfahren gegen eine obere Dreiecksmatrix, deren Diagonalelemente die Eigenwerte von A sind, und die Spalten von $Q^{(m)}$ konvergieren gegen die Eigenvektoren von A.

Bedingung (7.8) entspricht dabei der Voraussetzung in der Unterraumiteration, dass $S_k \cap U_k = \{0\}$ gilt. Sie ist in der Praxis so gut wie immer erfüllt. Aus der Herleitung ist auch ersichtlich, dass die Einträge auf der Diagonalen von $A^{(m)}$ nach ihrem Betrag sortiert sind.

Bemerkung. Hat A auch Paare von komplex konjugierten Eigenwerten, so kann man zeigen, dass die $A^{(m)}$ gegen die obere Block-Dreiecksmatrix in (7.1) konvergieren. Ist A dagegen symmetrisch, konvergieren die $A^{(m)}$ gegen eine Diagonalmatrix.

Das QR-Verfahren in der Form von Algorithmus 7.3 hat folgende Nachteile:

1. In jedem Schritt ist eine QR-Zerlegung notwendig, die $\mathcal{O}(n^3)$ Operationen benötigt.
2. Die Konvergenz hängt von dem Maximum über alle $\frac{|\lambda_{j+1}|}{|\lambda_j|}$, $1 \leqslant j \leqslant n-1$, ab. Liegen also zwei Eigenwerte sehr nahe beieinander, kann die Konvergenz (für alle) beliebig langsam sein.

In der Praxis bedient man sich deshalb verschiedener Techniken, um das Verfahren zu beschleunigen. Wir betrachten hier stellvertretend jeweils die einfachsten Varianten.

TRANSFORMATION AUF HESSENBERGFORM Eine Matrix A hat *obere Hessenbergform*, falls $a_{ij} = 0$ für $i > j+1$ gilt. Im Gegensatz zu einer oberen Dreiecksmatrix dürfen hier also auch die ersten Einträge direkt unterhalb der Diagonalen von Null verschieden sein. Hat nun A obere Hessenbergform, so kann man diese durch Givens-Rotationen in $\mathcal{O}(n^2)$ Schritten auf obere Dreiecksform bringen (da ja pro Zeile nur ein Element "wegrotiert" werden muss):

$$G_{n-1,n} \cdots G_{12} A^{(m)} = R^{(m)},$$

wobei $G_{i-1,i}$ so berechnet wird, dass das Element $a_{i-1,i}$ zu 0 wird. Geht man dabei wie zuvor von links nach rechts vor, bleibt im QR-Verfahren

$$A^{(m+1)} = R^{(m)} G_{12}^T \cdots G_{n-1,n}^T$$

die obere Hessenbergform erhalten. Wir betrachten dies anhand eines Beispiels $A \in \mathbb{R}^{4\times 4}$:

$$A^{(m)} = \begin{pmatrix} * & * & * & * \\ * & * & * & * \\ 0 & * & * & * \\ 0 & 0 & * & * \end{pmatrix} \xrightarrow{G_{34}G_{23}G_{12}} \begin{pmatrix} * & * & * & * \\ 0 & * & * & * \\ 0 & 0 & * & * \\ 0 & 0 & 0 & * \end{pmatrix} \xrightarrow{G_{12}^T} \begin{pmatrix} \circledast & \circledast & * & * \\ \circledast & \circledast & * & * \\ 0 & 0 & * & * \\ 0 & 0 & 0 & * \end{pmatrix}$$
$$\xrightarrow{G_{23}^T} \begin{pmatrix} * & \circledast & \circledast & * \\ * & \circledast & \circledast & * \\ 0 & \circledast & \circledast & * \\ 0 & 0 & 0 & * \end{pmatrix} \xrightarrow{G_{34}^T} \begin{pmatrix} * & * & \circledast & \circledast \\ * & * & \circledast & \circledast \\ 0 & * & \circledast & \circledast \\ 0 & 0 & \circledast & \circledast \end{pmatrix} = A^{(m+1)}.$$

Nutzt man die Assoziativität der Matrixmultiplikation, kann man die berechneten Givens-Rotationen sofort anwenden, ohne sie zwischenzuspeichern:

$$A^{(m+1)} = (G_{n-1,n}(\cdots(G_{12}A^{(m)})G_{12}^T)\cdots)G_{n-1,n}^T$$

(Beachten Sie, dass die Givens-Rotation $G_{k,k-1}$ weiterhin so berechnet wird, dass der $(k, k-1)$-te Eintrag von $(G_{k-1,k-2}\cdots G_{21})A^{(m)}$ annulliert wird.)[10]

Um nun eine beliebige Matrix A auf obere Hessenbergform zu bringen (ohne dass sich die Eigenwerte ändern), kann man wieder Givens-Rotationen verwenden: Analog zur QR-Zerlegung eliminiert man Einträge unterhalb der Diagonalen von unten nach oben, wobei man jede Givens-Rotation sofort wieder von rechts anwendet und die nächste Rotation für die so geänderte Matrix berechnet. Ein Beispiel soll dies wieder verdeutlichen:

$$A = \begin{pmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{pmatrix} \xrightarrow{G_{34}\cdot} \begin{pmatrix} * & * & * & * \\ * & * & * & * \\ \circledast & \circledast & \circledast & \circledast \\ 0 & \circledast & \circledast & \circledast \end{pmatrix} \xrightarrow{\cdot G_{34}^T} \begin{pmatrix} * & * & \circledast & \circledast \\ * & * & \circledast & \circledast \\ * & * & \circledast & \circledast \\ 0 & * & \circledast & \circledast \end{pmatrix} \xrightarrow{G_{23}\cdot} \begin{pmatrix} * & * & * & * \\ \circledast & \circledast & \circledast & \circledast \\ 0 & \circledast & \circledast & \circledast \\ 0 & * & * & * \end{pmatrix}$$

$$\xrightarrow{\cdot G_{23}^T} \begin{pmatrix} * & \circledast & \circledast & * \\ * & \circledast & \circledast & * \\ 0 & \circledast & \circledast & * \\ 0 & \circledast & \circledast & * \end{pmatrix} \xrightarrow{G'_{34}\cdot} \begin{pmatrix} * & * & * & * \\ * & * & * & * \\ 0 & \circledast & \circledast & \circledast \\ 0 & 0 & \circledast & \circledast \end{pmatrix} \xrightarrow{\cdot (G'_{34})^T} \begin{pmatrix} * & * & \circledast & \circledast \\ * & * & \circledast & \circledast \\ 0 & * & \circledast & \circledast \\ 0 & 0 & \circledast & \circledast \end{pmatrix} =: \tilde{A}.$$

Wegen

$$\tilde{A} = (G_{(m)}(\cdots(G_{(1)}A)G_{(1)}^T)\cdots)G_{(m)}^T =: Q^TAQ$$

mit Q orthogonal ist $\tilde{A}$ ähnlich zu A, hat also die selben Eigenwerte λ_i (mit Eigenvektoren Qv_i, für deren Berechnung man also Q speichern muss).

SPEKTRALVERSCHIEBUNG Die Konvergenzgeschwindigkeit des QR-Verfahrens hängt ab vom (größten) Verhältnis der Eigenwerte $\frac{|\lambda_{j+1}|}{|\lambda_j|}$. Haben wir also eine Näherung μ für einen Eigenwert λ_i so, dass

$$|\mu - \lambda_i| \ll |\mu - \lambda_j| \quad \text{für alle } j \neq i$$

gilt, und wenden das QR-Verfahren auf $A - \mu I$ an, dann wird der betragskleinste Eigenwert $\lambda_i - \mu$ von $A - \mu I$ sehr rasch angenähert. Macht man die Spektralverschiebung wieder

[10] Tatsächlich kann $A^{(m)}$ sofort bei Anwendung der berechneten Givens-Rotation von links und rechts überschrieben werden. Dass das funktioniert, ist nicht offensichtlich, und folgt aus der Eindeutigkeit der QR-Zerlegung sowie der linearen Abhängigkeit von Vektoren der Form $G(a, 0)^T$ und $G(b, 0)^T$ für jede Rotation G. Letzteres sorgt dafür, dass die zusätzlich erzeugten Subdiagonaleinträge bei der nächsten Rotation wieder verschwinden.

rückgängig, erhält man eine bessere Näherung $A^{(m+1)}$:

$$\begin{aligned} Q^{(m)}R^{(m)} &= A^{(m)} - \mu I, \\ A^{(m+1)} &= R^{(m)}Q^{(m)} + \mu I. \end{aligned}$$

Wegen

$$A^{(m+1)} = (Q^{(m)})^T(A^{(m)} - \mu I)Q^{(m)} + \mu I = (Q^{(m)})^T A^{(m)} Q^{(m)}$$

gilt immer noch $\sigma(A^{(m+1)}) = \sigma(A)$. (Die obere Hessenbergform bleibt durch Addition einer Diagonalmatrix natürlich erhalten.)

Wie bei der inversen Vektoriteration kann man nun in jedem Schritt einen neuen Parameter $\mu^{(m)}$ wählen, um die Konvergenzgeschwindigkeit weiter zu verbessern. Da die Diagonaleinträge von $A^{(m)}$ gegen die Eigenwerte von A konvergieren, liegt die Wahl eines Diagonaleintrags für $\mu^{(m)}$ nahe. Nun ist die ursprüngliche Spektralverschiebung so gewählt, dass $\lambda_i - \mu$ der betragskleinste Eigenwert von $A - \mu I$ ist. Da die Diagonalelemente im QR-Algorithmus betragsmäßig abfallend sortiert werden, strebt also der Eintrag $a_{nn}^{(m)} - \mu$ schnell gegen $\lambda_i - \mu$. Man wählt daher sinnvollerweise den *Rayleigh-Shift*

$$\mu^{(m)} = a_{nn}^{(m)}.$$

DEFLATION Bei Anwendung der Spektralverschiebung wird der Eintrag $a_{n,n-1}^{(m)}$ in der Regel quadratisch gegen 0 gehen. Sobald $a_{n,n-1}^{(m)} = \varepsilon$ klein genug ist, hat $A^{(m)}$ die Form

$$A^{(m)} = \begin{pmatrix} & & & * \\ & \bar{A} & & * \\ & & & * \\ 0 & 0 & \varepsilon & \tilde{\lambda}_n \end{pmatrix},$$

wobei $\tilde{\lambda}_n$ eine gute Näherung von λ_n und $\bar{A} \in \mathbb{R}^{(n-1)\times(n-1)}$ eine obere Hessenbergmatrix ist, deren Eigenwerte ungefähr gleich den Eigenwerten λ_j, $j \neq i$, der Matrix A sind. Es liegt also nahe, die Iteration mit $A^{(m+1)} = \bar{A}$ fortzusetzen, und so mit jedem berechneten Eigenwert die Dimension der auftretenden Matrizen zu reduzieren.

Algorithmus 7.4 beschreibt das QR-Verfahren inklusive der oben beschriebenen Modifikationen (mit expliziten Rayleigh-Shifts). Sind zusätzlich die Eigenvektoren von A gesucht, müssen zumindest im letzten Schritt die Givens-Rotationen aufmultipliziert werden, um die Matrix $Q^{(m)}$ zu erhalten (und genauso für die Givens-Rotationen bei der Transformation auf Hessenbergform).

Algorithmus 7.4 : Beschleunigtes QR-Verfahren

Input : $A \in \mathbb{R}^{n \times n}$, $m = 0, \tau$

Transformiere A auf obere Hessenbergform $\rightarrow A^{(0)}$
repeat
  $A^{(m)} \leftarrow A^{(m)} - a_{nn}^{(m)} I$
  $A^{(m+1)} \leftarrow A^{(m)}$
  for $k = 1, \ldots, n$ **do**
    Berechne Givens-Rotation $G_{k-1,k}$ für $A^{(m)}$
    $A^{(m+1)} \leftarrow G_{k-1,k} A^{(m+1)} G_{k-1,k}^{T}$
  $A^{(m+1)} \leftarrow A^{(m+1)} + a_{nn}^{(m)} I$
  $m \leftarrow m + 1$
  if $|a_{n-1,n}^{(m)}| < \tau$ **then**
    $\lambda_n \leftarrow a_{nn}^{m}$
    $A^{(m)} \leftarrow (a_{ij}^{(m)})_{i,j=1}^{n-1}$
    $n \leftarrow n - 1$
until $n = 0$
Output : λ_i

WEITERFÜHRENDE LITERATUR

K. Bryan und T. Leise (2006). *The $25,000,000,000 eigenvector: the linear algebra behind Google*. SIAM Rev. 48.3, 569–581 (electronic). URL: http://www.rose-hulman.edu/~bryan/googleFinalVersionFixed.pdf.

G. H. Golub und C. F. Van Loan (2013). *Matrix Computations*. 4. Aufl. Johns Hopkins University Press, Baltimore, MD.

D. S. Watkins (1982). *Understanding the QR algorithm*. SIAM Rev. 24.4, S. 427–440.

D. S. Watkins (2008). *The QR algorithm revisited*. SIAM Review 50.1, S. 133–145. DOI: 10.1137/060659454.

Teil III

NUMERISCHE ANALYSIS

POLYNOM-INTERPOLATION

8

Die Interpolation ist ein grundlegender Bestandteil einer konstruktiven Theorie der Funktionen: Zu einer diskreten Menge gegebener Werte soll eine Funktion konstruiert werden, die diese Werte annimmt. Präziser stellen wir folgende *Interpolationsaufgabe*: Gegeben seien die

Stützstellen $x_0, \ldots, x_n \in [a, b] \subset \mathbb{R}$ und

Funktionswerte $f(x_0), \ldots, f(x_n) \in \mathbb{R}$;

gesucht ist eine (einfache) Funktion $g : \mathbb{R} \to \mathbb{R}$, so dass gilt

$$g(x_i) = f(x_i) \quad \text{für alle } 0 \leqslant i \leqslant n.$$

Ursprünglich entstammt diese Fragestellung der *Tabelleninterpolation*; vor der Verbreitung von Rechenmaschinen waren spezielle Funktionen wie Kosinus, Logarithmus oder die Exponentialfunktion in lange Listen von Argumenten und Funktionswerten niedergeschrieben. War man etwa am Wert von $\log 0.456$ interessiert, so suchte man zum Beispiel den Eintrag für $\log 0.45$ und $\log 0.46$, und interpolierte linear:

$$\log 0.456 \approx \log 0.45 + \frac{\log 0.46 - \log 0.45}{0.46 - 0.45}(0.456 - 0.45) = -0.7925$$

(der exakte Wert ist etwa -0.785262). Eine höhere Genauigkeit ist natürlich zu erwarten, wenn statt einer linearen Funktion eine Funktion verwendet wird, die dem Verlauf der Logarithmusfunktion besser folgt. Heutzutage[1] wird die Interpolation hauptsächlich als Grundlage für Verfahren der numerischen Analysis verwendet: Um zum Beispiel eine Funktion näherungsweise zu integrieren oder zu differenzieren, bestimmt man für sie eine interpolierende Funktion, welche exakt integriert (oder differenziert) werden kann.

[1] Tatsächlich wird dieses Prinzip unter dem Namen *lookup table* immer noch in integrierten Schaltungen verwendet, wenn eine sehr große Zahl von Berechnungen pro Sekunde erforderlich ist. Insbesondere moderne Grafikprozessoren können lineare Interpolation und Tabellenzugriffe mit wesentlich höherer Geschwindigkeit ausführen, als sie z. B. die ersten Glieder der Sinusreihe aufsummieren können.

Als "einfache" Interpolationsfunktionen betrachten wir in diesem Kapitel hauptsächlich Polynome, genauer gesagt, Funktionen aus dem Vektorraum

$$P_n := \left\{ \sum_{j=0}^{n} a_j x^j \ : \ a_j \in \mathbb{R} \right\}$$

der *Polynome vom Höchstgrad* n. (Alternativen wären trigonometrische Funktionen oder stückweise Polynome, genannt *Splines*.) Aus der Definition ist sofort ersichtlich, dass die *Monome* $1, x, x^2, \ldots, x^n$ eine Basis von P_n darstellen und P_n also die Dimension $n+1$ hat.

8.1 LAGRANGE-INTERPOLATIONSFORMEL

Wir suchen also zu den gegebenen Stützstellen $x_0, \ldots, x_n \in \mathbb{R}$ mit zugehörigen Funktionswerten $f(x_0), \ldots, f(x_n)$ ein Polynom $p_n \in P_n$, so dass $p_n(x_i) = f(x_i)$ für alle $0 \leqslant i \leqslant n$ gilt. Es ist einleuchtend, dass die $n+1$ Interpolationsbedingungen an paarweise verschiedenen (was wir in Folge annehmen wollen) Stützstellen genügen, um die $n+1$ Koeffizienten eines Polynoms vom Grad n eindeutig festzulegen. Durch eine geschickte Wahl der Basis von P_n kann man diese Koeffizienten explizit berechnen, ohne ein Gleichungssystem lösen zu müssen.

Definition 8.1. Für $0 \leqslant j \leqslant n$ sind die *Lagrange-Polynome*[2] definiert durch

$$l_{jn}(x) = \prod_{\substack{k=0 \\ k \neq j}}^{n} \frac{x - x_k}{x_j - x_k}, \qquad x \in \mathbb{R}. \tag{8.1}$$

Man sieht sofort, dass die l_{jn} Polynome vom Höchstgrad n sind. Durch Einsetzen erkennt man auch, dass gilt

$$l_{jn}(x_i) = \begin{cases} 1 & \text{für } j = i, \\ 0 & \text{für } j \neq i, \end{cases}$$

(siehe Abb. 8.1).[3] Eine Lösung der Interpolationsaufgabe ist also gegeben durch

$$p_n = \sum_{j=0}^{n} f(x_j) l_{jn}.$$

[2] Joseph-Louis Lagrange hat in der zweiten Hälfte des 18. Jahrhunderts zahlreiche Beiträge zur Mechanik, Wahrscheinlichkeitstheorie, Algebra und Zahlentheorie geleistet. Auf das Interpolationsproblem führten ihn seine Arbeiten zur Umformung unendlicher Reihen.

[3] Aus dieser Eigenschaft folgt sofort, dass die l_{jn} linear unabhängig sind und daher eine Basis von P_n (genannt *nodale Basis*) bilden.

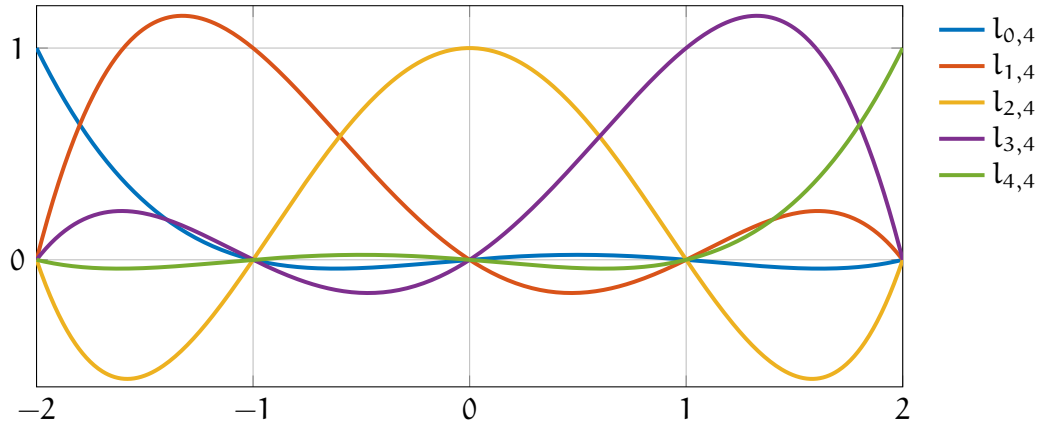


Abbildung 8.1: Die Lagrange-Polynome l_{j4} zu den Stützstellen $\{-2, -1, 0, 1, 2\}$.

Diese Lösung ist sogar eindeutig: Seien $p, q \in P_n$ zwei Polynome mit $p(x_i) = f(x_i) = q(x_i)$ für alle $0 \leqslant i \leqslant n$. Dann hat $p - q \in P_n$ die $n+1$ Nullstellen $x_0, \ldots, x_n$. Nach dem Hauptsatz der Algebra kann aber ein von Null verschiedenes Polynom vom Höchstgrad n höchstens n reelle Nullstellen haben; das Polynom $p - q$ muss also identisch Null sein, das heißt es gilt $p = q$. Wir halten fest:

Satz 8.2. *Für paarweise verschiedene $x_0, \ldots, x_n \in \mathbb{R}$ und $f(x_0), \ldots, f(x_n) \in \mathbb{R}$ existiert genau ein Polynom $p_n \in P_n$, so dass $p_n(x_i) = f(x_i)$ für alle $0 \leqslant i \leqslant n$ gilt.*

Die Lagrange-Darstellung hat jedoch wesentliche Nachteile: Einerseits besteht bei nahe beieinander liegenden Stützstellen die Gefahr der Auslöschung bei der Berechnung von (8.1). Andererseits müssen alle Lagrange-Polynome neu berechnet werden, falls sich eine Stützstelle ändert oder eine neue hinzukommt. Wir werden deshalb Algorithmen diskutieren, die – je nach Aufgabenstellung – besser geeignet sind.

8.2 POLYNOMAUSWERTUNG NACH AITKEN–NEVILLE

Ist man nur an (wenigen) Werten des Interpolationspolynoms interessiert (etwa bei der Tabelleninterpolation), so benötigt man die explizite Darstellung des Polynoms gar nicht. Der Algorithmus von Aitken und Neville wertet p_n rekursiv aus durch lineare Interpolation von Interpolationspolynomen niedrigeren Grades.

Um das Verfahren einfach darstellen zu können, definieren wir $p_{i,k}$ als Interpolationspolynom vom Höchstgrad k mit den Stützstellen $x_{i-k}, \ldots, x_i$. Das folgende Lemma stellt dann den Rekursionsschritt dar.

Lemma 8.3 (Aitken[4]). *Für alle* $n \in \mathbb{N}$ *und* $x \in \mathbb{R}$ *gilt*

$$p_n(x) = p_{n,n}(x) = \frac{x - x_0}{x_n - x_0} p_{n,n-1}(x) + \frac{x_n - x}{x_n - x_0} p_{n-1,n-1}(x). \tag{8.2}$$

Beweis. Wir überprüfen die Interpolationsbedingungen:

(i) Für $0 < i < n$ gilt

$$\frac{x_i - x_0}{x_n - x_0} p_{n,n-1}(x_i) + \frac{x_n - x_i}{x_n - x_0} p_{n-1,n-1}(x_i) = \frac{x_i - x_0}{x_n - x_0} f(x_i) + \frac{x_n - x_i}{x_n - x_0} f(x_i) = f(x_i),$$

da x_i Stützstelle sowohl für das Interpolationspolynom $p_{n,n-1}$ als auch für $p_{n-1,n-1}$ ist.

(ii) Für $i = 0$ gilt

$$\frac{x_0 - x_0}{x_n - x_0} p_{n,n-1}(x_0) + \frac{x_n - x_0}{x_n - x_0} p_{n-1,n-1}(x_0) = p_{n-1,n-1}(x_0) = f(x_0),$$

da x_0 Stützstelle für $p_{n-1,n-1}$ ist.

(iii) Für $i = n$ gilt analog

$$\frac{x_n - x_0}{x_n - x_0} p_{n,n-1}(x_n) + \frac{x_n - x_n}{x_n - x_0} p_{n-1,n-1}(x_n) = p_{n,n-1}(x_n) = f(x_n).$$

Nun sind $p_{n,n-1}$ und $p_{n-1,n-1}$ nach Definition Interpolationspolynome vom Höchstgrad $n - 1$. Die rechte Seite von (8.2) ist also auch ein Interpolationspolynom vom Höchstgrad n. Aus der Eindeutigkeit der Interpolationsaufgabe folgt daher, dass dieses identisch sein muss mit dem Interpolationspolynom p_n. □

Da es sich bei $p_{n,n-1}$ und $p_{n-1,n-1}$ ebenfalls um Interpolationspolynome (nun vom Höchstgrad $n - 1$) handelt, lassen sich diese ebenfalls mit Lemma 8.3 (mit entsprechend angepassten Stützstellen) berechnen; es gilt für alle $k = 0, \dots, n$ und $i = k, \dots, n$

$$p_{i,k}(x) = \frac{x - x_{i-k}}{x_i - x_{i-k}} p_{i,k-1}(x) + \frac{x_i - x}{x_i - x_{i-k}} p_{i-1,k-1}(x) \tag{8.3}$$

Dies führt auf das rekursive Schema von Neville[5] zur Auswertung von p_n an der (festen) Stelle x: Man berechnet jeweils für $k = 0, \dots, n$ und $i = k, \dots, n$ alle $p_{i,k}(x)$ (mit $p_{i,0}(x) = f(x_i)$).

[4] Alec Aitken (1895–1967) war einer der bedeutendsten Mathematiker Neuseelands. Er hatte ein phänomenales Gedächtnis und war ein Meister im Kopfrechnen.

[5] Eric Harold Neville (1889–1961) war ein britischer Mathematiker, hauptsächlich im Bereich der Geometrie tätig. Als er 1914 als Gastprofessor in Indien war, konnte er den indischen Ausnahmemathematiker Srinivasa Ramanujan überreden, mit ihm zurück nach England zu kommen.

Dies wird in folgendem Tableau verdeutlicht, in dem spaltenweise von oben nach unten jeder Wert durch Interpolation der beiden Werte direkt links und links oberhalb berechnet wird:

	p_{i0}	p_{i1}	p_{i2}	$\cdots$	p_{in}
x_0	$f(x_0)$				
x_1	$f(x_1)$	$p_{1,1}(x)$			
x_2	$f(x_2)$	$p_{2,1}(x)$	$p_{2,2}(x)$		
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	
x_n	$f(x_n)$	$p_{n,1}(x)$	$p_{n,2}(x)$	$\cdots$	$p_{n,n}(x) = p_n(x)$

Dabei sind für $k = 1$ insgesamt $6n$ Operationen nötig, für $k = 2$ sind es $6(n-1)$ Operationen, und so weiter, bis zur Berechnung für $k = n$ mit 6 Operationen. Nach Hinzunahme einer weiteren Stützstelle x_{n+1} kann man dann den Wert des Interpolationspolynon p_{n+1} in x einfach durch Berechnung einer weiteren Zeile aus den bisher bestimmten Werten konstruieren. Dafür muss für jedes k nur $p_{n+1,k}(x)$ berechnet werden, was nun in $\mathcal{O}(n)$ Operationen möglich ist.

8.3 POLYNOMDARSTELLUNG NACH NEWTON

Möchte man aber das Interpolationspolynom an vielen Stellen auswerten, ist es effizienter, das Polynom explizit zu berechnen. Auch für den Fall, dass man das Polynom differenzieren oder integrieren möchte, benötigt man eine Darstellung des Polynoms als Linearkombination von Basis-Polynomen. Wählt man die Basis geschickt, so lassen sich die entsprechenden Koeffizienten rekursiv mit Hilfe des Aitken–Neville-Schemas berechnen. Wir wählen dafür die so genannte *Newton-Form* des Interpolationspolynoms

$$p_n(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + a_n(x - x_0)\cdots(x - x_{n-1}).$$

Dabei handelt es sich offensichtlich um ein Polynom vom Grad n; außerdem gilt nach Konstruktion

$$p_n(x) = p_{n-1}(x) + a_n(x - x_0)\cdots(x - x_{n-1}). \tag{8.4}$$

Die Koeffizienten a_k können wir damit nacheinander aus den Interpolationsbedingungen berechnen:

$$\begin{aligned}
f(x_0) = p_0(x_0) = a_0 \qquad &\Rightarrow \qquad a_0 = f(x_0),\\
f(x_1) = p_0(x_1) + a_1(x_1 - x_0) \qquad &\Rightarrow \qquad a_1 = \frac{f(x_1) - f(x_0)}{x_1 - x_0},\\
f(x_2) = p_1(x_2) + a_2(x_2 - x_0)(x_2 - x_1) \qquad &\Rightarrow \qquad a_2 = \frac{f(x_2) - p_1(x_2)}{(x_2 - x_0)(x_2 - x_1)},
\end{aligned}$$

und so weiter. Für die Koeffizienten der Newton-Darstellung gilt also

$$a_k = \frac{f(x_k) - p_{k-1}(x_k)}{(x_k - x_0)\cdots(x_k - x_{k-1})} =: [x_0, \ldots, x_k]f, \qquad k = 0, \ldots, n$$

mit $a_0 = [x_0]f = f(x_0)$. Diese Koeffizienten $[x_0, \ldots, x_k]f$ heißen k-*te dividierte Differenzen*. Aus dem Aitken–Neville-Schema erhält man eine rekursive Formel für Ihre Berechnung: Einsetzen von (8.4) und (8.3) ergibt

$$\begin{aligned}
[x_0, \ldots, x_k]f\,(x - x_0)\cdots(x - x_{k-1}) &= p_k(x) - p_{k-1}(x) \\
&= \left(\frac{x - x_k}{x_k - x_0} p_{k,k-1}(x) - \frac{x_k - x}{x_k - x_0} p_{k-1,k-1}(x)\right) - p_{k-1,k-1}(x) \\
&= \frac{x - x_k}{x_k - x_0}\,(p_{k,k-1}(x) - p_{k-1,k-1}(x)) \\
&= \frac{x - x_k}{x_k - x_0}\big([x_1, \ldots, x_k]f\,(x - x_1)\cdots(x - x_{k-1}) \\
&\qquad - [x_0, \ldots, x_{k-1}]f\,(x - x_0)\cdots(x - x_{k-2}) + \mathcal{O}\left(x^{k-2}\right)\big),
\end{aligned}$$

wobei wir alle Polynom-Terme niedrigerer Ordnung im Landau-Symbol zusammengefasst haben. Führt man nun in dieser Gleichung einen Koeffizientenvergleich für x^k durch, so erhält man daraus die Rekursionsformel

$$[x_0, \ldots, x_k]f = \frac{[x_1, \ldots, x_k]f - [x_0, \ldots, x_{k-1}]f}{x_k - x_0}.$$

Das allgemeine Schema für $k = 0, \ldots, n$ und $i = k, \ldots, n$ lautet nun

$$\begin{cases} [x_i]f = f(x_i), \\ [x_i, \ldots, x_k]f = \dfrac{[x_{i+1}, \ldots, x_k]f - [x_i, \ldots, x_{k-1}]f}{x_k - x_i}, \end{cases} \tag{8.5}$$

bzw. in Tableau-Form

	$[x_i]f$	$[x_{i-1}, x_i]f$	$[x_{i-2}, x_{i-1}, x_i]f$	$[x_{i-3}, \ldots, x_i]f$	$\cdots$
x_0	$f(x_0)$				
x_1	$f(x_1)$	$[x_0, x_1]f$			
x_2	$f(x_2)$	$[x_1, x_2]f$	$[x_0, x_1, x_2]f$		
x_3	$f(x_3)$	$[x_2, x_3]f$	$[x_1, x_2, x_3]f$	$[x_0, x_1, x_2, x_3]f$	
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Wieder geht man spaltenweise von oben nach unten vor und liest am Schluss die Koeffizienten a_k auf der Diagonalen ab. Hat man die Koeffizienten in der Newton-Darstellung des Interpolationspolynoms p_n einmal berechnet, kann man die Auswertung an der Stelle x effizient mit dem Horner-Schema durchführen:

$$p_n(x) = a_0 + (x - x_0)\,(a_1 + (x - x_1)\,(\cdots + (x - x_{n-2})\,(a_{n-1} + (x - x_{n-1})a_n)))\,.$$

8.4 INTERPOLATIONSFEHLER

Wir untersuchen nun den Interpolationsfehler, d. h. die Differenz zwischen der zu interpolierenden Funktion und dem Interpolationspolynom *außerhalb* der Stützstellen.

Satz 8.4. *Sei* $f \in C^{n+1}([a,b])$, $a = x_0 < \cdots < x_n = b$, *und* $p_n \in P_n$ *das Interpolationspolynom zu diesen Stützstellen. Dann existiert für alle* $x \in [a,b]$ *ein* $\xi \in [a,b]$ *mit*

$$f(x) - p_n(x) = (x-x_0)\cdots(x-x_n)\frac{f^{(n+1)}(\xi)}{(n+1)!}. \tag{8.6}$$

Beweis. Sei also $x \in [a,b]$ gegeben. Für $x = x_i$ sind beide Seiten von (8.6) gleich Null; deshalb reicht es, nur $x \neq x_i$, $0 \leqslant i \leqslant n$, zu betrachten. Wir definieren für $K \in \mathbb{R}$

$$g(t) := f(t) - p_n(t) - K(t-x_0)\cdots(t-x_n).$$

Diese Funktion ist $n+1$-mal stetig differenzierbar, und hat die Nullstellen x_i, $0 \leqslant i \leqslant n$, da p_n die Interpolationsbedingungen erfüllt und der letzte Term in den Stützstellen verschwindet. Wählt man nun

$$K = \frac{f(x) - p_n(x)}{(x-x_0)\cdots(x-x_n)},$$

so ist auch x eine Nullstelle von g; insgesamt hat g also $n+2$ Nullstellen. Nach dem Satz von Rolle hat die Ableitung g' daher $n+1$ Nullstellen (je eine zwischen zwei Nullstellen von g). Weiter hat g'' also n Nullstellen, g''' hat $n-1$ Nullstellen, et cetera. Schließlich hat $g^{(n+1)}$ genau eine Nullstelle $\xi \in [a,b]$. Für diese gilt also:

$$\begin{aligned} 0 &= g^{(n+1)}(\xi) \\ &= f^{(n+1)}(\xi) - p_n^{(n+1)}(\xi) - \frac{f(x)-p_n(x)}{(x-x_0)\cdots(x-x_n)}[(t-x_0)\cdots(t-x_n)]^{(n+1)}|_{t=\xi} \\ &= f^{(n+1)}(\xi) - \frac{f(x)-p_n(x)}{(x-x_0)\cdots(x-x_n)}(n+1)!\,. \end{aligned} \tag{8.7}$$

Dabei haben wir verwendet, dass p_n ein Polynom vom Grad n ist (dessen $(n+1)$-te Ableitung daher verschwindet), und die letzte Klammer ein Polynom vom Grad $n+1$ ist, dessen höchster Koeffizient gleich 1 ist – also die Ableitung $(n+1)!$ hat. Lösen wir die Gleichung (8.7) nach $f(x) - p_n(x)$ auf, erhalten wir die Behauptung. □

Bemerkung. Gilt $\|f^{(n+1)}\|_\infty \leqslant M$, so erhält man aus (8.6) die Abschätzung

$$\|f - p_n\|_\infty \leqslant \|\omega_n\|_\infty \frac{M}{(n+1)!}.$$

für das *Stützstellenpolynom* $\omega_n(x) := (x-x_0)\cdots(x-x_n)$. Ist also die $n+1$-te Ableitung von f beschränkt, so kann der Fehler in der Interpolation trotzdem sehr groß sein, ω_n für große n sehr starke Oszillationen aufweisen und die rechte Seite daher beliebig groß sein kann (*Runge-Phänomen*).

Eine möglicher Ausweg ist, die Stützstellen so zu wählen, dass $\|\omega_n\|_\infty$ minimal ist. (Dieser Ansatz führt auf die Interpolation mit *Tschebyscheff-Polynomen*.) Eine andere Möglichkeit besteht darin, statt mit einem Polynom hohen Grades mit stückweisen Polynomen kleineren Grades zu arbeiten, was auf die *Spline-Interpolation* führt.

8.5 NUMERISCHE DIFFERENTIATION

Die Polynomdarstellung nach Newton lässt sich z. B. ausnützen, um die Ableitung einer Funktion numerisch zu berechnen: Wir wählen Stützstellen $x_0, \dots, x_n$, interpolieren f durch ein Polynom $p_n \in P_n$, und differenzieren dann das Polynom. Da wir Polynome exakt differenzieren können, wird nur der Interpolationsfehler die Genauigkeit dieser Approximation bestimmen.

Angenommen, wir wollen eine Funktion $f : [a, b] \to \mathbb{R}$ in einem Punkt $x_0 \in [a, b]$ auswerten. Wir betrachten dafür zu den Stützstellen $x_0, \dots, x_n$ das Interpolationspolynom in Newtonform:

$$p_n(x) = f(x_0) + [x_0, x_1]f\,(x - x_0) + \cdots + [x_0, \dots, x_n]f\,(x - x_0) \cdots (x - x_{n-1})$$

und den Interpolationsfehler

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

Ist $f \in C^{(n+2)}([a, b])$, so können wir beide Gleichungen differenzieren und $x = x_0$ einsetzen:

$$\begin{aligned} p_n'(x_0) &= [x_0, x_1]f\ + [x_0, x_1, x_2]f\,(x_0 - x_1) + \cdots \\ &\qquad + [x_0, \dots, x_n]f\,(x_0 - x_1) \cdots (x_0 - x_{n-1}), \\ r_n'(x_0) &= (x_0 - x_1) \cdots (x_0 - x_n) \frac{f^{(n+1)}(\xi(x_0))}{(n+1)!}. \end{aligned}$$

Dabei haben wir ausgenützt, dass alle Terme, die aufgrund der Produktregel entstehen und $(x - x_0)$ enthalten, beim Einsetzen von $x = x_0$ wegfallen; übrig bleiben lediglich die Terme, in denen der Faktor $(x - x_0)$ wegdifferenziert wurde. Dadurch erhalten wir:

$$f'(x_0) = p_n'(x_0) + r_n'(x_0).$$

Je mehr Stützstellen hinzugenommen werden, desto genauer wird also die Approximation sein. Andererseits können wir Interpolationspolynom und -Fehler auch weiter differenzieren, um höhere Ableitungen näherungsweise zu berechnen. Insbesondere erhalten wir für die n-te Ableitung:

Satz 8.5. *Ist* $f \in C^{n+2}([a,b])$ *und sind* $x_j \in [a,b]$, $j = 0,\dots,n$, *paarweise verschieden, so existiert ein* $\xi \in [a,b]$, *so dass gilt:*

$$f^{(n)}(x_0) = n![x_0,\dots,x_n]f + (x_0 - x_1)\cdots(x_0 - x_n)\frac{f^{(n+1)}(\xi)}{(n+1)!}.$$

Sind die Stützstellen im Abstand $h > 0$ um x_0 verteilt, erhält man die klassischen *Differenzenquotienten*:

Beispiel 8.6.

(i) *Vorwärtsdifferenzenquotient* für die erste Ableitung: $x_1 = x_0 + h$,

$$\begin{aligned} f'(x_0) &= p_1'(x_0) + r_1'(x_0) = [x_0,x_1]f + (x_0 - x_1)\tfrac{1}{2}f''(\xi) \\ &= \frac{f(x_0+h) - f(x_0)}{h} - \frac{h}{2}f''(\xi). \end{aligned}$$

Der Verfahrensfehler ist also $\mathcal{O}(h)$ für $h \to 0$.

(ii) *Zentrale Differenzenquotienten*: $x_1 = x_0 + h$, $x_2 = x_0 - h$. Die drei Stützstellen legen ein Interpolationspolynom vom Grad $n = 2$ fest, das wir verwenden können, um die erste oder die zweite Ableitung von f in x_0 berechnen zu können.

Für die zweite Ableitung verwenden wir Satz 8.5 und die Rekursionsformel (8.5) und berechnen (Übung!)

$$\begin{aligned} f''(x_0) &= 2![x_0,x_1,x_2]f + (x_0 - x_1)(x_0 - x_2)\frac{1}{3!}f'''(\xi) \\ &= \frac{f(x_0+h) - 2f(x_0) + f(x_0-h)}{h^2} - h^2\frac{1}{6}f'''(\xi). \end{aligned}$$

Die erste Ableitung erhalten wir wieder aus der ersten Ableitung des Interpolationspolynoms und der Rekursionsformel:

$$\begin{aligned} f'(x_0) &= p_2'(x_0) + r_2'(x_0) \\ &= [x_0,x_1]f + [x_0,x_1,x_2]f\,(x_0 - x_1) + (x_0 - x_1)(x_0 - x_2)\frac{1}{6}f'''(\xi) \\ &= \frac{f(x_0+h) - f(x_0-h)}{2h} - h^2\frac{1}{6}f'''(\xi). \end{aligned}$$

In beiden Fällen ist der Fehler $\mathcal{O}(h^2)$ für $h \to 0$.

8.6 TRIGONOMETRISCHE INTERPOLATION UND FFT

Für die Interpolation von Werten, die durch periodische Vorgänge wie zum Beispiel Schwingungen erzeugt werden, ist es sinnvoll, ebenfalls periodische Funktionen zu verwenden. Wir nehmen der Einfachheit an, dass die Periodenlänge 2π ist, also $f(x) = f(x + 2\pi)$ für alle x gilt.[6] Besonders einfach wird die Aufgabe dann, wenn wir unsere Interpolationspolynome auf dem *komplexen Einheitskreis* betrachten. Dazu verwenden wir die Variablentransformation $z : [0, 2\pi) \to \mathbb{C}$, $x \mapsto e^{ix}$ und definieren das *trigonometrische Polynom*

$$p_n(z(x)) = \sum_{k=0}^{n} c_k z(x)^k = \sum_{k=0}^{n} c_k \left(e^{ix}\right)^k = \sum_{k=0}^{n} c_k e^{ikx}.$$

Dabei ist es im Unterschied zu bisher vorteilhaft, n (statt $n + 1$) gleichverteilte Stützstellen zu verwenden. Wir betrachten also folgendes Problem:

Gegeben: $x_j = 2\pi\frac{j}{n}$ und $f_j := f(x_j) \in \mathbb{C}$ für $j = 0, \ldots, n-1$

Gesucht: $c_k \in \mathbb{C}$ für $k = 0, \ldots, n-1$, so dass gilt

$$\sum_{k=0}^{n-1} c_k e^{2\pi i k \frac{j}{n}} = f_j, \qquad j = 0, \ldots, n-1,$$

oder, in Matrixform

$$M\vec{c} = \vec{f},$$

mit $\vec{c} = (c_0, \ldots, c_{n-1})^T \in \mathbb{C}^n$, $\vec{f} = (f_0, \ldots, f_{n-1})^T \in \mathbb{C}^n$ und

$$M = (m_{kj})_{k,j=0}^{n-1} \in \mathbb{C}^{n\times n}, \qquad m_{kj} = e^{2\pi i \frac{kj}{n}}.$$

Dieses Problem hat eine eindeutige Lösung, da die Matrix M invertierbar ist. Die inverse Matrix lässt sich sogar direkt angeben als

$$M^{-1} = \frac{1}{n}\overline{M^T},$$

wobei $\overline{M^T}$ die transponierte Matrix mit den komplex konjugierten Einträgen von M ist. Mit $\overline{e^{ikx}} = e^{-ikx}$ folgt

$$(\overline{M^T})_{kj} = \overline{m_{jk}} = e^{-2\pi i \frac{jk}{n}}.$$

[6] Das lässt sich zum Beispiel durch eine geeignete Skalierung von x erzwingen.

Um dies zu zeigen, betrachten wir die Einträge des Produkts $\overline{M^T}M$:

$$(\overline{M^T}M)_{kj} = \sum_{l=0}^{n-1} \overline{m_{lk}} m_{lj} = \sum_{l=0}^{n-1} e^{-2\pi i \frac{lk}{n}} e^{2\pi i \frac{lj}{n}} = \sum_{l=0}^{n-1} e^{-2\pi i l \frac{(k-j)}{n}}$$
$$= \begin{cases} \sum_{l=0}^{n-1} 1 = n & \text{falls } k = j, \\ \sum_{l=0}^{n-1} \left(e^{-2\pi i \frac{(k-j)}{n}}\right)^l = \frac{1-e^{-2\pi i(k-j)}}{1-e^{-2\pi i \frac{(k-j)}{n}}} & \text{sonst,} \end{cases}$$

wobei wir im zweiten Fall die geometrische Summenformel $\sum_{k=0}^{n} q^k = \frac{1-q^n}{1-q}$ verwendet haben. Der letzte Bruch verschwindet aber, da gilt

$$e^{-2\pi i(k-j)} = (e^{-\pi i})^{2(k-j)} = \left((-1)^2\right)^{k-j} = 1.$$

Es ist also in der Tat $\overline{M^T}M = nI$, und die eindeutige Lösung der Interpolationsaufgabe ist gegeben durch

$$\vec{c} = \frac{1}{n}\overline{M^T}\vec{f},$$

oder, wieder komponentenweise für $k = 0, \ldots, n-1$:

$$c_k = \frac{1}{n} \sum_{j=0}^{n-1} f_j e^{-2\pi i \frac{jk}{n}}, \qquad k = 0, \ldots, n-1. \tag{8.8}$$

Dies bezeichnet man auch als *diskrete Fouriertransformation* (DFT). Mit ihrer Hilfe kann man zum Beispiel periodisch abgetastete Signale in ihre Grundschwingungen zerlegen (c_k gibt dabei die Amplitude der Grundschwingung mit der Frequenz k an), was fundamental für die gesamte digitale Signalverarbeitung (und damit die mathematische Basis für CDs, MP3-Musikkompression, JPEG-Bildkompression etc.) ist. Die *inverse DFT* entspricht der Polynomauswertung

$$f_j = \sum_{k=0}^{n-1} c_k e^{2\pi i \frac{jk}{n}}, \qquad j = 0, \ldots, n-1.$$

SCHNELLE FOURIERTRANSFORMATION

Ein Nachteil der diskreten Fouriertransformation ist ihr Aufwand von $\mathcal{O}(n^2)$ (entsprechend einem Matrix-Vektor-Produkt). Bei einer Anzahl von 10^6 gegebenen Werten[7] benötigt ein Prozessor, der eine Milliarde Operationen pro Sekunde schafft, bereits 17 Minuten. Die Berechnung kann aber mit der *schnellen Fouriertransformation* (FFT, "fast Fourier transform")[8] wesentlich beschleunigt werden.

[7] was etwa 20 Sekunden Musik in CD-Qualität entspricht

[8] Publiziert 1965 von James Cooley und John Tukey in einem Artikel, der immer noch zu den am öftesten zitierten mathematischen Artikeln zählt. Das Verfahren war aber – wie so oft – bereits Gauß bekannt.

Der klassische Ansatz geht davon aus, dass n gerade ist.[9] Dann lässt sich die Symmetrie der *Einheitswurzeln*[10]

$$\omega_n := e^{-2\pi i \frac{1}{n}}$$

ausnutzen, um den Berechnungsaufwand zu halbieren. Es gilt nämlich offensichtlich

$$\omega_{2n}^2 = \left(e^{-2\pi i \frac{1}{2n}}\right)^2 = e^{-2\pi i \frac{2}{2n}} = \omega_n = \omega_{n/2}^{1/2}.$$

Wir zerlegen jetzt die Summe in (8.8) in je eine Summe für die geraden und für die ungeraden j, die wir mit $j = 2j'$ beziehungsweise $j = 2j' + 1$ durchzählen:

$$\begin{aligned} nc_k &= \sum_{j \text{ gerade}} \omega_n^{kj} f_j + \sum_{j \text{ ungerade}} \omega_n^{kj} f_j \\ &= \sum_{j'=0}^{n/2-1} \omega_n^{k2j'} f_{2j'} + \sum_{j'=0}^{n/2-1} \omega_n^{k(2j'+1)} f_{2j'+1} \\ &= \sum_{j'=0}^{n/2-1} \omega_{n/2}^{kj'} f_{2j'} + \omega_n^k \sum_{j'=0}^{n/2-1} \omega_{n/2}^{kj'} f_{2j'+1} \\ &=: c'_k + \omega_n^k c''_k. \end{aligned}$$

Für $k \leqslant \frac{n}{2} - 1$ sind die c'_k gerade die DFT zu den $\frac{n}{2}$ Werten $f_0, f_2, \ldots, f_{n-2}$, die c''_k diejenige zu den $\frac{n}{2}$ Werten $f_1, f_3, \ldots, f_{n-1}$. Für diese k lassen sich die Koeffizienten c_k also durch zwei DFTs der Länge $\frac{n}{2}$ berechnen. Die Arbeitsersparnis entsteht dadurch, dass die restlichen c_k auch mit Hilfe der c'_k, c''_k ausgedrückt werden können. Ist nämlich $k \geqslant \frac{n}{2}$, also $k = \frac{n}{2} + k'$, $0 \leqslant k' \leqslant \frac{n}{2} - 1$, so gilt

$$\begin{aligned} \omega_n^{2jk} &= \omega_n^{2jk'+jn} = \omega_n^{2jk'} (\omega_n^n)^j = \omega_n^{2jk'}, \qquad j = 0, \ldots, \frac{n}{2} - 1, \\ \omega_n^k &= \omega_n^{k'+n/2} = \omega_n^{k'} \omega_n^{n/2} = \omega_n^{k'} e^{-\pi i} = -\omega_n^{k'}. \end{aligned}$$

Für $k = k' + \frac{n}{2} \geqslant \frac{n}{2} - 1$ haben wir daher

$$nc_k = c'_{k'} - \omega_n^{k'} c''_{k'}.$$

Der Aufwand $A(n)$ für eine DFT der Länge n entspricht also (bei Vorberechnung der Einheitswurzeln) dem von zweier DFTs halber Länge sowie n Additionen, $\frac{n}{2}$ Multiplikationen und n Divisionen:

$$A(n) = 2A\left(\frac{n}{2}\right) + \frac{5}{2}n.$$

[9] Es existieren Varianten, bei denen dies nicht notwendig ist.

[10] So genannt, da $\omega_n^n = 1$ gilt. Sie werden in diesem Kontext auch gerne als “twiddle factors” bezeichnet.

Ist nun $\frac{n}{2}$ wieder gerade, lassen sich die DFTs der Länge $\frac{n}{2}$ wieder durch zwei DFTs halber Länge ausdrücken. Dies lässt sich m mal wiederholen, wenn $n = 2^m$ eine Zweierpotenz ist, bis man bei der trivialen DFT der Länge 1, offensichtlich gegeben durch $c_0 = f_0$, ankommt (siehe Algorithmus 8.1). Für die rekursive Berechnung der DFT erhält man so per Induktion den Aufwand

$$A(n) = \mathcal{O}(n \log_2(n)).$$

Auf diese Weise lässt sich eine DFT der Länge 10^6 in $A(10^6) \approx 10^7$ Operationen durchführen, was bei einer Milliarde Operationen pro Sekunde etwa 0.1 Sekunden benötigt.

Algorithmus 8.1 : FFT

Input : $f_0, \ldots, f_{n-1}, n = 2^m$

1 **if** $n = 1$ **then**
2 $\quad c_0 \leftarrow f_0$
3 $a_0, \ldots, a_{\frac{n}{2}} \leftarrow \mathrm{FFT}(f_0, f_2, \ldots, f_{n-2})$
4 $b_0, \ldots, b_{\frac{n}{2}} \leftarrow \mathrm{FFT}(f_1, f_3, \ldots, f_{n-1})$
5 **for** $k = 0, \ldots, \frac{n}{2} - 1$ **do**
6 $\quad c_k \leftarrow (a_k + e^{-2\pi i k/n} b_k)/n$
7 $\quad c_{k+\frac{n}{2}} \leftarrow (a_k - e^{-2\pi i k/n} b_k)/2$

Output : $c_0, \ldots, c_{n-1}$

REELLE TRIGONOMETRISCHE INTERPOLATION

Sollen reelle Funktionswerte f_j interpoliert werden, möchte man in der Regel auch ein reelles Interpolationspolynom finden. Als periodische Ansatzfunktionen kommen Sinus und Kosinus in Frage. Wegen der Beziehung $e^{ix} = \cos(x) + i \sin(x)$ können wir auch die Koeffizienten des reellen trigonometrischen Interpolationspolynoms mit Hilfe der FFT berechnen.

Die Idee lässt sich besonders einfach darstellen, wenn $n = 2p + 1$ ungerade ist. Dann können wir die Koeffizienten des komplexen Interpolationspolynoms wie folgt umnummerieren:

$$\begin{array}{ccccccccc} c_0, & \ldots, & c_p, & c_{p+1}, & \ldots, & c_{n-1} \\ \downarrow & & \downarrow & \downarrow & & \downarrow \\ c_0 & \ldots, & c_p, & c_{-p} & \ldots, & c_{-1} \end{array}$$

Da analog $e^{2\pi i \frac{n-1}{n}} = e^{-2\pi i \frac{1}{n}}, \ldots, e^{2\pi i \frac{p+1}{n}} = e^{-2\pi i \frac{p}{n}}$ gilt,[11] können wir die Interpolationsbedingungen schreiben als

$$f(x_j) = \sum_{k=-p}^{p} c_k e^{ikx_j}.$$

[11] Es ist egal, ob wir $n - k$ Schritte gegen den Uhrzeigersinn oder k Schritte im Uhrzeigersinn gehen.

Wenn jetzt $f(x_j) \in \mathbb{R}$ für alle $j = 0, \ldots, n-1$ ist, so müssen sich die Imaginärteile auf der rechten Seite zu Null summieren. Wegen $\overline{e^{ikx}} = e^{-ikx}$ bedeutet das, dass $\overline{c_k} = c_{-k}$ (und insbesondere $c_0 \in \mathbb{R}$) sein muss.

Schreiben wir $c_k = c_k^r + ic_k^i$, erhalten wir damit (durch die Symmetrie von Sinus und Kosinus)

$$\begin{aligned} f(x_j) &= c_0 + \sum_{k=1}^{p} \left(c_k e^{ikx_j} + \overline{c_k} e^{-ikx_j} \right) \\ &= c_0 + \sum_{k=1}^{p} \left((c_k^r + ic_k^i)(\cos(kx_j) + i\sin(kx_j) + (c_k^r - ic_k^i)(\cos(kx_j) - i\sin(kx_j) \right) \\ &= c_0 + \sum_{k=1}^{p} \left(2c_k^r \cos(kx_j) - 2c_k^i \sin(kx_j) \right). \end{aligned}$$

Damit haben wir gezeigt, dass das eindeutige reelle Interpolationspolynom eine Linearkombination der $\{1, \cos(kx), \sin(kx) : k = 1, \ldots, \frac{n-1}{2}\}$ ist, und die Koeffizienten mit Hilfe der komplexen FFT berechnet werden können.

WEITERFÜHRENDE LITERATUR

G. Hämmerlin und K.-H. Hoffmann (1994). *Numerische Mathematik*. 4. Aufl. Springer-Verlag, Berlin.

C. Van Loan (1992). *Computational frameworks for the fast Fourier transform*. Bd. 10. Frontiers in Applied Mathematics. Society for Industrial und Applied Mathematics (SIAM), Philadelphia, PA.

NUMERISCHE INTEGRATION

9

Die numerische Berechnung bestimmter Integrale ist eine der ältesten Aufgaben der Mathematik: Sie liegt der Bestimmung des Inhalts krummliniger Flächen zugrunde. Das bekannteste solche Problem ist die Quadratur des Kreises (d. h. die Bestimmung der Seitenlänge desjenigen Quadrats, das den selben Flächeninhalt wie ein Kreis gegebenen Durchmessers hat). Aus diesem Grund wird die numerische Integration auch heute noch *Quadratur* genannt. Sie kommt auch zum Einsatz, falls eine Funktion integriert werden soll, für die eine Stammfunktion nicht bekannt oder schwierig auszuwerten ist. Als dritte, heutzutage wohl wichtigste, Anwendung ist wieder die numerische Lösung von Differentialgleichungen zu nennen. So kann zum Beispiel die Gleichung $y'(x) = f(y, x)$ äquivalent ausgedrückt werden als $y(x) = \int f(y, x)\,dx + C$. Wir betrachten daher folgende Aufgabe: Gegeben sei eine Funktion $f \in C([a, b])$ für $a < b \in \mathbb{R}$; gesucht ist ein Näherungswert $\tilde{I}$ für $I(f) := \int_a^b f(x)\,dx$.

Dabei berechnen wir diese Näherung wieder aus Funktionswerten an vorgegebenen Stützstellen.

Definition 9.1. Für *Stützstellen* $x_0, \ldots, x_n \in [a, b]$ und *Gewichte* $w_0, \ldots, w_n \in \mathbb{R}$ heißt

$$I_n(f) := \sum_{j=0}^{n} w_j f(x_j)$$

Quadraturformel (der Ordnung n*)*. Wir nennen sie *exakt vom Grad* m, falls gilt:

$$I_n(q) = \int_a^b q(x)\,dx \quad \text{für alle } q \in P_m.$$

Wir fordern also, dass Polynome vom Grad m durch die Quadraturformel exakt integriert werden. Dies ist wichtig bei der numerischen Integration von polynomialen Ansatzfunktionen, etwa in der Methode der Finiten Elemente, erlaubt aber in Kombination mit der Taylorentwicklung auch Aussagen über die Genauigkeit für andere Funktionen.

9.1 INTERPOLATIONSQUADRATUR: NEWTON–COTES-FORMELN

Eine nahe liegende Idee ist, die Stützstellen zur Bestimmung eines Interpolationspolynom zu verwenden, welches dann exakt integriert werden kann. Zu den Stützstellen $x_0, \dots, x_n$ betrachten wir also das Interpolationspolynom in Lagrangeform

$$p_n(x) = \sum_{j=0}^{n} f(x_j) l_{jn}(x),$$

und damit erhalten wir die Quadraturformel

$$I_n(f) := \int_a^b p_n(x)\,dx = \sum_{j=0}^{n} f(x_j) \int_a^b l_{jn}(x)\,dx.$$

Die Lagrangepolynome l_{jn} sind aus (8.1) bekannt; diese Integrale können also einmal berechnet und anschließend als Gewichte $w_j := \int_a^b l_{jn}(x)\,dx$ verwendet werden. Diese Quadraturformeln sind nach Konstruktion exakt vom Grad n, da wegen der Eindeutigkeit der Polynominterpolation jedes Polynom $q \in P_n$ sein eigenes Interpolationspolynom vom Höchstgrad n oder größer ist.

Für äquidistante Stützstellen mit $a = x_0 < x_n = b$ erhält man die *Newton–Cotes-Formeln*.[1] Wir betrachten zuerst die zwei einfachsten Beispiele, und untersuchen deren Exaktheit und Genauigkeit.

TRAPEZREGEL ($n = 1$) Der Einfachheit halber betrachten wir $a = 0$, $b = h$ und setzen $x_0 = 0$ und $x_1 = h$. Dann ist

$$I_1(f) = f(0) \int_0^h \frac{x-h}{0-h}\,dx + f(h) \int_0^h \frac{x}{h-0}\,dx = \frac{h}{2}(f(0) + f(h)).$$

Die Trapezregel ist exakt vom Grad 1. Für den *Quadraturfehler* folgt aus der Darstellung (8.6) des Interpolationsfehlers

$$\begin{aligned} |I(f) - I_1(f)| = |I(f) - I(p_1)| &= \left| \int_0^h f(x) - p_1(x)\,dx \right| \\ &= \left| \int_0^h \frac{1}{2} f''(\xi(x))(x-x_0)(x-x_1)\,dx \right| \leqslant \frac{1}{2}\|f''\|_\infty \left| \int_0^h x(x-h)\,dx \right| \\ &= \frac{1}{2}\|f''\|_\infty \frac{h^3}{6}. \end{aligned} \tag{9.1}$$

Je kleiner also das Integrationsintervall, desto genauer wird die Quadraturformel.

[1] Roger Cotes (1682–1716) war mit der Vorbereitung der zweiten Ausgabe von Newtons *Principia* betraut. Dabei arbeitete er Newtons Idee zur numerischen Integration aus und publizierte die Gewichte für die Quadraturformeln bis $n = 10$.

SIMPSON-REGEL ($n = 2$) Jetzt setzen wir $x_0 = 0$, $x_1 = \frac{h}{2}$ und $x_2 = h$. Wir erhalten dann analog die Simpson-Regel[2]

$$I_2(f) = \frac{h}{6}\left(f(0) + 4f\left(\frac{h}{2}\right) + f(h)\right).$$

Diese Formel ist sogar exakt vom Grad 3: Ist $q \in P_3$, und $p_2 \in P_2$ dessen Interpolationspolynom, so gilt

$$q(x) - p_2(x) = \frac{q'''}{6}(x-0)\left(x - \frac{h}{2}\right)(x-h),$$

wobei q''' nicht von x abhängt, da q ein kubisches Polynom ist. Der Interpolationsfehler erfüllt also:

$$I(q) - I_2(q) = \int_a^b \frac{q'''}{6} x\left(x - \frac{h}{2}\right)(x-h) = \frac{q'''}{6}\left[\frac{1}{4}x^4 - \frac{1}{2}hx^3 + \frac{1}{4}h^2x^2\right]_{x=0}^{x=h} = 0,$$

da der letzte Term sowohl für $x = 0$ als auch für $x = h$ verschwindet. Außerdem lässt sich zeigen, dass für beliebige f gilt:

$$|I(f) - I_2(f)| \leqslant \frac{1}{2880}\|f^{(4)}\|_\infty h^5.$$

Diese erhöhte Exaktheit und Fehlerordnung trifft auf alle Newton–Cotes-Formeln mit geradem n zu.

Allgemein lauten die Newton–Cotes Formeln für $\int_a^b f(x)\,dx$:

$$I_n(f) = (b-a)\sum_{j=0}^{n} w_j f(x_j) \qquad \text{für} \quad x_j = a + \frac{j}{n}(b-a).$$

Die Gewichte und Genauigkeiten der ersten vier Quadraturformeln sind in der folgenden Tabelle gesammelt:

n	Name	w_j	exakt	Fehler
1	Trapez	$\frac{1}{2}, \frac{1}{2}$	1	$\|f''\|_\infty \mathcal{O}(h^3)$
2	Simpson	$\frac{1}{6}, \frac{4}{6}, \frac{1}{6}$	3	$\|f^{(4)}\|_\infty \mathcal{O}(h^5)$
3	Newtons $\frac{3}{8}$	$\frac{1}{8}, \frac{3}{8}, \frac{3}{8}, \frac{1}{8}$	3	$\|f^{(4)}\|_\infty \mathcal{O}(h^5)$
4	Milne	$\frac{7}{90}, \frac{32}{90}, \frac{12}{90}, \frac{32}{90}, \frac{7}{90}$	5	$\|f^{(6)}\|_\infty \mathcal{O}(h^7)$

Ab $n = 8$ treten allerdings negative Gewichte w_j auf, so dass Auslöschung ein Problem werden kann.

[2] Thomas Simpson (1710–1761) war ein britischer Mathematiker und Autor zahlreicher, zu seiner Zeit beliebter, Lehrbücher. Die heute üblichen Bezeichnungen Sinus, Kosinus, Tangens und Kotangens wurden erstmals von ihm verwendet. Die nach ihm benannte, 1722 publizierte, Formel war allerdings schon Cavalieri (um 1639) bekannt.

9.2 SUMMIERTE QUADRATURFORMELN

Um eine höhere Genauigkeit zu erreichen ohne Quadraturformeln höherer Ordnung zu verwenden, kann man analog zur Spline-Interpolation das Integral stückweise berechnen.

Wir unterteilen dafür das komplette Integrations-Intervall $[a, b]$ in die N Teilintervalle $[x_0, x_1], [x_1, x_2], \ldots, [x_{N-1}, x_N]$, und wenden auf jedes Intervall eine Quadraturformel an. Definieren wir

$$I\left(f; [x_j, x_{j+1}]\right) := \int_{x_j}^{x_{j+1}} f(x)\,dx$$

und

$$I_n\left(f; [x_j, x_{j+1}]\right) := (x_{j+1} - x_j) \sum_{k=0}^{n} w_{j,k} f(t_{t,k})$$

(wobei $t_{j,k}$ die Stützstellen im Intervall $[x_j, x_{j+1}]$ und $w_{j,k}$ die entsprechenden Gewichte sind), so ist unsere *summierte Quadraturformel* gegeben durch

$$I_n^N(f) := \sum_{j=0}^{N-1} I_n\left(f; [x_j, x_{j+1}]\right).$$

Für jede Newton–Cotes-Formel kann man solch eine summierte Quadraturformel aufstellen. Wir betrachten als einfaches Beispiel die *summierte Trapezregel*. Wir nehmen wieder an, dass Intervalle die gleiche Länge haben, d. h. $x_{j+1} - x_j = h$ für alle $0 \leqslant j < N$. Dann ist

$$I_1\left(f; [x_j, x_{j+1}]\right) = \frac{h}{2}\left(f(x_j) + f(x_{j+1})\right),$$

und deshalb

$$I_1^N(f) = \sum_{j=0}^{N-1} \frac{h}{2}\left(f(x_j) + f(x_{j+1})\right) = h\left(\frac{1}{2}f(x_0) + f(x_1) + \cdots + f(x_{N-1}) + \frac{1}{2}f(x_N)\right).$$

Für den Gesamtfehler gilt wegen $(b - a) = Nh$

$$\begin{aligned} |I(f) - I_1^N(f)| &\leqslant \sum_{j=0}^{N-1} |I_n\left(f; [x_j, x_{j+1}]\right) - I\left(f; [x_j, x_{j+1}]\right)| \leqslant \sum_{j=0}^{N-1} \frac{h^3}{12}\|f''\|_\infty \\ &= \frac{h^3}{12}\|f''\|_\infty N = \frac{h^2}{12}\|f''\|_\infty (b - a). \end{aligned}$$

Durch die Summation verliert man also eine Fehlerordnung gegenüber der einfachen Quadraturformel; dafür kann man h (und damit den Fehler) durch feinere Unterteilung beliebig klein machen, ohne negative Gewichte zu bekommen. Die summierte Formel hat außerdem die gleiche Exaktheit wie die zugrunde liegende Newton–Cotes-Formel.

9.3 GAUSS-QUADRATUR

Die summierten Quadraturformeln erreichen zwar bei vergleichsweise wenig Aufwand eine beliebig hohe Genauigkeit, in manchen Anwendungen ist es jedoch wichtig, dass Polynome vorgegebenen Grades exakt integriert werden. Dies erfordert aber eine Quadraturformel entsprechend hohen Grades.[3] Um negative Gewichte zu vermeiden, sind äquidistante Stützstellen also nicht geeignet (man vergleiche das Runge-Phänomen bei der Polynominterpolation). Wir fragen uns also, welche Stützstellen (und Gewichte) wir verwenden sollten, um Polynome möglichst hohen Grades exakt zu integrieren.

Da wir nun für eine Quadraturformel I_n die $n+1$ Gewichte und $n+1$ Stützstellen (also $2n+2$ Freiheitsgrade) optimal wählen können, liegt die Vermutung nahe, dass wir Polynome vom Grad $2n+1$ exakt integrieren können. Das ist auch der höchstmögliche Grad: Angenommen, wir könnten mit den $n+1$ Stützstellen $x_0, \dots, x_n$ Polynome vom Grad $2n+2$ exakt integrieren. Dann gilt dies insbesondere für das Polynom

$$q(x) := (x - x_0)^2 \dots (x - x_n)^2 \in P_{2n+2}.$$

Da $q(x) > 0$ für $x \neq x_j$ gilt, ist wegen der Monotonie des Integrals auch

$$0 < \int_a^b q(x) = \sum_{j=0}^{n} w_j q(x_j) = 0,$$

was einen Widerspruch darstellt.

Wir suchen also Stützstellen $x_0, \dots, x_n \in [a, b]$ und Gewichte $w_0, \dots, w_n \in \mathbb{R}$ so, dass gilt

$$I_n(q) := \sum_{j=0}^{n} w_j q(x_j) = \int_a^b q(x)\,dx \quad \text{für alle } q \in P_{2n+1}.$$

Um die Bedingungen herzuleiten, die die Stützstellen erfüllen müssen, definieren wir wieder das Stützstellenpolynom $\omega_n(x) := (x - x_0) \dots (x - x_n) \in P_{n+1}$. Die Polynomdivision zeigt, dass jedes Polynom $q \in P_{2n+1}$ geschrieben werden kann als

$$q = \omega_n p + r \quad \text{mit } p, r \in P_n.$$

Setzen wir dies ein, erhalten wir

$$I(q) = \int_a^b \omega_n(x) p(x)\,dx + \int_a^b r(x)\,dx,$$
$$I_n(q) = \sum_{j=0}^{n} w_j(\omega_n(x_j)p(x_j)) + \sum_{j=0}^{n} w_j r(x_j).$$

[3] Freilich spricht nichts dagegen, die Ansätze dieses und des vorherigen Abschnitts zu kombinieren.

Nun ist $\omega_n(x_j) = 0$ für alle $0 \leqslant j \leqslant n$, der erste Term von $I_n(p)$ verschwindet also. Solange die Stützstellen $x_0, \dots, x_n$ paarweise verschieden sind, können wir wie in Abschnitt 9.1 die Gewichte $w_0, \dots, w_j$ so bestimmen, dass die Quadraturformel I_n (mindestens) exakt vom Grad n ist. Da $r \in P_n$ ist, gilt dann

$$I_n(r) = \sum_{j=0}^{n} w_j r(x_j) = \int_a^b r(x)\,dx.$$

Dies legt die Gewichte fest; für die Stützstellen erhalten wir damit die Bedingung

$$I(q) - I_n(q) = \int_a^b \omega_n(x)p(x)\,dx = 0 \qquad \text{für alle } q \in P_{2n+1}.$$

Anders gesagt müssen also die Stützstellen $x_0, \dots, x_n$ Nullstellen eines Polynoms ω_n sein, für das gilt

$$\int_a^b \omega_n(x)p(x)\,dx = 0 \qquad \text{für alle } p \in P_n. \tag{9.2}$$

Solche Polynome nennt man *Orthogonalpolynome*,[4] und sie sind sehr gründlich untersucht. Insbesondere weiß man, dass für gegebene $[a, b]$ solch ein Polynom in P_{n+1} eindeutig existiert und $n+1$ paarweise verschiedene, reelle Nullstellen hat. Diese kann man aus (9.2) bestimmen, indem man für p sukzessive alle Basisvektoren von P_n einsetzt.

Wir müssen also die Stützstellen x_j als Nullstellen eines Orthogonalpolynoms wählen, und dann die Gewichte w_j als Integrale über die zugehörigen Lagrange-Polynome berechnen.

Satz 9.2. *Zu $n \in \mathbb{N}$ und $a, b \in \mathbb{R}$ existieren $x_0, \dots, x_n \in [a, b]$ und $w_0, \dots, w_n \in \mathbb{R}$, so dass die zugehörige Interpolationsquadraturformel exakt ist vom Grad $2n + 1$.*

Aus der Exaktheit kann man wieder schliessen, das der Quadraturfehler (bei genügender Differenzierbarkeit von f) von der Ordnung $\mathcal{O}(h^{2n+3})$ ist.

Diese Quadraturformeln nennt man *Gauß-Formeln*. Für das Intervall $[a, b] = [-1, 1]$ erfüllen die so genannten *Legendre-Polynome* die Orthogonalitätsbedingung (9.2); ihre Nullstellen lassen sich in der Regel nur numerisch berechnen. Die ersten drei sind aber exakt bekannt sind und lauten

n	x_j	w_j	exakt
0	0	2	1
1	$-\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}$	1, 1	3
2	$-\sqrt{\frac{3}{5}}, 0, \sqrt{\frac{3}{5}}$	$\frac{5}{9}, \frac{8}{9}, \frac{5}{9}$	5

Formeln für beliebige Intervalle $[a, b]$ erhält man daraus durch Transformation auf das Intervall $[-1, 1]$ mit Hilfe der Substitutionsformel.

[4] Da sie senkrecht auf dem Raum der Polynome vom Grad n stehen bezüglich des Skalarprodukts $\langle p, q\rangle := \int_a^b p(x)q(x)\,dx$.

WEITERFÜHRENDE LITERATUR

G. Hämmerlin und K.-H. Hoffmann (1994). *Numerische Mathematik*. 4. Aufl. Springer-Verlag, Berlin.

NICHTLINEARE GLEICHUNGEN

10

Viele mathematische Modelle aus Physik (Mehrkörperprobleme in der Mechanik), Biologie (Populationswachstum) und Chemie (Reaktionskinetik) sind nicht linear. Wir betrachten daher nun die Aufgabe, für eine gegebene Funktion $f : [a, b] \to \mathbb{R}$ ein $x^* \in [a, b]$ zu bestimmen mit $f(x^*) = 0$. Wie das Beispiel $f(x) = x^5 - x + 1$ zeigt, kann es im Allgemeinen kein direktes Verfahren für die Lösung dieses Problems geben; wir müssen wieder iterative Verfahren betrachten.

Eine einfache Methode für skalare Funktionen $f : \mathbb{R} \to \mathbb{R}$ ist das *Bisektionsverfahren*, das auf der Tatsache beruht, dass eine stetige Funktion zwischen zwei Punkten mit unterschiedlichem Vorzeichen eine Nullstelle haben muss. Dieses Intervall, in dem die Nullstelle liegen muss, wird nun wiederholt halbiert:

Algorithmus 10.1 : Bisektionsverfahren

```
1 Wähle a_0, b_0 mit x* ∈ [a_0, b_0] und f(a_0)f(b_0) < 0
2 for k = 0, 1, ... do
3     x^(k) ← (a_k + b_k)/2
4     if f(x^(k))f(a_k) < 0 then
5         a_{k+1} ← a_k, b_{k+1} ← x^(k)
6     else
7         a_{k+1} ← x^(k), b_{k+1} ← b_k
```

In jedem Schritt wird dabei die Länge des Intervalls, in dem $x^{(k+1)}$ und x^* liegen müssen, um den Faktor 2 kleiner. Deshalb konvergiert die Iteration zwar lediglich linear, aber dafür unabhängig davon, wie weit a_0 und b_0 von x^* entfernt liegen. Die Erweiterung für Funktionen $f : \mathbb{R}^n \to \mathbb{R}$ ist allerdings nicht offensichtlich.[1]

[1] Allerdings enthalten viele kompliziertere Verfahren als Teilschritt eine *Schrittweitensuche* (wie etwa die Bestimmung von $\alpha^{(k)}$ im Gradientenverfahren), die mit dem Bisektionsverfahren durchgeführt werden kann.

10.1 FIXPUNKTITERATION

Eine sehr allgemeine Klasse[2] von Verfahren beruht auf der Formulierung der Nullstellensuche $f(x) = 0$ als *Fixpunktgleichung* $\Phi(x) = x$ für eine geeignete Funktion $\Phi(x)$. Wählen wir eine Funktion $g(x)$ mit $g(x^*) \neq 0$, so können wir zum Beispiel schreiben:

$$f(x^*) = 0 \quad \Leftrightarrow \quad x^* = x^* - g(x^*)f(x^*) =: \Phi(x^*). \tag{10.1}$$

(Oft sind nichtlineare Modelle, die etwa Gleichgewichtszustände beschreiben, auch direkt als Fixpunktpunktgleichungen gegeben.) Die Lösung dieser Fixpunktgleichung kann nun iterativ durch Fixpunktiteration bestimmt werden.

Algorithmus 10.2 : Fixpunktiteration

```
1 Wähle x^(0) ∈ [a, b]
2 for k = 0, 1, ... do
3     x^(k+1) ← Φ(x^(k))
```

Wir müssen uns nun fragen, in welchen Fällen diese Fixpunktiteration konvergiert, d. h. wann $\lim_{k\to\infty} x^{(k)} = x^*$ gilt. Der Banachsche Fixpunktsatz sagt uns, dass dies der Fall ist, wenn Φ eine *Kontraktion* ist.

Satz 10.1 (Banachscher Fixpunktsatz)**.** *Gilt* $\Phi \in C([a, b])$, $\Phi([a, b]) \subset [a, b]$, *und*

$$\|\Phi(x) - \Phi(y)\| \leqslant L|x - y| \quad \textit{für alle } x, y \in [a, b] \tag{10.2}$$

mit $L < 1$, *so konvergiert für jedes* $x^{(0)} \in [a, b]$ *die Fixpunktiteration gegen den (eindeutigen) Fixpunkt* x^* *von* Φ.

Beweis. Die Konvergenz der Fixpunktiteration folgt aus der Tatsache, dass die $x^{(k)}$ eine Cauchyfolge bilden. Aus der Iterationsvorschrift und der Bedingung (10.2) folgt nämlich, dass gilt

$$\|x^{(k+1)} - x^{(k)}\| = \|\Phi(x^{(k)}) - \Phi(x^{(k-1)})\| \leqslant L\|x^{(k)} - x^{(k-1)}\| \leqslant \cdots \leqslant L^k\|x^{(1)} - x^{(0)}\|.$$

Deshalb ist

$$\begin{aligned}\|x^{(m+k)} - x^{(k)}\| &\leqslant \|x^{(m+k)} - x^{(m+k-1)} + x^{(m+k-1)} - \cdots + x^{(k+1)} - x^{(k)}\| \\ &\leqslant \|x^{(m+k)} - x^{(m+k-1)}\| + \cdots + \|x^{(k+1)} - x^{(k)}\| \\ &\leqslant (L^{m+k} + \cdots + L^k)\|x^{(1)} - x^{(0)}\| = L^k\frac{1 - L^m}{1 - L}\|x^{(1)} - x^{(0)}\|.\end{aligned}$$

[2] Alle Ausführungen in diesem Abschnitt gelten unverändert, wenn man das Intervall $[a, b]$ durch ein Gebiet $D \subset \mathbb{R}^n$ ersetzt und die Begriffe der Norm und der Ableitung entsprechend anpasst.

Da $L < 1$ gilt, konvergiert $L^k \to 0$ für $k \to \infty$. Also gilt $\|x^{(m+k)} - x^{(k)}\| \to 0$ für alle $m \in \mathbb{N}$ und $k \to \infty$. Die Cauchyfolge $x^{(k)}$ konvergiert daher gegen einen Punkt $x^* \in [a, b]$.

Dass der Grenzwert x^* ein Fixpunkt von Φ ist, folgt aus der Stetigkeit von Φ:

$$x^* - \Phi(x^*) = x^* - \lim_{k\to\infty} \Phi(x^{(k)}) = x^* - \lim_{k\to\infty} x^{(k+1)} = x^* - x^* = 0.$$

Die Eindeutigkeit erhält man aus der Kontraktionsbedingung (10.2). Seien $x^* = \Phi(x^*)$ und $x^{**} = \Phi(x^{**})$, dann gil

$$\|x^* - x^{**}\| = \|\Phi(x^*) - \Phi(x^{**})\| \leqslant L\|x^* - x^{**}\|$$

mit $L < 1$. Dies kann aber nur gelten, wenn $\|x^* - x^{**}\| = 0$ ist. □

Für differenzierbare Funktionen kann man die Kontraktionsbedingung (10.2) recht einfach nachprüfen.

Folgerung 10.2. *Ist $\Phi \in C^1([a, b])$ mit $\Phi([a, b]) \subset [a, b]$, und gilt $\|\Phi'\|_\infty < 1$, so konvergiert die Fixpunktiteration für jeden Startwert $x^{(0)}$.*

Beweis. Aus dem Mittelwertsatz folgt für $x, y \in [a, b]$ sofort die Existenz eines $\xi \in [x, y]$ mit

$$\|\Phi(x) - \Phi(y)\| = \|\Phi'(\xi)(x - y)\| \leqslant \|\Phi'\|_\infty \|x - y\|. \qquad \square$$

Umgekehrt können wir damit auch entscheiden, wann eine Fixpunktiteration *nicht* konvergiert: Gilt im Fixpunkt $\|\Phi'(x^*)\| > 1$, so folgt aus der Stetigkeit von Φ', dass eine Umgebung U um x^* existiert, so dass $\|\Phi'(x)\| > 1$ gilt für alle $x \in U$. Dann liefert aber der Mittelwertsatz für $x^{(k)} \in U$ ein $\xi \in [x^{(k)}, x^*] \subset U$ mit

$$\|x^{(k+1)} - x^*\| = \|\Phi(x^{(k)}) - \Phi(x^*)\| = \|\Phi'(\xi)(x^{(k)} - x^*)\| > \|x^{(k)} - x^*\|.$$

Der Fehler wird also *größer*, sobald man in die Nähe von x^* kommt. Ein Fixpunkt x^*, für den $\Phi'(x^*) > 1$ ist, wird *abstoßend* genannt; die Fixpunktiteration wird niemals gegen solch einen Fixpunkt konvergieren. (Ist $\|\Phi'(x^*)\| = 1$, so kann man keine definitive Aussage treffen; eine Konvergenz ist jedoch in der Regel nicht zu erwarten.)

Falls die Fixpunktiteration konvergiert, ist auch ihre Konvergenzgeschwindigkeit interessant. Dazu benutzen wir den Begriff der Konvergenzordnung aus Definition 1.13. Um die Konvergenzordnung der Fixpunktiteration zu bestimmen, entwickeln wir $\Phi \in C^2([a, b])$ in eine Taylorreihe um x^*. Dann gilt für alle $x \in [a, b]$

$$\Phi(x) = \Phi(x^*) + \Phi'(x^*)(x - x^*) + \mathcal{O}(\|x - x^*\|^2).$$

Für $x = x^{(k)}$ folgt daraus

$$\|x^{(k+1)} - x^*\| = \|\Phi(x^{(k)}) - \Phi(x^*)\| \leqslant \|\Phi'(x^*)\|\|x^{(k)} - x^*\| + \mathcal{O}(\|x^{(k)} - x^*\|^2).$$

Wir halten fest.

(i) Die Fixpunktiteration wird in der Regel nur linear konvergieren. (Anschaulich: Der Fehler pro Iteration wird um den Faktor $c = \|\Phi'\|_\infty$ reduziert).

(ii) Gilt $\Phi'(x^*) = 0$, so wird die Iteration superlinear konvergieren, sobald $x^{(k)}$ nahe genug bei x^* liegt. Falls Φ zweimal stetig differenzierbar ist (man also obigen Fehlerterm in der Taylorentwicklung angeben kann), ist die Konvergenz sogar *quadratisch*. (Anschaulich: Die Anzahl korrekter Stellen verdoppelt sich in jedem Schritt.)

(iii) Ist $\|\Phi'(x^*)\| > 1$, so ist die Funktion Φ für die Fixpunktiteration ungeeignet; es gibt keine Konvergenz.

Ein Beispiel soll dies verdeutlichen:

Beispiel 10.3. Wir berechnen $\sqrt{2}$ als Nullstelle von $f(x) = x^2 - 2$ durch Fixpunktiteration. Jede der folgenden Funktionen hat $\sqrt{2}$ als Fixpunkt:

(i) Setze $\Phi : (0, \infty) \to (0, \infty)$, $\Phi(x) = \frac{x^3}{2}$. Dann ist aber $\Phi'(\sqrt{2}) = 3 > 1$; diese Fixpunktiteration konvergiert also nicht, da $\sqrt{2}$ ein abstoßender Fixpunkt ist.

(ii) Setze $\Phi : [1, 2) \to [1, 2)$, $\Phi(x) = 1 + x - \frac{1}{2}x^2$. Hier ist $\Phi'(x) = 1 - x \in (-1, 0]$ und damit $|\Phi'(x) < 1|$, weshalb die Fixpunktiteration für jedes $x^{(0)} \in [1, 2)$ konvergiert. Da $\Phi'(\sqrt{2}) = 1 - \sqrt{2} \neq 0$ ist, ist die Konvergenz lediglich linear.

(iii) Setze $\Phi : [1, 2] \to [1, 2]$, $\Phi(x) = \frac{1}{2}x + \frac{1}{x}$. Also ist $\Phi'(x) = \frac{1}{2} - \frac{1}{x^2}$, und $\|\Phi'\|_\infty = \frac{1}{2} < 1$. Diese Fixpunktiteration konvergiert deshalb für jeden Startwert $x^{(0)} \in [1, 2]$. Weiterhin ist $\Phi'(\sqrt{2}) = 0$, weshalb das Verfahren sogar quadratisch konvergiert.

Es liegt also auf der Hand, für die Nullstellenbestimmung die Iterationsfunktion Φ so zu wählen, dass $\Phi'(x^*) = 0$ erfüllt ist: Dies führt auf das Newton-Verfahren.

10.2 NEWTON-VERFAHREN

Zur Herleitung des Newton-Verfahrens zur Nullstellenbestimmung einer Funktion $f \in C^1([a, b])$ greifen wir wieder auf den Ansatz (10.1) zurück:

$$\Phi(x) = x - g(x)f(x),$$

wobei $g(x^*) \neq 0$ gelten soll. Wir bestimmen jetzt g so, dass $\Phi'(x^*) = 0$ ist und wir damit superlineare Konvergenz haben. Wir fordern also

$$0 = \Phi'(x^*) = 1 - g'(x^*)f(x^*) - g(x^*)f'(x^*) = 1 - g(x^*)f'(x^*).$$

Gilt $f'(x^*) \neq 0$, so können wir nach g auflösen und erhalten

$$g(x^*) = f'(x^*)^{-1}.$$

Da f' stetig ist, existiert eine Umgebung U um x^*, so dass $f'(x) \neq 0$ für alle $x \in U$ gilt. Innerhalb dieser Umgebung können wir also die Funktion

$$\Phi(x) = x - \frac{f(x)}{f'(x)} \tag{10.3}$$

betrachten, für die $f(x^*) = 0$ und $\Phi'(x^*) = 0$ gilt. In dieser Umgebung können wir also die Fixpunktiteration durchführen und erhalten daraus das *Newton-Verfahren*, das – wenn es konvergiert – nach Konstruktion superlinear konvergiert.

Aus der Stetigkeit von f' folgt nun, dass es eine Umgebung V von x^* gibt, so dass $\Phi'(x) < 1$ für alle $x \in V \cap U$ gilt. In dieser Umgebung $U \cap V$ konvergiert also die Fixpunktiteration nach Konstruktion superlinear; damit erhalten wir das *Newton-Verfahren*:

Algorithmus 10.3 : Newton-Verfahren

1 Wähle $x^{(0)}$ "hinreichend nahe" bei x^*
2 **for** $k = 0, 1, \ldots$ **do**
3 $\quad x^{(k+1)} \leftarrow x^{(k)} - \frac{f(x^{(k)})}{f'(x^{(k)})}$

Beispiel 10.4. Wir betrachten wieder $f(x) = x^2 - 2$. Das Newton-Verfahren ergibt die Iterationsvorschrift

$$x^{(k+1)} = x^{(k)} - \frac{(x^{(k)})^2 - 2}{2x^{(k)}} = x^{(k)} - \frac{x^{(k)}}{2} + \frac{1}{x^{(k)}} = \frac{1}{2}x^{(k)} + \frac{1}{x^{(k)}},$$

womit wir Beispiel 10.3 (iii) hergeleitet haben.[3]

Ist f zweimal stetig differenzierbar, konvergiert das Newtonverfahren, und das sogar sogar lokal quadratisch.

Satz 10.5. *Sei* $f \in C^2([a, b])$ *und gelte* $f(x^*) = 0$, $f'(x^*) \neq 0$ *für ein* $x^* \in [a, b]$. *Dann existiert eine Umgebung* U *von* x^*, *so dass für alle* $x^{(0)} \in U$ *die Iterierten* $x^{(k)}$ *im Newton-Verfahren gilt:*

$$x^* - x^{(k+1)} = \frac{1}{2}\frac{f''(\xi_k)}{f'(x^{(k)})}(x^* - x^{(k)})^2$$

für ein $\xi_k \in U$.

Beweis. Zuerst halten wir fest, dass wegen der Stetigkeit von f' und $f'(x^*) \neq 0$ eine Umgebung U_1 von x^* existiert, so dass $f'(x) \neq 0$ für alle $x \in U_1$ ist. Also gilt für Φ aus (10.3)

$$\Phi'(x) = \frac{f(x)f''(x)}{(f'(x))^2} \qquad \text{für alle} x \in U_1.$$

[3] Dieses Verfahren zur Wurzelberechnung war den Babyloniern schon lange vor Newton bekannt.

Aus der Stetigkeit von f, f', und f'' sowie $\Phi'(x^*) = 0$ folgt nun die Existenz einer Umgebung U_2 von x^* mit $\Phi'(x) < 1$ für alle $x \in U_2$. Aus Folgerung 10.2 folgt nun, dass das Newton-Verfahren auf $U := U_1 \cap U_2$ eine Kontraktion ist, d. h. für $x^{(0)} \in U$ gilt auch $x^{(k)} \in U$. Damit konvergiert das Newton-Verfahren lokal.

Für die Konvergenzordnung betrachten wir nun die Taylorentwicklung von f im Punkt $x^{(k)}$,

$$f(x) = f(x^{(k)}) + f'(x^{(k)})(x - x^{(k)}) + \frac{1}{2} f''(\xi_k)(x - x^{(k)})^2. \tag{10.4}$$

Für $x = x^*$ erhalten wir also

$$0 = f(x^*) = f(x^{(k)}) + f'(x^{(k)})(x^* - x^{(k)}) + \frac{1}{2} f''(\xi_k)(x^* - x^{(k)})^2.$$

Auflösen nach $x^{(k)} - x^*$ ergibt dann

$$x^{(k)} - \frac{f(x^{(k)})}{f'(x^{(k)})} - x^* = \frac{1}{2} \frac{f''(\xi_k)}{f'(x^{(k)})} (x^* - x^{(k)})^2,$$

woraus wegen $x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{f'(x^{(k)})}$ die Behauptung folgt. □

Für einen hinreichend guten Startwert wird das Newton-Verfahren also quadratisch gegen eine einfache Nullstelle einer zweimal stetig differenzierbaren Funktion konvergieren. Liegt eine mehrfache Nullstelle[4] vor, geht die superlineare Konvergenz verloren. Liegt der Startwert nicht nahe genug bei der gesuchten Nullstelle, kann man im Allgemeinen gar keine Konvergenz beobachten. Eine gute Anfangsnäherung für das Newton-Verfahren kann man zum Beispiel mit dem Bisektionsverfahren konstruieren, das global konvergent ist.

Das Newton-Verfahren hat auch eine einfache geometrische Interpretation. Betrachtet man die Taylorentwicklung (10.4) von f im Punkt $x^{(k)}$ mit $\xi_k \in [a, b]$ und vernachlässigt den quadratischen Term, so erhält man eine lineare Näherung an f im Punkt $x^{(k)}$,

$$y = f(x^{(k)}) + f'(x^{(k)})(x - x^{(k)}),$$

nämlich die Tangente an f im Punkt $(x^{(k)}, f(x^{(k)}))$. Ihr Schnittpunkt mit der x-Achse ist $y = 0$, d. h.

$$x^{(k+1)} := x^{(k)} - \frac{f(x^{(k)})}{f'(x^{(k)})},$$

welcher im Allgemeinen eine bessere Näherung an die Nullstelle x^* liefert.[5]

[4] Für eine allgemeine nichtlineare Funktion f ist die Mehrfachheit einer Nullstelle x^* definiert als die größte Zahl m, für die $f^{(k)}(x^*) = 0$, $0 \leqslant k \leqslant m$, gilt.

[5] Diese Idee der iterativen Linearisierung führte auch Newton auf das nach ihm benannte Verfahren. (Er entwickelte es 1669 zur Lösung einer kubischen Gleichung.)

Dieser Ansatz kann auch auf Funktionen $F : \mathbb{R}^n \to \mathbb{R}^n$ für $n > 1$ (d. h. Systeme von n nichtlinearen Gleichungen für n Unbekannte) verallgemeinert werden. Die lineare Näherung ist hier analog zur Taylorentwicklung (1.1)

$$F(x) = F(x^{(k)}) + \nabla F(x^{(k)})(x - x^{(k)}),$$

wobei $\nabla F(x) \in \mathbb{R}^{n\times n}$ die *Jacobi-Matrix* der partiellen Ableitungen bezeichnet. Ist diese Matrix nun invertierbar, so kann man dieses *lineare* Gleichungssystem – mit den Verfahren aus Teil II – nach $s^{(k)} := x - x^{(k)}$ lösen und dann $x^{(k+1)} := x^{(k)} + s^{(k)}$ setzen. Auch diese Iteration konvergiert (unter geeigneten Voraussetzungen an $F \in C^2$ und hinreichend gute Startwerte) lokal quadratisch, wobei der Beweis hier natürlich wesentlich aufwendiger ist.

WEITERFÜHRENDE LITERATUR

C. T. Kelley (1995). *Iterative methods for linear and nonlinear equations*. Bd. 16. Frontiers in Applied Mathematics. Society for Industrial und Applied Mathematics (SIAM), Philadelphia, PA.

C. T. Kelley (1999). *Iterative methods for optimization*. Bd. 18. Frontiers in Applied Mathematics. Society for Industrial und Applied Mathematics (SIAM), Philadelphia, PA.

Teil IV

NUMERISCHE LÖSUNG VON DIFFERENTIALGLEICHUNGEN

ANFANGSWERTPROBLEME

11

Gewöhnliche Differentialgleichungen charakterisieren Funktionen durch ihre (zeitliche) Änderungsrate. Wichtige Beispiele kommen aus der *Populationsdynamik* (zeitliche Änderung der Bevölkerungszahl), *Reaktionskinetik* (zeitliche Änderung von chemischen Konzentrationen) oder der *Mechanik* (zeitliche Änderung von Ort und Impuls).

11.1 THEORETISCHE GRUNDLAGEN

Zu einer gegebenen Funktion

$$f : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}, \qquad y^0 \in \mathbb{R}^n$$

(im Folgenden auch als *rechte Seite* bezeichnet), einer *Anfangszeit* $t_0 \in \mathbb{R}$, einem *Anfangswert* $y^0 \in \mathbb{R}^n$ und einer *Endzeit* $T \in (t_0, \infty]$ wird eine Funktion $y : \mathbb{R} \to \mathbb{R}^n$ gesucht, die das *Anfangswertproblem*

$$\begin{cases} y'(t) = f(t, y), & t \in (t_0, T], \\ y(t_0) = y^0, \end{cases} \tag{11.1}$$

erfüllt. Da nur die erste Ableitung von $y(t) = (y_1(t), \ldots, y_n(t))$ vorkommt, spricht man von einem *System erster Ordnung*; tauchen in der Gleichung auch m-te Ableitungen auf, liegt entsprechend ein System m-ter Ordnung vor. Diese lassen sich auf ein (größeres) System erster Ordnung zurückführen, indem man alle Ableitungen von y als neue Unbekannte einführt.

BEISPIEL: In einer idealisierten Feder ist die Rückstellkraft proportional zur Auslenkung. Da Kraft gleich Masse mal Beschleunigung ist, erfüllt die Auslenkung $x(t)$ die Differentialgleichung zweiter Ordnung

$$x''(t) = -\frac{k}{m} x(t),$$

wobei k die Federkonstante und m die Masse bezeichnet. Als Anfangsbedingungen müssen wir aus physikalischen Überlegungen Startposition $x(0) = x^0$ und Startimpuls $x'(0) = x^1$ vorschreiben, um eine eindeutige Lösung zu erhalten. Setzen wir jetzt

$$\begin{aligned} y_1(t) &:= x(t), \\ y_2(t) &:= x'(t), \end{aligned}$$

differenzieren jeweils auf beiden Seiten, und drücken überall die Ableitungen von x durch die entsprechenden y_i aus, so ergibt das

$$\begin{aligned} y_1'(t) &= y_2(t), & y_1(0) &= x^0, \\ y_2'(t) &= -\frac{k}{m} y_1(t), & y_2(0) &= x^1. \end{aligned}$$

Mit $f(t, (y_1, y_2)) = (y_2, -\frac{k}{m} y_1(t)), y^0 = (x^0, x^1)$ und $y = (y_1(t), y_2(t))$ erhalten wir wieder ein System erster Ordnung in der Form (11.1).

Der folgende zentrale Satz in der Theorie von Anfangswertproblem besagt, unter welchen Voraussetzungen das Problem (11.1) gut konditioniert ist.

Satz 11.1 (Picard–Lindelöf). *Sei* $f(t, y) : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^n$ *stetig in* t *und lokal Lipschitzstetig in* y*, d. h. es existieren* $\delta, L > 0$ *mit*

$$\|f(t, y) - f(t, z)\| \leqslant L\|y - z\|$$

für alle $y, z \in \mathbb{R}^n$ *in einer Umgebung von* y^0 *und alle* $t \in [t_0, t_0 + \delta]$*. Dann existiert eine eindeutige Lösung* $y(t)$, $t \in [t_0, t_0 + \varepsilon]$ *für ein* $\varepsilon > 0$*, von* (11.1)*. Weiterhin gilt für eine Lösung* y *zum Anfangswert* y^0 *und eine Lösung* $\tilde{y}$ *zum Anfangswert* $\tilde{y}^0$ *die Abschätzung*

$$\|y(t) - \tilde{y}(t)\| \leqslant e^{L(t-t_0)} \|y^0 - \tilde{y}^0\| \qquad \text{für alle } t \in [t_0, t_0 + \varepsilon]. \tag{11.2}$$

Ist die rechte Seite lokal Lipschitzstetig, so existiert also (lokal)[1] eine eindeutige Lösung, die stetig von den Anfangswerten abhängt. Wir nehmen daher in Folge an, dass die Voraussetzungen von Satz 11.1 erfüllt sind und eine Lösung auf dem gesamten Intervall $[0, T]$ existiert. Der Kürze halber setzen wir von nun an auch $t_0 = 0$.

Wir untersuchen nun die Stabilität des Anfangswertproblems für längere Zeiträume. Es ist zweckmäßig, dies durchzuführen am Beispiel der *Testgleichung*

$$y'(t) = \lambda y(t), \qquad \lambda \in \mathbb{C}. \tag{11.3}$$

Für $t \geqslant 0$ ist die Lösung gegeben durch

$$y(t) = e^{\lambda t} y(0), \tag{11.4}$$

[1] Stärkere Bedingungen, die für alle t und y, z gelten, erlauben freilich auch stärkere Aussagen über globale Existenz und Eindeutigkeit.

wie man leicht durch Differenzieren nachprüft. Wir betrachten nun zwei Anfangswerte y^0 und z^0 (letzteren darf man sich als "gestörten Anfangswert" vorstellen). Die zugehörigen Lösungen $y(t)$ und $z(t)$ der Testgleichung erfüllen dann für alle $t \geqslant 0$ die Abschätzung

$$\|y(t) - z(t)\| = \|e^{\lambda t}(y^0 - z^0)\| \leqslant e^{\Re(\lambda)t}\|y^0 - z^0\|,$$

wobei $\Re(\lambda)$ den Realteil der komplexen Zahl λ bezeichnet.

Wir können nun drei Fälle unterscheiden:

1. Gilt $\Re(\lambda) \leqslant 0$, so ist $e^{\Re(\lambda)t} \leqslant 1$ für alle $t \geqslant 0$, und die Anfangsstörung bleibt für alle Zeiten beschränkt; das Problem ist *stabil*.
2. Gilt $\Re(\lambda) < 0$, so geht $e^{\Re(\lambda)t} \to 0$ für $t \to \infty$, und die Differenz der Lösungen wird sogar immer kleiner; das Problem ist *asymptotisch stabil*.
3. Gilt $\Re(\lambda) > 0$, so geht $e^{\Re(\lambda)t} \to \infty$ für $t \to \infty$; ein beliebig kleiner Anfangsfehler wird also nach einer gewissen Zeit beliebig groß. Solch ein Problem ist *instabil*, und man kann nur für sehr kleine t eine sinnvolle Lösung erwarten.

Die Untersuchung allgemeiner Anfangswertprobleme $y'(t) = f(t, y)$ lässt sich (etwa durch Diagonalisierung linearer Systeme oder Linearisierung nichtlinearer Systeme) auf die Betrachtung von Testgleichungen dieser Art zurückführen. Die Testgleichung stellt damit einen geeigneten Prüfstein für die Stabilität numerischer Verfahren dar.

11.2 EINSCHRITTVERFAHREN: PROTOTYPEN UND FUNDAMENTALE KONZEPTE

Wir betrachten hier drei verschiedene Methoden zur numerischen Lösung des Anfangswertproblems (11.1), die beispielhaft die Konstruktion und Analyse solcher Verfahren aufzeigen.

Allen Ansätzen liegt die Idee zugrunde, die gesuchte Lösung für $t \in [t_0, T]$ durch eine Folge von diskreten Funktionswerten zu approximieren. Dafür definieren wir die *Zeitschritte*

$$0 = t_0 < t_1 < \cdots < t_N = T$$

mit der *Schrittweite*

$$h_j := t_j - t_{j-1}.$$

Die numerische Näherung der exakten Lösung $y(t)$ bezeichnen wir mit

$$y_j \approx y(t_j), \qquad j = 0, \ldots, N.$$

Da $y_0 \approx y(t_0) = y^0$ bekannt ist, brauchen wir nun eine Vorschrift, um aus y_{j-1} den Wert y_j zu konstruieren; man spricht daher von *Zeitschrittverfahren*. Diese bekommen wir durch *Diskretisierung* der Differentialgleichung.

METHODE 1: Die Ableitung $y'(t)$ wird durch den Vorwärtsdifferenzenquotienten in t_{j-1} ersetzt:

$$\frac{y(t_j) - y(t_{j-1})}{h_j} \approx y'(t_{j-1}) = f(t_{j-1}, y(t_{j-1})).$$

Durch Ersetzen von $y(t_j)$ durch die diskrete Näherung y_j und Auflösen erhalten wir daraus das *explizite Eulerverfahren*

$$\begin{aligned} y_j &= y_{j-1} + h_j f(t_{j-1}, y_{j-1}), \qquad j = 1, \dots, N, \\ y_0 &= y^0. \end{aligned}$$

METHODE 2: Die Ableitung $y'(t)$ wird durch den Rückwärtsdifferenzenquotienten in t_j ersetzt:

$$\frac{y(t_j) - y(t_{j-1})}{h_j} \approx y'(t_j) = f(t_j, y(t_j)).$$

Analog erhalten wir daraus das *implizite Eulerverfahren*

$$\begin{aligned} y_j &= y_{j-1} + h_j f(t_j, y_j), \qquad j = 1, \dots, N, \\ y_0 &= y^0. \end{aligned}$$

Um bei bekanntem y_{j-1} den neuen Wert y_j zu bestimmen, muss dabei die Gleichung (bzw. das Gleichungssystem) $y_j - h_j f(t_j, y_j) = y_{j-1}$ gelöst werden. Im Gegensatz dazu ist im expliziten Eulerverfahren y_j direkt durch Einsetzen von y_{j-1} in die rechte Seite gegeben. Allgemein nennt man Zeitschrittverfahren *implizit*, falls in jedem Zeitschritt ein Gleichungssystem gelöst werden muss, und *explizit*, falls das nicht erforderlich ist.

METHODE 3: Integrieren wir die Differentialgleichung $y' = f(t, y)$ auf beiden Seiten über einen Zeitschritt und wenden den Hauptsatz der Differential- und Integralrechnung an, erhalten wir die äquivalente Integralgleichung

$$y(t_j) = y(t_{j-1}) + \int_{t_{j-1}}^{t_j} f(t, y(t))\, dt. \tag{11.5}$$

Indem wir das Integral auf der rechten Seite durch eine Quadraturformel approximieren, erhalten wir eine weitere Klasse von Zeitschrittverfahren. Die Trapezregel

$$\int_{t_{j-1}}^{t_j} f(t, y(t))\, dt \approx \frac{t_j - t_{j-1}}{2} \Big(f(t_{j-1}, y(t_{j-1})) + f(t_j, y(t_j)) \Big)$$

führt auf das *Trapezverfahren*

$$\begin{aligned} y_j &= y_{j-1} + \frac{h_j}{2} \Big(f(t_{j-1}, y_{j-1}) + f(t_j, y_j) \Big), \qquad j = 1, \dots, N, \\ y_0 &= y^0. \end{aligned}$$

Auch das Trapezverfahren ist implizit, da y_j auch in der rechten Seite vorkommt.

Alle drei Methoden sind Beispiele von *Einschrittverfahren*, da nur der letzte Zeitschritt t_{j-1} zur Berechnung von t_j verwendet wird. Allgemein haben Einschrittverfahren die Form

$$\begin{cases} y_j = y_{j-1} + h_j \Phi_f(t_{j-1}, t_j, y_{j-1}, y_j), & j = 1, \dots, N, \\ y_0 = y^0. \end{cases} \tag{11.6}$$

mit einer *Verfahrensfunktion* Φ_f, deren Definition natürlich von der gegebenen rechten Seite f abhängt.[2]

Eine wesentliche Frage ist nun, wie genau diese Verfahren sind, wie gut also die mit ihnen berechnete diskrete Näherung y_j die exakte Lösung $y(t_j)$ annähert. Wir betrachten zuerst den lokalen Fehler, der in *einem* Schritt gemacht wird.

KONSISTENZ beschreibt, wie gut das diskrete Zeitschrittverfahren die gewöhnliche Differentialgleichung approximiert; wir untersuchen hier also den *Diskretisierungsfehler*. Um das zu präzisieren, schreiben wir das Einschrittverfahren (11.6) in der Form

$$\Psi_h y_j := \frac{y_j - y_{j-1}}{h_j} - \Phi_f(t_{j-1}, t_j, y_{j-1}, y_j) = 0$$

und untersuchen, wie gut die exakte Lösung $y(t)$ diese Gleichung erfüllt.[3]

Definition 11.2. Der *lokale Fehler* ist definiert als

$$d_j := \|\Psi_h y(t)\| = \left\| \frac{y(t_j) - y(t_{j-1})}{h_j} - \Phi_f(t_{j-1}, t_j, y(t_{j-1}), y(t_j)) \right\|.$$

Ein Verfahren hat *Konsistenzordnung* p, falls für alle $1 \leqslant j \leqslant N$ gilt

$$d_j = \mathcal{O}(h_j^p).$$

Es heißt *konsistent*, falls es mindestens Konsistenzordnung $p = 1$ besitzt.

Da es sich hier um einen Diskretisierungsfehler handelt, können wir die Konsistenzordnung direkt an der zur Konstruktion des Verfahrens verwendeten Approximation ablesen:

(i) *Explizites Eulerverfahren*: Hier ist

$$\Psi_h y(t) = \frac{y(t_j) - y(t_{j-1})}{h_j} - f(t_{j-1}, y(t_{j-1})) = \frac{y(t_j) - y(t_{j-1})}{h_j} - y'(t_{j-1}) = \mathcal{O}(h_j),$$

wie in Beispiel 8.6 gezeigt wurde. Die Konsistenzordnung ist daher $p = 1$.

[2] Für das explizite Eulerverfahren etwa ist $\Phi_f(t_{j-1}, t_j, y_{j-1}, y_j) = f(t_{j-1}, y_{j-1})$.

[3] Vergleichen Sie dies mit der Verwendung des Residuums $b - Ax^{(k)}$ zur Fehlermessung im Gradientenverfahren. Da die diskrete Näherung y_j nur an den Zeitschritten t_j definiert ist, ist eine Ableitung nicht definiert, und wir können y_j nicht in die Differentialgleichung einsetzen. Deshalb wird hier umgekehrt die exakte Lösung in das numerische Verfahren eingesetzt.

(ii) *Implizites Eulerverfahren*: Hier liegt auch die Konsistenzordnung $p = 1$ vor, da der Rückwärtsdifferenzenquotient die selbe Fehlerordnung besitzt.

(iii) *Trapezverfahren*: Mit Hilfe der Quadraturfehlerabschätzung (9.1) erhalten wir aus (11.5), dass gilt

$$y(t_j) = y(t_{j-1}) + \frac{h_j}{2}\Big(f(t_{j-1}, y(t_{j-1})) + f(t_j, y(t_j))\Big) + \mathcal{O}(h_j^3).$$

Umformen und durch Division durch h_j ergibt

$$\Psi_h y(t) = \frac{y(t_j) - y(t_{j-1})}{h_j} - \frac{1}{2}\Big(f(t_{j-1}, y(t_{j-1})) + f(t_j, y(t_j))\Big) = \mathcal{O}(h_j^2).$$

Das Trapezverfahren hat daher Konsistenzordnung $p = 2$.

KONVERGENZ beschreibt dagegen, wie nahe die diskrete Lösung an der exakten Lösung liegt. Im Gegensatz zu dem lokalen Fehler müssen wir hier alle bisherigen Zeitschritte berücksichtigen.

Definition 11.3. Der *globale Fehler* ist definiert als

$$e_j := \|y(t_j) - y^j\|.$$

Ein Verfahren hat *Konvergenzordnung* p, falls für alle $1 \leqslant j \leqslant N$ und $h := \max_{1\leqslant j\leqslant N} h_j$ gilt

$$e_j = \mathcal{O}(h^p).$$

Es heißt *konvergent*, falls es mindestens Konvergenzordnung $p = 1$ besitzt.

Der globale Fehler misst also zusätzlich, wie sich die Diskretisierungsfehler über die Zeitschritte aufsammeln.

NULLSTABILITÄT bedeutet, dass die Fehler in früheren Schritten durch die folgenden Schritte nicht übermässig verstärkt werden, der globale Fehler also durch den lokalen Fehler bestimmt wird.

Dazu definieren wir in den Zeitschritten $t_0, \dots, t_N$ zwei Folgen von Werten

$$x_0, \dots, x_N \qquad \text{und} \qquad z_0, \dots, z_N,$$

und setzen wieder $h := \max_{1\leqslant j\leqslant N}(t_j - t_{j-1})$.

Definition 11.4. Ein Verfahren heißt *nullstabil*, falls $h_0 > 0$ und $K > 0$ existieren, so dass für alle $x_0, \dots, x_N$ und $z_0, \dots, z_N$ mit $h \leqslant h_0$ die Abschätzung

$$\|x_n - z_n\| \leqslant K\Big[\|x_0 - z_0\| + \max_{1\leqslant j\leqslant n} \|\Psi_h x_j - \Psi_h z_j\|\Big] \tag{11.7}$$

für alle $1 \leqslant n \leqslant N$ gilt.

Satz 11.5. *Für ein nullstabiles Einschrittverfahren gilt:*

(i) Aus Konsistenz folgt Konvergenz;

(ii) Die Konsistenzordnung ist gleich der Konvergenzordnung.

Beweis. Setzen wir in (11.7) für x_j die exakte Lösung $y(t_j)$ und für z_j die diskrete Lösung y_j ein, so erhalten wir sofort aus der Definition des globalen und des lokalen Fehlers die Abschätzung

$$e_j \leqslant K \max_{1 \leqslant j \leqslant n} d_j. \qquad \square$$

Die Abschätzung (11.7) ist eine diskrete Form von (11.2). Tatsächlich kann man zeigen, dass ein Verfahren nullstabil ist, wenn Φ_f Lipschitzstetig bezüglich y_j (und y_{j-1}) ist. Da wir f als Lipschitzstetig vorausgesetzt haben, erkennen wir sofort, dass beide Eulerverfahren sowie das Trapezverfahren nullstabil sind. Die Eulerverfahren haben also auch Konvergenzordnung $p = 1$, während das Trapezverfahren Konvergenzordnung $p = 2$ hat.

A-STABILITÄT ist die Forderung, dass das numerische Verfahren, angewandt auf die Testgleichung (11.3), eine Näherung liefert, die das gleiche Abklingverhalten wie die exakte Lösung (11.4) liefert. Wir verlangen also, dass für $\Re(\lambda) \leqslant 0$ die diskrete Lösung

$$\|y_j\| \leqslant \|y_{j-1}\|, \qquad \text{für } 1 \leqslant j \leqslant N \tag{11.8}$$

erfüllt. Wir betrachten das am Beispiel des expliziten Eulerverfahrens:

$$\begin{aligned} y_j &= y_{j-1} + h_j f(t_{j-1}, y_{j-1}) = y_{j-1} + h_j \lambda y_{j-1} \\ &= (1 + h_j \lambda) y_{j-1}. \end{aligned}$$

Damit (11.8) erfüllt ist, muss also

$$|1 + h\lambda| \leqslant 1$$

mit $h = \max_j h_j$ gelten. Je kleiner also $\Re(\lambda) < 0$, desto kleiner muss die maximale Zeitschrittweite sein, um die gewünschte monoton fallende Näherung zu garantieren. Auch für andere Verfahren spielt das Produkt $h\lambda$ eine Rolle, was die folgende Definition rechtfertigt:

Definition 11.6. Der *absolute Stabilitätsbereich* eines Zeitschrittverfahrens ist

$$A = \{z \in \mathbb{C} : \text{ für } \lambda h = z \text{ gilt (11.8)}\}.$$

Das Verfahren heißt *A-stabil*, falls $\{z \in \mathbb{C} : \Re(z) \leqslant 0\} \subset A$ gilt.

A-stabile Verfahren produzieren für alle stabilen Testgleichungen monoton fallende Näherungen, unabhängig von der gewählten Schrittweite.

i) Für das *explizites Eulerverfahren* haben wir bereits gezeigt, dass

$$A = \{z \in \mathbb{C} : |1 + z| \leqslant 1\}$$

(der Kreis um -1 mit Radius 1) ist. Das explizite Eulerverfahren ist also *nicht* A-stabil.

ii) Für das *implizite Eulerverfahren* erhält man

$$(1 - h_j\lambda)y_j = y_{j-1},$$

und daraus die Bedingung

$$\left|\frac{1}{1 - h\lambda}\right| \leqslant 1.$$

Der absolute Stabilitätsbereich ist daher

$$A = \{z \in \mathbb{C} : |1 - z| \geqslant 1\}$$

(alles außerhalb des Kreises um 1 mit Radius 1). Das implizite Eulerverfahren ist also A-stabil.

iii) Das *Trapezverfahren* kann man umformen zu

$$\left(1 - \frac{h_j}{2}\lambda\right) y_j = \left(1 + \frac{h_j}{2}\lambda\right) y_{j-1},$$

woraus folgt:

$$A = \left\{z \in \mathbb{C} : \left|\frac{1 - \frac{z}{2}}{1 + \frac{z}{2}}\right| \leqslant 1\right\} = \{z \in \mathbb{C} : \mathfrak{R}(z) \leqslant 0\},$$

da der Zähler genau dann kleiner als der Nenner ist, wenn $\mathfrak{R}(z) \leqslant 0$ gilt. Auch das Trapezverfahren ist damit A-stabil.

Die A-Stabilität ist besonders wichtig für *steife Differentialgleichungen*. Diese sind charakterisiert durch ein extrem rasches "Einschwingen" auf einen stationären Zustand, unabhängig vom Anfangswert. Betrachten wir zum Beispiel für $y^0 > 0$ das Anfangswertproblem

$$y'(t) = -100(y(t) - \sin(t)), \qquad y(0) = y^0,$$

so sehen wir, dass sich für sehr kleine t die rechte Seite wie $f(t, y) \approx -100y(t)$ verhält, die Lösung $y(t)$ also exponentiell abfällt. Sobald aber $y(t) \approx \sin(t)$ erreicht wird, ist $f(t, y) \approx 0$ und daher $y(t) \approx \sin(t)$. Für große t reichen also sehr große Zeitschritte, um die langsam variierende Lösung korrekt zu berechnen. Trotzdem ist die schnell abklingende Komponente noch vorhanden, sodass sie über den absoluten Stabilitätsbereich die maximale Zeitschrittweite mitbestimmt. Wegen $-100 \ll 0$ muss diese daher sehr klein sein, obwohl für den Zweck der Genauigkeit eine viel größere Schrittweite ausreichen würde. Hier ist also ein A-stabiles Verfahren vorteilhaft.

Allgemein kann man zeigen, dass für explizite Verfahren der absolute Stabilitätsbereich $A = \{z \in \mathbb{C} : |g(z)| \leqslant 1\}$ durch ein Polynom $g(z)$ bestimmt ist. Polynome können aber prinzipiell die Exponentialfunktion nur für kleine $|z|$ gut annähern.[4] Dagegen führen implizite Verfahren

[4] Es gilt $e^x \to 0$ für $x \to -\infty$, aber $p(x) \to \pm\infty$ für jedes Polynom.

auf rationale Funktionen $g(z)$, die auch das asymptotische Verhalten der Exponentialfunktion wiedergeben können.

Merke: Explizite Verfahren sind für steife Differentialgleichungen ungeeignet!

(Dies wird auch oft als pragmatische *Definition* der Steifheit verwendet.)

WEITERFÜHRENDE LITERATUR

E. Hairer, S. P. Nørsett und G. Wanner (1993). *Solving ordinary differential equations. I*. 2. Aufl. Bd. 8. Springer Series in Computational Mathematics. Nonstiff problems. Springer-Verlag, Berlin.

E. Hairer und G. Wanner (2010). *Solving ordinary differential equations. II*. 2. Aufl. Bd. 14. Springer Series in Computational Mathematics. Stiff and differential-algebraic problems. Springer-Verlag, Berlin. DOI: 10.1007/978-3-642-05221-7.

A. Iserles (2009). *A first course in the numerical analysis of differential equations*. 2. Aufl. Cambridge Texts in Applied Mathematics. Cambridge University Press, Cambridge.

R. LeVeque (2007). *Finite Difference Methods for Ordinary and Partial Differential Equations*. Society for Industrial und Applied Mathematics. DOI: 10.1137/1.9780898717839.

ZWEIPUNKT-RANDWERTPROBLEME

12

Wir betrachten das folgende Problem: Gesucht ist $y \in C^2(a,b)$ mit

$$\begin{aligned} y''(t) &= f(t, y(t), y'(t), y''(t)) \qquad t \in [a,b], \\ y(a) &= y_a, \\ y(b) &= y_b. \end{aligned}$$

Da wir nicht zwei Anfangswerte, sondern die Werte in den beiden Randpunkten a und b haben, können wir diese Differentialgleichung zweiter Ordnung nicht mehr auf ein System von Differentialgleichungen erster Ordnung zurückführen. Es handelt sich um ein *Randwertproblem*. Ist in den Randpunkten der Funktionswert von y vorgeschrieben, spricht man von *Dirichlet-Bedingungen*; ist dagegen die Ableitung von y gegeben, handelt es sich um *Neumann-Bedingungen*. (Schreibt man in einem Punkt den Funktionswert, im anderen die Ableitung vor, so liegen *gemischte Randbedingungen* vor.)

Die eindeutige Lösbarkeit des Randwertproblems hängt dabei von der Art der Randbedingungen ab. So kann man leicht nachprüfen, dass für jedes $c \in \mathbb{R}$ die konstante Funktion $y(t) \equiv c \in \mathbb{R}$ Lösung des Neumann-Problems $y''(t) = 0, y'(a) = y'(b) = 0$ ist. Wir werden im weiteren annehmen, dass eine eindeutige Lösung existiert.

Im folgenden soll anhand eines einfachen Modellproblems eine der wichtigsten Methoden zur Lösung von Randwertproblemen vorgestellt werden: Die Methode der Finiten Elemente. Da diese nicht nur für gewöhnliche, sondern auch für partielle Differentialgleichungen von Bedeutung ist, passen wir die Notation entsprechend an.

Wir suchen also $u \in C^2(0,1)$, so dass

$$\begin{aligned} -(c(x)u'(x))' &= f(x) \qquad x \in (0,1), \\ u(0) &= 0, \\ u(1) &= 0, \end{aligned} \tag{12.1}$$

wobei $c(x) \in C^1(0,1)$ eine gegebene Funktion ist, die folgende (wichtige) Bedingung erfüllt:

$$c(x) \geqslant c_0 > 0 \qquad \text{für alle } x \in [0,1]. \tag{12.2}$$

Gleichung (12.1) beschreibt zum Beispiel die Temperaturverteilung in einem unendlich dünnen Stab mit dem Wärmeleitkoeffizienten c, der an beiden Enden auf 0 Grad gekühlt und im Inneren durch die Wärmequelle f geheizt wird.

12.1 SCHWACHE FORMULIERUNG

Wir bringen zuerst das Randwertproblem in eine günstigere Form, indem wir die Differentialgleichung (12.1) mit einer *Testfunktion* $v \in C^1(0,1)$ multiplizieren und dann über $(0,1)$ integrieren:

$$\begin{aligned}\int_0^1 f(x)v(x)\,dx &= -\int_0^1 (c(x)u'(x))'v(x)\,dx \\ &= -[c(1)u'(1)v(1) - c(0)u'(0)v(0)] + \int_0^1 c(x)u'(x)v'(x)\,dx.\end{aligned}$$

Fordern wir zusätzlich, dass auch v die Bedingung $v(0) = v(1) = 0$ erfüllt, so fällt der erste Term weg, und wir erhalten

$$\int_0^1 c(x)u'(x)v'(x)\,dx = \int_0^1 f(x)v(x)\,dx.$$

Hier kommt keine zweite Ableitung von u mehr vor. Akzeptieren wir als Lösung des Randwertproblems nun ein $u \in C^1(0,1)$, das die Gleichung für alle $v \in C^1(0,1)$ mit $v(0) = v(1) = 0$ erfüllt, so nennt man dies *Variationsformulierung*.[1] Dieser Lösungsbegriff lässt sich sogar noch weiter abschwächen, indem man sich überlegt, welche minimalen Anforderungen die Definition von Integral und Ableitung erfüllen müssen, um für möglichst viele Randwertprobleme eine (sinnvolle) Lösung zu erhalten. Dies ist wichtig, um etwa auch stückweise konstante Wärmeleitkoeffizienten (für die (12.1) nicht definiert ist) behandeln zu können.

Für den Integralbegriff führen diese Überlegungen auf das Lebesgue-Integral und damit auf den Funktionenraum

$$L^2(0,1) = \left\{ f : [0,1] \to \mathbb{R} \;:\; \int_0^1 |f(x)|^2\,dx \leqslant \infty \right\},$$

wobei natürlich das Lebesgue-Integral zu Grunde liegt. In diesem Vektorraum definiert man nun die Norm

$$\|f\|_{L^2}^2 = \int_0^1 |f(x)|^2\,dx$$

[1] Tatsächlich führen viele mathematische Modelle zuerst auf solche Variationsformulierungen, bevor in einem zweiten Schritt daraus ein Randwertproblem hergeleitet wird.

und das Skalarprodukt

$$\langle f, g\rangle = \int_0^1 f(x)g(x)\,dx.$$

Die Ableitungen kommen in der Variationsformulierung nur im Integranten vor, und zwar multipliziert mit Testfunktionen. Wir benötigen also nur einen Ableitungsbegriff, der in diesem Fall die partielle Integration erlaubt.

Definition 12.1. Die Funktion $w \in L^2(0,1)$ heißt *schwache Ableitung* von $f \in L^2(0,1)$, falls gilt

$$\int_0^1 w(x)\varphi(x)\,dx = -\int_0^1 f(x)\varphi'(x)\,dx \qquad \text{für alle } \varphi \in C^1(0,1) \text{ mit } \varphi(0) = \varphi(1) = 0.$$

Offensichtlich erfüllt eine klassische Ableitung diese Definition; sie existiert aber für eine grössere Klasse von Funktionen. Zum Beispiel gilt für die Funktion $f : (0,1) \to \mathbb{R}$, $f(x) = |x - \frac{1}{2}|$, und eine beliebige Testfunktion φ

$$\begin{aligned}
-\int_0^1 f(x)\varphi'(x)\,dx &= -\int_0^{\frac{1}{2}} -(x-\tfrac{1}{2})\varphi'(x)\,dx - \int_{\frac{1}{2}}^1 (x-\tfrac{1}{2})\varphi'(x)\,dx \\
&= -\left[(\tfrac{1}{2}-\tfrac{1}{2})\varphi(\tfrac{1}{2})\right] + \int_0^{\frac{1}{2}} (-1)\varphi(x)\,dx \\
&\quad -\left[-(\tfrac{1}{2}-\tfrac{1}{2})\varphi(\tfrac{1}{2})\right] + \int_{\frac{1}{2}}^1 (1)\varphi'(x)\,dx \\
&=: \int_0^1 w(x)\varphi(x)\,dx,
\end{aligned}$$

Weiterhin gilt für

$$w(x) = \begin{cases} 1 & x \geqslant \frac{1}{2}, \\ -1 & x < \frac{1}{2}, \end{cases}$$

auch

$$\|w\|_{L^2}^2 = \int_0^1 |w(x)|^2\,dx = \int_0^1 1\,dx = 1 < \infty,$$

und damit ist w schwache Ableitung von f. Allgemein kann man so zeigen, dass stetige und bis auf einzelne Punkte stetig differenzierbare Funktionen eine schwache Ableitung besitzen.

Wir bezeichnen die schwache Ableitung von f, wenn sie existiert, ebenfalls mit f'. Den Raum der Funktionen, die eine schwache Ableitung in $L^2(0,1)$ besitzen und zusätzlich die Randbedingungen erfüllen, bezeichnet man mit

$$H_0^1(0,1) = \left\{ f \in L^2(0,1) \,:\, f' \in L^2(0,1), f(0) = f(1) = 0 \right\}.$$

Die zugehörige Norm ist

$$\|f\|_{H^1}^2 = \int_0^1 |f'(x)|^2 \, dx. \tag{12.3}$$

Für alle Funktionen in $H_0^1(0,1)$ gilt die *Poincaré-Ungleichung*

$$\|f\|_{L^2} \leqslant C_P \|f\|_{H^1}. \tag{12.4}$$

Hier sind die Null-Randbedingungen wichtig, denn sonst wäre die Ungleichung für konstante Funktionen nicht erfüllt. Die einzige konstante Funktion in $H_0^1(0,1)$ ist aber die Nullfunktion, für die beide Seiten gleich Null sind. Der Übersichtlichkeit halber führen wir noch die Schreibweise

$$a(u,v) := \int_0^1 c(x) u'(x) v'(x) \, dx$$

ein. Da die Ableitung und das Integral linear sind, ist a eine *Bilinearform*: Hält man v fest, ist $a(u,v)$ eine lineare Funktion in u, und umgekehrt.

Damit erhalten wir die *schwache Formulierung* des Randwertproblems.

Definition 12.2. Die Funktion $u \in H_0^1(0,1)$ heißt *schwache Lösung* von (12.1), wenn gilt

$$a(u,v) = \langle f, v \rangle \qquad \text{für alle } v \in H_0^1(0,1). \tag{12.5}$$

Hier ist nun – im Unterschied zur Variationsformulierung – das Lebesgue-Integral und die schwache Ableitung gemeint. Eine schwache Lösung ist im Allgemeinen keine klassische Lösung von (12.1).

12.2 GALERKIN-VERFAHREN

Um eine diskrete Näherung für die schwache Lösung u von (12.5) berechnen, schränken wir unsere Suche auf einen endlichdimensionalen Unterraum V_h von H_0^1 ein, und verlangen lediglich, dass die Variationsformulierung für alle Testfunktionen in diesem Unterraum erfüllt ist. Mit Hilfe einer Basis von V_h werden wir sehen, dass dies auf endlich viele Gleichungen für endlich viele Koeffizienten führt. Man bezeichnet dieses Vorgehen als *Galerkin-Ansatz*, und es ist in gleicher Weise zur numerischen Lösung von partiellen Differentialgleichungen anwendbar.

Wir suchen also ein $u_h \in V_h$, für das gilt

$$a(u_h, v_h) = \langle f, v_h \rangle \qquad \text{für alle } v_h \in V_h. \tag{12.6}$$

Die Tatsache, dass u_h die selbe Gleichung wie u erfüllt (nur eben für eine kleinere Klasse von Testfunktionen) und ebenfalls in $H_0^1(0,1)$ ist, ist hierbei fundamental.

Wir zeigen zuerst, dass (12.6) eine eindeutige Lösung $u_h \in V_h$ hat. Da V_h die Dimension $n < \infty$ hat, existiert eine Basis $\{\varphi_1, \dots, \varphi_n\}$ von V_h. Damit lässt sich u_h darstellen als

$$u_h(x) = \sum_{i=1}^{n} u_i \varphi_i(x) \tag{12.7}$$

mit dem Koeffizientenvektor $(u_1, \dots, u_n)^T \in \mathbb{R}^n$. Außerdem genügt es, als Testfunktionen die Basiselemente φ_i für $i = 1, \dots, n$ zu betrachten, da jedes weitere v_h aufgrund der Linearität von (12.6) in v_h keine linear unabhängige Gleichung liefern würde. Einsetzen von (12.7) in den Galerkin-Ansatz (12.6) liefert also

$$\sum_{i=1}^{n} u_i a(\varphi_i, \varphi_j) = \langle f, \varphi_j \rangle, \qquad j = 1, \dots, n.$$

Der Galerkin-Ansatz führt damit auf ein lineares Gleichungssystem $Az = b$ für die (offensichtlich symmetrische) Matrix $A \in \mathbb{R}^{n \times n}$ mit $a_{ij} := a(\varphi_i, \varphi_j)$, den Vektor $b := (b_1, \dots, b_n)^T \in \mathbb{R}^n$ mit $b_j = \langle f, \varphi_j \rangle$ und die Lösung $z := (u_1, \dots, u_n)^T$.[2]

Wir zeigen jetzt dass dieses Gleichungssystem eine eindeutige Lösung besitzt, indem wir nachweisen, dass A positiv definit ist. Für einen beliebigen Vektor $y = (y_1, \dots, y_n) \in \mathbb{R}^n$ definieren wir

$$y_h := \sum_{i=1}^{n} y_i \varphi_i.$$

Dann gilt aufgrund der Bilinearität von a

$$\begin{aligned} y^T A y = \sum_{i,j=1}^{n} y_i a_{ij} y_j = \sum_{i,j=1}^{n} y_i a(\varphi_i, \varphi_j) y_j = a\left(\sum_{i=1}^{n} y_i \varphi_i, \sum_{i=1}^{n} y_j \varphi_j \right) \\ = a(y_h, y_h) = \int_0^1 c(x) y_h'(x) y_h'(x)\, dx \geqslant c_0 \|y_h\|_{H^1}^2 \end{aligned}$$

wegen (12.2) und der Definition (12.3). Aufgrund der Normeigenschaft (siehe Definition 1.1, (N1)) folgt daraus, dass $y^T A y \geqslant 0$ für alle $y \in \mathbb{R}^n$ gilt, und $y^T A y = 0$ dann und nur dann, wenn $y_h = 0$ und damit $y = 0$ ist. A ist also positiv definit und deshalb invertierbar, und die eindeutige Lösung x des linearen Gleichungssystems liefert uns über die Darstellung (12.7) die eindeutige Lösung von (12.6).

Als nächstes untersuchen wir, ob diese Lösung stabil ist, kleine Änderungen in f also nur kleine Änderungen in u_h erzeugen. Sei u_h eine Lösung zu f und $\tilde{u}_h$ eine Lösung zu $\tilde{f}$. Ziehen wir beide Gleichungen voneinander ab, so sehen wir, dass die Differenz der Lösungen $w_h := u_h - \tilde{u}_h$ Lösung ist der Gleichung

$$a(w_h, v_h) = \langle f - \tilde{f}, v_h \rangle \qquad \text{für alle } v_h \in V_h.$$

[2] Aus der Anwendung in der Elastizitätstheorie haben sich die Namen *Steifigkeitsmatrix* für A und *Lastvektor* für b eingebürgert.

Es genügt also zu zeigen, dass die Norm der Lösung von (12.6) durch die der rechten Seite abgeschätzt werden kann. Dazu setzen wir als Testfunktion v_h die Lösung $u_h \in V_h$ ein, und verwenden wieder die Abschätzung durch die $H^1_0(0,1)$-Norm und erhalten

$$c_0 \|u_h\|^2_{H^1} \leqslant a(u_h, u_h) = \langle f, u_h \rangle \leqslant \|f\|_{L^2} \|u_h\|_{L^2} \leqslant C_P \|f\|_{L^2} \|u_h\|_{H^1},$$

wobei wir zuerst die Cauchy-Schwarzsche-Ungleichung und dann die Poincaré-Ungleichung (12.4) verwendet haben. Wenn $u_h \neq 0$ ist, dürfen wir durch $\|u_h\|_{H^1}$ dividieren, und erhalten

$$\|u_h\|_{H^1} \leqslant \frac{C_p}{c_0} \|f\|_{L^2}.$$

(Aufgrund der Eindeutigkeit der Lösung kann $u_h = 0$ nur für $f = 0$ gelten, und dann ist die Ungleichung trivialerweise erfüllt.)

Die Lösung u_h kann also für kleine Daten f nicht beliebig groß werden. Die gerade bewiesene Existenz, Eindeutigkeit und Stabilität ist ein Spezialfall des *Satzes von Lax–Milgram*.

Wir untersuchen nun den *Diskretisierungsfehler* $\|u - u_h\|$. Da sowohl u als auch u_h die selbe Gleichung erfüllen, können wir in der schwachen Formulierung (12.5) als Testfunktion ein beliebiges $v_h \in V_h \subset H^1_0(0,1)$ einsetzen; es gilt also

$$a(u, v_h) = \langle f, v_h \rangle \qquad \text{für alle } v_h \in V_h.$$

Subtrahieren wir von dieser Gleichung den Galerkin-Ansatz (12.6), erhalten wir die sogenannte *Galerkin-Orthogonalität*

$$a(u - u_h, v_h) = 0 \qquad \text{für alle } v_h \in V_h.$$

(Der Fehler $e := u - u_h$ steht also senkrecht auf dem Unterraum V_h.) Da mit u und u_h auch e in V liegt, können wir wieder abschätzen

$$\begin{aligned} c_0 \|e\|^2_{H^1} &\leqslant a(e,e) = a(e, u - w_h) + a(e, \underbrace{w_h - u_h}_{\in V_h}) \\ &= a(e, u - w_h) = \int_0^1 c(x) e'(x) (u - w_h)'(x)\, dx \\ &\leqslant \max_{x \in [0,1]} |c(x)| \, \|e\|_{H^1} \|u - w_h\|_{H^1}. \end{aligned}$$

Dabei haben durch Addition mit der "produktiven Null" ein beliebiges $w_h \in V_h$ eingefügt und die Galerkin-Orthogonalität ausgenutzt. Da diese Abschätzung für alle w_h gilt, gilt sie auch für das Minimum, und wir erhalten

$$\|u - u_h\|_{H^1} \leqslant \frac{C}{c_0} \min_{w_h \in V_h} \|u - w_h\|_{H^1}. \tag{12.8}$$

Der Diskretisierungsfehler ist also nur durch den *Approximationsfehler* beschränkt, d. h. dadurch, wie gut wir die Lösung u überhaupt mit Funktionen in V_h annähern können. Dies ist wieder ein Spezialfall eines allgemeineren Resultats, des *Céa-Lemmas*.

Bislang haben wir uns noch nicht darum kümmern müssen, wie der Unterraum V_h definiert ist. Um den Approximationsfehler weiter abschätzen zu können, ist das aber notwendig. Eine spezielle Wahl von V_h sind die Finiten Elemente.

12.3 METHODE DER FINITEN ELEMENTE

Die *Methode der Finiten Elemente* ("finite element method", FEM) entsteht aus dem Galerkin-Verfahren, indem der Unterraum V_h durch stückweise Polynome konstruiert wird. Wir betrachten hier nur den einfachsten Fall. Wir definieren dafür eine *Zerlegung* $x_0, \dots, x_n$ des Intervalls $[0, 1]$,

$$0 = x_0 < x_1 < \cdots < x_n = 1,$$

mit $h_j := x_{j+1} - x_j$ und $h = \max_j h_j$. Dann betrachten wir die *linearen Splines*, die die vorgeschriebenen Randbedingungen erfüllen:

$$V_h := \left\{ f \in C(0,1) \,:\, f|_{[x_j, x_{j+1}]} \in P_1, f(0) = f(1) = 0 \right\}.$$

Die Funktionen in V_h sind nach Konstruktion stetig, und auf jedem Teilstück $[x_j, x_{j+1}]$ eine lineare Funktion und damit stetig differenzierbar. Es gilt also wie gefordert, dass $V_h \subset H^1_0(0,1)$ ist.

Wir können nun den Approximationsfehler abschätzen. Dafür genügt es, den *Interpolationsfehler* zu betrachten, denn $\min_{w_h \in V_h} \|u - w_h\|_{H^1}$ ist sicher kleiner als $\|u - I_h u\|_{H^1}$, wenn $I_h u \in V_h$ derjenige lineare Spline ist, der u in den Knoten $x_0, \dots, x_n$ interpoliert. Indem man den Fehler auf jedem Teilintervall $[x_j, x_{j+1}]$ separat betrachtet, kann man dort die Fehlerdarstellung (8.6) verwenden und damit durch Integration und Differentiation die Abschätzungen

$$\|u - I_h u\|_{L^2} \leqslant \frac{1}{2} h^2 \|u''\|_{L^2},$$
$$\|u - I_h u\|_{H^1} \leqslant \frac{1}{2} h \|u''\|_{L^2},$$

herleiten, falls $u'' \in L^2(0,1)$ ist; vgl. Kapitel 9.2 und 8.5. Ist die schwache Lösung u also glatt genug, folgt daraus und aus dem Céa-Lemma (12.8) die Fehlerabschätzung

$$\|u - u_h\|_{H^1} \leqslant Ch \|u''\|_{L^2}. \tag{12.9}$$

Insbesondere gilt $\|u - u_h\|_{H^1} \to 0$ für $h \to 0$; für immer feinere Zerlegungen konvergiert also die Galerkin-Näherung u_h gegen die schwache Lösung u.

Für das Aufstellen der Steifigkeitsmatrix A müssen wir jetzt eine Basis von V_h angeben. Dafür bestimmen wir zuerst die Dimension von V_h, indem wir untersuchen, wieviele Freiheitsgrade ein $f \in V_h$ hat. Wir betrachten zunächst das Intervall $[x_0, x_1]$. Dort ist $f|_{[x_0,x_1]}$ ein lineares

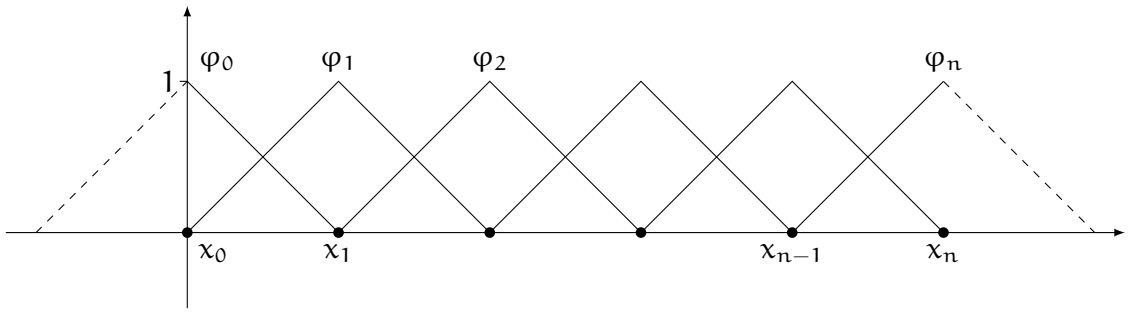


Abbildung 12.1: Die Hut-Funktionen φ_i für $i = 0, \dots, n$ im Intervall $[x_0, x_n]$.

Polynom, hat also 2 Freiheitsgrade, von denen aber einer durch die Bedingung $f(0) = 0$ festgelegt ist. Das nächste Teilstück $f|_{[x_1,x_2]}$ ist auch ein Polynom vom Grad 1, muss aber in x_1 die Bedingung erfüllen, dass

$$\left(f|_{[x_0,x_1]}\right)(x_1) = \left(f|_{[x_1,x_2]}\right)(x_1)$$

ist. Übrig bleibt also nur noch ein Freiheitsgrad. Die gleiche Überlegung gilt für $f|_{[x_i,x_{i+1}]}$ im Punkt x_i für $2 \leqslant i \leqslant n-2$. Im letzten Teilstück $f|_{[x_{n-1},x_n]}$ muss zusätzlich die Randbedingung $f(x_n) = 1$ erfüllt sein, so das kein Freiheitsgrad übrigbleibt. Insgesamt ergibt dies $(n-1)$ Freiheitsgrade.

Wir definieren nun für $0 \leqslant i \leqslant n$ die *linearen B-Splines* (oder *Hut-Funktionen*)

$$\varphi_i(x) := \begin{cases} \frac{x-x_{i-1}}{x_i-x_{i-1}} & \text{für } x_{i-1} < x \leqslant x_i, \\ \frac{x_{i+1}-x}{x_{i+1}-x_i} & \text{für } x_i < x \leqslant x_{i+1}, \\ 0 & \text{sonst,} \end{cases}$$

siehe Abbildung 12.1. Durch Einsetzen erkennt man leicht, dass gilt

$$\varphi_i(x_j) = \begin{cases} 1 & \text{falls } i = j, \\ 0 & \text{falls } i \neq j. \end{cases}$$

Wir können also (analog zu den Legendre-Polynomen) jedes $f \in V_h$ als Linearkombination der φ_i darstellen. Die Bedingungen $f(0) = f(1) = 0$ legen dabei die Koeffizienten für φ_0 und φ_n bereits als Null fest. Übrig bleiben die $n-1$ linear unabhängigen Funktionen $\varphi_1, \dots, \varphi_{n-1}$, die die gewünschte Basis von V_h sind.

Für die praktische Realisierung ist nützlich, dass die B-Splines einen kompakten Träger haben: nur benachbarte Hut-Funktionen können in einem Punkt von Null verschieden sein. Genauer gilt

$$\varphi_i(x)\varphi_j(x) = 0 \text{ falls } |i-j| > 1.$$

Daraus folgt sofort, dass $a_{ij} = a(\varphi_i, \varphi_j) = 0$ für $|i - j| > 1$; die Steifigkeitsmatrix A ist also dünn besetzt und insbesondere tridiagonal. Außerdem ist φ' konstant auf jedem Intervall, was die Berechnung von a_{ij} vereinfacht.

Zum Abschluss skizzieren wir kurz das praktische Vorgehen zur Lösung des Randwertproblems (12.1) mit der Methode der Finiten Elemente:

1. Konstruiere eine Zerlegung $\{x_0, \dots, x_n\}$ von $[0, 1]$, für die h klein genug ist, um nach (12.9) die gewünschte Genauigkeit zu erreichen.
2. Für jedes Intervall $[x_k, x_{k+1}]$ werden alle i, j bestimmt, für die $\varphi_i \varphi_j$ von Null verschieden ist.
3. Durch Transformation auf das Intervall $[0, 1]$ und Gauß-Quadratur (mit genügend hoher Ordnung, damit φ_j und $\varphi_i' \varphi_j'$ exakt integriert wird) berechnet man dann die Integrale $\int_{x_k}^{x_{k+1}} c(x)\varphi_i'(x)\varphi_j'(x)\,dx$ und $\int_{x_k}^{x_{k+1}} f(x)\varphi_j(x)\,dx$.
4. Durch Aufsummieren über alle k erhält man dadurch a_{ij} und b_j, $j = 1, \dots, n-1$.
5. Das tridiagonale, symmetrisch positiv definite lineare Gleichungssystem löst man je nach Größe mit einer Cholesky-Zerlegung oder dem CG-Verfahren.
6. Als Lösung erhält man den Vektor $(u_1, \dots, u_{n-1})$ mit $u_i = u_h(x_i) \approx u(x_i)$.

Dieses Vorgehen lässt sich effizient in Software realisieren.

WEITERFÜHRENDE LITERATUR

W. Zulehner (2008). *Numerische Mathematik: eine Einführung anhand von Differentialgleichungsproblemen. Band 1. Stationäre Probleme*. Mathematik Kompakt. [Compact Mathematics]. Birkhäuser Verlag, Basel.

Zum Thema Finite Elemente für partielle Differentialgleichungen:

D. Braess (2013). *Finite Elemente*. 5. Aufl. Theorie, schnelle Löser und Anwendungen in der Elastizitätstheorie. Springer.

S. C. Brenner und L. R. Scott (2008). *The mathematical theory of finite element methods*. 3. Bd. 15. Texts in Applied Mathematics. Springer, New York.

Software-Pakete:

M. S. Alnæs u. a. (2015). *The FEniCS Project Version 1.5*. Archive of Numerical Software 3.100. DOI: 10.11588/ans.2015.100.20553.

W. Bangerth, R. Hartmann und G. Kanschat (2016). *deal.II Differential Equations Analysis Library, Technical Reference*. URL: http://www.dealii.org/8.4.0.

ANHANG

EINE KURZE EINFÜHRUNG IN MATLAB

A

A.1 ALLGEMEINES

MATLAB (Matrix Laboratory) ist eine interaktive Umgebung für wissenschaftliche numerische Berechnung und Visualisierung. Die besondere Stärke für unsere Zwecke liegt in dem sehr einfachen *Debugging*: sämtliche Zwischenergebnisse lassen sich leicht anzeigen und manipulieren.[1]

Wichtigstes Merkmal: MATLAB kennt nur einen Datentypen: (reelle und komplexe) Matrizen. Das bedeutet:

- *Skalare* sind 1×1-Matrizen
- *Zeilenvektoren* der Dimension n sind $1 \times n$-Matrizen
- *Spaltenvektoren* der Dimension n sind $n \times 1$-Matrizen

A.1.1 DIE KOMMANDOZEILE

Startet man MATLAB, so sieht man zuerst eine *Kommandozeile* (>>, Octave: `octave:1>`). Dort kann man:

- *Befehle* eingeben: `sin(1)`, `exp(-1)`, `sqrt(2)`. Das Ergebnis wird sofort angezeigt. Schließt man einen Befehl mit `;` ab, so wird die Ausgabe unterdrückt. Mit `,` kann man mehrere Befehle in einer Zeile verbinden, ohne dass die Ausgabe unterdrückt wird.
- *Variablen zuweisen*: `x = cos(pi);`, `null = 4-4;`. Die so zugewiesenen Variablen können in folgenden Zeilen verwendet werden: `y=x*2;`. Der Befehl `who` zeigt alle bekannten Variablen an. Der Wert einer Variable kann durch Eingabe des Variablennamens ausgegeben werden: `x`, `y`. Der Befehl `clear x` löscht die Variable `x` aus dem Arbeitsspeicher.
- *Hilfe erhalten*: `help befehl` zeigt eine kurze Dokumentation zu `befehl`.

[1] Alle Ausführungen hier gelten – wo nicht anders angegeben – unverändert auch für OCTAVE

- *Laufzeiten messen*: `tic; befehl; toc` führt `befehl` aus und gibt die dafür nötige Rechenzeit aus. (Diese kann auch mit `t = toc` einer Variablen zugewiesen werden.)
- Mit den Pfeiltasten *auf* und *ab* lassen sich frühere Befehle zurückholen.
- MATLAB rechnet intern mit *double precision*; dies läßt sich auch nicht ändern. Jedoch kann das Ausgabeformat mit dem Befehl `format long` beeinflusst werden (siehe `help format`).

A.1.2 SKRIPTE

Möchte man eine Serie von Befehlen später wieder verwenden, so kann man sie in Form eines *Skripts* speichern. Dazu gibt man die Befehle im *Editor-Fenster*[2] ein, und speichert sie mit der Erweiterung `.m` ab (etwa `meinskript.m`).

- Skripte lassen sich wie Befehle aufrufen: `meinskript;`. Dafür müssen sie im aktuellen Arbeitsverzeichnis liegen. Das aktuelle Verzeichnis erhält man durch Eingabe von `pwd`, durch `cd verzeichnis` wechselt man in das Unterverzeichnis `verzeichnis`. Den Verzeichnisinhalt gibt `ls` aus.
- Skripte können mit Kommentaren versehen werden: Alles, was auf ein `%` folgt, wird von MATLAB ignoriert.
- Skripte haben Zugriff auf sämtliche Variablen, die auf der Kommandozeile (oder in früheren Skripten) definiert wurden. Genauso sind nach Beendigung eines Skriptes sämtliche Variablen, die im Skript definiert wurden (und Änderungen an bereits definierten Variablen) auf der Kommandozeile verfügbar.

A.1.3 FUNKTIONEN

Eine besondere Art von Skripten sind *Funktionen*, die neue Befehle definieren. Um ein Skript als Funktion zu deklarieren, gibt man als *erste Zeile* ein:

```
function [out1,out2,...,outn] = meinefunktion(in1,in2,...,inm)
```

Diese Funktion sollte dann als `meinefunktion.m` abgespeichert werden. Von der Kommandozeile (bzw. aus Skripten oder weiteren Funktionen) lässt sich die Funktion durch

```
[out1,out2,...,outn] = meinefunktion(in1,in2,...,inm);
```

[2] Octave bringt keinen eigenen Editor mit; Skripte lassen sich aber mit jedem Texteditor (wie Notepad) erstellen. Komfortabler ist ein Editor, der farbliche Hervorhebung von Schlüsselwörtern beherrscht, unter Windows z. B. Atom.

aufrufen. Gibt es nur eine Ausgabevariable, so können die eckigen Klammern weggelassen werden:

```
out = meinefunktion(in);
```

Im Unterschied zu Skripten sind bei Beginn der Funktionsausführung nur die Eingabevariablen `in1,...inm` bekannt; genauso sind nach Durchführung nur noch die Ausgabevariablen verfügbar. Auch sind alle Änderungen, die die Funktion an den Eingabevariablen durchführt, nach der Ende der Funktion "vergessen".

Wird in der Funktion der Befehl `keyboard` eingefügt, so wird die aktuelle Funktion angehalten, sobald der Befehl erreicht wird. Man kann dann auf der Kommandozeile den aktuellen Stand sämtlicher Variablen anzeigen und verändern. Der Ablauf der Funktion wird durch den Befehl `return` fortgesetzt.

A.2 MATRIZEN

Für die *Erzeugung* von Matrizen stehen folgende Möglichkeiten zur Verfügung:

- `A=[1,2,3;4,5,6;7,8,9]` erzeugt
$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$
Die Eingabe erfolgt also zeilenweise: Kommata (oder Leerzeichen) trennen Spalten, Strichpunkte Zeilen.
- `zeros(m,n)` erzeugt eine Matrix mit `m` Zeilen und `n` Spalten, die nur `0` als Einträge hat.
- `ones(m,n)` erzeugt eine Matrix mit `m` Zeilen und `n` Spalten, die nur `1` als Einträge hat.
- `rand(m,n)` erzeugt eine Matrix mit `m` Zeilen und `n` Spalten, die als Einträge gleichverteilte Zufallszahlen zwischen `0` und `1` hat.
- `eye(m,n)` erzeugt eine Matrix mit `m` Zeilen und `n` Spalten, die auf der Diagonale `1` und sonst `0` als Einträge hat. (Insbesondere erzeugt `eye(n,n)` die `n`×`n` Einheitsmatrix.)
- `i:j` erzeugt den Zeilenvektor `[i,i+1,...,j-1,j]`.

Der einfache *Zugriff* auf Matrixelemente ist die Stärke von MATLAB:

- `A(i,j)` ist das *Element* der Matrix `A` in der `i`-ten Zeile und `j`-ten Spalte. Für Spalten- oder Zeilenvektoren `x` liefert `x(i)` die `i`-te *Komponente*. (Achtung: `A(i)` liefert den `i`-ten Eintrag der Matrix `A`, spaltenweise gezählt (d. h. `A(2)`=4 im obigen Beispiel).)
- `A(i1:i2,j1:j2)` ist die *Submatrix* aus der `i1`-ten bis `i2`-ten Zeile sowie der `j1`-ten bis `j2`-ten Spalte.

- `A(:,j)` ist die komplette `j`-te *Spalte* von `A`; genauso ist `A(i,:)` ist die komplette `i`-te *Zeile* von `A`. (Mehrere Spalten bzw. Zeilen lassen sich mit `A(:,j1:j2)` bzw. `A(i1:i2,:)` auswählen.)
- `size(A,1)` bzw. `size(A,2)` ist die *Zeilen-* bzw. *Spaltenanzahl* von `A`; `size(A)` liefert den Zeilenvektor [*Zeilenanzahl, Spaltenanzahl*].
- `length(x)` gibt die Anzahl der Elemente des (Zeilen- oder Spalten-)Vektors `x` aus.
- Der Index `end` bezeichnet den *größtmöglichen Index*: `x(end)` ist die letzte Komponente des (Zeilen- oder Spalten-)Vektors `x`; `A(:,end)` ist die letzte Spalte, `A(end,:)` die letzte Zeile der Matrix `A`.

Die folgenden *Operationen* sind für Matrizen definiert:

- `+,-,*` bezeichnet die Addition, Subtraktion bzw. Multiplikation von Matrizen (bzw. Vektoren). Dabei müssen natürlich die Dimensionen der Operanden stimmen!
- `.*`, `./` bezeichnet die *komponentenweise* Multiplikation bzw. Division von Matrizen (bzw. Vektoren). Beispiel: `[1 2 3].*[1 2 3]` liefert `[1 4 9]`.
- `^` ist die (Matrix-)*Potenz* (`A^2` entspricht der Matrixmultiplikation `A*A`); die komponentenweise Potenz ist `.^` (`[1 2 3].^2` liefert also `[1 4 9]`.).
- `A'` ist die *Transponierte* der Matrix `A` (bzw. die hermitesch-konjugierte Matrix, falls `A` komplex ist). Insbesondere erzeugt `(i:j)'` einen Spaltenvektor.
- `A\b` (der *Backslash-Operator*) gibt die Lösung `x` des linearen Gleichungssystems `Ax=b` aus.
- Die eingebauten Funktionen wie `sin` und `exp` arbeiten komponentenweise, falls ihnen eine Matrix bzw. ein Vektor übergeben wird. Beispiel: `exp([0 1 2])` liefert `[1.0000 2.7183 7.3891]`.

A.3 ABLAUFSTEUERUNG

Wie die meisten höheren Programmiersprachen bietet auch MATLAB:

- *Bedingte Anweisungen*:

```
if (AUSDRUCK)
    ANWEISUNGEN
else
    ANWEISUNGEN2
end
```

führt den Block ANWEISUNGEN aus, falls AUSDRUCK wahr ist, ansonsten ANWEISUNGEN2. Für AUSDRUCK wird in der Regel ein *Vergleich* stehen, etwa x==y, x<y, x>=y oder x~=y (letzteres steht für $x \neq y$). Mehrere Ausdrücke können per && (logisches *und*) bzw. || (logisches *oder*) verknüpft werden.

- *Schleifen*:

```
for VARIABLE = VEKTOR
    ANWEISUNGEN
end
```

  führt den Block ANWEISUNGEN wiederholt aus, wobei VARIABLE der Reihe nach als Wert die Komponenten von VEKTOR zugewiesen bekommt. Beispiel: `for i = 1:n`

- *Bedingte Schleifen*:

```
while (AUSDRUCK)
    ANWEISUNGEN
end
```

  führt den Block ANWEISUNGEN aus, solange AUSDRUCK wahr ist. Dabei wird AUSDRUCK *vor* jedem Schleifendurchlauf geprüft, insbesondere wird der Block ANWEISUNGEN gar nicht ausgeführt, wenn AUSDRUCK vor Beginn der Schleife falsch ist.

- *Vorzeitiger Abbruch:* Die Anweisung `return` verlässt die aktuelle Schleife, bedingte Anweisung oder Funktion.

A.4 GRAPHISCHE AUSGABE

- `plot(x,y)` zeichnet einen *Graphen*, wobei für die x-Achse die Komponenten des Vektors `x` und für die y-Achse die Komponenten des Vektors `y` verwendet werden (beide müssen die selbe Länge haben). `plot(x,y1,x,y2)` zeichnet sowohl `y1` als auch `y2` in Abhängigkeit von `x`, `plot(y)` ist gleichbedeutend mit `plot(1:length(y),y)`.
- `surf(z)` zeichnet die Matrix `z` als zweidimensionale Funktion; die Höhe des "Funktionswerts" an der Stelle `(i,j)` entspricht dabei `z(i,j)`.
- `spy(A)` zeigt die von Null verschiedenen Einträge der Matrix `A` an.