Numerische Lineare Algebra

Stefan Funken
Karsten Urban

Universität Ulm

Wintersemester 2023/24

Version 16. Februar 2024

Vorwort

Dieses Manuskript kann und soll nicht ganz den Wortlaut der Vorlesung wiedergeben und kann den Besuch von Vorlesungen und Übungen nicht ersetzen. Es soll das Nacharbeiten des Inhalts der Vorlesung erleichtern. Notwendig zum Verstehen der Vorlesung sind Kenntnisse aus der Vorlesung „Lineare Algebra“ und Programmierkenntnisse, bevorzugt in MATLAB®. Dazu stellen wir im Anhang einige der Konzepte zusammen, von denen im Skript besonders häufig Gebrauch gemacht wird.

Wie zu jeder Vorlesung ist das Studium der Literatur unerläßlich zum vertieften Verständnis der vorgestellten Inhalte. Da es sich bei „Numerische Lineare Algebra“ um eine Standardvorlesung handelt, gibt es zahlreiche (Lehr-)Bücher zu diesem Thema. Wir nennen hier nur einige:

- Deuflhard und Hohmann: *Numerische Mathematik 1*, 2019: [DH19].
- Stoer: *Numerische Mathematik. 1*, 1994: [Sto94],
  Stoer und Bulirsch: *Numerische Mathematik. 2*, 1990: [SB90],
  siehe auch: Freund und Hoppe: *Stoer/Bulirsch: Numerische Mathematik 1*, 2007: [FH07].
- Richter und Wick: *Einführung in die Numerische Mathematik - Begriffe, Konzepte und zahlreiche Anwendungsbeispiele*, 2017: [RW17].
- Quarteroni, Sacco und Saleri: *Numerical mathematics*, 2007: [QSS07].
- Meister: *Numerik linearer Gleichungssysteme: Eine Einführung in moderne Verfahren*, 2015: [Mei15].
- Kanzow: *Numerik linearer Gleichungssysteme*, 2005: [Kan05].
- Plato: *Numerische Mathematik kompakt*, 2004: [Pla04].
- Hämmerlin und Hoffmann: *Numerische Mathematik*, 1994: [HH94].
- Hanke-Bourgeois: *Grundlagen der numerischen Mathematik und des wissenschaftlichen Rechnens*, 2009: [Han09].

Darüber hinaus gibt es auch weiterführende Literatur für spezielle Aspekte dieser Vorlesung (teilweise in englisch):

- Golub und Van Loan: *Matrix computations*, 2013: [GV13].
- Saad: *Iterative Methods for Sparse Linear Systems*, 2003: [Saa03].
- Bornemann: *Numerische lineare Algebra*, 2016: [Bor16].
- Hackbusch: *Iterative solution of large sparse systems of equations*, 2016: [Hac16].
- Quarteroni, Saleri und Gervasio: *Scientific computing with MATLAB and Octave*, 2014: [QSG14].
- Schwarz: *Numerische Mathematik*, 1997: [Sch97],
  Schwarz: *Numerik symmetrischer Matrizen.* 1972: [Sch72].

- Harbrecht: *Algorithmische Mathematik*, 2022: [Har22].

Es gibt wohl kaum ein (Fach-)Buch oder Vorlesungsmanuskript ohne Druckfehler, Flüchtigkeitsfehler, Unzulänglichkeiten oder sonstige Inkorrektheiten. Wir freuen uns über entsprechende Hinweise, die dazu beitragen können, das vorliegende Manuskript zu verbessern.

Ulm, im Februar 2024
Stefan A. Funken, *Karsten Urban*
(`stefan.funken@uni-ulm.de`, `karsten.urban@uni-ulm.de`)

Inhaltsverzeichnis

Kapitel 1

Einleitung

Was ist Numerik?

Numerik ist ein *Teilgebiet der Angewandten Mathematik* und bezeichnet allgemein die mathematische Untersuchung von Berechnungsverfahren, Algorithmen oder Methoden zur näherungsweisen Berechnung („Approximation“) bestimmter Größen (s.u.). Die kann –muss aber nicht– mithilfe eines Computers geschehen. Die mathematische Untersuchung von Verfahren an sich ist aber unabhängig von der Verwendung eines Computers.

Die oben erwähnten zu berechnende Größen können z.B. sein:

- Die **Auswertung von Funktionen** wie z.B. $\sin(1), e^2$ und ähnliche, deren Werte nicht unmittelbar durch eine Zahl angegeben werden können. Solche Berechnungen geschehen auch in jedem Taschenrechner. Dort sind die entsprechenden Berechnungsverfahren in der Regel durch Schaltkreise oder spezielle Microchips realisiert, also in der Hardware fest verdrahtet.
- Die **Lösung von Gleichungen** z.B. lineare $Ax = b$ oder nichtlineare Gleichungen. d.h. $f(x) = 0$, wobei der gesuchte Lösungsvektor $x^* \in \mathbb{R}^n$ oft aus sehr vielen Koordinaten besteht, also $n > 1.000.000$ gilt!
  Andere Typen von Gleichungen sind z.B. Differenzialgleichungen etwa $u''(x) = f(x),\ x \in (0,1)$, wobei die Funktion $u : [0,1] \to \mathbb{R}$ bei gegebenem $f : [0,1] \to \mathbb{R}$ gesucht ist. Ein Beispiel hierfür ist etwa die gesuchte Auslenkung u einer Geigen- oder Gitarrensaite unter der äußeren Last f.
- Die **Näherung von Größen**, die nicht exakt berechnet werden können (z.B. Ableitungen – Sensitivitäten, Integrale – Mittel- oder Erwartungswerte). Manchmal will man solche Größen auch gar nicht exakt bestimmen, etwa, wenn die Berechnung zu lange dauern würde. Andere Beispiele wären optimale Steuerungen oder optimale Anlage–Strategien.
- Computer-gestützte **Simulationen** komplexer Vorgänge.
  In vielen Bereichen sind Experimente sehr teuer, aufwendig, zu gefährlich, ethisch nicht vertretbar oder gar nicht möglich. Beispiele sind:
  - *Wettervorhersage*: Dies bedeutet u.a. die Simulation der turbulenten Wolkenströmungen.
  - *Strömungsmechanik*, also etwa Aerodynamik, Flugzeug-, Automobil- oder Schiffsbau.
  - *Bauingenieurwesen*, z.B. die Simulation der Statik oder der Eigenschwingung

von Brücken und anderen Bauwerken.
  - *Medizin*: Simulation der Knochenheilung oder die Therapie von Gehirn–Tumoren.
  - *Wirtschaftswissenschaften*: Simulation von Aktienkursen, Bewertung von komplexen Finanz- oder Versicherungsprodukten, Bestimmung optimaler Anlage–Strategien. Diese Liste ließe sich fortführen.

Speziell beschäftigt sich die Numerik mit
- der Konstruktion **„geeigneter“ Lösungsverfahren**, die insbesondere
  - schnell („**effizient**“) sind, teilweise in oder sogar schneller als in „Echtzeit“ (wobei zu definieren wäre, was das ist),
  - **zuverlässig**, also mit *beweisbarer* Abschätzung z.B. der folgenden Form

$$\|x_{\text{numerisch}} - x_{\text{exakt}}\| \leq \text{ Toleranz}$$

    (wobei die exakte Größe x_{exakt} unbekannt ist), und
  - **robust** gegenüber Störungen wie z.B. Messfehlern, Modell–Unsicherheiten etc.

  sind;
- mit der **mathematischen Analyse** dieser Verfahren (Konvergenz, Geschwindigkeit, Aufwand, Robustheit etc.) und
- deren effizienter Realisierung (**Implementierung**).

In diesem Sinne liegt die Numerik an der Schnittstelle von Mathematik und Informatik.

Die Numerische Mathematik selbst hat eine Reihe von Teilgebieten bzw. Gebiete, die unmittelbar an sie angrenzen, etwa:
- Numerische Lineare Algebra („Numerik 1“),
- Numerische Analysis („Numerik 2“),
- Numerische Optimierung („Numerik 3“),
- Numerik von (gewöhnlichen) Differenzialgleichungen („Numerik 4“),
- Numerik von Partiellen Differenzialgleichungen,
- Numerical Finance,
- Computational Physics, Computational Science, High Performance Computing (HPC),
- CFD: Computational Fluid Dynamics,
- Wissenschaftliches Rechnen,
- Numerische Methoden in Data Science, Maschinellem Lernen und Künstlicher Intelligenz,
- ...

Insbesondere wird klar, dass die Numerik einen starken interdisziplinären Bezug aufweist.

Modellierung und Simulation

In vielen Fällen stammt das Problem, für das man ein numerisches Näherungsverfahren konstruieren, analysieren und realisieren möchte, von einem *realen* Problem, z.B. aus Naturwissenschaft, Technik, Wirtschaftswissenschaften oder Medizin. Bevor wir jedoch ein numerisches Verfahren entwickeln können, muss das reale Problem in die Sprache der Mathematik übersetzt werden, man sucht also Gleichungen oder andere mathematische Konstrukte (Ungleichungen, Optimierungsaufgaben, ...), die das reale Problem möglichst gut darstellen. Diesen Vorgang nennt man *(mathematische) Modellierung*. Wenn wir dann ein mathematisch formuliertes Problem haben, können wir ein numerisches Verfahren für dieses Problem konstruieren, analysieren und umsetzen. Damit können wir dann eine numerische Simulation des realen Problems durchführen. Dieses Vorgehen ist im folgenden Schaubild vereinfacht dargestellt.

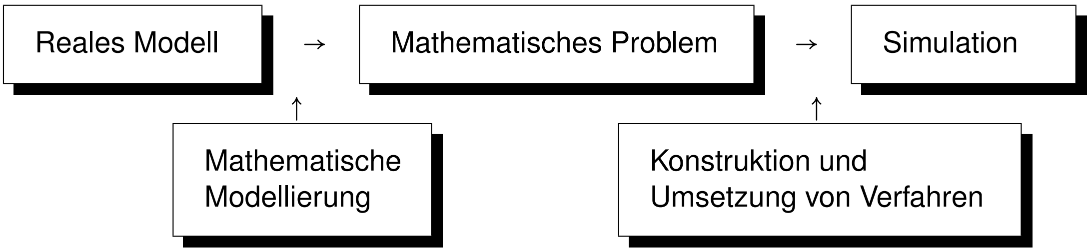


Abbildung 1.1: Modellierung und Simulation.

In vielen Fällen folgt auf die Simulation noch ein vergleichendes Experiment, numerische Simulationen werden am Experiment „validiert". Das kann auch zu einer Verfeinerung des mathematischen Modells führen, so dass aus dem Vorgehen in Abbildung 1.1 ein Kreislauf wird, der auch als *Modellierungzirkel* bezeichnet wird.

Wir beschreiben dieses abstrakte Vorgehen an einem konkreten Beispiel.

■ **Beispiel 1.0.1 — Bestimmung des Abraums bei der Braunkohleförderung, [DR08].** Nehmen wir an, dass wir den Abraum bestimmen wollen, der durch die Förderung von Braunkohle entstanden ist. Würde man die Form des entstandenen Lochs exakt kennen, so würde sich der Abraum als der Wert des *Volumenintegrals* dieses Loches ergeben. Da man aber die Form nicht exakt kennt, fliegt z.B. ein Flugzeug oder eine Drohne über die Braunkohle-Halde und macht Tiefenmessungen an einzelnen Punkten. Daraus ergibt sich dann folgendes Vorgehen:

1) *Reales Modell:* Volumen des Abraums; mit konkreten *Messungen (Experiment)* in Form von Tiefenmessungen durch Stereofotographien aus Flugzeugen, Satelliten;
2) *Mathematisches Problem:* Bestimme das Volumen des Lochs. Dies bedeutet die Berechnung von Volumenintegralen; verwende dabei die Mess-Ergebnisse als Daten (*Modellierung*);
3) *Konstruktion und Umsetzung von Verfahren:* Verwende sogenannte Kubaturformeln (siehe Vorlesung *Numerische Analysis*) zur näherungsweisen Berechnung der dreidimensionalen Integrale;
4) *Simulation:* Programmiere das o.g. Verfahren und erzeuge so entsprechende Simulationen. ■

Umsetzung bedeutet Programmierung

Die Numerische Mathematik birgt für viele Studierende zunächst die Herausforderung, dass Inhalte aus den Grundvorlesungen aus den Bereichen Lineare Algebra und Analysis gleichermaßen benötigt werden. So verwenden wir etwa bei der numerischen Lösung von linearen Gleichungssystemen sowohl Matrixzerlegungen aus der Linearen Algebra als auch die Taylor–Entwicklung im Mehrdimensionalen zur Konvergenzanalyse. Diese Kombination von Inhalten ist am Anfang nicht einfach.

Es kommt aber noch eine Herausforderung dazu: um das Verhalten numerischer Verfahren studieren zu können, müssen wir diese Verfahren umsetzen, also in Software implementieren – für einige Studierende eine echte Hürde. Programmieren kann man numerische Methoden z.B. in einer höheren Programmiersprache wie `C`, `C++`, `C#`, `Java` oder `Python`, es gibt aber auch mathematische Standardsoftware, die einige der Hürden von höheren Programmiersprachen deutlich niedriger machen und so den Einstieg in die Numerik erleichtern. Wir haben für diese Vorlesung und dieses Skript eine solche mathematische Standardsoftware gewählt, MATLAB®.

MATLAB®

Wir verwenden MATLAB® insbesondere zur Darstellung von Algorithmen. Es ist nicht unbedingt erforderlich, dass hier eine bestimmte Programmiersprache verwendet wird, wir könnten die Algorithmen auch in einem Pseudocode oder einer anderen Programmiersprache darstellen. Allerdings hat sich MATLAB® in vielen Unternehmen und Bereichen von Forschung und Wirtschaft so durchgesetzt, dass man von einem Standard sprechen kann. Insofern gehen wir davon aus, dass Kenntnisse in MATLAB® für ihre spätere berufliche Tätigkeit von großer Bedeutung sein können.

Für die Verwendung in einer Lehrveranstaltung hat MATLAB® den Vorteil, dass viele Probleme zwischen Daten-Ein- und -Ausgabe, Plotten usw. auf einen Schlag gelöst sind, weil MATLAB® die entsprechenden Funktionalitäten bereitstellt. Datenstrukturen werden standardisiert, indem man sich auf einheitliche Befehle stützt. Diese Ziele können auch auf andere Weise erreicht werden. Es ist jedoch hilfreich, ein Paket zu haben, das auf (fast) allen Betriebssystemen stabil läuft, kompatibel ist und die Details so vereinfacht, so dass sich die Studierenden auf die mathematischen Probleme konzentrieren können. Anhang B enthält daher ein kurzes MATLAB®-Tutorial, das als erste Einführung für Studierende oder als Referenz für bereits Vertraute verwendet werden kann.

Suchmaschinen und ChatGBT

Im Zuge der Programmieraufgaben war es bislang über Jahrzehnte üblich, dass Studierende aufgefordert waren, ein bestimmtes Verfahren z.B. in MATLAB® umzusetzen und anhand von einigen Beispieldaten zu validieren. Natürlich finden Sie die allermeisten Codes (insbesondere auch in MATLAB®) längst per Suchmaschinen oder in `github`. `ChatGBT` hat die Verfügbarkeit von Programmcode für numerische Verfahren noch einmal verstärkt und bietet Möglichkeiten, von denen wir vor wenigen Jahren noch keine Ahnung haben konnten. Wir greifen diesen Trend in diesem Manuskript auf und geben zu den MATLAB®-Programmen auch Antworten von `ChatGBT`.

In diesem Zusammenhang möchten wir aus der Stellungnahme der Deutschen Forschungsgemeinschaft (DFG) vom September 2023 zur Verwendung generativer Modelle wie `ChatGBT` verweisen:

Der Einsatz generativer Modelle im Rahmen des wissenschaftlichen Arbeitens sollte angesichts der erheblichen Chancen und Entwicklungspotenziale keinesfalls ausgeschlossen werden. Ihr Einsatz erfordert jedoch bestimmte verbindliche Rahmenbedingungen, um die *gute wissenschaftliche Praxis und die Qualität wissenschaftlicher Ergebnisse zu sichern.*

- *Transparenz und Nachvollziehbarkeit des Forschungsprozesses und der gewonnenen Erkenntnisse für Dritte sind wesentliche Grundprinzipien wissenschaftlicher Integrität. Dieses Wertesystem bietet im Hinblick auf den Umgang mit generativen Modellen weiterhin wertvolle Leitlinien.*
- *Es entspricht dem Berufsethos von Wissenschaftlerinnen und Wissenschaftlern, dass sie selbst für die Einhaltung der Grundprinzipien wissenschaftlicher Integrität einstehen. Der Einsatz generativer Modelle kann Wissenschaftlerinnen und Wissenschaftler von dieser inhaltlichen und formalen Verantwortung nicht entbinden.*
- *Wissenschaftlerinnen und Wissenschaftler sollten bei der öffentlichen Zugänglichmachung ihrer Ergebnisse im Sinne wissenschaftlicher Integrität offenlegen, ob und welche generativen Modelle sie zu welchem Zweck und in welchem Umfang eingesetzt haben.*
- *In wissenschaftlichen Publikationen können nur die verantwortlich handelnden natürlichen Personen als Autorinnen und Autoren in Erscheinung treten. Sie müssen sicherstellen, dass durch die Verwendung generativer Modelle kein fremdes geistiges Eigentum verletzt wird und kein wissenschaftliches Fehlverhalten etwa in Form von Plagiaten entsteht.*

Warum hier keine höhere Programmiersprache?

Zu unseren Zeiten als Studierende haben wir die Programmieraufgaben in `Fortran77` geschrieben. Später kam dann `Turbo Pascal`, `C`, `C++` und heute wäre es dann vielleicht `Java` oder `Python`. Wir erkennen schon, wie schnelllebig die Entwicklung von jeweils als „besonders effizient“ angesehenen Programmiersprachen in der Numerik ist. Zwangsläufig riskieren wir bei der Verwendung einer Programmiersprache, dass diese in kurzer Zeit schon durch eine andere abgelöst wird.

Ein weiterer Grund ist, dass unsere Erfahrung gezeigt hat, dass die Verwendung einer höheren Programmiersprache eine wesentliche zusätzliche Herausforderung für Studierende darstellt. Natürlich sind wir uns bewußt, dass hocheffiziente oder z.B. parallele Programmierung nur mit der entsprechenden Programmiersprache zu realisieren ist – aber das können Sie auch noch in fortgeschrittenen Vorlesungen der Numerik lernen.

Kapitel 2

Kondition und Stabilität

Ein Algorithmus wird verwendet, um ein mathematisches Problem zu lösen. Wir wollen nun untersuchen, welche Probleme „einfach“ bzw. „schwer“ sind und welche Algorithmen „gut“ bzw. „schlecht“ sind. Wir beginnen mit der Untersuchung der Problemstellungen.

2.1 Kondition eines Problems

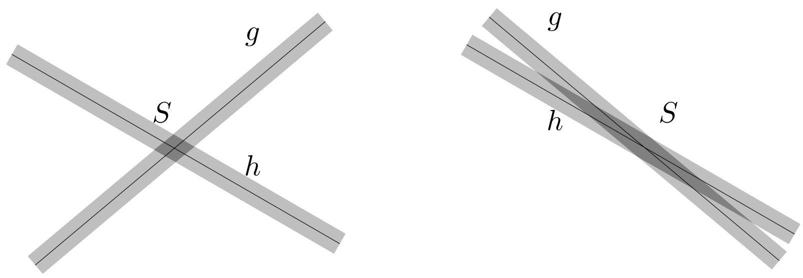


Abbildung 2.1: Schnittpunkt S zweier Geraden g und h. Links gut konditioniert, rechts schlecht konditioniert.

■ **Beispiel 2.1.1** Diskutieren wir am Anfang zuerst folgendes geometrische Problem: die zeichnerische Bestimmung des Schnittpunkts S zweier Geraden g und h in der Ebene, vgl. Abbildung 2.1. Schon beim Zeichnen haben wir Schwierigkeiten, die Geraden ohne Fehler darzustellen. Die Frage, die wir näher untersuchen wollen, lautet: wie stark hängt die Ungenauigkeit bei der Bestimmung des Schnittpunkts S (Output) von den Zeichenfehlern (Fehler im Input) ab.

Wie wir der Grafik direkt entnehmen können, hängt der Fehler in der Ausgabe stark davon ab, in welchem Winkel $\measuredangle(g, h)$ sich die Geraden schneiden. Stehen g und h annähernd senkrecht aufeinander, so variiert der Schnittpunkt S etwa im gleichen Maße wie der Fehler beim Zeichnen von g und h. In diesem Fall bezeichnet man das Problem, den Schnittpunkt S zeichnerisch zu bestimmen, als *gut konditioniert*.

Ist jedoch der Winkel $\measuredangle(g, h)$ sehr klein (g und h sind fast parallel), so kann man schon mit dem Auge keinen genauen Schnittpunkt S ausmachen, und eine kleine Lageänderung von g oder h liefert einen gänzlich anderen Schnittpunkt. Man spricht dann von einem *schlecht konditionierten* Problem. ■

Kommen wir nun zu einer mathematischen Präzisierung des Konditionsbegriffs. Dazu brauchen wir einen etwas allgemeineren Rahmen und werden später sehen, dass dieser Rahmen für die Untersuchung sehr gut geeignet ist.

Aufgabe 2.1.2 Seien X,Y zwei Mengen und $\varphi : X \to Y$. Wir betrachten das Problem:

$$\text{Gegeben sei } x \in X, \quad \text{gesucht } y := \varphi(x).$$

Wir untersuchen, wie sich Störungen in den Daten x auf das Ergebnis y auswirken. Man beachte, dass dies **nichts mit dem Algorithmus** (der Realisierung auf dem Computer) zu tun hat, sondern einzig eine **Eigenschaft der Problemstellung** ist.

■ **Beispiel 2.1.3 — Noch einmal der Geradenschnittpunkt.** Die beiden Geraden seien in folgender Form gegeben

$$G_1 = \{(x_1, x_2) \in \mathbb{R}^2 : a_{1,1}x_1 + a_{1,2}x_2 = b_1\}, \quad G_2 = \{(x_1, x_2) \in \mathbb{R}^2 : a_{2,1}x_1 + a_{2,2}x_2 = b_2\},$$

wobei $b = (b_1, b_2)^T \in \mathbb{R}^2$ und die Koeffizienten $a_{i,j}$ für $i, j = 1, 2$ gegeben sind. Mit $A := (a_{ij})_{i,j=1,2} \in \mathbb{R}^{2\times 2}$ ist der gesuchte Schnittpunkt $x = (x_1, x_2)^T$ von G_1, G_2 gegeben durch

$$Ax = b\ ,$$

also die Lösung eines linearen Gleichungssystems. Falls A regulär ist können wir schreiben $x = A^{-1}b$. Also sind A und b die Eingaben und x die Ausgabe, d.h. $X := \mathbb{R}^{2\times 2} \times \mathbb{R}^2$, $Y := \mathbb{R}^2$ und $\varphi(A, b) := A^{-1}b = x$. ■

■ **Beispiel 2.1.4 — Kohleaushub, Beispiel 1.0.1.** Beim Beispiel 1.0.1 des Kohleaushubs gehen wir davon aus, dass wir aus den Messungen bereits eine Höhenfunktion $h(x) = z$ mit $h : \mathbb{R}^2 \to \mathbb{R}$, gewonnen haben, die jedem Punkt $x \in \mathbb{R}^2$ einer Karte die Höhe zuweist. Weiterhin sei das Kohlerevier in einem Rechteck $R := [a, b] \times [c, d] \subset \mathbb{R}^2$ enthalten. Dann lautet die Formel für den Kohleaushub

$$f(h) = \int_a^b \int_c^d h(x)\, dx_2\, dx_1,$$

also ein Doppelintegral. Damit ist klar, dass $Y = \mathbb{R}$, aber für welche Eingaben ist φ definiert? Offenbar muss die Funktion h, die hier als Eingabe fungiert, integrierbar sein, etwa $h \in X := \mathcal{R}(\mathbb{R}^2)$.[1] Man beachte, dass X hier (im Gegensatz zu den beiden ersten Beispielen) unendlich-dimensional ist. ■

Beide Beispiele zeigen, dass der in Problem 2.1.2 beschriebene Rahmen in der Tat angemessen ist.

■ **Beispiel 2.1.5** Wir betrachten den Spezialfall

$$\varphi : \mathbb{R}^n \to \mathbb{R} \quad \text{mit} \quad \varphi(x) := \langle a, x\rangle + b, \quad a \in \mathbb{R}^n, \quad b \in \mathbb{R},$$

[1] $\mathcal{R}(\mathbb{R})$ bezeichne die Menge der Rieman-integrierbaren Funktionen auf $\mathbb{R}$.

wobei $\langle\cdot,\cdot\rangle$ das (Euklidische) Skalarprodukt bezeichne, also $\langle x,y\rangle = x^T y$. Seien x und $\widetilde{x} \in \mathbb{R}^n$ zwei Eingaben. Der relative Fehler von $\varphi(\widetilde{x})$ bezüglich $\varphi(x)$ lässt sich folgendermaßen abschätzen (falls $\varphi(x) \neq 0$ und $x_j \neq 0$ für alle j):

$$\left|\frac{\varphi(x)-\varphi(\widetilde{x})}{\varphi(x)}\right| = \frac{|\langle a, x-\widetilde{x}\rangle|}{|\varphi(x)|} \leq \frac{1}{|\varphi(x)|}\sum_{j=1}^{n}|a_j||x_j-\widetilde{x}_j| = \sum_{j=1}^{n}\frac{|x_j|}{|\varphi(x)|}|a_j|\cdot\frac{|x_j-\widetilde{x}_j|}{|x_j|}. \tag{2.1}$$

Wir können davon ausgehen, dass der relative Eingabefehler immer die gleiche Größenordnung haben, $\mathrm{eps} := \frac{|x_j-\widetilde{x}_j|}{|x_j|}$, dann ist

$$\mathrm{eps}\cdot\sum_{j=1}^{n}\frac{|x_j|}{|\varphi(x)|}|a_j|$$

der **unvermeidbaren Fehler** (selbst bei exakter Rechnung), der bei der Berechnung von φ auftritt. Sei nun $\varphi:\mathbb{R}^n \to \mathbb{R}$ differenzierbar in x. Dann gilt mit der Taylor–Entwicklung[2]

$$\varphi(x)-\varphi(\widetilde{x}) = \langle\nabla\varphi(x), x-\widetilde{x}\rangle + o(\|x-\widetilde{x}\|),$$

und (2.1) wird zu

$$\frac{|\varphi(x)-\varphi(\widetilde{x})|}{|\varphi(x)|} \leq \sum_{j=1}^{n}\frac{|x_j|}{|\varphi(x)|}\left|\frac{\partial\varphi}{\partial x_j}(x)\right|\cdot\frac{|x_j-\widetilde{x}_j|}{|x_j|} + o(\|x-\widetilde{x}\|).$$

Somit gibt der Faktor $\frac{|x_j|}{|\varphi(x)|}\left|\frac{\partial\varphi}{\partial x_j}(x)\right|$ (der unabhängig von $\widetilde{x}$ ist) an, wie stark sich ein relativer Fehler in der Eingabe x_j auf den relativen Fehler der Ausgabe $\varphi(x)$ auswirkt. Ist schließlich $\varphi:\mathbb{R}^n\to\mathbb{R}^m$, so können wir die bisherigen Überlegungen auf jede Komponente von φ einzeln anwenden. ∎

Dies motiviert den folgenden Begriff.

Definition 2.1.6 — Konditionszahlen. Sei $\varphi:\mathbb{R}^n\to\mathbb{R}^m$ differenzierbar in $x\in\mathbb{R}^n$ sowie $\varphi_i(x)\neq 0$ für $1\leq i\leq m$. Die Zahlen

$$\kappa_{ij}(x) = \frac{|x_j|}{|\varphi_i(x)|}\cdot\left|\frac{\partial\varphi_i}{\partial x_j}(x)\right|, \qquad 1\leq i\leq m,\ 1\leq j\leq n, \tag{2.2}$$

heißen *Konditionszahlen* von φ in x.

■ **Beispiel 2.1.7** (a) **Multiplikation**: $\varphi:\mathbb{R}^2\to\mathbb{R}, \varphi(x_1,x_2) := x_1\cdot x_2$; es gilt:

$$\kappa_1(x) = \frac{|x_1|}{|x_1x_2|}\cdot\left|\frac{\partial\varphi}{\partial x_1}(x)\right| = 1 \quad \text{und} \quad \kappa_2(x) = 1.$$

Es tritt keine Verstärkung des relativen Fehlers auf; die Multiplikation ist „gut konditioniert“.

[2] Die Landau-Symbole o und $\mathcal{O}$ werden hier als bekannt vorausgesetzt.

(b) **Addition**: $\varphi : \mathbb{R}^2 \to \mathbb{R}, \varphi(x_1, x_2) := x_1 + x_2$,

$$\kappa_1(x) = \frac{|x_1|}{|x_1 + x_2|} \cdot \left| \frac{\partial \varphi}{\partial x_1}(x) \right| = \frac{|x_1|}{|x_1 + x_2|} \quad \text{und} \quad \kappa_2(x) = \frac{|x_2|}{|x_1 + x_2|}.$$

Im Falle der **Auslöschung**[3], d.h. für $|x_j| \gg |x_1 + x_2|$ tritt eine große Verstärkung des relativen Fehlers auf; die Addition ist in diesem Fall „schlecht konditioniert".

(c) **Lösen der quadratischen Gleichung** $x^2 + 2px - q = 0$ im Fall $p, q > 0$: Die größere der beiden Nullstellen ergibt sich aus der „Mitternachtsformel":

$$\varphi(p, q) := -p + \sqrt{p^2 + q}, \qquad \varphi : \mathbb{R}^2 \to \mathbb{R}. \tag{2.3}$$

Eine einfache Rechnung ergibt

$$\kappa_p = \frac{p}{\sqrt{p^2 + q}} \qquad \text{und} \qquad \kappa_q = \frac{p + \sqrt{p^2 + q}}{2\sqrt{p^2 + q}}, \tag{2.4}$$

also $\kappa_p \leq 1$ und $\kappa_q \leq 1$; das Problem ist ebenfalls „gut konditioniert". Im Fall $q < 0$, $q \approx -p^2$ hingegen wäre das Problem „schlecht konditioniert".

(d) **Skalarprodukt**: $\varphi : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$, $\varphi(x, y) := x^T y$. Wir fassen $(x, y) \in \mathbb{R}^n \times \mathbb{R}^n$ als Vektor der Länge $2n$ auf. Dementsprechend haben wir Konditionszahlen κ_{1j}, $j = 1, ..., n$ für die Komponenten von x und $\kappa_{1,j+n}$, $j = 1, ..., n$, für diejenigen von y. Hier gilt

$$\begin{aligned} \kappa_{1j}(x, y) &= \frac{|x_j|}{|\varphi(x, y)|} \left| \frac{\partial \varphi(x, y)}{\partial x_j} \right| = \frac{|x_j|}{|x^T y|} |y_j| = \frac{|x_j \, y_j|}{|x^T y|}, \\ \kappa_{1,j+n}(x, y) &= \frac{|y_j|}{|\varphi(x, y)|} \left| \frac{\partial \varphi(x, y)}{\partial y_j} \right| = \frac{|y_j|}{|x^T y|} |x_j| = \frac{|x_j \, y_j|}{|x^T y|} = \kappa_{1j}(x, y), \end{aligned} \tag{2.5}$$

für $j = 1, ..., n$. Also ist das Problem schlecht konditioniert, falls x und y fast orthogonal sind ($x^T y \approx 0$), ansonsten gut konditioniert. ■

■ **Beispiel 2.1.8** Als nächstes untersuchen wir die Kondition der Lösung eines **linearen Gleichungssystems** $Ax = b$ in Abhängigkeit der Matrix $A \in \mathbb{R}^{n \times n}$. Sei also $b \in \mathbb{R}^n$ fest und setze $\varphi : \mathbb{R}^{n \times n} \to \mathbb{R}^n$ mit $\varphi(A^{-1}) := A^{-1}b$. Es gilt also $\varphi'(A^{-1}) = b = Ax$. Die Kondition hängt nun von der gewählten Norm $\|\cdot\|$ ab, deswegen verwenden wir die Notation $\kappa_{\|\cdot\|}$ und erhalten

$$\begin{aligned} \kappa_{\|\cdot\|}(A) &= \frac{\|A^{-1}\|}{\|A^{-1}b\|} \|b\| = \frac{\|b\|}{\|A^{-1}b\|} \|A^{-1}\| = \frac{\|Ax\|}{\|x\|} \|A^{-1}\| \\ &\leq \|A^{-1}\| \sup_{x \neq 0} \frac{\|Ax\|}{\|x\|} = \|A\| \, \|A^{-1}\| \end{aligned} \tag{2.6}$$

mit einer beliebigen Vektornorm $\|\cdot\|$ und der induzierten Matrixnorm $\|A\| := \sup_{x \neq 0} \frac{\|Ax\|}{\|x\|}$, vgl. Anhang A. Der Ausdruck auf der rechten Seite von (2.6) wird häufig zur Definition der Konditionszahl einer Matrix verwendet, vgl. Definition 3.1.2, S. 17. ■

Die Konditionszahlen erlauben die Untersuchung der Fehlerverstärkung für jede ein-

[3] Vgl. Bemerkung 7.5.3.

zelne Komponente von φ. Manchmal ist man eher am Gesamtfehler (im Sinne eines Fehlermaßes, einer Norm) interessiert. Dazu führen wir einige Begriffe ein.

Definition 2.1.9 (a) Es sei $\widetilde{x} \in X$ eine Approximation von $x \in X$. Als **Fehler** bezeichnet man $\Delta x := x - \widetilde{x}$ und zu einer gegebenen Norm $\|\cdot\|_X$ ist

$$\|x - \widetilde{x}\|_X \tag{2.7}$$

der **absolute Fehler** von $\widetilde{x}$. Für $x \neq 0$ beschreibt

$$\frac{\|x - \widetilde{x}\|_X}{\|x\|_X} = \frac{\|\Delta x\|_X}{\|x\|_X} \tag{2.8}$$

den **relativen Fehler** von $\widetilde{x}$.

(b) Seien $\|\cdot\|_X$, $\|\cdot\|_Y$ geeignete Normen auf X bzw. Y, $\Delta x := x - \tilde{x}$, $\Delta y := y - \tilde{y}$ und

$$\delta_x := \frac{\|\Delta x\|_X}{\|x\|_X}, \qquad \delta_y := \frac{\|\Delta y\|_Y}{\|y\|_Y}$$

die **relativen Ein- bzw. Ausgabefehler**. Dann heißt

$$\kappa_\varphi := \frac{\delta_y}{\delta_x} \tag{2.9}$$

die **(relative) Kondition** des Problems φ beschrieben durch die Funktion $\varphi : X \to Y$. Die Größe

$$\kappa_{\varphi,\mathrm{abs}} := \frac{\|\Delta y\|_Y}{\|\Delta x\|_X} \tag{2.10}$$

heißt **absolute Kondition** von $y = \varphi(x)$.

(c) Ein Problem heißt **gut konditioniert**, wenn „kleine“ Schranken für κ_φ für $\delta_x \to 0$ existieren. Offenbar wäre $\kappa_\varphi = 1$ (bzw. $\kappa_\varphi \approx 1$) optimal.

■ **Beispiel 2.1.10** Wir untersuchen Beispiel 2.1.7 (b), die Addition, hinsichtlich absoluter und relativer Kondition. Hier gilt $X = \mathbb{R}^2$, $x = (x_1, x_2)$ und $\|x\|_X^2 = x_1^2 + x_2^2$. Weiterhin ist $Y = \mathbb{R}$, $y = x_1 + x_2$ und $\|y\|_Y = |x_1 + x_2|$. Damit erhalten wir

$$\begin{aligned}
\kappa_{\varphi,\mathrm{abs}}^2 &= \frac{|x_1 - \widetilde{x}_1 + x_2 - \widetilde{x}_2|^2}{(x_1 - \widetilde{x}_1)^2 + (x_2 - \widetilde{x}_2)^2} = \frac{(x_1 - \widetilde{x}_1)^2 + (x_2 - \widetilde{x}_2)^2 + 2\,|x_1 - \widetilde{x}_1|\,|x_2 - \widetilde{x}_2|}{(x_1 - \widetilde{x}_1)^2 + (x_2 - \widetilde{x}_2)^2} \\
&= 1 + 2\frac{|x_1 - \widetilde{x}_1|\,|x_2 - \widetilde{x}_2|}{(x_1 - \widetilde{x}_1)^2 + (x_2 - \widetilde{x}_2)^2} \leq 2,
\end{aligned}$$

wobei der letzte Schritt mit der Young-Ungleichung $ab \leq \frac{1}{2}(a^2 + b^2)$ folgt. Also haben wir $\kappa_{\varphi,\mathrm{abs}} \leq \sqrt{2}$, eine sehr moderate Schranke. Hingegen gilt

$$\kappa_\varphi = \kappa_{\varphi,\mathrm{abs}} \frac{\|x\|_X}{\|y\|_Y} = \kappa_{\varphi,\mathrm{abs}} \frac{\sqrt{x_1^2 + x_2^2}}{|x_1 + x_2|} \leq \sqrt{2}\,\frac{\sqrt{x_1^2 + x_2^2}}{|x_1 + x_2|},$$

und dieser Term lässt sich für $x_1 \approx -x_2$ nicht von oben beschränken. Wir erkennen deutlich den Effekt der Auslöschung, vgl. Bemerkung 7.5.3. ■

Bemerkung 2.1.11 Relative Fehler in der ∞-Norm können durch eine Aussage über die Anzahl korrekter Stellen von $\widetilde{x}$ ausgedrückt werden: die Abschätzung

$$\frac{\|x-\widetilde{x}\|_\infty}{\|x\|_\infty} \approx 10^{-p},$$

bedeutet, dass die betragsgrößte Komponente von $\widetilde{x}$ näherungsweise p korrekte signifikante Stellen hat.

2.2 Stabilität eines Algorithmus

Fehlerverstärkung von der Eingabe zur Ausgabe tritt nicht nur beim zu lösenden Problem auf, sondern auch beim verwendeten Algorithmus. Dies führt auf den Begriff der **Stabilität** eines Algorithmus. Jeder Algorithmus zur Lösung von Problem 2.1.2 lässt sich auffassen als eine Abbildung $\widetilde{\varphi}: X \to Y$, welche die (exakte) Lösungsabbildung φ approximiert.

Von einem guten Algorithmus wird man erwarten, dass er die relativen Fehler nur unwesentlich mehr verstärkt als die Konditionszahlen $\kappa_{ij}(x)$ des Problems φ es erwarten lassen. Für $\widetilde{\varphi}: \mathbb{R}^n \to \mathbb{R}^m$ sollte also eine Abschätzung der Form

$$\left|\frac{\widetilde{\varphi}_i(\widetilde{x})-\widetilde{\varphi}_i(x)}{\widetilde{\varphi}_i(x)}\right| \le C_{i1} \underbrace{\sum_{j=1}^n \kappa_{ij}(x)\frac{|\widetilde{x}_j-x_j|}{|x_j|}}_{\le \frac{|\varphi_i(\widetilde{x})-\varphi_i(x)|}{|\varphi_i(x)|}+o(\|\tilde{x}-x\|)} + C_{i2}\, n \operatorname{eps} \quad (i=1,\dots,m), \tag{2.11}$$

(oder zumindest in ähnlicher Form) gelten mit Konstanten $C_{i1}, C_{i2} \ge 0$, welche nicht viel größer als 1 sind.

Definition 2.2.1 — Numerische Stabilität eines Algorithmus. Ein Algorithmus definiert durch die Eingabe-Ausgabe-Funktion $\widetilde{\varphi}: \mathbb{R}^n \to \mathbb{R}^m$ zur Lösung eines Problems $\varphi: \mathbb{R}^n \to \mathbb{R}^m$ heißt *numerisch stabil*, falls (2.11) gilt mit „vernünftigen" Konstanten C_{i1}, C_{i2}. Andernfalls heißt der Algorithmus *numerisch instabil*.

Aus (2.11) ist nicht ersichtlich, ob $\widetilde{\varphi}(x)$ tatsächlich eine gute Approximation der exakten Lösung $\varphi(x)$ für gegebenes $x \in X$ ist. Dies sieht man an einem einfachen Beispiel: $\widetilde{\varphi}(x) \equiv 0$ ist stabil mit $C_{i1} = C_{i2} = 0$, aber in aller Regel keine gute Approximation von φ. Mit der Frage nach einer guten Approximation (man spricht auch von *Konsistenz*) werden wir uns hier nicht näher beschäftigen, sehr wohl aber in der Vorlesung *Numerik Gewöhnlicher Differenzialgleichungen*.

▪ **Beispiel 2.2.2 — Quadratische Gleichung.** Wir betrachten erneut die Lösung der quadratischen Gleichung (2.3) in Beispiel 2.1.7, also $x^2 + 2px - q = 0$ mit der größeren Wurzel, d.h.

$$\varphi: \mathbb{R}^2 \to \mathbb{R} \quad \text{mit} \quad \varphi(p,q) := -p + \sqrt{p^2+q}.$$

Dazu untersuchen zwei numerische Verfahren:

Algorithmus 2.2.3

$$u := \sqrt{p^2+q}, \quad \tilde{\varphi}_1(p,u) := -p + u. \tag{2.12}$$

Algorithmus 2.2.4 — Satz von Viëta.

$$u := \sqrt{p^2 + q}, \quad \tilde{\varphi}_2(p, q, u) := \frac{q}{p + u}. \tag{2.13}$$

Falls $u \approx p$, d.h. falls $p \gg q$, sind die Konditionszahlen von $\tilde{\varphi}_1$ in Algorithmus 2.2.3 erheblich größer als 1, es tritt Auslöschung[4] auf. Hingegen sind die Konditionszahlen von $\tilde{\varphi}_2$ kleiner als 1. Das Verfahren in Algorithmus 2.2.4 beruht auf dem Satz von Viëta, der besagt, dass $-(p + u)$ die andere Lösung und das Produkt der beiden Lösungen gleich $-q$ ist.

Es stellt sich heraus, dass Algorithmus 2.2.3 numerisch instabil, aber Algorithmus 2.2.4 numerisch stabil ist. Man beachte jedoch, dass beim Algorithmus 2.2.3 die numerische Auswertung der Funktion $\tilde{\varphi}_1$ für sich genommen (trotz Auslöschung) nicht numerisch instabil ist! Schlecht konditioniert ist hier die Auswertung, also die Berechnung von $\tilde{\varphi}_1(p, \sqrt{p^2 + q})$ im Fall $|q| \ll p^2$. Daran kann man auch sehen, dass das Zusammensetzen zweier stabiler Algorithmen sehr wohl einen instabilen Algorithmus für das Gesamtproblem ergeben kann. Diese Tatsache steht in unangenehmem, aber unvermeidlichem Kontrast zu dem Wunsch, ein Problem in unabhängig voneinander zu bearbeitende Teilprobleme zu zerlegen. ∎

[4] Vgl. Bemerkung 7.5.3.

Kapitel 3

Direkte Löser Linearer Gleichungssysteme

Lineare Gleichungssysteme (LGSe) begegnen uns an vielen Stellen. Wir betrachten die Form $Ax = b$ mit $A \in \mathbb{R}^{n \times n}$ und $b \in \mathbb{R}^n$ gegeben sowie $x \in \mathbb{R}^n$ gesucht. Ist n moderat, so können wir das Gleichungssystem mit Papier und Stift exakt lösen; aber was tun wir, wenn n „groß“ ist? Diese Fragestellung hat die Menschheit schon seit Jahrtausenden beschäftigt wie das folgende Beispiel aus dem chinesischen Rechenbuch „Neun Bücher arithmetischer Technik“ (Jiǔ Zhāng Suánshú) aus der frühen Han-Zeit (1. Jahrhunderts n. Chr) zeigt:

> Drei Bauern handeln auf einem Markt Rinder, Schafe und Schweine. Der erste Bauer verkauft 2 Rinder und 5 Schafe und kauft 13 Schweine, wobei 1000 Geldstücke übrig bleiben. Der zweite verkauft 3 Rinder und 3 Schweine, wofür er genau 9 Schafe bekommt. Der dritte Bauer verkauft 6 Schafe und 8 Schweine und kauft 5 Rinder, wobei er noch 600 Geldstücke drauflegen muss. Wie hoch ist der Preis eines Rinds, eines Schafs und eines Schweins?

Später stellte u.a. Euler[1] Berechnungen für die Zugbahnen der Planeten an. Damals gab es keine Computer. Man war also auf Rechenmethoden angewiesen, die von Hand schnell („effizient“) und zuverlässig („stabil“) auszuführen waren. Gauß[2] führte dann die nach ihm benannte Eliminationsmethode ein, die eine wesentliche Effizienzsteigerung bewirkte, vgl. §3.4.

Insofern sind „direkte“ Methoden (so werden Verfahren bezeichnet, die nach einer festen Anzahl von Rechenschritten die *exakte* Lösung des Gleichungssystem ergeben) nicht mit der Existenz moderner Computer verknüpft; diese geben aber ganz andere Möglichkeiten insbesondere was die maximale Dimension n der handhabbaren Gleichungssysteme angeht. Direkte Löser sind zwar „klassisch“, aber selbst auf modernen Parallel- oder sogar Quantencomputern äußerst relevant.

[1] Leonard Euler, 1707-1783.

[2] Carl Friedrich Gauß, 1777-1855.

Einführung und Motivation

Es ist oftmals so, dass die Kenntnis der Herkunft eines linearen Gleichungssystems dazu verwendet werden kann, um besonders „gute“ (was das auch heißen mag) Verfahren zu konstruieren. Betrachten wir daher ein motivierendes Beispiele für lineare Gleichungssysteme. Die schwingende Saite ist ein klassisches Beispiel aus der Physik. Sie ist außerdem ein Modellbeispiel dafür, dass viele lineare Gleichungssysteme in Naturwissenschaften und Technik von Differenzialgleichungen stammen. Das mehrdimensionale Gegenstück ist die schwingende Membran.

■ **Beispiel 3.0.1 — Auslenkung einer Saite.** Gegeben sei eine elastische Saite der Länge 1, die an beiden Enden fixiert ist. Die Saite wird nun durch eine äußere Kraft f ausgelenkt. Wir wollen die Auslenkung u der Saite aus ihrer Ruhelage als Funktion von $x \in [0,1]$ berechnen. Die gesuchte Auslenkung $u : [0,1] \to \mathbb{R}$ ist Lösung des folgenden linearen Randwertproblems zweiter Ordnung

$$-u''(x) + \lambda(x)u(x) = f(x)\ ,\ x \in (0,1), \qquad u(0) = u(1) = 0 \tag{3.1}$$

mit gegebenen $f : (0,1) \to \mathbb{R}$ und $\lambda : (0,1) \to \mathbb{R}$ mit $\lambda(x) \geq 0$ für $x \in (0,1)$. Genaueres zur Modellierung findet man z.B. in [AU18].

Wir wollen (3.1) näherungsweise lösen. Dazu unterteilen wir $[0,1]$ in Teilintervalle gleicher Länge. Die Anzahl der Intervalle sei $n_h + 1 > 1$, $n_h \in \mathbb{N}$ und $h = \frac{1}{n_h+1}$ die Schrittweite. Dann setzt man $x_i := ih\ ,\ i = 0,\dots,n_h+1 \quad (x_0 = 0, x_{n_h+1} = 1)$, die x_i werden als *Knoten* bezeichnet. Die *Schrittweite* ist $h := x_{i+1} - x_i$ für alle i, man spricht von einem *äquidistanten Gitter*, weil der Abstand von zwei Knoten immer gleich ist. Wir wollen die Lösung an den Knoten x_i approximieren und ersetzen hierzu (wie aus der Analysis bekannt) die zweite Ableitung durch den zentralen Differenzenquotienten

$$u''(x_i) \approx \frac{1}{h^2}\big(u(x_{i-1}) - 2u(x_i) + u(x_{i+1})\big) =: D^2_{c;h}u(x_i).$$

Es gilt bekanntlich $\|u'' - D^2_{c;h}u\| = \mathcal{O}(h^2)$, falls $u \in C^4[0,1]$. Damit erhält man für die Näherung $u_i \approx u(x_i)$ also folgende Bedingungen (mit den Abkürzungen $\lambda_i = \lambda(x_i), f_i = f(x_i)$):

$$\begin{cases} \dfrac{1}{h^2}(-u_{i-1} + 2u_i - u_{i+1}) + \lambda_i u_i = f_i, & \text{für } 1 \leq i \leq n_h, \\ u_0 = u_{n_h+1} = 0, & \end{cases}$$

also ein lineares Gleichungssystem der Form

$$\underbrace{\begin{bmatrix} (2+h^2\lambda_1) & -1 & & 0 \\ -1 & (2+h^2\lambda_2) & \ddots & \\ & \ddots & \ddots & -1 \\ 0 & & -1 & (2+h^2\lambda_n) \end{bmatrix}}_{=:A_h} \underbrace{\begin{bmatrix} u_1 \\ \vdots \\ \vdots \\ u_{n_h} \end{bmatrix}}_{=:u_h} = \underbrace{\begin{bmatrix} h^2 f_1 \\ \vdots \\ \vdots \\ h^2 f_{n_h} \end{bmatrix}}_{=:f_h},$$

d.h. $A_h u_h = f_h$. Für $n_h \to \infty$ ($h \to 0$) konvergiert die *diskrete Lösung* u_h gegen die Lösung u von (3.1). Allerdings wächst die Dimension der Matrix A_h mit kleiner werdendem h. Bei mehrdimensionalen Problemen führt dies leicht zu sehr großen LGS.

Wir nennen diese Matrix auch *Standardmatrix*, weil sie typische Eigenschaften von Matrizen besitzt, die von Differenzialgleichungen stammen: A_h ist tridiagonal (insbesondere *dünn besetzt*, d.h. es gibt „wenige" Nicht-Null-Einträge – diesen Begriff definieren wir später noch genauer), symmetrisch und positiv definit. ■

3.1 Kondition eines linearen Gleichungssystems

Im Folgenden soll eine Abschätzung für den Fehler, der beim Lösen eines LGS auftritt, untersucht werden. Dies führt auf die Kondition eines linearen Gleichungssystems, also die Frage, wann ein solches System „schwer" und wann „leicht" zu lösen ist.

3.1.1 Fehler und Residuum

Definition 3.1.1 Sei $x := A^{-1}b$ die exakte (aber im Allgemeinen unbekannte) Lösung des linearen Gleichungssystems $Ax = b$ und $\widetilde{x}$ eine Näherungslösung. Dann heißt $e := x - \widetilde{x}$ **Fehler** und $r := b - A\widetilde{x}$ das **Residuum.**

Den Fehler können wir im Allgemeinen nicht berechnen, da wir x nicht kennen. Das Residuum hingegen ist für eine gegebene Approximation $\widetilde{x}$ (z.B. das Ergebnis eines Algorithmus) berechenbar. Damit können wir hoffen, dass das Residuum zur Abschätzung des Fehlers verwendet werden kann. Es gilt folgender Zusammenhang:

$$Ae = Ax - A\widetilde{x} = b - A\widetilde{x} = r. \tag{3.2}$$

Daraus folgt mit einer Vektornorm $\|\cdot\|$ und einer damit verträglichen Matrixnorm (vgl. (A.23))

$$\|e\| = \|A^{-1}r\| \le \|A^{-1}\|\,\|r\| \quad \text{und} \quad \|b\| = \|Ax\| \le \|A\|\,\|x\|. \tag{3.3}$$

Wir hatten schon in (2.6) die Konditionszahl einer Matrix gesehen. Sie ist eine entscheidende Größe bei der Untersuchung der Fehlerverstärkung.

Definition 3.1.2 Die **Konditionszahl** einer regulären Matrix A bezüglich der verwendeten Matrixnorm lautet

$$\kappa(A) \equiv \kappa_{\|\cdot\|}(A) := \|A\|\,\|A^{-1}\|. \tag{3.4}$$

Damit erhalten wir nun eine einfache Abschätzung für den relativen Fehler (vgl. Definition 2.1.9 und (3.3))[3]

$$\frac{\|x-\widetilde{x}\|}{\|x\|} = \frac{\|e\|}{\|x\|} \le \frac{\|A^{-1}\|\,\|r\|}{\|b\|\cdot\frac{1}{\|A\|}} = \|A^{-1}\|\,\|A\|\cdot\frac{\|r\|}{\|b\|} = \kappa(A)\,\frac{\|r\|}{\|b\|}. \tag{3.5}$$

Ist nun $\widetilde{x} = (1-\varepsilon)x$ mit $\varepsilon > 0$, dann lautet der relative Fehler

$$\frac{\|x-\widetilde{x}\|}{\|x\|} = \frac{\|e\|}{\|x\|} = \frac{\|\varepsilon x\|}{\|x\|} = \varepsilon$$

[3] Dabei verwenden wir $\|b\| = \|AA^{-1}b\| \le \|A\|\,\|A^{-1}b\|$, also $\|A^{-1}b\| \ge \|b\|\frac{1}{\|A\|}$.

und das Residuum lautet dann $r = b - A\widetilde{x} = Ax - A\widetilde{x} = A(x-(1-\varepsilon)x) = \varepsilon Ax = \varepsilon b$, also

$$\frac{\|r\|}{\|b\|} = \frac{\|\varepsilon b\|}{\|b\|} = \varepsilon = \frac{\|x-\widetilde{x}\|}{\|x\|}.$$

In diesem Fall ist die Abschätzung (3.5) offenbar zu pessimistisch, insbesondere, wenn die Konditionzahl sehr groß ist; (3.5) ist eine „worst case Abschätzung". Dies kann aber dennoch scharf sein, wie folgendes Beispiel zeigt.

Beispiel 3.1.3 Betrachte das LGS $Ax = b$ mit

$$A = \begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} \\ \frac{1}{3} & \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} \\ \frac{1}{4} & \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & \frac{1}{8} \\ \frac{1}{5} & \frac{1}{6} & \frac{1}{7} & \frac{1}{8} & \frac{1}{9} \end{pmatrix}, \qquad b := (b_j)_{j=1,\ldots,5} \text{ mit } b_j := \sum_{k=1}^{5} \frac{1}{k+j-1}.$$

Die exakte Lösung lautet $x = (1,\ldots,1)^T$, was man leicht nachrechnet.

Für die Matrixnormen und die Konditionszahlen erhält man in den Fällen $p \in \{1,2,\infty\}$

$$\begin{aligned} \|A\|_1 &= 2.28\overline{3}, & \|A^{-1}\|_1 &= 413280, & \kappa_1(A) &= 943656, \\ \|A\|_2 &\approx 1.5670507, & \|A^{-1}\|_2 &\approx 304142.84, & \kappa_2(A) &\approx 476607.25, \\ \|A\|_\infty &= 2.28\overline{3}, & \|A^{-1}\|_\infty &= 413280, & \kappa_\infty(A) &= \kappa_1(A). \end{aligned}$$

Betrachten wir nun die folgende (speziell gewählte) Näherung

$$\widetilde{x} = (0.993826,\ 1.116692,\ 0.493836,\ 1.767191,\ 0.623754)^T. \tag{3.6}$$

Diese Wahl erklärt sich wie folgt: Wir bezeichnen mit $0 \le \lambda_1 < \cdots < \lambda_5$ die Eigenwerte von A mit den zugehörigen normierten Eigenvektoren $\varphi_1, \ldots, \varphi_5$, $\|\varphi_j\|_2 = 1$. Dann gilt $\widetilde{x} - x \approx \varphi_1$ und $x \approx -0.004\,\varphi_1 - 0.042\,\varphi_2 + 0.244\,\varphi_3 + 0.972\,\varphi_4 + 1.998\,\varphi_5$, die Lösung x wird also von φ_5 dominiert. Damit erhalten wir

$$\frac{\|e\|_1\,\|b\|_1}{\|x\|_1\,\|r\|_1} \approx 0.41\,\kappa_1(A), \quad \frac{\|e\|_2\,\|b\|_2}{\|x\|_2\,\|r\|_2} \approx 0.89\,\kappa_2(A), \quad \frac{\|e\|_\infty\|b\|_\infty}{\|x\|_\infty\,\|r\|_\infty} \approx 0.74\,\kappa_\infty(A),$$

die Abschätzung (3.5) ist also in der richtigen Größenordung, sie ist nahezu scharf. ■

3.1.2 Störungen der Daten

Nun untersuchen wir, welchen Einfluss (kleine) Störungen in den Eingabedaten (also A und b) auf die Lösung x des linearen Gleichungssystems haben können, d.h., wir sind interessiert an der **Empfindlichkeit** (Sensitivität, Robustheit) der Lösung x bezüglich Störungen in den Daten. Dazu seien die gestörten Daten

$$A + \delta A \quad \text{und} \quad b + \delta b$$

und sei $x+\delta x$ die Lösung des gestörten LGS

$$(A+\delta A)(x+\delta x)=b+\delta b,$$

wobei wir annehmen, dass $A+\delta A$ regulär ist. Wir wollen nun also δx durch die Störungen δA und δb ausdrücken. Einfaches Ausmultiplizieren ergibt mit $Ax=b$

$$b+\delta b=(A+\delta A)(x+\delta x)=Ax+\delta Ax+(A+\delta A)\delta x=b+\delta Ax+(A+\delta A)\delta x,$$

also $\delta b=\delta Ax+(A+\delta A)\delta x$ und somit $\delta x=(A+\delta A)^{-1}(\delta b-\delta A\,x)=(I+A^{-1}\delta A)^{-1}A^{-1}(\delta b-\delta A\,x)$. Aufgrund der Submultiplikativität gilt $\|\delta x\|\le\|(I+A^{-1}\delta A)^{-1}\|\,\|A^{-1}\|\,(\|\delta b\|+\|\delta A\|\,\|x\|)$ und wir erhalten die Abschätzung

$$\frac{\|\delta x\|}{\|x\|}\le\|(I+A^{-1}\delta A)^{-1}\|\,\|A^{-1}\|\left(\frac{\|\delta b\|}{\|x\|}+\|\delta A\|\right). \tag{3.7}$$

Um $\|(I+A^{-1}\delta A)^{-1}\|$ abschätzen zu können, verwenden wir folgendes bekannte Resultat.

Lemma 3.1.4 Sei $B\in\mathbb{R}^{n\times n}$ mit $\|B\|<1$. Dann ist $(I-B)$ regulär mit

$$(I-B)^{-1}=\sum_{k=0}^{\infty}B^k\quad\textbf{(Neumann-Reihe)}\quad\text{und}\quad\|(I-B)^{-1}\|\le\frac{1}{1-\|B\|}. \tag{3.8}$$

Beweis. Mit der Voraussetzung $\|B\|<1$, der Submultiplikativität und der Summenformel für die geometrische Reihe erhalten wir für alle $m\in\mathbb{N}$, dass

$$\sum_{k=0}^{m}\|B^k\|\le\sum_{k=0}^{m}\|B\|^k\le\sum_{k=0}^{\infty}\|B\|^k=\frac{1}{1-\|B\|}<\infty. \tag{3.9}$$

Der Raum $\mathbb{R}^{n\times n}$ ist vollständig bezüglich $\|\cdot\|$ (er ist ein Banachraum) und aus der absoluten Konvergenz in (3.9) folgt die Konvergenz der Reihe $\sum_{k=0}^{\infty}B^k$. Ebenso folgt $A^k\to 0$ wegen $\|B\|<1$ aus $\|B^k\|\le\|B\|^k\to 0$ für $k\to\infty$. Die folgende Gleichung ist eine „Teleskopsumme“

$$\left(\sum_{k=0}^{m}B^k\right)(I-B)=I-B^{m+1}\quad\text{für alle } m\in\mathbb{N}. \tag{3.10}$$

Der Grenzübergang von (3.10) mit $m\to\infty$ führt aufgrund der Konvergenz der Reihe zur Gleichung

$$\left(\sum_{k=0}^{\infty}B^k\right)(I-B)=I. \tag{3.11}$$

Das bedeutet, dass $I-B$ regulär ist und $(I-B)^{-1}=\sum_{k=0}^{\infty}B^k$. Mit der Summenformel für die geometrische Reihe erhält man schließlich

$$\|(I-B)^{-1}\|=\left\|\lim_{m\to\infty}\sum_{k=0}^{m}B^k\right\|\le\lim_{m\to\infty}\sum_{k=0}^{m}\|B^k\|\le\lim_{m\to\infty}\sum_{k=0}^{m}\|B\|^k=\frac{1}{1-\|B\|}, \tag{3.12}$$

womit der Satz bewiesen ist. ∎

Wir wollen diesen Satz für $B = A^{-1}\delta A$ anwenden. Dazu muss $\|B\| < 1$ gelten. Dies realisieren wir wie folgt

$$\|B\| = \|A^{-1}\delta A\| \le \|A^{-1}\|\,\|\delta A\| = \kappa(A)\frac{\|\delta A\|}{\|A\|} \overset{!}{<} 1,$$

der relative Datenfehler der Matrix ist also so klein, dass er multipliziert mit der Konditionszahl kleiner eins ist. Je größer die Konditionszahl, desto kleiner die zugelassenen Datenstörungen. Damit haben wir also mit Lemma 3.1.4

$$\|(I + A^{-1}\delta A)^{-1}\| \le \frac{1}{1 - \|A^{-1}\delta A\|} \le \frac{1}{1 - \|A^{-1}\|\,\|\delta A\|}.$$

Damit erhalten wir schließlich aus (3.7)

$$\begin{aligned}\frac{\|\delta x\|}{\|x\|} &\le \|(I + A^{-1}\delta A)^{-1}\|\,\|A^{-1}\|\left(\frac{\|\delta b\|}{\|x\|} + \|\delta A\|\right) \le \frac{\|A^{-1}\|\,\|A\|}{1 - \|A^{-1}\|\,\|\delta A\|}\left(\frac{\|\delta b\|}{\|A\|\,\|x\|} + \frac{\|\delta A\|}{\|A\|}\right)\\ &\le \frac{\kappa(A)}{1 - \kappa(A)\frac{\|\delta A\|}{\|A\|}}\left(\frac{\|\delta b\|}{\|b\|} + \frac{\|\delta A\|}{\|A\|}\right).\end{aligned}$$

Wir halten das Ergebnis nun in einem Satz fest.

Satz 3.1.5 Sei $A \in \mathbb{R}^{n\times n}$ regulär und $x := A^{-1}b \in \mathbb{R}^n$. Es seien δb und δA Störungen von b bzw. A, so dass $A + \delta A$ regulär sei und

$$\kappa(A)\,\|\delta A\|/\|A\| < 1$$

gelte. Dann gilt für $\widetilde{x} := (A + \delta A)^{-1}(b + \delta b)$ die Fehlerabschätzung

$$\frac{\|x - \widetilde{x}\|}{\|x\|} \le \frac{\kappa(A)}{1 - \kappa(A)\frac{\|\delta A\|}{\|A\|}}\left(\frac{\|\delta b\|}{\|b\|} + \frac{\|\delta A\|}{\|A\|}\right). \tag{3.13}$$

Bemerkung 3.1.6 Für sehr kleine Störungen, d.h. für $\kappa(A)\,\frac{\|\delta A\|}{\|A\|} \ll 1$ wird (3.13) zu

$$\frac{\|x - \widetilde{x}\|}{\|x\|} \lesssim \kappa(A)\left(\frac{\|\delta b\|}{\|b\|} + \frac{\|\delta A\|}{\|A\|}\right), \tag{3.14}$$

die Konditionszahl ist also die entscheidende Größe, welche die Empfindlichkeit der Lösung bzgl. der Daten-Störungen δA und δb ausdrückt.

Zum Ende dieses Abschnittes wollen wir den praktischen Nutzen der Abschätzung (3.13) erörtern: Bei einer d-stelligen dezimalen Gleitkomma-Rechnung können die relativen Fehler der Ausgangsgrößen für beliebige, verträgliche Normen von der Größenordnung

$$\frac{\|\delta A\|}{\|A\|} \approx 5\cdot 10^{-d} \qquad \frac{\|\delta b\|}{\|b\|} \approx 5\cdot 10^{-d}$$

sein. Ist die Konditionszahl in der Größenordnung $\kappa(A) \approx 10^{\alpha}$ mit $5\cdot 10^{\alpha-d} \ll 1$, so ergibt die Abschätzung (3.13)

$$\|\delta x\| \leq 10^{\alpha-d+1} \|x\|.$$

Dies bedeutet, dass die Größe der Störung $\|\delta x\|$ maximal in der Größenordnung der $(d-\alpha-1)$-ten Dezimalstelle von $\|x\|$ liegen kann. Dies motiviert folgende **Faustregel: Wird ein LGS $Ax = b$ mit d-stelliger dezimaler Gleitkomma-Rechnung gelöst, und beträgt die Konditionszahl $\kappa(A) \approx 10^{\alpha}$, so sind (bezogen auf die betragsgrößte Komponente) nur $(d-\alpha-1)$ Dezimalstellen sicher.**

3.2 Determinanten-basierte Methoden

Aus der Linearen Algebra ist bekannt, dass ein LGS $Ax = b$ für jede rechte Seite $b \in \mathbb{R}^n$ genau dann lösbar ist, wenn A regulär (invertierbar) ist, was wiederum äquivalent ist zu

$$\det A \neq 0.$$

Vielleicht bereits aus der Schule ist bekannt, dass man die Determinate auch verwenden kann, um die Lösung eines LGS auszurechnen, mit der Cramerschen Regel. Im Folgenden sei stets $A = (a_{ij})_{i,j=1,\dots,n} \in \mathbb{R}^n$ und $b = (b_j)_{j=1,\dots,n} \in \mathbb{R}^n$.

Falls $\det A \neq 0$, so lässt sich $x = A^{-1}b$ mit der **Cramerschen Regel** berechnen, d.h.

$$x_i = \frac{1}{\det A} \begin{vmatrix} a_{11} & \dots & b_1 & \dots & a_{1n} \\ a_{21} & \dots & b_2 & \dots & a_{2n} \\ \vdots & & \vdots & & \vdots \\ a_{nn} & \dots & b_n & \dots & a_{nn} \end{vmatrix} = \frac{D_i}{D} \qquad (i = 1, \dots, n).$$

Dabei geht die im Zähler stehende Determinante D_i dadurch aus $D = \det A$ hervor, dass man die i-te Spalte der Matrix A durch den Vektor b der rechten Seite ersetzt.

Bemerkung 3.2.1 Man beachte, dass die Cramersche Regel eine Verbindung der Methode mit der Existenz- und Eindeutigkeits-Aussage herstellt in dem Sinne, dass die Methode genau dann zu einem Ergebnis führt, wenn das Problem lösbar ist (und sonst abbricht).

Berechnung der Determinante

Wir brauchen noch eine Berechnungsmethode für die Determinante, um die Cramersche Regel durchführen zu können. Hierfür gibt es zwei gängige Methoden, die Leibnizsche Darstellung und den Laplaceschen Entwicklungssatz. Wir stellen beide kurz vor und untersuchen insbesondere den damit verbundenen Rechenaufwand, d.h. die Anzahl der arithmetischen Operationen.

Bemerkung 3.2.2 — Rechenaufwand. Im Folgenden werden die Verknüpfungen Multiplikation, Addition, Division und Subtraktion in ihrem Rechenaufwand nicht unterschieden und unter dem Begriff **Gleitkommaoperation** zusammengefasst (1 Gleitkommaoperation $\simeq 1\,\mathrm{FLOP}$[4]). Multiplikationen mit ± 1 bleiben im Allgemeinen unberücksichtigt.

Anderweitige Operationen wie z.B. Wurzelziehen werden gesondert betrachtet.

Nun zu den beiden Methoden.

Die Determinante von $A \in \mathbb{R}^{n\times n}$ lautet in der **Leibnizschen Darstellung**

$$\det A = \sum_{\pi} (-1)^{j(\pi)} a_{1i_1}\, a_{2i_2} \cdots a_{ni_n}\,,$$

wobei die Summe über alle möglichen $n!$ Permutationen π der Zahlen $1, 2, \ldots, n$ zu berechnen ist. Der Wert des Ausdrucks $(-1)^{j(\pi)}$ ergibt sich aus der Anzahl $j(\pi)$ der Inversionen der Permutation $\pi = \begin{pmatrix} 1 & 2 & \ldots & n \\ i_1 & i_2 & \ldots & i_n \end{pmatrix}$.

Satz 3.2.3 — Aufwand Leibnizsche Darstellung. Der Aufwand zur Berechnung von $\det A$ für $A \in \mathbb{R}^{n\times n}$ mit der Leibnizschen Darstellung beträgt

$$\mathrm{FLOP}(\det A) = n\, n! - 1\,.$$

Beweis. Der Aufwand (unter Vernachlässigung der Multiplikation mit $(-1)^{j(\pi)}$) ergibt sich wie folgt:

$$\begin{aligned}\mathrm{FLOP}(\det A) &= \text{„}(n!-1) \text{ Additionen } + n! \text{ Produkte mit } n \text{ Faktoren“} \\ &= n! - 1 + n!(n-1) = n\, n! - 1,\end{aligned}$$

womit die Aussage des Satzes bewiesen ist. ∎

Die Leibnizsche Darstellung wird oft auch als Definition der Determinante verwendet. Nun zu zweiten Methode.

Die Determinante von $A \in \mathbb{R}^{n\times n}$ lässt sich rekursiv mit Hilfe des **Laplaceschen Entwicklungssatzes** berechnen:

$$\det A = \sum_{j=1}^{n} (-1)^{1+j} a_{1j} \det(A_{1j}),$$

wobei A_{1j} diejenige Matrix ist, die aus A durch Streichen der ersten Zeile und der j-ten Spalte entsteht.

Wir können auch hier den Aufwand bestimmen.

Satz 3.2.4 — Aufwand Laplacescher Entwicklungssatz. Sei $A \in \mathbb{R}^{n\times n}$ $(n \geq 2)$. Der Aufwand zur Berechnung von $\det A$ mit dem Laplaceschen Entwicklungssatz beträgt

$$\mathrm{FLOP}(\det A) = \sum_{k=0}^{n} \frac{n!}{k!} - 2 < e\, n! - 2\,.$$

Beweis. Die Ungleichung ist klar. Die Gleichung lässt sich induktiv beweisen. Für $n = 2$ (Induktionsanfang) lautet die Determinante von $A = (a_{ij}) \in \mathbb{R}^{2\times 2}$ bekanntermaßen $det(A) =$

[4] FLOP=Floating Point OPeration; nicht zu verwechseln mit FLOP/s oder flops (FLOPs per second), welches als Maßeinheit für die Geschwindigkeit von Computersystemen verwendet wird.

$a_{11}a_{22} - a_{21}a_{12}$, also ist der Aufwand zwei Multiplikationen und eine Addition, d.h.

$$\mathrm{FLOP}\left(\det(\mathbb{R}^{2\times 2})\right) = 3 = \frac{2!}{0!} + \frac{2!}{1!} + \frac{2!}{2!} - 2\,.$$

Nun sei die Behauptung für ein $n \geq 2$ bewiesen (Induktionsannahme, IA), dann gilt

$$\begin{aligned}
\mathrm{FLOP}\Big(\det(\mathbb{R}^{(n+1)\times(n+1)})\Big) &= \text{„}n \text{ Additionen } + (n+1) \text{ Multiplikationen} \\
&\qquad + (n+1) \text{ Berechnungen von } \det(\mathbb{R}^{n\times n})\text{“} \\
&= n + (n+1) + (n+1)\cdot \mathrm{FLOP}(\det(\mathbb{R}^{n\times n})) \\
&\overset{\text{IA}}{=} 2n + 1 + (n+1)\cdot\Big(\sum_{k=0}^{n}\frac{n!}{k!} - 2\Big) \\
&= 2n + 1 + \sum_{k=0}^{n}\frac{(n+1)!}{k!} - 2(n+1) \\
&= \sum_{k=0}^{n}\frac{(n+1)!}{k!} - 1 = \sum_{k=0}^{n+1}\frac{(n+1)!}{k!} - 2,
\end{aligned}$$

also die Behauptung. ■

■ **Beispiel 3.2.5** Ein einfaches Beispiel soll zeigen, dass man die Determinante überhaupt nur für sehr kleine allgemeine Matrizen mit dem Laplaceschen Entwicklungssatz berechnen kann. Auch die rasante Leistungssteigerung der Computer hilft dabei wenig.

Stand Juli 2023 führte der Computer namens „Frontier“ im Oak Ridge National Laboratory in den USA mit einer Rechenleistung von 1.1 ExaFLOPs $= 1.1\cdot 10^{18}$ FLOPs die Liste der Top 500 an. Wir wollen dies vergleichen mit dem im Jahr 2008 schnellsten europäischen Rechner JUGENE im Forschungszentrum Jülich bei Aachen mit einer Leistung von 180 TFLOPs $= 180\cdot 10^{12}$ FLOPs.

n	2008		2023	
20	10.21	[hrs]	0.10	[min]
21	214.32	[hrs]	2.10	[min]
22	0.54	[yrs]	0.77	[hrs]
23	12.37	[yrs]	17.75	[hrs]
24	296.92	[yrs]	0.60	[mon]
25	7422.88	[yrs]	1.21	[yrs]
26	192994.95	[yrs]	31.58	[yrs]

Wir sehen also, dass selbst auf den schnellsten Rechnern $n = 30$ außerhalb jeder Reichweite ist. Außerdem sehen wir zwar deutlich, wie sehr die Rechenleistung in den 15 Jahren von 2008 bis 2023 zugenommen hat, aber die Steigerung von $n = 20$ in 10.21 Stunden auf $n = 23$ in ähnlicher Zeit von 17.75 Stunden nicht wirklich signifikant ist.

Nun kann man vielleicht hoffen, dass Quantencomputer hier eine erhebliche Verbesserung bewirken können, aber eine deutliche Beschleunigung der numerischen Verfahren ist essenziell, damit eine Chance hat, realistische Problemgrößen von $n \sim 10^9$ oder gar mehr behandeln zu können. ■

3.3 Gestaffelte Systeme

Wir haben hoffentlich hinreichend motiviert, warum wir effiziente numerische Verfahren zur Lösung von LGS benötigen. Ein übliches Vorgehen ist es, ein allgemeines Problem zunächst so zu modifizieren, dass man es auf (möglicherweise mehrere) einfachere Spezialfälle reduzieren kann, die man dann besonders effizient behandeln kann. Im Fall von $Ax = b$ könnte man beispielsweise die Matrix A geeignet zerlegen und dann Gleichungslöser für die Teile der Zerlegung konstruieren. Daher betrachten wir zunächst einige Spezialfälle von Matrizen, die eine besonders effiziente Lösung der entsprechenden Systeme erlauben.

3.3.1 Diagonalmatrizen

Man bezeichnet eine Matrix $A = (a_{ij})$ als **Diagonalmatrix**, falls $a_{ij} = 0$ für $i \neq j$ gilt. Häufig schreibt man eine Diagonalmatrix A mit Diagonaleinträgen $a_{11}, \ldots, a_{nn}$ auch einfach

$$A = \operatorname{diag}(a_{11}, \ldots, a_{nn}).$$

Falls $a_{ii} \neq 0$, kann man die Inverse sofort hinschreiben $A^{-1} = \operatorname{diag}(1/a_{11}, \ldots, 1/a_{nn})$ und damit kann man das LGS sofort durch $x_i = b_i/a_{ii}$ lösen.[5]

Algorithmus 3.3.1 — Lösen von $Ax = b$ mit Diagonalmatrix A.

Input: $A \in \mathbb{R}^{n\times n}$ Diagonalmatrix mit $a_{ii} \neq 0$, $b \in \mathbb{R}^n$

1: **for** $i = 1, \ldots, n$ **do**
2: $\quad x_i = b_i/a_{ii}$
3: **end for**

Output: $x = (x_1, \ldots, x_n)^T$

Ganz offensichtlich ist der Aufwand n FLOP, was optimal für die Bestimmung von $x \in \mathbb{R}^n$ ist. Eine einfache MATLAB®-Realisierung ist im Folgenden dargestellt.

MATLAB-Beispiel:

Man löse $Ax = b$ mit

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 4 \end{pmatrix}, \quad b = \begin{pmatrix} 5 \\ 3 \\ 2 \end{pmatrix}.$$

```
>> A=diag([1,2,4]);
>> b=[5;3;2];
>> for j=1:3, x(j)=b(j)/A(j,j); end
>> x
x =
     5.0000     1.5000     0.5000
```

3.3.2 Dreiecksmatrizen – gestaffelte Systeme

Der „nächstschwierigere“ Fall ist der einer **Dreiecksmatrix** A.

Definition 3.3.2 Sei $A = (a_{ij})_{i,j=1,\ldots,n} \in \mathbb{R}^{n\times n}$. Dann heißt A *obere Dreiecksmatrix*, falls $a_{ij} = 0$ für $i > j$ und *untere Dreiecksmatrix* (häufig als L bezeichnet), falls $a_{ij} = 0$ für $i < j$ (oft mit R bezeichnet, in der englischen Literatur U)[6]. Eine Matrix heißt *Dreiecksmatrix*, wenn sie entweder eine untere oder eine obere Dreiecksmatrix ist; sie

[5] Diagonalmatrizen sind i.W. die einzige Ausnahme, bei der man ein LGS über die Bestimmung der Inversen berechnet. Im Allgemeinen ist die Berechnung der Inversen um ein Vielfaches teurer als das Lösen des LGS.

heißt *unipotente Dreiecksmatrix*, falls zusätzlich $a_{ii} = 1$. Ein Gleichungssystem $Ax = b$ mit einer Dreiecksmatrix A heißt *gestaffelt*.

Obere Dreiecksmatrizen

Sei R eine rechte (obere) Dreiecksmatrix. Das entsprechende gestaffelte Gleichungssystem $Rx = z$ hat die Form[7]

$$\begin{array}{ccccccccc} r_{11}x_1 & + & r_{12}\,x_2 & + & \cdots & + & r_{1n}\,x_n & = & z_1 \\ & & r_{22}\,x_2 & + & \cdots & + & r_{2n}\,x_n & = & z_2 \\ & & & & \ddots & & \vdots & & \vdots \\ & & & & & & r_{nn}\,x_n & = & z_n \end{array} \tag{3.15}$$

Für die obere Dreiecksmatrix R gilt, dass

$$\det(R) = r_{11} \cdot r_{22} \cdot \dots \cdot r_{nn}$$

und daher ist R regulär genau dann, wenn $r_{ii} \neq 0$ für alle $i = 1, \dots, n$, was wir im Folgenden annehmen wollen. Unter dieser Voraussetzung können wir die Lösung x durch sukzessive Auflösung erhalten, beginnend mit der letzten Zeile:

$$\begin{array}{lcll} x_n & := & z_n/r_{nn}, & \text{falls} \quad r_{nn} \neq 0, \\ x_{n-1} & := & (z_{n-1} - r_{n-1,n}x_n)/r_{n-1,n-1}, & \text{falls} \quad r_{n-1,n-1} \neq 0, \\ \vdots & \vdots & \quad \vdots & \quad \vdots \\ x_k & := & (z_k - \sum_{i=k+1}^{n} r_{ki}x_i)/r_{kk}, & \text{falls} \quad r_{kk} \neq 0, \\ \vdots & \vdots & \quad \vdots & \quad \vdots \\ x_1 & := & (z_1 - r_{12}x_2 - \ldots - r_{1n}x_n)/r_{11}, & \text{falls} \quad r_{11} \neq 0. \end{array} \tag{3.16}$$

Dieser Algorithmus wir auch als *Rückwärtseinsetzen* oder *Rückwärtssubstitution* bezeichnet und ist wiederum genau dann anwendbar, wenn das LGS für jede rechte Seite eindeutig lösbar ist, wir haben also wieder eine Verknüpfung von Algorithmus und der Wohlgestelltheit des Problems.

Untere Dreiecksmatrizen

Analog zum obigen Vorgehen lässt sich auch ein gestaffeltes lineares Gleichungssystem der Form

$$Lx = z$$

mit einer unteren Dreiecksmatrix $L \in \mathbb{R}^{n \times n}$ lösen. In diesem Fall beginnt man in der ersten Zeile mit der Berechnung von x_1 und arbeitet sich dann bis zur letzten Zeile zur Bestimmung von x_n vor, was man als *Vorwärtssubstitution* oder *Vorwärtseinsetzen* bezeichnet.

Satz 3.3.3 — Rechenaufwand für Dreiecksmatrizen. Sei $A \in \mathbb{R}^{n \times n}$ eine reguläre Dreiecksmatrix und $x, b \in \mathbb{R}^n$. Dann gilt für die Matrix-Vektor-Multiplikation Ax und die Lösung

[6] L: links/lower; R: rechts, U: upper.
[7] Es wird später klar, warum die rechte Seite hier z und nicht b heißt.

$A^{-1}b$ des gestaffelten Systems

$$\mathrm{FLOP}(Ax) = n^2 \quad \text{und} \quad \mathrm{FLOP}(A^{-1}b) = n^2 .$$

Bei der Berechnung von $A^{-1}b$ ist im Gegensatz zu Ax die Reihenfolge, in der die Zeilen „abgearbeitet" werden, festgelegt.

Beweis. Es genügt, den Fall einer regulären oberen Dreiecksmatrix $R \in \mathbb{R}^{n\times n}$ zu betrachten. Für den Rechenaufwand zur Lösung von $Rx = b$ ergibt sich:

i) für die i-te Zeile: je $(n-i)$ Additionen und Multiplikationen sowie eine Division

ii) insgesamt für die Zeilen n bis 1: Betrachten wir zuerst die Summenformel

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

Mit i) erhält man insgesamt einen Aufwand von $2\,\frac{n^2-n}{2} + n = n^2$ Operationen. Der zweite Teil der Aussage folgt analog. ∎

Bemerkung 3.3.4 Die Bezeichnung $\mathrm{FLOP}(A^{-1}b)$ **bedeutet nicht**, dass man erst die Inverse A^{-1} bestimmt und diese dann mit einem Vektor multipliziert, sondern ist ein Synonym für die Lösung des linearen Gleichungssystems. Man könnte die Inverse dadurch bestimmen, dass man die n LGS mit den kanonischen Einheitsvektoren als rechte Seite löst. Der Aufwand wäre also $\mathrm{FLOP}(A^{-1}) = n \cdot \mathrm{FLOP}(A^{-1}b)$, weswegen die Berechnung der Inversen aufwendig ist.

Eine MATLAB®-Realisierung zur Lösung von $Rx = z$, bei der die Matrix R zeilenweise durchlaufen wird, und ein Anwendungsbeispiel sind im Folgenden dargestellt.

MATLAB-Funktion: Rinvb.m

```
function x = Rinvb(R,z)
% compute solution of R * x = z with upper triangular matrix R
n = size(R,1);
x = zeros(n,1);
for j = n:-1:1
  for k=j+1:n
    z(j) = z(j) - R(j,k) * x(k);
  end
  x(j) = z(j) / R(j,j);
end
```

MATLAB-Beispiel:

Man löse $Rx = b$ mit

$$R = \begin{pmatrix} 4 & 1 & 1 \\ 0 & 3 & 2 \\ 0 & 0 & 1 \end{pmatrix}, \quad z = \begin{pmatrix} 9 \\ 12 \\ 3 \end{pmatrix}.$$

```
>> R = [4,1,1;0,3,2;0,0,1];
>> z = [9;12;3];
>> x = Rinvb(R,b);
>> x'
ans =
      1      2      3
```

Eine alternative Realisierung, bei der die Matrix R spaltenweise durchlaufen wird, ist mit `Rinvb2.m` gegeben. Was ist der Unterschied? Machen Sie sich dies klar!

MATLAB-Funktion: Rinvb2.m

```
function x = Rinvb2(R,z)
% compute solution of R * x = z with upper triangular matrix R
n = size(R,1);
x = zeros(n,1);
for j = n:-1:1
  x(j) = z(j) / R(j,j);
  for k=1:j-1
    z(k) = z(k) - R(k,j) * x(j);
  end
end
```

Für dieses Beispiel ziehen wir `ChatGBT` heran:

```
Gib mir ein Matlab-Programm für gestaffelte Gleichungssysteme mit
einer oberen Dreiecksmatrix
```

```
Gerne! Hier ist ein einfaches MATLAB-Programm, das ein gestaffeltes
Gleichungssystem mit einer oberen Dreiecksmatrix löst:
```

MATLAB-Funktion per ChatGBT: solve_upper_triangular.m

```
function x = solve_upper_triangular(A, b)
    % A ist die obere Dreiecksmatrix
    % b ist der Vektor auf der rechten Seite des Gleichungssystems

    n = length(b);
    x = zeros(n, 1);

    % Rueckwaertssubstitution
    for i = n:-1:1
        x(i) = b(i) / A(i, i);
        for j = i-1:-1:1
            b(j) = b(j) - A(j, i) * x(i);
        end
    end
end
```

Vergleichen Sie nun die drei Algorithmen.

3.4 Gaußsche Eliminationsmethode

Im Folgenden werden wir uns mit der Gaußschen[8] Eliminationsmethode zur Lösung von LGS $Ax = b$ mit regulärer Matrix $A \in \mathbb{R}^{n \times n}$ beschäftigen. Obwohl mit dem Namen Gauß verbunden, wurde schon von Lagrange[9] im Jahr 1759 eine Methode veröffentlicht, welches die grundlegenden Ideen der Methode enthält. Gauß veröffentlichte das von ihm im Rahmen seiner Entwicklung und Anwendung der Methode der kleinsten Quadrate (Kapitel 4) modifizierte Verfahren in seiner 1809 erschienenen Schrift *Theoria motus corporum coelestium in sectionibus conicis solem ambientium* (Theorie der Bewegung der Himmelskörper, die in Kegelschnitten die Sonne umlaufen). Bereits damals erkannte man, dass Verfahren unter Verwendung der Determinante wegen des großen Rechenaufwands nicht anwendbar sind. Die heute unter dem Namen Gauß-Verfahren bekannte Methode ist mit dem vor über 2100 Jahren in China verwendeten Verfahren (vgl. die Einleitung zu diesem Kapitel auf S. 15) identisch.

Die Idee besteht in einer multiplikativen Zerlegung der Matrix A der Form

$$A = L \cdot R, \tag{3.17}$$

mit einer unteren und einer oberen Dreiecksmatrix L bzw. R, wobei zunächst die Frage offen bleibt, ob und ggf. wann eine solche Zerlegung existiert. Wenn aber (3.17) gilt, dann könnte man ein beliebiges lineares Gleichungssystem $Ax = b$ durch eine Rückwärts- und Vorwärtssubstitution lösen mittels der beiden Schritte

i) löse $Lz = b$,

[8]Carl Friedrich Gauß, 1777 - 1855.
[9]Joseph-Louis Lagrange, 1736 - 1813.

ii) löse $Rx = z$.[10]

Wir brauchen also *nur* noch einen Algorithmus zur Bestimmung der Zerlegung (3.17) – und das ist die Gaußsche Eliminationsmethode, die wir nun beschreiben und die Sie vielleicht auch schon aus der Linearen Algebra kennen. Wir betonen hier insbesondere die numerischen Aspekte.

Definition 3.4.1 — LR-Zerlegung. Seien $A \in \mathbb{R}^{n\times n}$ sowie $R \in \mathbb{R}^{n\times n}$ rechte obere und $L \in \mathbb{R}^{n\times n}$ eine unipotente linke untere Dreiecksmatrix. Falls $A = L \cdot R$, so nennt man dies eine **LR-Zerlegung** von A.

Seien $A \in \mathbb{R}^{n\times n}$ regulär sowie $b \in \mathbb{R}^n$ gegeben und wir betrachten ein lineares Gleichungssystem $Ax = b$, also komponentenweise

$$\begin{array}{ccccccccc} a_{11}\,x_1 & + & a_{12}\,x_2 & + & \cdots & + & a_{1n}\,x_n & = & b_1 \\ a_{21}\,x_1 & + & a_{22}\,x_2 & + & \cdots & + & a_{2n}\,x_n & = & b_2 \\ \vdots & & \vdots & & & & \vdots & & \vdots \\ a_{n1}\,x_1 & + & a_{n2}\,x_2 & + & \cdots & + & a_{nn}\,x_n & = & b_n \end{array} \tag{3.18}$$

Wenn wir auf (3.18) ausschließlich Äquivalenzoperationen, also

1. Vertauschung von Zeilen,
2. Multiplikation einer Zeile mit einem Skalar $\neq 0$,
3. Addition eines Vielfachen einer Zeile zu einer anderen,

anwenden, dann wird die Lösungsmenge des Gleichungssystems (3.18) nicht verändert. Also sucht man nach einer Kombination dieser Äquivalenzoperationen in der Form, dass man das LGS in ein gestaffeltes umformen kann. Das Vorgehen ist wie folgt: Um (3.18) in ein gestaffeltes System umzuformen, muss die erste Zeile nicht verändert werden. Die restlichen Zeilen werden so modifiziert, dass die Koeffizienten vor x_1 verschwinden, d.h., die Variable x_1 wird aus den Gleichungen in den Zeilen 2 bis n eliminiert. Es soll also ein System der Art

$$\begin{array}{ccccccccc} a_{11}x_1 & + & a_{12}x_2 & + & \cdots & + & a_{1n}x_n & = & b_1 \\ & & a'_{22}x_2 & + & \cdots & + & a'_{2n}x_n & = & b'_2 \\ & & \vdots & & & & \vdots & & \vdots \\ & & a'_{n2}x_2 & + & \cdots & + & a'_{nn}x_n & = & b'_n\,. \end{array} \tag{3.19}$$

entstehen. Falls $a_{11} \neq 0$, so können wir dies wie folgt realisieren: Mit den so genannten *Multiplikatoren*

$$l_{i1} := a_{i1}/a_{11} \qquad (i = 2, 3, \dots, n) \tag{3.20}$$

sind die Elemente in (3.19) gegeben durch

$$a'_{ik} = a_{ik} - l_{i1}\,a_{1k}, \qquad b'_i = b_i - l_{i1}\,b_1.$$

Damit ist der erste Eliminationsschritt unter der Annahme $a_{11} \neq 0$ ausführbar. Wir wenden nun dasselbe Verfahren auf die Restmatrix $A^{(2)} := (a'_{i,j})_{i,j=2,\dots,n} \in \mathbb{R}^{(n-1)\times(n-1)}$ an. Dies

[10] Jetzt erkennt man, warum wir oben die rechte Seite z und nicht b verwendet haben.

setzt man rekursiv fort und erhält so eine Folge

$$A = A^{(1)} \to A^{(2)} \to \cdots \to A^{(n)} =: R$$

von (Rest-)Matrizen der speziellen Gestalt

$$A^{(k)} = \begin{pmatrix} a_{11}^{(1)} & a_{12}^{(1)} & \dots & & & a_{1n}^{(1)} \\ & a_{22}^{(2)} & \dots & & & a_{2n}^{(2)} \\ & & \ddots & & & \\ & & & a_{kk}^{(k)} & \dots & a_{kn}^{(k)} \\ & & & \vdots & & \vdots \\ & & & a_{nk}^{(k)} & \dots & a_{nn}^{(k)} \end{pmatrix} \in \mathbb{R}^{(n-k+1)\times(n-k+1)}, \tag{3.21}$$

auf die wir den folgenden Eliminationsschritt ausführen können, wenn das sogenannte **Pivotelement**[11] $a_{kk}^{(k)}$ nicht verschwindet:

$$\boxed{\begin{aligned} l_{ik} &:= a_{ik}^{(k)}/a_{kk}^{(k)} && \text{für} \quad i = k+1\dots,n, \\ a_{ij}^{(k+1)} &:= a_{ij}^{(k)} - l_{ik}\, a_{kj}^{(k)} && \text{für} \quad i,j = k+1\dots,n, \\ b_i^{(k+1)} &:= b_i^{(k)} - l_{ik}\, b_k^{(k)} && \text{für} \quad i = k+1,\dots,n, \end{aligned}} \tag{3.22}$$

Da jeder Eliminationsschritt eine lineare Operation auf den Zeilen von A ist, lässt sich der Übergang von $A^{(k)}$ und $b^{(k)}$ zu $A^{(k+1)}$ und $b^{(k+1)}$ als Multiplikation mit einer Matrix $L_k \in \mathbb{R}^{n\times n}$ von links darstellen, d.h.

$$A^{(k+1)} = L_k A^{(k)}, \quad b^{(k+1)} = L_k b^{(k)}. \tag{3.23}$$

Die entsprechende Matrix entsteht aus den Multiplikatoren

$$L_k = \begin{pmatrix} 1 & & & & & \\ & \ddots & & & & \\ & & 1 & & & \\ & & -l_{k+1,k} & 1 & & \\ & & \vdots & & \ddots & \\ & & -l_{n,k} & & & 1 \end{pmatrix} = I - \begin{pmatrix} 0 \\ \vdots \\ 0 \\ l_{k+1,k} \\ \vdots \\ l_{n,k} \end{pmatrix} \cdot e_k^T =: I - \ell_k \cdot e_k^T\,, \tag{3.24}$$

wobei e_k den k-te Einheitsvektor bezeichnet. Man sieht leicht, dass die Inverse L_k^{-1} aus L_k durch einen Vorzeichenwechsel der Multiplikatoren l_{ik}, $i > k$, entsteht und für das Produkt der L_k^{-1} gilt, dass dies die gesuchte unipotente linke untere Dreiecksmatrix ist, d.h.

$$L := L_1^{-1} \cdot \dots \cdot L_{n-1}^{-1} = \begin{pmatrix} 1 & 0 & \dots & \dots & 0 \\ l_{21} & 1 & \ddots & & \vdots \\ l_{31} & l_{32} & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & 0 \\ l_{n1} & \dots & & l_{n,n-1} & 1 \end{pmatrix}. \tag{3.25}$$

[11] In der Bedeutung: der „Dreh- und Angelpunkt“.

Die Matrix R ergibt sich dann

$$R = L^{-1}A = (L_1^{-1} \cdot \dots \cdot L_{n-1}^{-1})^{-1}A = L_{n-1} \cdot \dots \cdot L_1\, A. \tag{3.26}$$

Damit haben wir alle drei Schritte des Gaußschen Elimintationsverfahrens zusammen:

Das **Gaußsche Eliminationsverfahren** lautet für $A \in \mathbb{R}^{n\times n}$ und $b \in \mathbb{R}^n$:
i) $L \cdot R = A$ (LR-Zerlegung),
ii) $Lz = b$ (Vorwärtseinsetzen),
iii) $Rx = z$ (Rückwärtseinsetzen).

Wir haben bisher nichts zur Existenz und Eindeutigkeit der LR-Zerlegung gesagt.

Satz 3.4.2 — Eindeutigkeit der LR-Zerlegung. Existiert eine LR-Zerlegung, so sind L und R eindeutig bestimmt.

Beweis. Übung. ■

Rechenaufwand

Nun zur Bestimmung des Rechenaufwands. Dabei unterscheiden wir hier nicht zwischen $\pm$- und $\cdot$-Operationen, so dass sich die folgende Abschätzung teilweise von denen in der Literatur unterscheidet.

Satz 3.4.3 — Rechenaufwand. Sei $A \in \mathbb{R}^{n\times n}$ regulär mit existierender LR-Zerlegung $A = LR$ und $b \in \mathbb{R}^n$. Dann gilt

$$\begin{aligned} \mathrm{FLOP}(\text{LR-Zerlegung}) &= \tfrac{1}{6}(4n^3 - 3n^2 - n), \\ \mathrm{FLOP}(A^{-1}b) &= \tfrac{1}{6}(4n^3 + 9n^2 - n), \end{aligned}$$

d.h. die Lösung von $Ax = b$ kann in $\mathcal{O}(n^3)$ FLOPs berechnet werden.

Beweis. Die beiden gestaffelten Systeme können nach Satz 3.3.3 mit je n^2 Operationen gelöst werden, so dass

$$\mathrm{FLOP}(A^{-1}b) = \mathrm{FLOP}(\text{LR-Zerlegung}) + 2n^2.$$

Wir können uns also auf die Bestimmung des Rechenaufwandes der LR-Zerlegung beschränken:

(i) im k-ten Eliminationsschritt sind je $(n-k)$ Divisionen zur Berechnung der Multiplikatoren ℓ_{ik} und je $(n-k)^2$ Multiplikationen und Subtraktionen zur Berechnung der $a_{ij}^{(k+1)}$ erforderlich, also insgesamt $2\,(n-k)^2 + (n-k)$ Operationen;
(ii) Summation über alle Schritte ergibt mit bekannten Summenformeln

$$\begin{aligned} \sum_{k=1}^{n-1}\left((n-k) + 2\,(n-k)^2\right) &= \sum_{k=1}^{n-1}\left(k + 2\,k^2\right) = \frac{n(n-1)}{2} + \frac{(2n-1)n(n-1)}{3} \\ &= \tfrac{1}{6}n(n-1)(3+4n-2) = \tfrac{1}{6}(4n^3 - 3n^2 - n) \end{aligned}$$

und damit die Behauptung. ■

Speicheraufwand

Der Aufwand eines Algorithmus bestimmt sich nicht alleine aus der Anzahl der Rechenoperationen, sondern auch aus dem benötigten Speicher. Es stellt sich heraus, dass eine clevere Strategie bei der Berechnung der LR-Zerlegung den benötigten Speicheraufwand signifikant reduzieren kann. Würden wir die Zwischenmatrizen $A^{(k)}$ aus (3.21) speichern, beginnend mit $A^{(0)} := A$, würden wir einen Speicheraufwand von (im Allgemeinen) $\mathcal{O}(n^3)$ benötigen. Wir können dies aber signifikant reduzieren, wie wir nun sehen werden.

Nun wissen wir, an welchen Stellen die Elimination Nullen in $A^{(k)}$ erzeugt – diese Nullen müssen wir nicht speichern und können den freiwerdenden Speicherplatz verwenden, um dort die Multiplikatoren l_{ik} zu speichern. Auch bei den Matrizen L_k in (3.24) können wir Speicherplatz sparen, weil wir wissen, dass auf der Diagonalen nur 1er stehen, die wir nicht speichern müssen. Damit erhalten wir folgendes **Speicherschema**:

$$A \to \begin{pmatrix} a_{11}^{(1)} & \dots & \dots & a_{1n}^{(1)} \\ l_{21} & a_{22}^{(2)} & \dots & a_{2n}^{(2)} \\ \vdots & \vdots & & \\ l_{n1} & a_{n2}^{(2)} & & a_{nn}^{(2)} \end{pmatrix} \to \dots \to \begin{pmatrix} a_{11}^{(1)} & \dots & & & a_{1n}^{(1)} \\ l_{21} & a_{22}^{(2)} & & & a_{2n}^{(2)} \\ \vdots & & \ddots & & \\ \vdots & & & \ddots & \\ l_{n1} & \dots & \dots & l_{n,n-1} & a_{nn}^{(n)} \end{pmatrix}$$

$$\Longrightarrow L = \begin{pmatrix} 1 & 0 & \dots & \dots & 0 \\ l_{21} & \ddots & & & \vdots \\ \vdots & & \ddots & & \vdots \\ \vdots & & & \ddots & 0 \\ l_{n1} & \dots & \dots & l_{n,n-1} & 1 \end{pmatrix}, \qquad R = \begin{pmatrix} a_{11}^{(1)} & \dots & \dots & \dots & a_{1n}^{(1)} \\ 0 & \ddots & & & \vdots \\ \vdots & & \ddots & & \vdots \\ \vdots & & & \ddots & \vdots \\ 0 & \dots & \dots & 0 & a_{nn}^{(n)} \end{pmatrix}.$$

Wir können also durch dieses geschickte Überschreiben der Einträge von A alle Informationen dort ablegen und haben insgesamt einen Speicheraufwand von $\mathcal{O}(n^2)$.

3.5 Pivotisierung

Bisher haben wir uns noch nicht mit der Frage beschäftigt, ob für jede reguläre Matrix A immer eine LR-Zerlegung existiert. Ein weiterer Aspekt ist, dass wir bisher angenommen, dass alle Operationen exakt ausgeführt werden, wir haben also die Rechnerarithmetik außer Acht gelassen. Betrachten wir hierzu zwei Beispiele:

▪ **Beispiel 3.5.1** i) Betrachte die Matrix $A \in \mathbb{R}^{2\times 2}$

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

Offensichtlich ist $\det(A) = -1 \neq 0$, die Matrix ist also regulär. Sie besitzt aber *keine LR-Zerlegung*, was man wie folgt sieht. Wenn es eine LR-Zerlegung geben würde, dann gäbe es $a, b, c, d \in \mathbb{R}$ mit

$$A = \begin{pmatrix} 1 & 0 \\ a & 1 \end{pmatrix} \begin{pmatrix} b & c \\ 0 & d \end{pmatrix} = \begin{pmatrix} b & c \\ ab & ac + d \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix},$$

also $b = 0$, $c = 1$, $ab = 1$ und $ac + d = 0$, was ein Widerspruch ist!
Wenn wir nun die beiden Zeilen vertauschen, erhalten wir die Einheitsmatrix, die trivialerweise eine LR-Zerlegung mit $L = R = I$ besitzt.
In diesem Beispiel **liefert eine Permutation die LR–Zerlegung**!

ii) Für $0 < \varepsilon \ll 1$ betrachten wir die Matrix

$$A = \begin{pmatrix} \varepsilon & 1 \\ 1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ \frac{1}{\varepsilon} & 1 \end{pmatrix} \begin{pmatrix} \varepsilon & 1 \\ 0 & 1 - \frac{1}{\varepsilon} \end{pmatrix}$$

Vertauschen wir die beiden Zeilen mithilfe der Permutation $P = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$, so erhalten wir

$$PA = \begin{pmatrix} 1 & 1 \\ \varepsilon & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ \varepsilon & 1 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 0 & 1 - \varepsilon \end{pmatrix}$$

Die LR-Zerlegung von PA vermeidet die Division durch ε, ist also stabiler.
In diesem Beispiel liefert eine **Permutation bessere Stabilität**! ■

Fassen wir zusammen, was uns dieses Beispiel gezeigt hat:

- nicht jede reguläre Matrix besitzt eine LR-Zerlegung;
- Vertauschung der Zeilen (Permutationen des Indizes) liefert u.U. eine LR-Zerlegung und bessere Stabilität.

Betrachten wir zunächst die Gauß-Elimination mit **Spaltenpivotisierung**.

Algorithmus 3.5.2 — Gauß-Elimination mit Spaltenpivotisierung.

a) Wähle im Eliminationsschritt $A^{(k)} \to A^{(k+1)}$ einen Index $p \in \{k, \dots, n\}$, so dass

$$|a_{pk}^{(k)}| \geq |a_{jk}^{(k)}| \qquad \text{für } j = k, \dots, n\,.$$

Die Zeile p soll die **Pivotzeile**, $a_{pk}^{(k)}$ das **Pivotelement**, werden.

b) Vertausche die Zeilen p und k

$$A^{(k)} \to \widetilde{A}^{(k)} \quad \text{mit} \quad \widetilde{a}_{ij}^{(k)} = \begin{cases} a_{kj}^{(k)}, & \text{falls} \quad i = p, \\ a_{pj}^{(k)}, & \text{falls} \quad i = k, \\ a_{ij}^{(k)}, & \text{sonst.} \end{cases}$$

Nun gilt $|l_{ik}| = \left| \dfrac{\widetilde{a}_{ik}^{(k)}}{\widetilde{a}_{kk}^{(k)}} \right| = \left| \dfrac{\widetilde{a}_{ik}^{(k)}}{a_{pk}^{(k)}} \right| \leq 1$.

c) Führe den nächsten Eliminationsschritt angewandt auf $\widetilde{A}^{(k)}$ aus, $\widetilde{A}^{(k)} \to A^{(k+1)}$.

Anstelle der Spaltenpivotisierung mit Zeilentausch kann man auch eine **Zeilenpivotisierung** mit Spaltentausch durchführen. Beide Strategien benötigen im schlimmsten Fall $\mathcal{O}(n^2)$ zusätzliche Operationen. In der Praxis werden im Gegensatz zum oben genannten Algorithmus die Zeile bzw. Spalte nicht umgespeichert, sondern besondere Indexvektoren verwendet (siehe MATLAB®-Funktion `mylu.m`). Die Kombination von Spalten- und Zeilenpivotisierung führt zur **vollständigen Pivotsuche**, bei der die gesamte Restmatrix nach dem betragsgrößten Eintrag durchsucht wird, welches dann als **Pivotelement** verwendet wird. Wegen des Aufwands $\mathcal{O}(n^3)$ wird dies jedoch seltener angewandt.

MATLAB-Funktion: mylu.m

```
function [L,R,P] = mylu(A)
n = size(A,1);                          % get leading dimension of A
p = 1 : n;                              % pivot element vector
for k = 1 : n-1                         % consider k-th column
  [m,mptr] = max(abs(A(p(k:end),k))); % find pivot element
  tmp = p(k);                           % interchange in vector p
  p(k) = p(k-1+mptr);
  p(k-1+mptr) = tmp;

  for j = k+1 : n                       % modify entries in
    A(p(j),k) = A(p(j),k)/A(p(k),k);   % compute  l_jk, store in A
    for i = k+1 : n                     % (n-k-1)*(n-k-1) submatrix
      A(p(j),i) = A(p(j),i) ...
                  - A(p(j),k)*A(p(k),i);
    end
  end
end
L = tril(A(p,:),-1)+eye(n);             % these lines could be
R = triu(A(p,:));                       % neglected, all information
P = eye(n);                             % allready in A and p
P(:,p) = P;
```

Existenz der LR-Zerlegung

Wir kommen nun zurück zur Untersuchung der Existenz der LR-Zerlegung und werden sehen, dass die Pivotisierung tatsächlich der entscheidende Schlüssel dafür ist. Wir benötigen einige Vorbereitungen.

Satz 3.5.3 Sei $A \in \mathbb{R}^{n\times n}$ regulär. Dann existiert vor dem k-ten Eliminationsschritt des Gauß-Algorithmus stets eine Zeilen-/Spaltenpermutation derart, dass das k-te Diagonalelement von Null verschieden ist. Bei Zeilenpermutation (Spaltenpivotisierung) sind alle Einträge von L vom Betrag kleiner oder gleich eins.

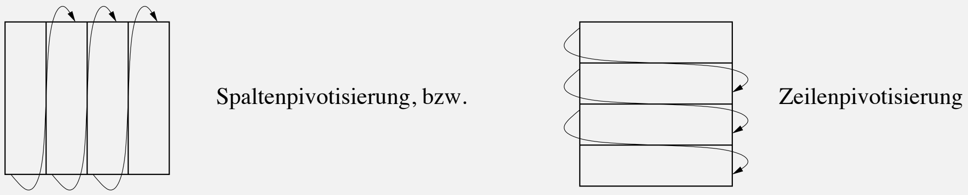
Beweis. Nach Voraussetzung gilt $\det A \neq 0$. Angenommen, es sei $a_{11} = 0$. Dann existiert eine Zeilenvertauschung, so dass das erste Pivotelement $a_{11}^{(1)}$ von Null verschieden und das betragsgrößte Element in der Spalte ist, d.h.

$$0 \neq |a_{11}^{(1)}| \geq |a_{i1}^{(1)}| \qquad \text{für} \quad i = 1, \ldots, n,$$

denn andernfalls wäre die erste Spalte eine Nullspalte und $\det(A) = 0$, im Widerspruch zur Voraussetzung. Die Zeilenvertauschung hat einen Vorzeichenwechsel in der Determinante zur Folge, ändert aber $|\det(A)|$ nicht, insbesondere bleibt diese von Null verschieden, die so entstandene Matrix ist wiederum regulär. Also können wir die gleiche Strategie auf die folgenden Restmatrizen anwenden.

Diese Schlussfolgerungen gelten analog bei der Zeilenpivotisierung. ■

Bemerkung 3.5.4 Bezüglich des Aufwandes macht es keinen Unterschied, ob man Spalten- und Zeilenpivotisierungen durchführt. Die Auswahl hängt von der Speichermethode der Matrix A ab: man favorisiert die Spaltenpivotisierung, wenn die Einträge der Matrix spaltenweise im Speicher abgelegt sind, z.B. in MATLAB®oder `Fortran`; die Zeilenpivotisierung ist bei zeilenweise Ablage (wie etwa in `C`) sinnvoller. Die Speicherzugriffe verdeutlicht die folgende Abbildung.



Zeilen- oder Spaltenpivotisierungen lassen sich formal durch die Multiplikation mit einer geeigneten Permutationsmatrix P von links bzw. rechts beschreiben, Zeilenvertauschung durch $P \cdot A$ und Spaltenpermutationen mittels $A \cdot P$.

Definition 3.5.5 Eine Matrix $P \in \{0,1\}^{n\times n}$ heißt **Permutationsmatrix**, welche jede Zeile und jede Spalte genau eine Eins und sonst nur Nullen enthält.

Lemma 3.5.6 Sei P eine Permutationsmatrix, dann gilt $\det P = \pm 1$ und $P^{-1} = P^T$.

Beweis. Übung. ■

Satz 3.5.7 — Existenz einer Zerlegung $PA = LR$. Zu jeder regulären Matrix A existiert eine Permutationsmatrix P, so dass

$$P \cdot A = L \cdot R.$$

Beweis. Die Aussage folgt direkt aus Satz 3.5.3. ■

Natürlich ist die Wahl der Permutationsmatrix, man denke etwa an den Fall, in dem es zwei Pivotelemente mit dem gleichen Betrag gibt. Die Pivotisierung sichert aber die Existenz einer LR-Zerlegung für jede reguläre Matrix. Natürlich kostet eine Pivotisierung Operationen, also Rechenzeit. Es stellt sich die Frage, ob eine Pivotisierung immer notwendig ist. Dies würde man natürlich gerne a priori anhand eines einfach (und effizient) überprüfbaren Kriteriums prüfen.

Definition 3.5.8 Eine Matrix $A = (a_{ij})_{i,j=1,\dots,n} \in \mathbb{R}^{n\times n}$ heißt **schwach diagonaldominant**, falls

$$|a_{ii}| \geq \sum_{\substack{k=1\\k\neq i}}^{n} |a_{ik}| \qquad \forall\, i = 1,\dots,n, \tag{3.27}$$

gilt; sie heißt **strikt diagonaldominant**, falls in (3.27) statt „≥“ für alle $i = 1,\dots,n$ stets „>“

gilt; A heißt **diagonaldominant**, falls zusätzlich zu (3.27) „>" für *einen* Index $i \in \{1, ..., n\}$ gilt.

Satz 3.5.9 Sei $A \in \mathbb{R}^{n \times n}$ regulär und diagonaldominant. Dann gibt es eine LR-Zerlegung

$$A = L \cdot R.$$

Beweis. Nach Voraussetzung ist $A^{(0} = A$ regulär und diagonaldominant. Angenommen, dies gelte nun auch für $A^{(k)}$ mit einem $k \geq 0$ (Induktionsannahme). Nach Voraussetzung gilt

$$|a_{ii}^{(k)}| \geq \sum_{\substack{j=k \\ j \neq i}}^{n} |a_{ij}^{(k)}| \geq 0 \qquad \forall\, i \in \{k, ..., n\} \qquad \text{und „>" für ein } i \in \{k, ..., n\}.$$

Da $A^{(k)}$ regulär ist, gilt daher entweder $|a_{kk}^{(k)}| \geq \sum_{j=k+1}^{n} |a_{kj}^{(k)}| > 0$ oder $\sum_{j=k+1}^{n} |a_{kj}^{(k)}| = 0$ und $|a_{kk}^{(k)}| > 0$. In beiden Fällen folgt, dass $a_{kk}^{(k)} \neq 0$. Der Eliminationsschritt lautet

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} - \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}} a_{kj}^{(k)} \qquad \forall\, i, j \in \{k+1, ..., n\}$$

und damit gilt für $i \in \{k+1, ..., n\}$ die folgende Ungleichung:

$$\begin{aligned}
\sum_{\substack{j=k+1 \\ j \neq i}}^{n} |a_{ij}^{(k+1)}| &= \sum_{\substack{j=k+1 \\ j \neq i}}^{n} \left| a_{ij}^{(k)} - \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}} a_{kj}^{(k)} \right| \leq \sum_{\substack{j=k+1 \\ j \neq i}}^{n} |a_{ij}^{(k)}| + \left| \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}} \right| \sum_{\substack{j=k+1 \\ j \neq i}}^{n} |a_{kj}^{(k)}| \\
&= \sum_{\substack{j=k \\ j \neq i}}^{n} |a_{ij}^{(k)}| - |a_{ik}^{(k)}| + \left| \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}} \right| \left\{ \sum_{j=k+1}^{n} |a_{kj}^{(k)}| - |a_{ki}^{(k)}| \right\} \\
&\leq |a_{ii}^{(k)}| - |a_{ik}^{(k)}| + \left| \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}} \right| \left\{ |a_{kk}^{(k)}| - |a_{ki}^{(k)}| \right\} \\
&= |a_{ii}^{(k)}| - \left| \frac{a_{ik}^{(k)} a_{ki}^{(k)}}{a_{kk}^{(k)}} \right| = \left| |a_{ii}^{(k)}| - \left| \frac{a_{ik}^{(k)} a_{ki}^{(k)}}{a_{kk}^{(k)}} \right| \right| \\
&\leq \left| a_{ii}^{(k)} - \frac{a_{ik}^{(k)} a_{ki}^{(k)}}{a_{kk}^{(k)}} \right| = |a_{ii}^{(k+1)}|.
\end{aligned}$$

Damit ist auch $A^{(k+1)}$ diagonaldominant; sie ist auch regulär, weil die Elimination aus Äquivalenztransformationen besteht. ■

Der obige Beweis zeigt auch, dass eine schwach diagonaldominante, reguläre Matrix auch diagonaldominant ist.

3.6 Nachiteration

Die oben diskutierten Pivotstrategien schließen nicht aus, dass die so berechnete Lösung x immer noch „ziemlich ungenau“ ist. Wie kann man x ohne großen Aufwand verbessern?

Häufig lässt sich dies durch eine so genannte **Nachiteration** mit Hilfe einer expliziten Auswertung des **Residuums** $r := b - A\widetilde{x}$ erreichen, wobei $\widetilde{x}$ die aktuelle Näherung an x ist, vgl. Definition 3.1.1. Die Idee besteht nun darin, $\widetilde{x}$ durch eine additive Korrektur „aufzudatieren“. Wir kennen die exakte Korrektur, nämlich den Fehler $e := x - \widetilde{x}$, denn $\widetilde{x} + e = x$, wenn $x = A^{-1}b$. Wir kennen ja bereiots den Zusammenhang zwischen Fehler und Residuum

$$Ae = Ax - A\widetilde{x} = b - A\widetilde{x} = r. \tag{3.28}$$

Da wir diese Gleichung nicht exakt lösen können, erhalten wir eine numerische Approximation $\widetilde{e}$ und setzen als neue Approximation

$$\widetilde{\widetilde{x}} := \widetilde{x} + \widetilde{e}.$$

Das Gleichungssystem (3.28) für die Nachiteration unterscheidet sich vom ursprünglichen LGS nur durch die rechte Seite, wir können also z.B. die LR-Zerlegung von A erneut ohne zusätzlichen Aufwand verwenden.

Den Prozess der Nachiteration kann man natürlich iterieren. In der Praxis genügen bei Spalten- oder Zeilenpivotisierung meist wenige Nachiterationen, um eine Lösung auf eine dem Problem angepasste Genauigkeit zu erhalten.

3.7 Cholesky-Verfahren

Die Gauß-Elimination mit Pivotisierung ist ein *black box*-Verfahren, es funktioniert immer, wenn die Matrix des LGS invertierbar ist. Wir haben keinerlei zusätzliche Annahmen an A gebraucht oder gemacht. Nun zeigt sich, dass man schnellere Verfahren für bestimmte Klassen von Matrizen konstruieren kann, zusätzliche Eigenschaften führen zu schnelleren Verfahren, die speziell für die jeweilige Problemklasse konstruiert sind und nicht mehr *black box* sind. Ein erstes solches Verfahren ist das *Cholesky-Verfahren*, welches für symmetrisch positiv definite Matrizen nur etwa den halben Aufwand an Operationen der Gauß-Elimination benötigt.

3.7.1 Symmetrisch positiv definite Matrizen

Definition 3.7.1 Eine Matrix $A \in \mathbb{R}^{n\times n}$ heißt

(i) *positiv definit*, wenn $x^T Ax > 0$ für alle $x \in \mathbb{R}^n$ mit $x \neq 0$ ist;

(ii) *symmetrisch positiv definit (s.p.d.)*, wenn sie symmetrisch und positiv definit ist.

Satz 3.7.2 — Eigenschaften positiv definiter Matrizen. Sei $A \in \mathbb{R}^{n\times n}$ s.p.d., dann gilt:

i) A ist invertierbar,

ii) $a_{ii} > 0$ für alle $i = 1, \dots, n$,

iii) $\max\limits_{i,j=1,\dots,n} |a_{ij}| = \max\limits_{i=1,\dots,n} a_{ii}$,

iv) Bei der Gauß-Elimination ohne Pivotsuche ist jede Restmatrix wiederum symmetrisch positiv definit.

Beweis. Wäre A nicht invertierbar, gäbe es wegen $\det A = 0$ einen Eigenwert $\lambda = 0$ mit einem zugehörigen Eigenvektor $x \neq 0$. Für diesen Eigenvektor gilt dann aber $x^T Ax = 0$, was im Widerspruch zu Definition 3.7.1 ist. Damit ist i) gezeigt.

Für den kanonischen Einheitsvektor e_i gilt $a_{ii} = e_i^T Ae_i > 0$ wegen Definition 3.7.1 und daher folgt die Aussage ii).

Behauptung iii) sei als Übung überlassen.

Um die verbleibende Behauptung iv) zu beweisen, schreiben wir $A = A^{(1)}$ in der Form

$$A^{(1)} = \left(\begin{array}{c|c} a_{11} & z^T \\ \hline z & B^{(1)} \end{array}\right),$$

wobei $z = (a_{12}, \ldots, a_{1n})^T$ sei. Nach einem Eliminationsschritt ergibt sich

$$A^{(2)} = L_1 A^{(1)} = \left(\begin{array}{c|c} a_{11} & z^T \\ \hline 0 & \\ \vdots & B^{(2)} \\ 0 & \end{array}\right) \quad \text{mit} \quad L_1 = \begin{pmatrix} 1 & & & \\ -l_{21} & 1 & & \\ \vdots & & \ddots & \\ -l_{n1} & & & 1 \end{pmatrix}.$$

Multipliziert man nun $A^{(2)}$ von rechts mit L_1^T, so wird auch z^T in der ersten Zeile eliminiert und die Teilmatrix $B^{(2)}$ bleibt unverändert, d.h.

$$L_1 A^{(1)} L_1^T = \left(\begin{array}{c|ccc} a_{11} & 0 & \cdots & 0 \\ \hline 0 & & & \\ \vdots & & B^{(2)} & \\ 0 & & & \end{array}\right).$$

Damit ist bewiesen, dass die Restmatrix $B^{(2)}$ symmetrisch ist. Können wir nun noch zeigen, dass $B^{(2)}$ auch wiederum positiv definit ist, so können wir unsere Argumentation sukzessive fortsetzen. Es sei $y \in \mathbb{R}^{n-1}$ mit $y \neq 0$. Da L_1 (und damit auch L_1^T) regulär ist, gilt $x := L_1^T (0, y_1, \ldots, y_{n-1})^T \neq 0$. Nach Voraussetzung ist A symmetrisch positiv definit, also gilt

$$0 < x^T Ax = (0, y^T) L_1 A L_1^T \begin{pmatrix} 0 \\ y \end{pmatrix} = (0, y^T) \left(\begin{array}{c|c} a_{11} & 0 \\ \hline 0 & B^{(2)} \end{array}\right) \begin{pmatrix} 0 \\ y \end{pmatrix} = y^T B^{(2)} y\,.$$

Somit haben wir gezeigt, dass $B^{(2)}$ symmetrisch positiv definit ist. ■

3.7.2 Konstruktion der Cholesky-Zerlegung

Mit Hilfe des letzten Satzes können wir die **rationale** Cholesky-Zerlegung für symmetrisch positiv definite Matrizen herleiten.

Satz 3.7.3 — rationale Cholesky-Zerlegung. Für jede symmetrische, positiv definite Matrix A existiert eine eindeutig bestimmte Zerlegung der Form

$$A = LDL^T,$$

wobei L eine unipotente untere Dreiecksmatrix und D eine positive Diagonalmatrix ist.

Beweis. Wir setzen die Konstruktion im Beweis von Satz 3.7.2 iv) für $k = 2, \dots, n-1$ fort und erhalten so unmittelbar L als Produkt der $L_1^{-1}, \dots, L_{n-1}^{-1}$ und D als Diagonalmatrix der Pivotelemente. Die Eindeutigkeit folgt aus der Konstruktion. ∎

Definition 3.7.4 — Cholesky-Zerlegung. Sei $A \in \mathbb{R}^{n \times n}$ s.p.d. mit rationaler Cholesky-Zerlegung $A = LDL^T$. Dann heißt

$$A = \overline{L}\,\overline{L}^T,$$

mit $\overline{L} := LD^{1/2}$, $D^{1/2} := \operatorname{diag}(\pm\sqrt{d_i})$ **Cholesky-Zerlegung** von A.

Bemerkung 3.7.5 Die Cholesky-Zerlegung ist nicht eindeutig, da $D^{1/2}$ nur bis auf Vorzeichen der Elemente in der Diagonalen eindeutig bestimmt ist und somit auch $\overline{L} = LD^{1/2}$ nicht eindeutig ist.

3.7.3 Cholesky-Algorithmus

Zur Herleitung des Algorithmus zur Berechnung einer Cholesky-Zerlegung betrachten wir folgende Gleichung

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & & & \vdots \\ \vdots & & & \vdots \\ a_{n1} & \cdots & \cdots & a_{nn} \end{pmatrix} \overset{!}{=} \begin{pmatrix} l_{11} & & & \\ l_{21} & l_{22} & & \\ \vdots & & \ddots & \\ l_{n1} & l_{n2} & \cdots & l_{nn} \end{pmatrix} \begin{pmatrix} l_{11} & l_{21} & \cdots & l_{n1} \\ & l_{22} & & l_{n2} \\ & & \ddots & \vdots \\ & & & l_{nn} \end{pmatrix}$$

$$= \begin{pmatrix} l_{11}^2 & l_{11}l_{21} & l_{11}l_{31} & \cdots & l_{11}l_{n1} \\ l_{11}l_{21} & l_{21}^2 + l_{22}^2 & l_{21}l_{31} + l_{22}l_{32} & \cdots & l_{21}l_{n1} + l_{22}l_{n2} \\ l_{11}l_{31} & l_{21}l_{31} + l_{22}l_{32} & l_{31}^2 + l_{32}^2 + l_{33}^2 & \cdots & l_{31}l_{n1} + l_{32}l_{n2} + l_{33}l_{n3} \\ \vdots & \vdots & \vdots & & \vdots \\ l_{11}l_{n1} & l_{21}l_{n1} + l_{22}l_{n2} & l_{31}l_{n1} + l_{32}l_{n2} + l_{33}l_{n3} & \cdots & \sum_{k=1}^{n} l_{nk}^2 \end{pmatrix},$$

d.h. für $i = k$ gilt $a_{kk} = \sum_{j=1}^{k} l_{kj}^2$ und für $i \neq k$ gilt $a_{ik} = \sum_{j=1}^{k} l_{ij} \cdot l_{kj}$, $k+1 \leq i \leq n$.

Diese Identitäten werden nun in einer geschickten Reihenfolge ausgewertet, was folgenden Algorithmus ergibt.

Cholesky-Verfahren zur Berechnung von L mit $A = LL^T$
Spaltenweise berechnet man für $k = 1, 2, ..., n$

$$l_{kk} = \left(a_{kk} - \sum_{j=1}^{k-1} l_{kj}^2\right)^{1/2}$$

$$l_{ik} = \frac{1}{l_{kk}}\left(a_{ik} - \sum_{j=1}^{k-1} l_{ij} \cdot l_{kj}\right), \quad k+1 \le i \le n.$$

Bemerkung 3.7.6 i) Aus der Cholesky-Zerlegung $A = LL^T$ ergibt sich für $1 \le k \le n$ die Abschätzung

$$\sum_{j=1}^{k} l_{kj}^2 \le \max_{1\le j\le n} |a_{jj}|.$$

Folglich sind alle Elemente der Matrix L betragsweise durch $\max\limits_{1\le j\le n} \sqrt{|a_{jj}|}$ beschränkt. Die Elemente können damit nicht allzu stark anwachsen, was sich günstig auf die Stabilität des Verfahrens auswirkt.

ii) Da A symmetrisch ist, wird nur Information oberhalb und einschließlich der Hauptdiagonalen benötigt. Wenn man die Diagonalelemente l_{kk} separat in einem Vektor der Länge n speichert, und die Elemente l_{jk}, $k < j$ unterhalb der Diagonale, so kann man die Information der Matrix A bewahren.

iii) Bei der algorithmischen Durchführung der Cholesky-Zerlegung liefert das Verfahren auch die Information, ob die Matrix positiv definit ist (Übung)

Satz 3.7.7 — Rechenaufwand Cholesky-Zerlegung. Sei $A \in \mathbb{R}^{n\times n}$ symmetrisch positiv definit, dann gilt

$$\mathrm{FLOP}(\text{Cholesky-Zerl. von } A) = \frac{2n^3 + 3n^2 - 5n}{6} + n\,\text{Wurzeln} = \frac{1}{3}n^3 + \mathcal{O}(n)\,.$$

Beweis. Untersuchen wir die Komplexität der Cholesky-Zerlegung zuerst für einen k-ten Zerlegungsschritt und bestimmen dann den Gesamtaufwand.

i) Für den k-ten Zerlegungsschritt, d.h. die Berechnung der Elemente l_{ik} für festen Spaltenindex, sind jeweils $(k-1)$ Multiplikationen, $(k-1)$ Subtraktionen und eine Wurzelberechnung zur Berechnung des Diagonalelements nötig. Für jedes Nebendiagonalelement werden $(k-1)$ Multiplikationen, $(k-1)$ Subtraktionen und eine Division benötigt, d.h., bei $(n-k)$ Elementen unterhalb des k-ten Diagonalelements sind dies in der Summe

$$\begin{aligned}&\big(2(k-1) + (n-k)(2k-1)\big)\ \mathrm{FLOP}\ \text{und } 1 \text{ Wurzelberechnung}\\ &= \big(-2k^2 + (3+2n)k - (n+2)\big)\ \mathrm{FLOP}\ \text{und } 1 \text{ Wurzelberechnung.}\end{aligned}$$

ii) Insgesamt ergibt sich dann für die Summe über alle Zerlegungsschritte

$$n\,\text{Wurzelberechnungen} + \sum_{k=1}^{n}\big(-2k^2 + (3+2n)k - (n+2)\big)$$

$$
\begin{aligned}
&= n\,\text{Wurzelberechnungen} + \Big(-2\,\frac{(2n+1)(n+1)n}{6} + (3+2n)\,\frac{(n+1)n}{2} - (n+2)n \Big) \\
&= n\,\text{Wurzelberechnungen} + \frac{-(4n^3+6n^2+2n)+(6n^3+15n^2+9n)-(6n^2+12n)}{6} \\
&= n\,\text{Wurzelberechnungen} + \frac{2n^3+3n^2-5n}{6}\,.
\end{aligned}
$$

Da eine einzelne Wurzelberechnung an Aufwand ein konstantes Vielfaches einer Gleitkommaoperation benötigt, kann man den Term n Wurzelberechnungen $+ (3n^2 - 5n)/6$ zu $\mathcal{O}(n^2)$ Operationen zusammenfassen. ■

3.8 Bandmatrizen

Bandmatrizen tauchen z.B. bei der Diskretisierung von Differenzialgleichungen auf, vgl. die „Standardmatrix" in Beispiel 3.0.1. Für die Anwendungen sind insbesondere symmetrisch positiv definiten Bandmatrizen wichtig.

Definition 3.8.1 Unter der **Bandbreite** m einer symmetrischen Matrix $A \in \mathbb{R}^{n\times n}$ versteht man die kleinste natürliche Zahl $m < n$, so dass gilt

$$a_{ik} = 0 \quad \text{für alle } i \text{ und } k \text{mit } |i-k| > m.$$

Die Bandbreite gibt also die Anzahl der Nebendiagonalen unterhalb bzw. oberhalb der Diagonalen an, welche die von Null verschiedenen Matrixelemente enthalten. In verallgemeinerter Form spricht man auch von *unterer* und *oberer Bandbreite*, welche sich auf naheliegende Weise definieren.

Satz 3.8.2 Sei A eine s.p.d. Bandmatrix mit Bandbreite m und Cholesky-Zerlegung $A = LL^T$. Dann besitzt L einer dieselbe Bandstruktur, d.h.

$$l_{ik} = 0 \quad \text{für alle } i, k \text{ mit } i - k > m\,.$$

Beweis. Übung. ■

Speicherschema

Symmetrisch positiv definite Bandmatrizen kann man in ökonomischer Weise speichern. Wir verzichten auf eine formale Beschreibung, zeigen aber die Idee in der folgenden Abbildung 3.1.

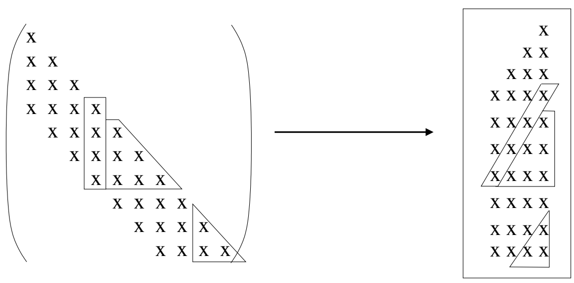


Abbildung 3.1: Reduktion und Speicherung einer symmetrisch positiv definiten Bandmatrix.

Algorithmus

Analog zur Herleitung des Cholesky-Verfahrens auf Seite 40 erhalten wir die Rechenvorschrift des Cholesky-Verfahrens für Bandmatrizen, wobei nun die Koeffizienten zeilenweise bestimmt werden. Für $n = 5$ und $m = 2$ sieht die Situation schematisch wie folgt aus:

$$\begin{pmatrix} l_{11} & 0 & 0 & 0 & 0 \\ l_{21} & l_{22} & 0 & 0 & 0 \\ l_{31} & l_{32} & l_{33} & 0 & 0 \\ 0 & l_{42} & l_{43} & l_{44} & 0 \\ 0 & 0 & l_{53} & l_{54} & l_{55} \end{pmatrix} \begin{pmatrix} l_{11} & l_{21} & l_{31} & 0 & 0 \\ 0 & l_{22} & l_{32} & l_{42} & 0 \\ 0 & 0 & l_{33} & l_{43} & l_{53} \\ 0 & 0 & 0 & l_{44} & l_{54} \\ 0 & 0 & 0 & 0 & l_{55} \end{pmatrix}$$

$$= \begin{pmatrix} l_{11}^2 & & \text{sym.} & & \\ l_{21}l_{11} & l_{21}^2 + l_{22}^2 & & & \\ l_{31}l_{11} & l_{31}l_{21} + l_{32}l_{22} & l_{31}^2 + l_{32}^2 + l_{33}^2 & & \\ & l_{42}l_{22} & l_{42}l_{32} + l_{43}l_{33} & l_{42}^2 + l_{43}^2 + l_{44}^2 & \\ & & l_{53}l_{33} & l_{53}l_{43} + l_{54}l_{44} & l_{53}^2 + l_{54}^2 + l_{55}^2 \end{pmatrix}.$$

Daraus ergibt sich dann folgendes Verfahren:

Cholesky-Verfahren zur Berechnung von L mit $A = LL^T$, wobei $A \in \mathbb{R}^{n\times n}$ mit Bandbreite $0 \leq m < n$. Zeilenweise berechnet man für $k = 1, 2, \dots, n$

$$l_{ki} = \frac{1}{l_{ii}}\Big(a_{ki} - \sum_{j=\max\{1,k-m\}}^{i-1} l_{kj} \cdot l_{ij}\Big), \quad \max\{1, k-m\} \leq i \leq k-1,$$

$$l_{kk} = \Big(a_{kk} - \sum_{j=\max\{1,k-m\}}^{k-1} l_{kj}^2\Big)^{1/2}.$$

Aufwand

Satz 3.8.3 Es sei $A \in \mathbb{R}^{n\times n}$ eine s.p.d. Bandmatrix mit Bandbreite $1 \leq m \leq n$. Dann beträgt der Aufwand zur Berechnung der Cholesky-Zerlegung von A

$$\begin{aligned} \mathrm{FLOP}(\text{Cholesky-Zerl. von } A) &= n(m^2 + 2m) - \frac{4m^3 + 9m^2 + 5m}{6} \\ &\quad + n \text{ Wurzelberechnungen} = \mathcal{O}(n\,m^2). \end{aligned}$$

Beweis. Wendet man die Cholesky-Zerlegung auf eine s.p.d. Bandmatrix A an, so ist die resultierende untere Dreiecksmatrix L mit $A = LL^T$ wieder eine Bandmatrix mit der gleichen Bandbreite, wie A sie hat.

i) Gehen wir davon aus, dass die Matrix L zeilenweise berechnet wird und betrachten zuerst den Aufwand für die ersten m Zeilen. Hier kann man das Ergebnis aus Satz 3.7.7 verwenden und erhält folgende Anzahl von Operationen:

$$m \text{ Wurzelberechnungen} + \frac{2m^3 + 3m^2 - 5m}{6}.$$

ii) In den verbleibenden $n - m$ Zeilen, in denen jeweils $m + 1$ Koeffizienten zu bestimmen sind, ergibt sich Folgendes: Um den j-ten von Null verschiedenen Nebendiagonaleintrag von L in der k-ten Zeile $m + 1 \leq k \leq n$ zu berechnen, sind jeweils $j - 1$

Multiplikationen, $j-1$ Subtraktionen und eine Division notwendig. Zur Berechnung des Diagonalelements werden m Multiplikationen, m Subtraktionen und eine Wurzelberechnung benötigt. Bei m Nebendiagonalelementen sind dies in der Summe

$$\begin{aligned} 1\,\text{Wurzelberechnung} + 2m + \sum_{j=1}^{m}(2j-1) &= 1\sqrt{\cdot} + m + 2\sum_{j=1}^{m} j \\ &= 1\sqrt{\cdot} + m + m(m+1) = 1\sqrt{\cdot} + m(m+2)\,. \end{aligned}$$

iii) Der gesamte Aufwand beträgt also

$$\begin{aligned} &n\,\text{Wurzelberechnungen} + \frac{2m^3+3m^2-5m}{6} + (n-m)m(m+2) \\ &= n\,\text{Wurzelberechnungen} + n(m^2+2m) - \frac{4m^3+9m^2+5m}{6}\,. \end{aligned}$$

Damit ist die Behauptung bewiesen. ■

Bemerkung 3.8.4 Ist für eine Klasse von s.p.d. Bandmatrizen aus $\mathbb{R}^{n\times n}$ für beliebiges n die Bandbreite beschränkt, so wächst der Aufwand zur Berechnung der Cholesky-Zerlegung asymptotisch nur **linear** in n, vgl. Beispiel 3.0.1. Man mache sich klar, dass dies „asymptotisch optimal" ist.

Kapitel 4

Lineare Ausgleichsprobleme

Lineare Ausgleichsprobleme (engl. *least squares problems*) treten immer dann auf, wenn der Zusammenhang von Parametern (Eingabe) und Ausgabe linear ist (oder als solches angenommen wird) und das entsprechende Gleichungssystem überbestimmt ist, wir also mehr Gleichungen als Unbekannte haben. Wir beginnen zur Motivation mit einem einfachen Beispiel.

■ **Beispiel 4.0.1** Gesucht sei eine Gerade $g(t) = \alpha t + c$, deren y-Werte den kleinsten Quadratsummenabstand (mittlerer quadratischer Abstand) zu den vorgegebenen Daten (t_i, b_i) haben, die sogenannte *Ausgleichsgerade*. Die geometrische Veranschaulichung dazu ist in der nachfolgenden Abbildung dargestellt. ■

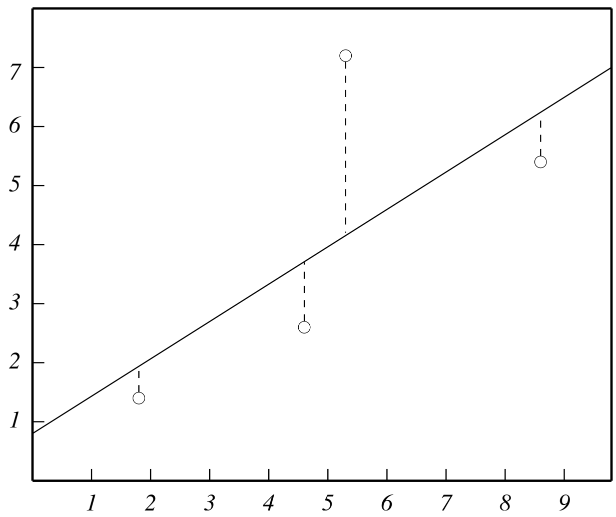


Abbildung 4.1: Die Ausgleichsgerade, die die Quadratsumme der Abstände minimiert

4.1 Die Methode der kleinsten Quadrate

Das Beispiel 4.0.1 können wir wie folgt mathematisch modellieren: zu gegebenen Punktepaaren $(t_1, b_1), (t_2, b_2), ..., (t_m, b_m)$ sind die in der Geradengleichung $g(t) = \alpha t + c$ freien

Parameter $x_1 := c$ (y-Achsenabschnitt) und $x_2 := \alpha$ (Steigung) so zu bestimmen, dass gilt

$$F(x) := \sum_{i=1}^{m} \big(g(t_i) - b_i\big)^2 = \sum_{i=1}^{m} \big(\alpha\, t_i + c - b_i\big)^2 = \sum_{i=1}^{m} \big(t_i x_2 + x_1 - b_i\big)^2 \to \min_{x \in \mathbb{R}^2},$$

oder in alternativer Formulierung

$$\min_{x \in \mathbb{R}^2} \left\| A\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} - b \right\|_2^2,$$

wobei wir nachfolgende Definitionen verwenden

$$A = \begin{pmatrix} 1 & t_1 \\ \vdots & \vdots \\ 1 & t_m \end{pmatrix} \in \mathbb{R}^{m \times 2}, \quad b = \begin{pmatrix} b_1 \\ \vdots \\ b_m \end{pmatrix} \in \mathbb{R}^m.$$

Etwas allgemeiner kann man ein *lineares Ausgleichsproblem* basierend auf der *Methode der kleinsten Fehlerquadrate* so formulieren: Gegeben seien $m \in \mathbb{N}$ Messungen $b_1, \ldots, b_m$, von denen man einen linearen Zusammenhang zu $n \leq m$ Parametern $x_1, \ldots, x_n$ vermutet (oben war $n = 2$). Gesucht sind dann diejenigen optimalen Parameter $x \in \mathbb{R}^n$, die ein Minimum der Fehlerquadratsumme bilden, d.h.

$$\|Ax - b\|_2^2 \to \min_{x \in \mathbb{R}^n}, \tag{4.1}$$

wobei $A \in \mathbb{R}^{m \times n}$.

Bemerkung 4.1.1 (i) Für $n = m$ wird das Minimum in (4.1) für die Lösung x des linearen Gleichungssystems $Ax = b$ angenommen. Also ist ein lineares Ausgleichungsproblem eine Verallgemeinerung linearer Gleichungssysteme für überbestimmte Systeme.

(ii) Offensichtlich ist (4.1) äquivalent zu $\|Ax - b\|_2 \to \min_{x \in \mathbb{R}^n}$. Die Formulierung in (4.1) hat aber den Vorteil, dass die zu minimierende Funktion quadratisch, insbesondere konvex ist. Wir werden beide Formulierungen synonym verwenden.

(iii) In realen Problemen kann es wichtig sein, anstelle der Fehlerquadratsumme ein anderes Fehlermaß zu betrachten. Man denke etwa an die maximale Abweichung oder die Summe der Fehler. Dann wird (4.1) zu

$$\|Ax - b\| \to \min_{x \in \mathbb{R}^n}.$$

mit einer Norm $\|\cdot\|$, die zum gewünschten Fehlermaß passt. Für $\|\cdot\| = \|\cdot\|_\infty$ erhält man ein sogenanntes MinMax-Problem, für $\|\cdot\| = \|\cdot\|_1$ ein lineares Optimierungsproblem. Es sei betont, dass alle folgenden Verfahren *ausschließlich* für die Methode der kleinsten Fehlerquadrate geeignet sind. Für andere Fehlermaße benötigt man andere numerische Verfahren, die über diese einführende Vorlesung hinausgehen.

Nun zur (analytischen) Lösung von (4.1). Wir wollen also eine konvexe Funktion

$$F(x) := \|Ax - b\|_2^2 = (Ax - b, Ax - b)_2 = (Ax - b)^T (Ax - b)$$

$$= (Ax)^T Ax - (Ax)^T b - b^T Ax + b^T b = x^T A^T Ax - 2x^T A^T b + b^T b$$

minimieren. Wie üblich bei der Suche eines Minimums bilden wir Ableitung, also

$$0 \stackrel{!}{=} F'(x) = \nabla F(x) = 2A^T Ax - 2A^T b,$$

was man leicht nachrechnet. Diese letzte Gleichung ist offenbar äuivalent zu

$$A^T Ax \stackrel{!}{=} A^T b, \tag{4.2}$$

den sogenannten **Gaußschen Normalengleichungen**. Nun ist (4.2) nur eine notwendige, aber noch keine hinreichende Bedingung für das gesuchte Minimum. Dazu betrachten wir die zweite Ableitung, also

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

Falls A vollen Rang hat, ist $F''(x)$ positiv definit und damit ist eine Lösung x von (4.2) tatsächlich Lösng des linearen Ausgleichsproblems (4.1). Die Lösung ist eindeutig.

Nun könnte man sich damit zufrieden geben, die Normalengleichungen (4.2) numerisch zu lösen, etwa mit dem Cholesky-Verfahren. Dies ist aus zwei Gründen ungünstig:

1. Zunächst ist $A^T A \in \mathbb{R}^{n\times n}$ für $A \in \mathbb{R}^{m\times n}$, $m \gg n$, und die Matrix-Matrix-Multiplikation benötigt $\mathcal{O}(mn^2)$ Operationen. Wir haben also die Problemgröße deutlich vergrößert.
2. Ein weiterer (und entscheidender) Nachteil besteht in der (sehr schlechten) Kondition der Normalengleichungen.

Um den zweiten Punkt genauer beschreiben zu können, brauchen wir einen Konditionsbegriff für nicht-quadratische Matrizen, da Definition 3.1.2 (S. 17) die Inverse der Matrix benötigt. Im Beispiel 2.1.8 (Seite 10) hatten wir die Kondition bei der Lösung eines LGS betrachtet. Dies können wir nun auf die Normalengleichung (4.2) anwenden und erhalten folgende Abschätzung:

$$\begin{aligned}\kappa_{\|\cdot\|}(A^T A) &= \max_{x\neq 0} \frac{\|A^T Ax\|}{\|x\|} \max_{y\neq 0} \frac{\|(A^T A)^{-1} y\|}{\|y\|} = \max_{x\neq 0} \frac{\|A^T Ax\|}{\|x\|} \max_{z\neq 0} \frac{\|z\|}{\|A^T Az\|} \\ &= \frac{\max\limits_{\|x\|=1} \|A^T Ax\|}{\min\limits_{\|x\|=1} \|A^T Ax\|}.\end{aligned} \tag{4.3}$$

Den Ausdruck auf der rechten Seite von (4.3) kann man offenbar als Definition der Konditionszahlen für allgemeine (auch nicht-quadratische) Matrizen verwenden:

Definition 4.1.2 Sei $B \in \mathbb{R}^{m\times n}$ und $\|\cdot\|$ eine Vektornorm, dann heißt

$$\kappa(B) \equiv \kappa_{\|\cdot\|}(B) := \frac{\max\limits_{\|x\|=1} \|Bx\|}{\min\limits_{\|x\|=1} \|Bx\|}.$$

Kondition von B bzgl. $\|\cdot\|$.

Bemerkung 4.1.3 Die beiden folgenden Aussagen lassen sich leicht als Übungsaufgabe zeigen:

i) Sei $A \in \mathbb{R}^{n\times n}$ regulär. Dann gilt

$$\frac{\max\limits_{\|x\|=1} \|Ax\|}{\min\limits_{\|x\|=1} \|Ax\|} = \|A\|\,\|A^{-1}\|,$$

d.h., die beiden Konditionsbegriffe stimmen überein.

ii) Falls A vollen Rang besitzt, gilt

$$\kappa_{\|\cdot\|}(A^TA) = \kappa_{\|\cdot\|}(A)^2.$$

Da die Matrix A Ergebnisse von Messungen beinhaltet, müssen wir von Redundanzen ausgehen. In diesem Fall wird der Nenner von (4.3) sehr klein und damit $\kappa_{\|\cdot\|}(A)$ sehr groß werden. Dann verstärkt das Quadrat aber die sowieso schon schlechte Kondition dramatisch. Also reagiert die Lösung der Normalengleichungen im Allgemeinen sehr sensitiv auf Störungen – wir brauchen einen anderen Zugang.

4.2 Die *QR*-Zerlegung

Der beschriebene notwendige andere Zugang beruht auf der QR-Zerlegung einer Matrix. Dazu führen wir zunächst einige Begriffe ein.

Definition 4.2.1 Eine quadratische Matrix $Q \in \mathbb{R}^{m\times m}$ heißt *orthogonal*, falls gilt:

$$QQ^T = Q^TQ = I.$$

Man sieht leicht, dass orthogonale Abbildungen stets längen- und winkeltreu sind, dass also für alle $x, y \in \mathbb{R}^m$ gilt

$$\begin{aligned} \|Qx\|_2^2 &= (Qx)^TQx = x^TQ^TQx = x^Tx = \|x\|_2^2, \quad \text{d.h. } \|Qx\|_2 = \|x\|_2,\\ (Qx, Qy)_2 &= x^TQ^TQy = x^Ty = (x,y)_2, \end{aligned}$$

mit der Euklidischen Norm definiert in (A.3).

Motivation: Zur Lösung des Minimierungsproblems $\min_{x\in\mathbb{R}^n} \|Ax-b\|_2^2$ setzen wir zunächst voraus, dass zu $A \in \mathbb{R}^{m\times n}$ mit vollem Rang $\operatorname{rang}(A) = n < m$ stets eine Zerlegung $A = QR$ existiert (und wir zeigen dies später auch), wobei $Q \in \mathbb{R}^{m\times m}$ orthogonal ist und $R \in \mathbb{R}^{m\times n}$ folgende Gestalt besitzt

$$R = \begin{pmatrix} * & * & * \\ & * & * \\ & & * \\ & & \\ 0 & & \end{pmatrix} = \begin{pmatrix} \widehat{R} \\ \hline 0 \end{pmatrix}.$$

Die Matrix $\widehat{R} \in \mathbb{R}^{n\times n}$ ist dabei eine reguläre obere Dreiecksmatrix. Damit können wir unser bisheriges Minimierungsproblem wie folgt modifizieren (mit $A = QR$ und $\|Q^T x\|_2 = \|x\|_2$):

$$\begin{aligned}\|Ax-b\|_2^2 = \|Q^T(Ax-b)\|_2^2 = \|Q^T Ax - Q^T b\|_2^2 = \|Rx - Q^T b\|_2^2 \\ = \left\| \begin{pmatrix} \widehat{R} \\ \hline 0 \end{pmatrix} x - \begin{pmatrix} (Q^T b)_{i=1,\dots,n} \\ \hline (Q^T b)_{i=n+1,\dots,m} \end{pmatrix} \right\|_2^2 \\ = \sum_{i=1}^{n} [(\widehat{R}x)_i - (Q^T b)_i]^2 + \sum_{i=n+1}^{m} (Q^T b)_i^2.\end{aligned}$$

Wir setzen $b_1 := (Q^T b)_{i=1,\dots,n}$ und $b_2 := (Q^T b)_{i=n+1,\dots,m}$ und erhalten

$$\|Ax-b\|_2^2 = \|\widehat{R}x - b_1\|_2^2 + \|b_2\|_2^2.$$

Nun wollen wir ja das Minimum dieses Ausdruckes bzgl. $x \in \mathbb{R}^n$ bestimmen. Offenbar ist $\|b_2\|_2^2$ unabhängig von x. Diesen Anteil nennt man den **unvermeidbaren Modellfehler**. Im Beispiel der Ausgleichsgeraden wäre er Null, wenn alle Messpunkte exakt auf einer Geraden liegen würden. Also sehen wir

$$\|Ax-b\|_2 \longrightarrow \min_{x\in\mathbb{R}^n}! \iff \widehat{R}x = b_1.$$

Also: wenn wir eine QR-Zerlegung von A kennen würden, könnten wir das lineare Ausgleichsproblem durch das Lösen eines gestaffelten linearen Gleichungssystems mit Rückwärtseinsetzen lösen. Dieser Faktorisierungsansatz ($A = QR$) führt uns nun zu folgenden Fragen:

1. Zu welchen Matrizen $A \in \mathbb{R}^{m\times n}$ existiert eine derartige QR-Zerlegung?
2. Wie bestimmen wir eine solche Zerlegung numerisch möglichst effizient?

Wir beantworten zunächst die erste Frage.

Satz 4.2.2 Zu jeder Matrix $A \in \mathbb{R}^{m\times n}$ mit Maximalrang $n < m$ existiert eine orthogonale Matrix $Q \in \mathbb{R}^{m\times m}$ derart, dass

$$A = QR \text{ mit } R = \begin{pmatrix} \widehat{R} \\ 0 \end{pmatrix} \in \mathbb{R}^{m\times n} \text{ mit } \widehat{R} \in \mathbb{R}^{n\times n}$$

gilt, wobei $\widehat{R}$ eine reguläre obere Dreiecksmatrix darstellt.

Beweis. Der Kern des (konstruktiven) Beweises liegt in der Verwendung von orthogonalen Drehmatrizen $\mathcal{U}(p,q;\varphi) \in \mathbb{R}^{m\times m}$ der Form

$$
\mathcal{U}(p,q;\varphi) :=
\begin{array}{c}
\begin{array}{ccccccccc} & & & \text{p-te Spalte}\downarrow & & \text{q-te Spalte}\downarrow & & & \end{array} \\
\begin{pmatrix}
1 & & & & & & & & 0 \\
 & 1 & & & & & & & \\
 & & \ddots & & & & & & \\
 & & & \cos\varphi & & \sin\varphi & & & \\
 & & & & \ddots & & & & \\
 & & & -\sin\varphi & & \cos\varphi & & & \\
 & & & & & & \ddots & & \\
 & & & & & & & 1 & \\
0 & & & & & & & & 1
\end{pmatrix}
\end{array}
\begin{array}{l}
\\ \\ \\ \\ \leftarrow \text{p-te Zeile} \\ \\ \leftarrow \text{q-te Zeile} \\ \\ \\ \\
\end{array}
\tag{4.4}
$$

Die Orthogonalität von $\mathcal{U}(p,q;\varphi)$ ergibt sich unmittelbar, denn es gilt $\mathcal{U}^T(p,q;\varphi)\cdot\mathcal{U}(p,q;\varphi) = I$. Das Produkt $\mathcal{U}^T(p,q;\varphi)\cdot A$ liefert dagegen für $j = 1,...,n$

$$
\begin{array}{c}
\begin{array}{ccccccc} & & \text{p-te Spalte}\downarrow & & \text{q-te Spalte}\downarrow & & \end{array} \\
\begin{pmatrix}
1 & & & & & & 0 \\
 & \ddots & & & & & \\
 & & \cos\varphi & & -\sin\varphi & & \\
 & & & \ddots & & & \\
 & & \sin\varphi & & \cos\varphi & & \\
 & & & & & \ddots & \\
0 & & & & & & 1
\end{pmatrix}
\end{array}
A =
\begin{pmatrix}
a_{11} & \cdots & a_{1n} \\
\vdots & & \vdots \\
a_{p-1,1} & \cdots & a_{p-1,n} \\
a_{p1}\cos\varphi - a_{q1}\sin\varphi & \cdots & a_{pn}\cos\varphi - a_{qn}\sin\varphi \\
a_{p+1,1} & \cdots & a_{p+1,n} \\
\vdots & & \vdots \\
a_{q-1,1} & \cdots & a_{q-1,n} \\
a_{p1}\sin\varphi + a_{q1}\cos\varphi & \cdots & a_{pn}\sin\varphi + a_{qn}\cos\varphi \\
a_{q+1,1} & \cdots & a_{q+1,n} \\
\vdots & & \vdots \\
a_{m1} & \cdots & a_{mn}
\end{pmatrix}.
$$

Demzufolge werden die Zeilen p und q von A durch Linearkombinationen der $\cos$- und $\sin$-Terme ersetzt. Wir wählen φ so, dass $\widetilde{a}_{p1} := a_{p1}\cos\varphi - a_{q1}\sin\varphi = 0$.

Die sukzessive Multiplikation der Matrix A von links mit Rotationsmatrizen $\mathcal{U}^T(p,q;\varphi)$ (φ dabei so gewählt, dass $\widetilde{a}_{qp} = 0$) mit den Rotationsindexpaaren

$$(1,2),(1,3),...,(1,m),(2,3),(2,4),...,(2,m),(3,4),...,(n,m)$$

eliminiert dann die Matrixelemente $a_{21}, a_{31}, ..., a_{mn}$. Sukzessive werden dabei unterhalb des Diagonalelements Nullen erzeugt. Nach $k := \frac{1}{2}n(2m-n-1)$ Transformationsschritten gilt dann $A = QR$ mit

$$\mathcal{U}_k^T \cdot \ldots \cdot \mathcal{U}_2^T \mathcal{U}_1^T A = Q^T A = R \quad \text{bzw.} \quad A = QR.$$

Da die orthogonale Matrix Q regulär ist, ist der Rang von A und der Rang von R gleich n und folglich ist die Rechtecksdreiecksmatrix $\widehat{R}$ regulär, da A nach Voraussetzung Maximalrang hat. ∎

Der obige Existenzbeweis ist rein konstruktiv und liefert damit eine erste Möglichkeit eine Zerlegung QR zu bestimmen, in dem wir Drehmatrizen verwenden. Rein geometrisch gibt es zwei Klassen von orthogonalen Transformationen, Drehungen und Spiegelungen. Wir beschreiben im folgenden die beiden Vorgehensweisen:

- *Givens*[1]-Rotationen;
- *Householder*[2]-Spiegelungen.

4.3 Givens-Rotationen

Wir beschreiben nun das algorithmische Vorgehen zur Realisierung der Drehungen im Beweis von Satz 4.2.2. Es bezeichne im Folgenden stets $c = \cos\varphi$ und $s = \sin\varphi$. Die Grundidee der *Givens*-Rotation besteht darin, durch die Multiplikation mit einer geeigneten orthogonalen (Dreh-)Matrix einen Vektor so zu drehen, dass möglichst viele seiner Komponenten verschwinden.

Wie in obigem Beweis auch, wendet man dies zunächst auf Vektoren mit zwei Komponenten an. Sei dazu $(a, b) \in \mathbb{R}^2$; wir suchen $c, s \in \mathbb{R}$, so dass

$$\begin{pmatrix} c & -s \\ s & c \end{pmatrix}\begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} r \\ 0 \end{pmatrix}, \quad \text{mit } c^2 + s^2 = 1 \text{ und damit } r^2 = a^2 + b^2,$$

gilt. Setzen wir zuerst $a,\ b \neq 0$ voraus, dann erhalten wir folgende Gleichung

$$as = -bc \tag{4.5}$$

und mit $s^2 + c^2 = 1$ gilt

$$r^2 = (ac - bs)^2 = a^2c^2 - abcs - abcs + b^2s^2 \overset{(4.5)}{=} a^2c^2 + a^2s^2 + b^2c^2 + b^2s^2 = a^2 + b^2\ .$$

Weiter ergibt sich für $a \neq 0$

$$\begin{aligned} ac - bs = r \Rightarrow c = \frac{1}{a}(r + bs) &\Rightarrow as + \frac{b}{a}(r + bs) = 0 \\ &\Leftrightarrow a^2s + br + b^2s = 0 \\ &\Leftrightarrow br = -(a^2 + b^2)s \\ &\Leftrightarrow s = \frac{-br}{a^2 + b^2} \overset{(r^2=a^2+b^2)}{=} \frac{-b}{r}. \end{aligned}$$

Und damit erhalten wir für $b \neq 0$

$$-bc = as \ \Rightarrow\ c = -\frac{a}{b}s = \frac{a}{r}.$$

Offen ist noch die Wahl des Vorzeichens des Vorzeichens $r = \pm\sqrt{a^2 + b^2}$. Setzen wir $r = \operatorname{sign}(a)\sqrt{a^2 + b^2}$, so folgt $c > 0$, falls $a \neq 0$, und für $a = 0$ setzen wir $c = 0$ und $s = 1$. Somit sind auch s und c für die anderen Spezialfälle $b = 0$ und $a = b = 0$ wohldefiniert. Damit gilt (obwohl der Winkel bisher gar nicht explizit genannt wurde) $\varphi \in (-\frac{\pi}{2}, \frac{\pi}{2}]$ und somit lässt sich $\cos\varphi$ eindeutig aus $\sin\varphi$ mit $\cos\varphi = \sqrt{1 - \sin^2\varphi}$ bestimmen.

[1] James Wallace Givens, 1910-1993.
[2] Alston Scott Householder, 1904-1993.

Seien p und q zwei Zeilenindizes von A. Zur Eliminierung des Matrixelements a_{qj} gehen wir daher wie folgt vor:

$$\begin{array}{llll}
\text{Falls} & a_{pj} \neq 0: & \cos\varphi = \dfrac{|a_{pj}|}{\sqrt{a_{pj}^2 + a_{qj}^2}}, & \sin\varphi = \dfrac{-\operatorname{sign}(a_{pj})\, a_{qj}}{\sqrt{a_{pj}^2 + a_{qj}^2}}, \\[2ex]
\text{falls} & a_{pj} = 0: & \cos\varphi = 0, & \sin\varphi = 1.
\end{array}$$

Da sich $\cos\varphi$ aus $\sin\varphi$ bestimmen lässt, ermöglicht das Verfahren eine effiziente Speicherung, denn es genügt $\sin\varphi$ an den entsprechenden freiwerdenden Stellen in A abzulegen:

$$A = \begin{pmatrix} a_{11}^{(0)} & \cdots & a_{1n}^{(0)} \\ \vdots & & \vdots \\ a_{m1}^{(0)} & \cdots & a_{mn}^{(0)} \end{pmatrix} \rightsquigarrow \begin{pmatrix} a_{11}^{(1)} & \cdots & \cdots & a_{1n}^{(1)} \\ \sin\varphi_{21} & a_{22}^{(1)} & \dots & a_{2n}^{(1)} \\ a_{31}^{(0)} & a_{32}^{(0)} & \cdots & a_{3n}^{(0)} \\ a_{41}^{(0)} & a_{42}^{(0)} & \cdots & a_{4n}^{(0)} \\ \vdots & & & \vdots \\ a_{m1}^{(0)} & \cdots & \cdots & a_{mn}^{(0)} \end{pmatrix} \rightsquigarrow \begin{pmatrix} a_{11}^{(2)} & \cdots & \cdots & a_{1n}^{(2)} \\ \sin\varphi_{21} & a_{22}^{(1)} & \cdots & a_{2n}^{(1)} \\ \sin\varphi_{31} & a_{32}^{(2)} & \cdots & a_{3n}^{(2)} \\ a_{41}^{(0)} & a_{42}^{(0)} & \cdots & a_{4n}^{(0)} \\ \vdots & & & \vdots \\ a_{m1}^{(0)} & \cdots & \cdots & a_{mn}^{(0)} \end{pmatrix}$$

$$\rightsquigarrow \begin{pmatrix} a_{11}^{(3)} & \cdots & \cdots & a_{1n}^{(3)} \\ \sin\varphi_{21} & a_{22}^{(1)} & \cdots & a_{2n}^{(1)} \\ \sin\varphi_{31} & a_{32}^{(2)} & \cdots & a_{3n}^{(2)} \\ \sin\varphi_{41} & a_{42}^{(3)} & \cdots & a_{4n}^{(3)} \\ \vdots & & & \vdots \\ a_{m1}^{(0)} & \cdots & \cdots & a_{mn}^{(0)} \end{pmatrix} \rightsquigarrow \begin{pmatrix} a_{11}^{(m-1)} & \cdots & \cdots & a_{1n}^{(m-1)} \\ \sin\varphi_{21} & a_{22}^{(1)} & \cdots & a_{2n}^{(1)} \\ \sin\varphi_{31} & a_{32}^{(2)} & \cdots & a_{3n}^{(2)} \\ \sin\varphi_{41} & a_{42}^{(3)} & \cdots & a_{4n}^{(3)} \\ \vdots & & & \vdots \\ \sin\varphi_{m1} & a_{m2}^{(m-1)} & \cdots & a_{mn}^{(m-1)} \end{pmatrix} \rightsquigarrow \begin{pmatrix} \ddots & & \widehat{R} \\ & \ddots & \\ \text{„}Q\text{“} & & \ddots \end{pmatrix}.$$

Die *QR*-Zerlegung liefert also QR in kompakter Form, gespeichert in A. Der Aufwand dieser Vorgehensweise liegt bei ungefähr $\mathcal{O}(n^3)$ Operationen.

Bemerkung 4.3.1 Die Multiplikation eines gegebenen Vektors $z \in \mathbb{R}^m$ mit der Drehmatrix $\mathcal{U}(p, q; \varphi)$ ist äquivalent zur Drehung von z um einen Winkel φ entgegen dem Uhrzeigersinn in der Koordinatenebene, vgl. Abbildung 4.2.

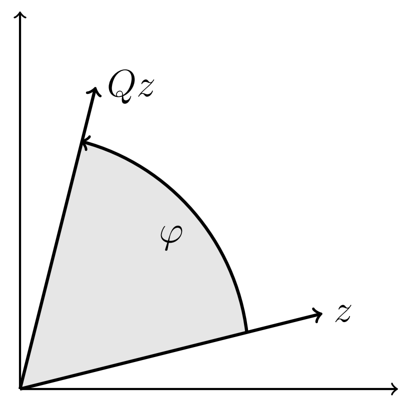


Abbildung 4.2: Drehung um einen Winkel φ in der Ebene.

Wir zeigen einige MATLAB®-Codes zur Realisierung.

MATLAB-Funktion: GivensRotMat.m

```
function rot = GivensRotMat(ap,aq)
if ap == 0
  s = 1; c = 0;
else
  dist2 = sqrt(ap^2+aq^2);
  c = abs(ap)/dist2;
  s = -sign(ap)*aq/dist2;
end
rot = [c,s;-s,c];
```

MATLAB-Funktion: Givens.m

```
function [Q,A] = Givens(A)
Q = eye(size(A,1));
for j=1:size(A,2)
  for i=j+1:size(A,1)
    rot = GivensRotMat(A(j,j),A(i,j));
    A([j,i],j:end) = rot' * A([j,i],j:end);
    Q(:,[j,i]) = Q(:,[j,i]) * rot;
  end
  A(j+1:end,j) = 0;
end
```

MATLAB-Beispiel:

Alternativ zu der MATLAB®-Funktion `qr` können wir auch die Routine `Givens` verwenden, die eine QR-Zerlegung, wie oben diskutiert, berechnet.

```
>> A = [1,2;3,4;5,6];
>> [Q,R] = Givens(A);
>> Q*R
ans =
     1.0000     2.0000
     3.0000     4.0000
     5.0000     6.0000
```

4.4 Householder-Spiegelungen

Nun zur zweiten Klasse von Äquivalenztransformationen, den Spiegelungen.

Definition 4.4.1 Sei $\omega \in \mathbb{R}^m$. Die Matrix

$$P = I - 2\frac{\omega\omega^T}{\omega^T\omega} \in \mathbb{R}^{m\times m} \tag{4.6}$$

wird als **Householder-Transformation** und ω als **Householder-Vektor** bezeichnet.

Der Householder-Vektor wird als Normalenvektor der (Hyper-)Ebene fungieren, die zur Spiegelung verwendet wird.

Satz 4.4.2 Eine *Householder*-Transformation $P = I - 2(\omega\omega^T)/(\omega^T\omega)$ ist symmetrisch und orthogonal, d.h. $P^{-1} = P^T$.

Beweis. Da $\omega\omega^T$ symmetrisch ist $\big((\omega\omega^T)^T = \omega\omega^T\big)$, folgt

$$P^T = \left(I - 2\frac{\omega\omega^T}{\omega^T\omega}\right)^T = I - 2\frac{\omega\omega^T}{\omega^T\omega} = P\,.$$

Des Weiteren ergibt sich

$$P\,P^T = \left(I - 2\frac{\omega\omega^T}{\omega^T\omega}\right)\left(I - 2\frac{\omega\omega^T}{\omega^T\omega}\right) = I - 2\frac{\omega\omega^T}{\omega^T\omega} - 2\frac{\omega\omega^T}{\omega^T\omega} + 4\frac{\omega\omega^T\omega\omega^T}{(\omega^T\omega)^2}$$

und mit $\omega\omega^T\omega\omega^T = (\omega^T\omega)\,(\omega\omega^T)$ somit die Behauptung. ∎

Betrachten wir nun folgende Frage: Sei $z \in \mathbb{R}^m$. Wie müsste ein $\omega \in \mathbb{R}^m$ aussehen, so dass $Pz = \alpha e_1$ gelte ($\alpha \in \mathbb{R}$, e_1 erster kanonischer Einheitsvektor)? Mit $Pz = z - 2\frac{\omega\omega^T}{\omega^T\omega}z = z - \lambda\omega \overset{!}{=} \alpha e_1$ folgt $\omega \in \operatorname{span}\{z - \alpha e_1\}$. Wir setzen nun $\omega = z - \alpha e_1$ in $Pz = \alpha e_1$ ein und erhalten

$$Pz = z - 2\frac{\omega^T z}{\omega^T\omega}\,\omega = \left(1 - \frac{2(z-\alpha e_1)^T z}{\|z-\alpha e_1\|^2}\right) z + \alpha\frac{2(z-\alpha e_1)^T z}{\|z-\alpha e_1\|^2} e_1 = \alpha e_1\,.$$

Damit in der letzten Gleichung der Faktor vor z verschwindet, muss

$$1 = \frac{2(z-\alpha e_1)^T z}{\|z-\alpha e_1\|^2} \;\Leftrightarrow\; (z-\alpha e_1)^T(z-\alpha e_1) = 2z^Tz - 2\alpha z_1 \;\Leftrightarrow\; \alpha = \pm\sqrt{z^Tz}$$

gelten. Wie ist nun das Vorzeichen zu wählen, $\alpha = \pm\sqrt{z^Tz}$? Die Wahl $\alpha = \|z\|_2$ hat die schöne Eigenschaft, dass Pz ein positives Vielfaches von e_1 ist. Aber das Rezept ist gefährlich, wenn z annähernd ein positives Vielfaches von e_1 ist, da Auslöschungen[3] auftreten können. Berechnet man ω_1 mittels $\omega_1 = z_1 - \|z\|_2$ treten für $z_1 \le 0$ keine Auslöschungen auf und mit der Umformung

$$\omega_1 = z_1 - \|z\|_2 = \frac{z_1^2 - \|z\|_2^2}{z_1 + \|z\|_2} = \frac{-(z_2^2 + \cdots + z_m^2)}{z_1 + \|z\|_2}$$

treten auch für $z_1 > 0$ keine Auslöschungen auf. Man beachte außerdem $(\omega_2, \ldots, \omega_m) = (z_2, \ldots, z_n)$. Normiert man den Vektor ω so, dass $\omega_1 = 1$ gilt, d.h.

$$\omega := \frac{z - \alpha e_1}{z_1 - \alpha} \quad \text{mit } \alpha^2 = \|z\|^2 \quad (\omega^T e_1 = 1),$$

dann folgt

$$\omega^T\omega = \frac{(z-\alpha e_1)^T(z-\alpha e_1)}{(z_1-\alpha)^2} = \frac{\|z\|^2 - 2\alpha z^T e_1 + \alpha^2}{(z_1-\alpha)^2} = \frac{2\alpha(\alpha - z_1)}{(z_1-\alpha)^2} = \frac{2\alpha}{\alpha - z_1}\,.$$

Bei der Speicherung der Matrix $P = I - 2(\omega\omega^T)/(\omega^T\omega)$, $P \in \mathbb{R}^{m\times m}$, auch des Vektors

[3] Vgl. Bemerkung 7.5.3.

$(\omega_2, \ldots, \omega_m)^T$ ist zu berücksichtigen, der sich auf den $m-1$ freigewordenen Einträgen speichern lässt.

Algorithmus 4.4.3 — Berechnung Householder-Vektor.

function $[\omega, \beta] = \texttt{housevector}(z)$

Input: $z \in \mathbb{R}^m$

$m = \texttt{length}(z)$

$\sigma = z(2:m)^T z(2:m)$

$\omega = \begin{bmatrix} 1 \\ z(2:m) \end{bmatrix}$

if $\sigma = 0$ **then**

  $\beta = 0$

else

  $\mu = \sqrt{z(1)^2 + \sigma}$

  if $z(1) \leq 0$ **then**

    $\omega(1) = z(1) - \mu$

  else

    $\omega(1) = -\sigma/(z(1) + \mu)$

  end if

  $\beta = 2\,\omega(1)^2/(\sigma + \omega(1)^2)$

  $\omega = \omega/\omega(1)$

end if

Output: $\omega \in \mathbb{R}^m$ mit $\omega(1) = 1$ und $\beta \in \mathbb{R}$,
so dass $P = I_m - \beta\omega\omega^T$ orthogonal ist und $Pz = \|z\|_2 e_1$

Bemerkung 4.4.4 Wir haben hier den Eingabevektor mit z bezeichnet und dessen Dimension mit m, da die Householder-Transformation auf die Spaltenvektoren der Matrix $A \in \mathbb{R}^{m\times n}$ angewendet wird und wir eine Verwechselung mit der Lösung $x \in \mathbb{R}^n$ des linearen Ausgleichsproblems vermeiden wollten. Dennoch sind viele der in der Literatur vorhandenen Programme auf Eingaben $x \in \mathbb{R}^n$ bezogen.

MATLAB-Funktion: HouseholderVektor.m

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

```
      x(2:end) = x(2:end)/tmp;
    end
    v = [1;x(2:end)];
else
    beta = 0;
    v = 1;
end
```

Mit den obigen Überlegungen haben wir konstruktiv gezeigt, dass sich mit Hilfe der *Householder*-Transformation eine Matrix $A \in \mathbb{R}^{m\times n}$ in eine Matrix der folgenden Form transformieren können:

$$H_1A = \begin{pmatrix} * & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \end{pmatrix}$$

Gehen wir nun davon aus, dass wir nach einigen Schritten die Ausgangsmatrix A auf folgende Gestalt gebracht haben:

$$H_2H_1A = \begin{pmatrix} * & * & * & * & * \\ 0 & * & * & * & * \\ 0 & 0 & \boxplus & * & * \\ 0 & 0 & \boxplus & * & * \\ 0 & 0 & \boxplus & * & * \\ 0 & 0 & \boxplus & * & * \end{pmatrix}$$

Im nächsten Schritt soll nun die nächste Subspalte unterhalb der Diagonalen eliminiert werden. Es ist also eine Householder-Matrix zu bestimmen, so dass

$$\widetilde{H}_3 \begin{pmatrix} \boxplus \\ \boxplus \\ \boxplus \\ \boxplus \end{pmatrix} = \begin{pmatrix} * \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

gilt. Definiert man H_3 als Blockdiagonalmatrix $H_3 := \begin{pmatrix} I & 0 \\ 0 & \widetilde{H}_3 \end{pmatrix}$, so erhalten wir

$$H_3H_2H_1A = \begin{pmatrix} * & * & * & * & * \\ 0 & * & * & * & * \\ 0 & 0 & * & * & * \\ 0 & 0 & 0 & * & * \\ 0 & 0 & 0 & * & * \\ 0 & 0 & 0 & * & * \end{pmatrix}$$

Mit dieser Vorgehensweise erhalten wir folgende Faktorisierung:

$$R = H_{n-1}H_{n-2}\cdots H_1 A \quad \Leftrightarrow \quad A = (H_1\cdots H_{n-1})R \quad \rightsquigarrow Q = H_1\cdots H_{n-1}$$

Wir stellen nun dar, wie A überschrieben wird. Es sei

$$\omega^{(j)} = (\underbrace{0,\ldots,0}_{\text{j-1}}, 1, \omega^{(j)}_{j+1}, \ldots, \omega^{(j)}_m)^T$$

der j-te Householder-Vektor, dann erhält man nach Durchführung der QR-Zerlegung

$$A = \begin{pmatrix} r_{11} & r_{12} & r_{13} & r_{14} & r_{15} \\ \omega_2^{(1)} & r_{22} & r_{23} & r_{24} & r_{25} \\ \omega_3^{(1)} & \omega_3^{(2)} & r_{33} & r_{34} & r_{35} \\ \omega_4^{(1)} & \omega_4^{(2)} & \omega_4^{(3)} & r_{44} & r_{45} \\ \omega_5^{(1)} & \omega_5^{(2)} & \omega_5^{(3)} & \omega_5^{(4)} & r_{55} \\ \omega_6^{(1)} & \omega_6^{(2)} & \omega_6^{(3)} & \omega_6^{(4)} & \omega_6^{(5)} \end{pmatrix}$$

Algorithmus 4.4.5 — Householder-QR.

function $A = \mathtt{housematrix}(A)$

Input: $A \in \mathbb{R}^{m\times n}$ mit $m \geq n$

for $j = 1, \ldots, n$ **do**
  $[\omega, \beta] = \mathtt{housevector}(A(j:m, j))$
  $A(j:m, j:n) = (I_{m-j+1} - \beta\omega\omega^T)A(j:m, j:n)$
  if $j < m$ **then**
    $A(j+1:m, j) = \omega(2:m-j+1)$
  end if
end for

Output: Householder-Matrizen $H_1, \ldots, H_n$, so dass $Q = H_1\cdots H_n$ und $R = Q^T A$,
Der obere Dreiecksteil von A wird mit R überschrieben und unterhalb der Diagonalen werden die nichttrivialen Komponenten der Householder-Vektoren gespeichert.

Bemerkung 4.4.6 Analog zur *Givens*-Rotation möchten wir auch an dieser Stelle kurz die geometrische Anschauung der *Householder*-Spiegelung anführen. Für einen gegebenen Vektor $x \in \mathbb{R}^n$ ist der Vektor $y = Px$ die Spiegelung von x an der Ebene $span\{\omega\}^\perp$. Hierzu Abbildung 4.3.

MATLAB-Funktion: Householder.m

```
function [A,R] = Householder(A)
for j = 1:size(A,2)
  [v,beta(j)] = HouseholderVector(A(j:end,j));
  A(j:end,j:end) = A(j:end,j:end)- v * (beta(j) * v'*A(j:end,j:
      end));
  if j<size(A,1)
```

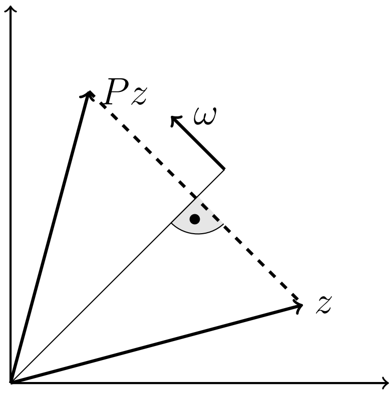


Abbildung 4.3: Spiegelung an einer Ebene mit Normalenvektor ω.

```
    A(j+1:end,j) = v(2:end);
  end
end
if nargout == 2
  R = A;
  A = eye(size(A,1));
  for j = size(R,2):-1:1
    v = [1;R(j+1:end,j)];
    A(j:end,j:end) = A(j:end,j:end) - v*(beta(j)*v'*A(j:end,j:
        end));
  end
  R = triu(R);
end
```

MATLAB-Funktion: HouseholderMult.m

```
function y = prod_rx(A,x)
% extract R from A and multiply with x
y = triu(A) * x;

function y = prod_qx(A,x)
y = x;
for j = size(A,2)-1:-1:1
    v = [1;A(j+1:end,j)];
    beta = 2/(v'*v);
    y(j:end) = y(j:end) - v*(beta*v'*y(j:end));
end

function y = prod_qtx(A,x)
y = x;
for j = 1:size(A,2)-1
    v = [1;A(j+1:end,j)];
    beta = 2/(v'*v);
    y(j:end) = y(j:end) - v*(beta*v'*y(j:end));
end
```

MATLAB-Beispiel:

Mit der Funktion `Householder` wird zum einen die Matrix A kompakt überschrieben, d.h., der Rückgabewert benötigt nur den Speicherplatz der ursprünglichen Matrix; zum anderen kann man sich auch Q und R mit $A = QR$ ausgeben lassen. Die Berechnung von Rx, Qx und $Q^T x$, wenn die kompakte Form vorliegt, ist in den Routinen `prod_qx`, etc. realisiert.

```
>> A = rand(3);
>> x = rand(3,1);
>> [Q,R] = Householder(A);
>> A-Q*R
ans =
  1.0e-015 *
   -0.1110         0   -0.1110
   -0.0035   -0.1110         0
   -0.1110         0    0.2220
>> C = Householder(A);
>> Q'*x - prod_qtx(C,x)
ans =
  1.0e-015 *
    0.2220
    0.0555
    0.5551
```

Bemerkung 4.4.7 — Vergleich Givens – Householder. Will man einen kompletten Vektor mittels einer Ähnlichkeitstransformation auf einen Einheitsvektor transformieren, dann ist die Householder-Spiegelung effizienter. Hat man jedoch nur einzelne Einträge, dann ist die Givens-Rotation besser geeignet. Ist also die Matrix A dünnbesetzt, dann sollte man Givens bevorzugen, bei einer vollbesetzten Matrix hingegen Householder.

4.5 Update einer *QR*-Zerlegung

Bei verschiedenen numerischen Verfahren müssen wiederholt QR-Zerlegungen bestimmt werden, wobei sich die Matrizen nur wenig voneinander unterscheiden. Beispiele sind u.a. Zeitreihen oder wenn eine Matrix so gross ist, dass sie nicht in den Speicher passt und man spaltenweise Daten hinzufügt. In solchen Fällen sind Aufdatierungs-Strategien geeignet, von denen wir eine vorstellen.

Sei die Zerlegung $A_k = Q_k R_k$ gegeben und wir wollen die QR-Zerlegung von

$$A_{k+1} := A_k + st^T \text{ mit } s \in \mathbb{R}^m, t \in \mathbb{R}^n$$

(einem sogenannten *Rang-1-Update*) bestimmen. Anstatt von $\mathcal{O}(n^3)$ Operationen für eine vollständige QR-Zerlegung kann man einen Update-Schritt in $\mathcal{O}(n^2)$ Operationen verwenden. Wir benötigen dazu einige Hilfsmittel, auf die wir auch in anderen Themen noch zurückgreifen werden.

Definition 4.5.1 — Hessenberg-Matrix. Eine Matrix $A \in \mathbb{R}^{n\times n}$ der Form

$$A = \begin{pmatrix} * & \cdots & \cdots & \cdots & * \\ * & & & & \vdots \\ 0 & \ddots & & & \vdots \\ \vdots & \ddots & \ddots & & \\ 0 & \cdots & 0 & * & * \end{pmatrix}, \qquad \text{d.h. } a_{ij} = 0 \text{ für } i - j \geq 2,$$

heißt **obere Hessenberg-Matrix**. Gilt $a_{ij} = 0$ für $j - i \geq 2$, so hat die Matrix A **untere Hessenberg-Gestalt**.

Lemma 4.5.2 Es seien $u, v \in \mathbb{R}^n$ und $R \in \mathbb{R}^{n\times n}$ eine obere Dreiecksmatrix. Dann benötigt man $n-2$ Givens-Rotationen, um $R + uv^T$ auf obere Hessenberg-Gestalt zu transformieren.

Beweis. Sei Q eine orthogonale Matrix, dann gilt $Q(R + uv^T) = QR + (Qu)v^T$. Wir wählen nun Drehmatrizen so, dass $Qu = (*, *, 0, \ldots, 0)^T$ gilt. Dies kann mit $n - 2$ Rotationen bewerkstelligt werden. Weiterhin gilt für diese so gewählt Matrix Q, dass $QR = R$ gilt, weil die Drehungen nur auf die Einträge unterhalb der Diagonale wirken, die allesamt Null sind. Damit hat $Qu)v^T$ obere Hessenberg-Form und damit auch $Q(R + uv^T)$, womit die Aussage bewiesen ist. ∎

Bemerkung 4.5.3 (i) Beachtet man, dass sich jede Givens-Rotation angewandt auf $A \in \mathbb{R}^{n\times n}$ durch $\mathcal{O}(n)$ Operationen realisieren lässt, so genügen $\mathcal{O}(n^2)$ Operationen, um eine orthogonale Matrix $\widetilde{Q}$ und eine Hessenberg-Matrix $\widetilde{H}$ mit $\widetilde{Q}\widetilde{H} = R + uv^T$ zu bestimmen.

(ii) Es genügen $n - 1$ Givens-Rotation um eine Hessenberg-Matrix $A \in \mathbb{R}^{n\times n}$ auf Dreiecksgestalt zu transformieren.

Effiziente Update-Berechnung: Der obige Ansatz

$$A_{k+1} = Q_{k+1}R_{k+1} = Q_k(R_k + uv^T) = A_k + Q_k uv^T = A_k + st^T$$

liefert $v = t$ und $s = Q_k u$ bzw. $u = Q_k^T s$. Das Update besteht nun aus zwei Schritten.

1. Mittels $n-2$ Givens-Rotationen wird $R_k + uv^T$ auf eine obere Hessenberg-Form gebracht, d.h. wir bestimmen eine orthogonale Matrix Q'_{k+1} und eine Hessenberg-Matrix H_{k+1} mit $Q'_{k+1}H_{k+1} = R_k + uv^T = R_k + (Q_k^T s)t^T$.
2. Mittels $n-1$ Givens-Rotationen wird diese obere Hessenberg-Form zur neuen Matrix R_{k+1} transformiert, d.h., wir bestimmen eine orthogonale Matrix Q''_{k+1} und eine obere Dreiecksmatrix R_{k+1} mit $Q''_{k+1}R_{k+1} = H_{k+1}$. Die Matrix Q_{k+1} ergibt sich dann als Produkt $Q_k \cdot Q'_{k+1} \cdot Q''_{k+1}$.

MATLAB-Funktion: QR2Hessenberg.m

```
function [Q,R] = QR2Hessenberg(Q,R,s,t)
% computes QH-decomposition of Q(R+s*t), H upper Hessenberg
    form
for j=size(R,1):-1:3
  rot = GivensRotMat(s(j),s(j-1));
  s([j,j-1]) = rot * s([j,j-1]);
  Q(:,[j,j-1]) = Q(:,[j,j-1]) * rot';
  R([j,j-1],:) = rot * R([j,j-1],:);
end
R(1:2,:) = R(1:2,:)+ s(1:2) * t';
```

MATLAB-Funktion: GivensHessenberg.m

```
function [Q,H] = GivensHessenberg(Q,H)
% computes QR-decomposition of QH, H Hessenberg matrix
for j=1:min(size(H,1)-1,size(H,2))
  rot = GivensRotMat(H(j+1,1),H(j,1));
  H([j+1,j],j:end) = rot * H([j+1,j],j:end);
  Q(:,[j+1,j]) = Q(:,[j+1,j]) * rot';
  H(j+1,j) = 0;
end
```

MATLAB-Beispiel:

Ein einfaches Beispiel zeigt, dass man das $\mathcal{O}(n^2)$-Verhalten des Updates auch im Vergleich zur optimierten MATLAB®-Routine `qr` sieht und nutzen kann. Verdoppelt man z.B. n auf $n = 4000$ im nebenstehenden Listing, so benötigt das Update nur $2^2 = 4$-mal länger, die interne Funktion jedoch $2^3 = 8$-mal. Testen Sie es einmal!

```
>> n = 2000;
>> A=rand(n); s=rand(n,1); t=rand(n,1);
>> An = A + s * t';
>> [Q,R] = qr(A);
>> tic
>> [Qtmp,H] = QR2Hessenberg(Q,R,Q'*s,t);
>> [Qn,Rn] = GivensHessenberg(Qtmp,H);
>> toc
Elapsed time is 1.078000 seconds.
>> tic, [Qn,Rn] = qr(An); toc
Elapsed time is 5.766000 seconds.
```

4.6 Singulärwertzerlegung

Eine weitere Möglichkeit zur Lösung linearer Ausgleichsprobleme ist die Singulärwertzerlegung (engl. *singular value decomposition (SVD)*), die jedoch auch viele weitere Anwendungen hat, z.B.

- Berechnung von Eigenwerten und -vektoren, vgl. Kapitel 5;
- Hauptkomponentenanalyse;
- Daten- und Bildkompression;
- Data Science.

Zu einigen dieser Themen und auch zum numerischen Algorithmus verweisen wir auf die Vorlesung *Numerical Methods in Data Science*. Die SVD kann auch als Analogon zur Schur-Zerlegung für quadratische Matrizen (Sätze A.3.8 und A.3.9) für nichtquadratische Matrizen $A \in \mathbb{R}^{m\times n}$ mit beliebigen $m, n \in \mathbb{N}$ angesehen werden.

Satz 4.6.1 — Singulärwertzerlegung. Zu jeder Matrix $A \in \mathbb{R}^{m\times n}$ existieren orthogonale Matrizen $U = [u_1, \dots, u_m] \in \mathbb{R}^{m\times m}$ und $V = [v_1, \dots, v_n] \in \mathbb{R}^{n\times n}$, sodass

$$U^T AV = \Sigma = \operatorname{diag}(\sigma_1, \dots, \sigma_p) \in \mathbb{R}^{m\times n} \qquad \text{mit } p = \min\{m, n\}$$

und mit $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_r > \sigma_{r+1} = \dots = \sigma_p = 0$, $r = \operatorname{rang}(A)$ (diese Zahlen heißen **Singulärwerte** von A).

Beweis. Für $A = 0$ gilt die Behauptung trivialerweise. Sei also $A \neq 0$ und setze

$$\sigma_1 := \|A\|_2 = \max_{0\neq x\in\mathbb{R}^n} \frac{\|Ax\|_2}{\|x\|_2} = \max_{\|y\|_2=1} \|Ay\|_2. > 0.$$

Da dieses Maximum[4] angenommen wird, existiert ein $v_1 \in \mathbb{R}^n$ mit $\|v_1\|_2 = 1$ und $\|Av_1\|_2 = \sigma_1$. Setze $u_1 := \frac{1}{\sigma_1} Av_1 \in \mathbb{R}^m$, dann gilt $\|u_1\|_2 = 1$. Nun ergänzen wir (mit dem Basis-Ergänzungssatz aus der Linearen Algebra und ggf. der Gram-Schmidt-Orthogonlisierung, Satz A.4.1) u_1 und v_1 zu Orthonormalbasen

$$\{v_1, \tilde{v}_2, \dots, \tilde{v}_n\} \text{ des } \mathbb{R}^n \qquad \text{und} \qquad \{u_1, \tilde{u}_2, \dots, \tilde{u}_m\} \text{ des } \mathbb{R}^m.$$

[4] Hier stimmen also Supremum und Maximum überein.

Damit sind die Matrizen $U_1 = [u_1, \tilde{u}_2, ..., \tilde{u}_m] \in \mathbb{R}^{m\times m}$, $V_1 = [v_1, \tilde{v}_2, ..., \tilde{v}_n] \in \mathbb{R}^{n\times n}$ orthogonal und es gilt mit $w := (u_1^T A\tilde{v}_j)_{2=1,...,n} \in \mathbb{R}^{n-1}$ und $\widetilde{A}_1 := (\tilde{u}_i^T A\tilde{v}_j)_{i=2,...,m;j=2,...n} \in \mathbb{R}^{(m-1)\times(n-1)}$.

$$\begin{aligned}
A_1 := U_1^T A V_1 &= \begin{pmatrix} u_1^T A v_1 & (u_1^T A\tilde{v}_j)_{2=1,...,n} \\ (\tilde{u}_i^T A v_1)_{i=2,...,m} & (\tilde{u}_i^T A\tilde{v}_j)_{i=2,...,m;j=2,...n} \end{pmatrix} \\
&= \begin{pmatrix} \sigma_1\, u_1^T u_1 & w^T \\ \sigma_1\, (\tilde{u}_i^T u_1)_{i=2,...,m} & \widetilde{A}_1 \end{pmatrix} \\
&= \begin{pmatrix} \sigma_1 & w^T \\ 0 & \widetilde{A}_1 \end{pmatrix}.
\end{aligned}$$

Wir wollen nun zeigen, dass $w = 0$. Dazu sei $z := (\sigma_1, w^T)^T$. Dann gilt $\|z\|_2^2 = \sigma_1^2 + w^T w$ und

$$\|A_1 z\|_2^2 = \left\| \begin{pmatrix} z^T z \\ * \end{pmatrix} \right\|_2^2 \geq \left(z^T z\right)^2 = \|z\|_2^4.$$

Mit der Definition der Matrixnorm gilt

$$\|A_1\|_2^2 = \max_{x\in\mathbb{R}^n} \frac{\|A_1 x\|_2^2}{\|x\|_2^2} \geq \frac{\|A_1 z\|_2^2}{\|z\|_2^2} \geq \frac{\|z\|_2^4}{\|z\|_2^2} = \|z\|_2^2 = \sigma_1^2 + w^T w.$$

Aufgrund der Submultiplikativität der Matrixnorm und der Orthogonalität von U_1, V_1 folgt

$$\sigma_1^2 + \|w\|_2^2 = \sigma_1^2 + w^T w \leq \|A_1\|_2^2 = \|U_1^T A V_1\|_2^2 \leq \|U_1\|_2^2\, \|A\|_2^2\, \|V_1\|_2^2 = \|A\|_2^2 = \sigma_1^2.$$

Damit folgt $w = 0$ und wir erhalten

$$A_1 = \begin{pmatrix} \sigma_1 & 0 \\ 0 & \widetilde{A}_1 \end{pmatrix}.$$

Nun können wir induktiv mit $\widetilde{A}_1$ fortfahren: falls in einem Schritt $\sigma_i = \|\widetilde{A}_{i-1}\|_2 = 0$ gilt, bricht das Verfahren ab. Dann ist $\widetilde{A}_{i-1} = 0$ und die Singulärwertzerlegung ist konstruiert.

Um die Sortierung zu zeigen, beachte, dass

$$\begin{aligned}
\sigma_1 = \|A\|_2 = \|U_1 A_1 V_1^T\|_2 = \|A_1\|_2 &= \sup_{x\in\mathbb{R}^n, \|x\|_2=1} \|A_1 x\|_2 \\
&\geq \sup_{x=(x_1,...,x_n)^T\in\mathbb{R}^n, x_1=0, \|x\|_2=1} \|A_1 x\|_2 = \sup_{\tilde{x}\in\mathbb{R}^{n-1}, \|\tilde{x}\|_2=1} \left\| \begin{pmatrix} \sigma_1 & 0 \\ 0 & \widetilde{A}_1 \end{pmatrix} \begin{pmatrix} 0 \\ \tilde{x} \end{pmatrix} \right\|_2 \\
&= \sup_{\tilde{x}\in\mathbb{R}^{n-1}, \|\tilde{x}\|_2=1} \|\widetilde{A}_1 \tilde{x}\|_2 = \|\widetilde{A}_1\|_2 = \sigma_2,
\end{aligned}$$

also $\sigma_1 \geq \sigma_2$. Die weiteren Ungleichungen folgen induktiv. Es bleibt nur noch, $\sigma_r > 0$ und $\sigma_{r+1}, \ldots, \sigma_n = 0$ zu zeigen. Dazu, setze

$$\tilde{r} := \#\{j = 1, ..., n : \sigma_j \neq 0\} = \operatorname{rang}(\Sigma).$$

Da die Singulärwerte sortiert sind, erhalten wir $\sigma_{\tilde{r}} > 0$ und $\sigma_{\tilde{r}+1}, ..., \sigma_n = 0$. Da U und V orthogonal sind, folgt $\tilde{r} = \operatorname{rang}(\Sigma) = \operatorname{rang}(U\Sigma V^T) = \operatorname{rang}(A) = r$. ■

Korollar 4.6.2 Mit Satz 4.6.1 gilt:

i) Die Sinulärwerte sind die Wurzeln der Eigenwerte von A^TA und bis auf Permutation eindeutig bestimmt;
ii) $\mathrm{Kern}(A) = \mathrm{span}\{v_{r+1}, \dots, v_n\}$ und $\mathrm{Bild}(A) = \mathrm{span}\{u_1, \dots, u_r\}$;
iii) $\|A\|_F^2 = \sigma_1^2 + \dots + \sigma_p^2$ und $\|A\|_2 = \sigma_1$;
iv) $u_i^TA = \sigma_i v_i^T, \quad Av_i = \sigma_i u_i, \quad i = 1, \dots, p.$
Die Vektoren u_i und v_i werden linke bzw. rechte **Singulärvektoren** genannt.

Beweis. Übung. ∎

Die SVD erlaubt auch die Definition einer verallgemeinerten Inversen (für nicht-invertierbare Matrizen) und auch den Begriff der Wurzel (und auch allgemeiner Potenzen) einer Matrix. Diese Begriffe führen wir nun ein.

Die Pseudoinverse

Definition 4.6.3 Sei $A = U\Sigma V^T \in \mathbb{R}^{m\times n}$ gemäß Satz 4.6.1 und $\Sigma^+ := \mathrm{diag}(\frac{1}{\sigma_1}, \dots, \frac{1}{\sigma_r}, 0, \dots, 0) \in \mathbb{R}^{n\times m}$. Dann heißt

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

die **Pseudoinverse**[5] von A.

Lemma 4.6.4 Die Pseudoinverse hat folgende Eigenschaften:

(i) $AA^+A = A$ (A^+ ist eine verallgemeinerte Inverse);
(ii) $A^+AA^+ = A^+$ (A^+ ist eine schwache Inverse);
(iii) $(AA^+)^\top = AA^+$ (Symmetrie von AA^+);
(iv) $(A^+A)^\top = A^+A$ (Symmetrie von A^+A).

Beweis. Übung. ∎

Die Eigenschaften (i)-(iv) werden auch als *Moore-Penrose-Axiome* bezeichnet und sind eine äquivalente Definition von A^+ (Übung).

Mit der Singulärwertzerlegung kann man das lineare Ausgleichsproblem wie folgt lösen.

Proposition 4.6.5 Für alle $x \in \mathbb{R}^n$ gilt $\|Ax^* - b\|_2 \le \|Ax - b\|_2$ mit $x^* := A^+b$.

Beweis. Es gilt $\|Ax - b\|_2 = \|U^TAx - U^Tb\|_2 = \|U^TAV(V^Tx) - U^Tb\|_2 = \|\Sigma z - d\|_2$ mit $z := V^Tx \in \mathbb{R}^n$ und $d := U^Tb \in \mathbb{R}^m$. Damit folgt

$$\|Ax - b\|_2^2 = \sum_{j=1}^m |d_j - \sigma_j z_j|^2 = \sum_{j=1}^r |d_j - \sigma_j z_j|^2 + \sum_{j=r+1}^m |d_j|^2$$

und dieser Ausdruck wird minimal für $z_j^* := \frac{1}{\sigma_j} d_j$, $j = 1, \dots, r$ und (z.B.) $z_j^* := 0$, $j = r+1, \dots, m$. Für dieses z^* gilt einerseits $z^* = V^Tx^*$ und andererseits $z^* = \Sigma^+d = \Sigma^+U^Tb$, also $x^* = V\Sigma^+U^Tb = A^+b$. ∎

[5] Auch *Moore-Penrose-Inverse* genannt, nach E.H. Moore, 1920-; Roger Penrose, 1955-.

Korollar 4.6.6 Die Lösung des linearen Ausgleichsproblem ist gegeben durch $x^* = A^+ b$.

Man beachte, dass Proposition 4.6.5 auch besagt, dass $x^* := A^+ b$ die Lösung des linearen Ausgleichsproblems mit der kleinsten Euklidischen Norm ist. In der Tat muss die Lösung des linearen Ausgleichsproblems nicht eindeutig sein.

Die Wurzel einer Matrix

Mithilfe der Singulärwertzerlegung definiert man auch Wurzeln von symmetrisch positiv definiten Matrizen.

Definition 4.6.7 Sei $A \in \mathbb{R}^{n \times n}$ s.p.d. mit SVD $A = U \Sigma U^T$. Dann definiert man $A^{\frac{1}{2}} := U \Sigma^{\frac{1}{2}} U^T$ mit $\Sigma^{\frac{1}{2}} := \operatorname{diag}(\sqrt{\sigma_1}, ..., \sqrt{\sigma_n})$.

Man sieht leicht, dass $A^{\frac{1}{2}}$ wohldefiniert ist. Das A s.p.d. ist, gilt $V = U$ und $\sigma_i > 0$ for alle $i = 1, ..., n$. Weiter gilt

$$A^{\frac{1}{2}} A^{\frac{1}{2}} = U \Sigma^{\frac{1}{2}} U^T U \Sigma^{\frac{1}{2}} U^T = U \Sigma U^T = A,$$

wie man es von einer Wurzel erwartet. Zudem ist $A^{\frac{1}{2}}$ s.p.d., wenn man ausschließlich positive Wurzeln nimmt. Allgemeiner kann man so $A^s := U \Sigma^s U^T$ für $s \in \mathbb{R}$ definieren.

Der SVD-Algorithmus

Die Beschreibung des Algorithmus zur effizienten Berechnung der Singulärwertzerlegung würde den Rahmen dieser Vorlesung sprengen. Wir behandeln ihn ausführlich in der Vorlesung *Numerical Methods for Data Science*. Die SVD wurde erstmals 1965 von Golub und Kahan publiziert [GK96], der Algorithmus in seiner heutigen Form stammt von Golub und Reinsch aus dem Jahr 1970, [GR70]. Er besteht aus zwei Schritten:

1. Reduziere A auf eine Matrix B in Bidiagonalform,

$$U_B^T A V_B = \begin{pmatrix} B \\ \hline 0 \end{pmatrix} = \left(\begin{array}{c} \ddots \;\ddots \\ \hline 0 \end{array}\right) \in \mathbb{R}^{m \times n}.$$

2. Führe „impliziten" QR-Schritt auf $B^T B$ aus (d.h., ohne Aufstellung von $B^T B$).

Für Details verweisen wir auf die Vorlesung *Numerical Methods for Data Science*, [GR70] oder [GV13].

Kapitel 5

Numerische Berechnung von Eigenwerten und -vektoren

Viele reale Fragestellungen führen auf Eigenwertprobleme, man denke etwa an Eigenfrequenzen von Bauteilen wie Gebäude oder Brücken. Zu einer gegebenen Matrix $A \in \mathbb{R}^{n\times n}$ sucht man $\lambda \in \mathbb{C}$ und einen Vektor $0 \neq v \in \mathbb{C}^n$, $v \neq 0$, sodass die *Eigenwertgleichung*

$$Av = \lambda v$$

erfüllt ist. Die Zahl λ heißt *Eigenwert* und der Vektor v *Eigenvektor* zum Eigenwert λ. Wir lassen hier auch komplexe Eigenwerte und -vektoren zu, da nicht jede reelle Matrix ausschließlich reelle Eigenwerte besitzen muss. Wir haben einige Grundlagen aus der Linearen Algebra zu Eigenwertproblemen in Anhang A.3 zusammen gestellt.

5.1 Einführung

5.1.1 Ein Anwendungsbeispiel

Betrachten wir nun zunächst ein konkretes Beispiel, das auf ein Eigenwertproblem führt.

■ **Beispiel 5.1.1 — Sturm-Liouville-Problem.** Wir erinnern an Beispiel 3.0.1 (S. 16). Dort hatten wir die Auslenkung einer Saite unter Einwirkung einer äußeren Kraft gesucht. Nun sind wir an Eigenschwingungen und -frequenzen interessiert.[1] Dies führt auf das sogenannte *Sturm-Liouville Problem*: Zu einer bekannten stetigen Funktion $r(x) > 0$, $x \in [0,1]$, bestimme $\lambda \in \mathbb{R}$ und $u : [0,1] \to \mathbb{R}$, welche die Differenzial-Eigenwertgleichung

$$-u''(x) - \lambda\, r(x)\, u(x) = 0, \quad x \in (0,1), \tag{5.1}$$

mit den Randbedingungen

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

[1] Die mathematische Modellierung zur Beschreibung der Überlagerung von Schwingungsvorgängen wird etwa beim Brückenbau durchgeführt wird, um Resonanzen zu vermeiden, vgl. das berühmte Beispiel der Tacoma Bay Bridge, https://www.youtube.com/watch?v=j-zczJXSxnw.

erfüllen. Für den einfachen Fall $r(x) \equiv 1$ sind die Lösungen von (5.1) bekannt:

$$\lambda_k = (k\pi)^2, \qquad u_k(x) = \sin(k\pi x), \qquad k = 0, 1, 2, \ldots$$

Wir haben also unendlich viele Lösungen. Man nennt u_k auch *Eigenfunktionen*.

Wie in Beispiel 3.0.1 diskretisieren wir diese Gleichung mit einem äquidistanten Gitter mit Schrittweite $h = \frac{1}{n_h+1}$ und verwenden den zentralen Differenzenquotienten zweiter Ordnung zur Approximation der zweiten Ableitung. Wir erhalten das (algebraische) System

$$A_h u_h - \lambda R_h u_h = 0 \tag{5.2}$$

für die Unbekannten $\lambda \in \mathbb{R}$ und $u_h = (u_i)_{i=1,\ldots,n_h}$, $u_i \approx u(x_i)$, $i = 1, \ldots, n_h$, wobei

$$A_h = \frac{1}{h^2}\begin{pmatrix} 2 & -1 & & & \\ -1 & 2 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 2 & -1 \\ & & & -1 & 2 \end{pmatrix}, \quad R_h = \begin{pmatrix} r(x_1) & & & 0 \\ & r(x_2) & & \\ & & \ddots & \\ 0 & & & r(x_{n-1}) \end{pmatrix}. \tag{5.3}$$

Ein Problem der Form (5.2) heißt auch *verallgemeinertes* Eigenwertproblem (weil auf der rechten Seite noch eine Matrix R_h steht), welches sich aber in ein herkömmliches Eigenwertproblem überführen lässt: Mit $R_h^{1/2} = \operatorname{diag}(\sqrt{r(x_1)}, \ldots, \sqrt{r(x_{n-1})})$, $R_h^{-1/2} := (R_h^{1/2})^{-1}$, $B_h := R_h^{-1/2} A_h R_h^{-1/2}$ sowie $v_h = R_h^{1/2} u$ erhält man aus (5.2) die transformierte Gleichung

$$B_h v_h = \lambda v_h,$$

also ein (symmetrisches) Eigenwertproblem, da B_h symmetrisch ist. Da R_h regulär ist, hätte man die Gleichung (5.2) auch von links mit R_h^{-1} multiplizieren und so das Eigenwertproblem $R_h^{-1} A_h u_h = \lambda u_h$ erhalten können. Im Gegensatz zur Matrix B_h ist die Matrix $R_h^{-1} A_h$ jedoch nicht symmetrisch. ▪

5.1.2 Abschätzungen und geometrische Lage der Eigenwerte

Im Folgenden werden einige Abschätzungen für das Spektrum einer Matrix $A \in \mathbb{C}^{n\times n}$ vorgestellt. Solche Abschätzungen können später bei der Wahl geeigneter Startwerte für lokal konvergente[2] Iterationsverfahren zur Bestimmung der Eigenwerte nützlich sein. Eine erste Eingrenzung der Lage der Eigenwerte von A liefert der folgende Satz.

Satz 5.1.2 — Abschätzung mittels Matrixnorm. Sei $\|\cdot\|$ eine konsistente Matrixnorm auf $\mathbb{C}^{n\times n}$, dann gilt $|\lambda| \leq \|A\|$ für alle $\lambda \in \sigma(A)$.

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

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

sodass $|\lambda| \leq \|A\|$ folgt. ■

[2] Das bedeutet, dass ein Startwert „nahe genug“ an der gesuchten Lösung sein muss.

Diese Abschätzung ist in vielen Fällen sehr grob. Außerdem kann die Berechnung der Matrixnorm sehr aufwendig sein. Daher ist man an anderen Abschätzungen interessiert.

Definition 5.1.3 — Wertebereich einer Matrix. Sei $A \in \mathbb{K}^{n\times n}$ ($\mathbb{K} = \mathbb{R}$ oder $\mathbb{K} = \mathbb{C}$). Die Menge aller *Rayleigh-Quotienten*

$$\varrho(A;x) := \frac{x^H Ax}{x^H x}$$

mit $x \in \mathbb{C}^n \setminus \{0\}$, also

$$W(A) := \{\varrho(A;x) : x \in \mathbb{C}^n \setminus \{0\}\}$$

wird als *Wertebereich der Matrix* A bezeichnet.

Bemerkung 5.1.4 i) Selbst für eine reelle Matrix $A \in \mathbb{R}^{n\times n}$ werden beim Wertebereich alle komplexen Vektoren $x \in \mathbb{C}^n \setminus \{0\}$ durchlaufen.

ii) Der Wertebereich enthält das Spektrum: $\sigma(A) \subseteq W(A)$. Ist $x \in \mathbb{C}^n \setminus \{0\}$ ein Eigenvektor von A, dann ist der zugehörige Rayleigh-Quotient $\varrho(A;x)$ gerade der Eigenwert zu x.

Lemma 5.1.5 — Eigenschaften des Wertebereichs. Sei $A \in \mathbb{K}^{n\times n}$, dann gilt:

i) W(A) ist zusammenhängend.

ii) Ist A hermitesch, dann gilt $W(A) = \sigma(A) = [\lambda_{\min}, \lambda_{\max}] \subset \mathbb{R}$.

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

Satz 5.1.6 — Bendixson. Sei $A \in \mathbb{K}^{n\times n}$ und $R := W\left(\frac{A+A^H}{2}\right) + W\left(\frac{A-A^H}{2}\right)$ ein „Rechteck", wobei $A + B := \{a + b : a \in A, b \in B\}$. Dann gilt

$$W(A) \subseteq R.$$

Beweis. Sei $x \in \mathbb{C}^n \setminus \{0\}$. Dann gilt

$$\begin{aligned} \frac{x^H Ax}{x^H x} &= \frac{1}{x^H x} x^H \left(\frac{A + A^H}{2} + \frac{A - A^H}{2}\right) x \\ &= \frac{1}{x^H x} x^H \left(\frac{A + A^H}{2}\right) x + \frac{1}{x^H x} x^H \left(\frac{A - A^H}{2}\right) x, \end{aligned}$$

und dies ist in $W\left(\frac{A+A^H}{2}\right) + W\left(\frac{A-A^H}{2}\right)$ enthalten. ∎

Bemerkung 5.1.7 Da $\frac{1}{2}(A + A^H)$ hermitesch und $\frac{1}{2}(A - A^H)$ schiefhermitesch sind, besagt der Satz von Bendixson, dass für jedes $\lambda \in \sigma(A)$ für Real- und Imaginärteil gilt

$$\lambda_{\min}\left(\tfrac{1}{2}(A + A^H)\right) \le \Re(\lambda) \le \lambda_{\max}\left(\tfrac{1}{2}(A + A^H)\right),$$
$$\min\left\{\Im(\mu) \,|\, \mu \in \sigma\left(\tfrac{1}{2}(A - A^H)\right)\right\} \le \Im(\lambda) \le \max\left\{\Im(\mu) \,|\, \mu \in \sigma\left(\tfrac{1}{2}(A - A^H)\right)\right\}.$$

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

Satz 5.1.8 — Gerschgorin Kreise. Sei $A \in \mathbb{C}^{n\times n}$. Dann gilt

$$\sigma(A) \subseteq \mathcal{S}_{\mathcal{R}} := \bigcup_{i=1}^{n} \mathcal{R}_i, \qquad \mathcal{R}_i = \Big\{ z \in \mathbb{C} : |z - a_{ii}| \le \sum_{\substack{j=1\\ j\neq i}}^{n} |a_{ij}| \Big\}. \tag{5.4}$$

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

Beweis. Sei $\lambda \in \sigma(A)$ und $Ax = \lambda x$ für ein $x \neq 0$. Dann existiert ein x_i mit $|x_j| \le |x_i|$ für alle $i \neq j$ und es gilt

$$\lambda x_i = (Ax)_i = \sum_{j=1}^{n} a_{ij}\, x_j \quad \Leftrightarrow \quad \lambda - a_{ii} = \sum_{\substack{j=1\\ j\neq i}}^{n} a_{ij}\, \frac{x_j}{x_i}$$

und damit

$$|\lambda - a_{ii}| = \left| \sum_{\substack{j=1\\ j\neq i}}^{n} a_{ij}\, \frac{x_j}{x_i} \right| \le \sum_{\substack{j=1\\ j\neq i}}^{n} |a_{ij}| \frac{|x_j|}{|x_i|} \le \sum_{\substack{j=1\\ j\neq i}}^{n} |a_{ij}|, \qquad \text{also} \qquad \lambda \in \mathcal{R}_i \subset \bigcup_{i=1}^{n} \mathcal{R}_i,$$

was die Behauptung zeigt. ■

Da A und A^T das gleiche Spektrum haben, gilt Satz 5.1.8 auch in der Form

$$\sigma(A) \subseteq \mathcal{S}_{\mathcal{C}} := \bigcup_{j=1}^{n} \mathcal{C}_j, \qquad \mathcal{C}_j = \Big\{ z \in \mathbb{C} : |z - a_{jj}| \le \sum_{\substack{i=1\\ i\neq j}}^{n} |a_{ij}| \Big\}. \tag{5.5}$$

Die Aussagen (5.4) und (5.5) liefern zusammen das erste Gerschgorin Theorem.

Satz 5.1.9 — Erstes Gerschgorin Theorem. Sei $A \in \mathbb{C}^{n\times n}$, dann gilt für alle $\lambda \in \sigma(A)$

$$\lambda \in \mathcal{S}_{\mathcal{R}} \cap \mathcal{S}_{\mathcal{C}}.$$

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

Satz 5.1.10 — Zweites Gerschgorin Theorem. Seien

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

Falls $\mathcal{M}_1 \cap \mathcal{M}_2 = \emptyset$, so enthält $\mathcal{M}_1$ genau k und $\mathcal{M}_2$ die übrigen $n-k$ Eigenwerte, jeder entsprechend seiner algebraischen Vielfachheit gezählt.

Beweis. Sei $D := \operatorname{diag}(a_{11}, \dots, a_{nn})$ und für $t \in [0,1]$ definieren wir $A_t := D + t(A - D)$, also insbesondere $A_0 = D$ und $A_1 = A$. Die Eigenwerte $\lambda(t)$ von A_t bilden eine stetige Funktion in t.[3] Wendet man Satz 5.1.8 auf $A_0 = D$ an, so erhält man für $t = 0$ die Aussage des

[3] Dies ist tatsächlich das in der Originalarbeit von Gerschgorin verwendete Argument, welches zwar intuitiv klar zu sein scheint, aber schwer mathematisch sauber zu formulieren ist. In [HJ01, Thm. 6.1.1] findet man einen rigorosen Beweis, der jedoch auf Methoden der Funktionentheorie basiert, was hier zu weit führen

Satzes. Für $0 \le t \le 1$ müssen alle Eigenwerte von A_t ebenfalls in den Kreisen liegen. Daher und aufgrund der Stetigkeit von λ als Funktion in t folgt, dass auch k Eigenwerte von $A_1 = A$ in $\mathcal{M}_1$ und die übrigen $n-k$ in $\mathcal{M}_2$ liegen.[4] ∎

Um das dritte Gerschgorin Theorem formulieren zu können, benötigen wir eine Eigenschaft von Matrizen. Für den Beweis dieses Satzes und auch für weitere Gerschgorin-Typ-Abschätzungen verweisen wir auf [HJ01, §6.2].

Definition 5.1.11 Eine Matrix $A \in \mathbb{C}^{n\times n}$ heißt *reduzibel*, wenn eine Permutationsmatrix P und quadratische Matrizen B_{11}, B_{22} existieren, sodass

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

gilt. Die Matrix A heißt irreduzibel, wenn A nicht reduzibel ist. [a]

[a] Wir werden später diese Definition auch noch eine Permitationsmatrix P sehen, vgl. Definition 6.1.6 auf Seite 97.

Satz 5.1.12 — Drittes Gerschgorin Theorem. Sei $A \in \mathbb{C}^{n\times n}$ eine irreduzible Matrix. Ein Eigenwert $\lambda \in \sigma(A)$ kann nicht auf dem Rand von $\mathcal{S}_{\mathcal{R}}$ liegen, es sei denn, er liegt auf dem Rand eines jeden Kreises $\mathcal{R}_i$ für $i = 1, \dots, n$. ∎

▪ **Beispiel 5.1.13** Wir betrachten die reelle Matrix

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

und werden zunächst den Satz von Bendixson, Satz 5.1.6, anwenden, um die Eigenwerte der Matrix A abzuschätzen. Wir erhalten

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

Wendet man auf diese beiden Matrizen Satz 5.1.8 an, so ergeben sich folgende erste Abschätzungen eines Eigenwertes λ von A

$$-5 \le \Re(\lambda) \le 6, \qquad |\Im(\lambda)| \le 1\,.$$

Die Gerschgorin Kreise von A ergeben sich zu

$$\mathcal{R}_1 = \{z \in \mathbb{C} : |z-4| \le 3\}, \qquad \mathcal{R}_2 = \{z \in \mathbb{C} : |z+1| \le 1\}, \qquad \mathcal{R}_3 = \{z \in \mathbb{C} : |z+2| \le 2\},$$

sowie diejenigen von A^T zu

$$\mathcal{C}_1 = \{z \in \mathbb{C} : |z-4| \le 1\}, \qquad \mathcal{C}_2 = \{z \in \mathbb{C} : |z+1| \le 1\}, \qquad \mathcal{C}_3 = \{z \in \mathbb{C} : |z+2| \le 4\}\,.$$

würde.

[4] Diese Beweismethode ist auch als *Homotopie* bekannt.

Die Abschätzung nach Bendixson und die Gerschgorin Kreise sind in Abb. 5.1 dargestellt. Hierbei ist zu beachten, dass man die Menge $\mathcal{R}_1 \cap \mathcal{C}_3$ durch Anwenden des zweiten Gerschgorin Theorems auf A und A^T als in Frage kommenden Bereich für die Eigenwerte von A ausschließen kann. ∎

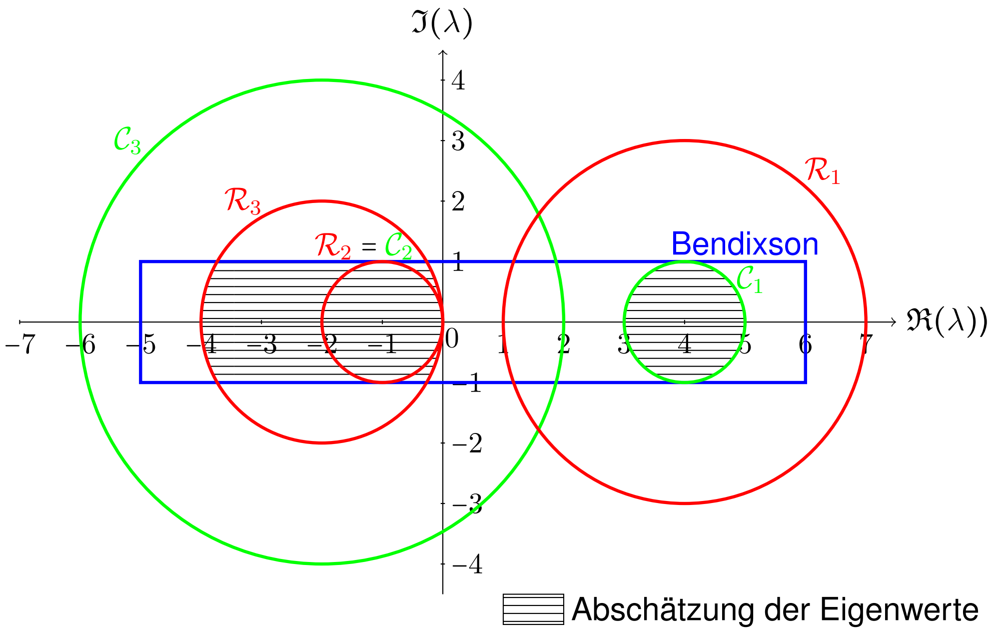


Abbildung 5.1: Abschätzung nach Bendixson und Gerschgorin Kreise.

5.2 Potenzmethode

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

$$|\lambda_1| > |\lambda_2| \geq |\lambda_3| \geq \cdots \geq |\lambda_n| \tag{5.6}$$

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

$$x^{(0)} = \sum_{i=1}^{n} \alpha_i v_i \,. \tag{5.7}$$

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

$$a^{(k)} = A^k x^{(0)} = A^k \left(\sum_{i=1}^{n} \alpha_i v_i \right) = \sum_{i=1}^{n} \alpha_i \lambda_i^k \, v_i = \lambda_1^k \sum_{i=1}^{n} \alpha_i \left(\frac{\lambda_i}{\lambda_1} \right)^k v_i \,. \tag{5.8}$$

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

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

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

Algorithmus 5.2.1 — Potenzmethode.

Input: $x^{(0)}$ mit $v_1^T x^{(0)} \neq 0$ und $\|x^{(0)}\| = 1$

1: **for** $k = 0, 1, \ldots$ **do**
2: $\quad a^{(k)} = A x^{(k)}$
3: $\quad \varrho^{(k)} = x^{(k)\,T} a^{(k)}$ {Rayleigh-Quotient}
4: $\quad x^{(k+1)} = \frac{a^{(k)}}{\|a^{(k)}\|}$
5: **end for**

Satz 5.2.2 Sei λ_1 ein einfacher Eigenwert der diagonalisierbaren Matrix $A \in \mathbb{C}^{n \times n}$ mit (5.6) und $x^{(0)} \in \mathbb{C}^n$ sei ein Vektor, der nicht senkrecht auf dem Eigenraum von λ_1 steht und $\|x^{(0)}\| = 1$ erfüllt. Dann konvergiert die Folge $x^{(k+1)} := a^{(k)}/\|a^{(k)}\|$ mit $a^{(k)} = Ax^{(k)}$ gegen einen normierten Eigenvektor von A zum Eigenwert λ_1.

Beweis. Wir wissen aus (5.8), dass

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

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

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

was die Behauptung zeigt. ■

Wir schließen einige Bemerkungen zur **Konvergenzgeschwindigkeit** an. Dazu zunächst folgende Definition. Eine ausführliche Behandlung der Konvergenzgeschwindigkeit iterativer Verfahren wird in der Vorlesung *Numerische Analysis* vorgestellt.

Definition 5.2.3 Eine Folge $(x^{(k)})_{k \in \mathbb{N}} \subset X$ in einem normierten Raum X heißt *linear konvergent* gegen $x^* \in X$, wenn es ein $K \in \mathbb{N}$ und ein $\varrho < 1$ (den *Konvergenzfaktor*) gibt mit

$$\|x^{(k+1)} - x^*\|_X \leq \varrho \, \|x^{(k)} - x^*\|_X$$

für alle $k \geq K$. Gilt diese Ungleichung mit $\varrho_k \to 0$ mit $k \to \infty$ anstelle von ϱ, dann spricht man von *superlinearer Konvergenz*.

Bemerkung 5.2.4 i) Der Rayleigh-Quotient $\varrho^{(k)} = \varrho(A; x^{(k)})$ (beachte, dass $x^{(k)}$ normiert ist) in Algorithmus 5.2.1 liefert im Grenzwert den betragsgrößten Eigenwert, d.h. $\lim_{k\to\infty} \varrho^{(k)} = \lambda_1$.

ii) Der Beweis des Satzes 5.2.2 läßt sich auch auf diagonalisierbare Matrizen mit eindeutig bestimmten betragsgrößtem Eigenwert λ_1, welcher aber nicht einfach zu sein braucht, d.h.

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

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

iii) Man sieht in (5.8), dass für die Potenzmethode lineare Konvergenz mit Konvergenzfaktor $|\lambda_2/\lambda_1|$ bzw. $|\lambda_{r+1}/\lambda_1|$ vorliegt. Das Verfahren konvergiert also umso besser, je mehr die Beträge der Eigenwerte von A getrennt sind.
Falls A *symmetrisch* ist, sind die Eigenvektoren v_i orthogonal. Mit Hilfe dieser Orthogonalitätseigenschaft kann man zeigen, dass in diesem Fall der Konvergenzfaktor sogar $|\lambda_2/\lambda_1|^2$ bzw. $|\lambda_{r+1}/\lambda_1|^2$ beträgt.

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

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

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

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

Sei nun $A \in \mathbb{C}^{n\times n}$ eine *nicht diagonalisierbare* Matrix mit eindeutig bestimmtem betragsgrößtem Eigenwert λ_1, d.h. aus $|\lambda_1| = |\lambda_i|$ folgt $\lambda_1 = \lambda_i$. Ersetzt man die Darstellung (5.7) des Startvektors $x^{(0)}$ durch eine Linearkombination von Eigen- und Hauptvektoren von A, so kann man auf dieselbe Weise wie im diagonalisierbaren Fall zeigen, dass unter analogen Voraussetzungen an $x^{(0)}$ in der Potenzmethode 5.2.1 der Rayleigh-Quotient $\varrho^{(k)}$ gegen λ_1 und $x^{(k)}$ gegen einen zu λ_1 gehörigen Eigenvektor konvergieren.

■ **Beispiel 5.2.5** Wir betrachten das Eigenwertproblem aus Beispiel 5.1.1 mit $R_h = I_h$, also $A_h x_h = \lambda x_h$ mit $A_h \in \mathbb{R}^{n_h\times n_h}$ wie in (5.3). Die Eigenwerte der Matrix A_h lassen sich explizit angeben:

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

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

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

ist für $h \ll 1$ eine sehr langsame Konvergenz mit dem Konvergenzfaktor $\left|\frac{\lambda_2}{\lambda_1}\right|^2 = 1 - \frac{3}{2}\pi^2 h^2$ (quadratisch, da die Matrix symmetrisch ist) zu erwarten. Dies wird durch die numerischen Ergebnisse bestätigt, vgl. Tabelle 5.1 und Abbildung 5.2, welche die Ergebnisse der Anwendung der Vektoriteration auf die Matrix A_h mit $h = \frac{1}{30}$ und dem Startvektor $x^{(0)} = y^{(0)}/\|y^{(0)}\|_2$, $y^{(0)} = (1,2,\dots,29)^T$ zeigen. ∎

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

Tabelle 5.1: Konvergenz der Vektoriteration angewandt auf Beispiel 5.1.1.

▪ **Beispiel 5.2.6** Es soll der betragsgrößte Eigenwert der Matrix

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

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

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

Dies wird von den numerischen Resultaten der Vektoriteration angewandt auf A mit Startvektor $x^{(0)} = (0.5, 0.5, 0.5, 0.5)^T$ in Tabelle 5.2 bestätigt. ∎

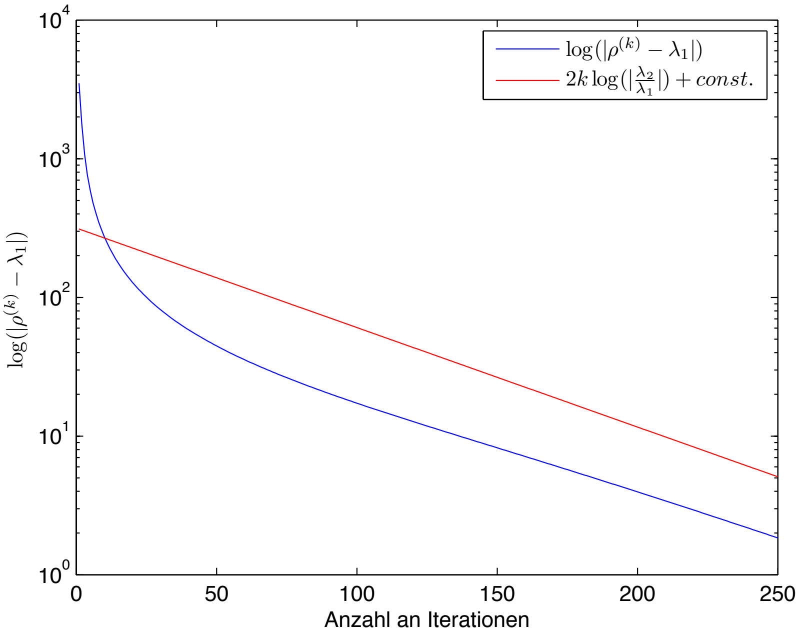


Abbildung 5.2: Semilogarithmische Darstellung der Entwicklung des Fehlers.

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

Tabelle 5.2: Konvergenz der Vektoriteration angewandt auf Beispiel 5.2.6

5.3 Inverse Iteration nach Wielandt und Shift-Strategien

In vielen (insbesondere technischen) Anwendungen wird der **betrags<u>kleinste</u>** Eigenwert λ_n einer regulären Matrix $A \in \mathbb{C}^{n\times n}$

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

gesucht. Um diesen zu finden, kann man die Tatsache ausnutzen, dass der betragskleinste Eigenwert von A der Kehrwert des betragsgrößten Eigenwert von A^{-1} ist, d.h. für die Eigenwerte ν_i von A^{-1} gilt

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

Die Anwendung der Potenzmethode auf A^{-1} liefert also eine Möglichkeit, den betragskleinsten Eigenwert $\nu_n = \frac{1}{\lambda_n}$ von A zu approximieren. In jeder Iteration ist dann der Vektor $a^{(k)} = A^{-1}x^{(k)}$ zu berechnen; dies entspricht dem Lösen des linearen Gleichungssystems $Aa^{(k)} = x^{(k)}$. Das Verfahren heißt *inverse Iteration nach Wielandt*.[5]

Mit Hilfe einer Verschiebung (shift) kann man auch die **Berechnung anderer Eigenwerte** erreichen. Hierbei wird vorausgesetzt, dass man eine gute Näherung μ für einen Eigenwert λ_j von A kennt, in dem Sinne, dass

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

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

Algorithmus 5.3.1 — Inverse Iteration nach Wielandt mit Spektralverschiebung.

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

1: **for** $k = 0, 1, \ldots$ **do**
2: Löse $(A - \mu I)\, a^{(k)} = x^{(k)}$
3: $\varrho^{(k)} = (x^{(k)})^T a^{(k)}$ {Rayleigh-Quotient}
4: $x^{(k+1)} = \frac{a^{(k)}}{\|a^{(k)}\|}$
5: **end for**

Bemerkung 5.3.2 Pro Iteration muß also ein lineares Gleichungssystem gelöst werden, dessen Koeffizientenmatrix $A - \mu I$ aber bzgl. der Iteration konstant ist. Bestimmt man also *einmal* eine LR-Zerlegung von $A - \mu I$ (und speichert diese), so sind pro Iterationsschritt zwei gestaffelte Systeme zu lösen, was einem Aufwand von $\mathcal{O}(n^2)$ entspricht.

Mit Satz 5.2.2 können wir nun folgern, dass der Rayleigh-Quotient $\varrho^{(k)}$

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

erfüllt. Gemäß der Konvergenzanalyse der Potenzmethode in Abschnitt 5.2 ergibt sich die Konvergenzgeschwindigkeit aus dem Verhältnis zwischen $\frac{1}{\lambda_j - \mu}$ und dem betragsmäßig zweitgrößten Eigenwert von $(A - \mu I)^{-1}$, also durch den Faktor

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

bestimmt wird.

[5] Helmut Wielandt, 1910-2001.

Bemerkung 5.3.3 Ist μ eine gute Schätzung von λ_j, so gilt

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

und das Verfahren konvergiert in diesem Fall sehr rasch.

Durch geeignete Wahl des Spektralverschiebungsparameters μ kann man also mit der inversen Vektoriteration in Algorithmus 5.3.1 einzelne Eigenwerte und Eigenvektoren der Matrix A bestimmen. In der Praxis ist aber oft nicht klar, wie man für einen beliebigen Eigenwert λ_j diesen Parameter μ geeignet wählen kann. Die Konvergenzgeschwindigkeit der Methode kann man noch erheblich verbessern, wenn man den Parameter μ nach jedem Schritt auf die aktuelle Annäherung $\lambda^{(k)} := \frac{1}{\varrho^{(k)}} + \mu$ von λ_j setzt. Da die LR-Zerlegung dann aber in jedem Schritt neu berechnet werden muß, steigt damit der Rechenaufwand sehr stark an.

■ **Beispiel 5.3.4** Wir betrachten die Matrix aus Beispiel 5.2.6 und wenden zur Berechnung des Eigenwerts $\lambda_3 = 4.4616$ dieser Matrix den Algorithmus 5.3.1 mit $\mu = 3.5$ und Startvektor $x^{(0)} = (0.5, 0.5, 0.5, 0.5)^T$ an. Die Resultate in Tabelle 5.3 (links) bestätigen die obigen theoretischen Betrachtungen zur Konvergenz des Wielandt-Verfahrens, nach denen wir lineare Konvergenz mit einem Konvergenzfaktor in der Größenordnung

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

erwarten.

Für die inverse Vektoriteration mit Anpassung des Parameters μ nach jedem Schritt auf die jeweils aktuelle Annäherung $\lambda^{(k)}$ vom λ_3,

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

sind einige Ergebnisse in Tabelle 5.3 (rechts) dargestellt. Die Resultate zeigen, dass die Konvergenzgeschwindigkeit wesentlich schneller (genauer: quadratisch statt linear; diese Begriffe werden in der Vorlesung *Numerische Analysis* rigoros eingeführt) ist. ■

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

■ **Beispiel 5.3.5** Gegeben sei die Matrix $A = \begin{pmatrix} -1 & 3 \\ -2 & 4 \end{pmatrix}$ mit dem Spektrum $\sigma(A) = \{1, 2\}$. Für eine Approximation $\mu := 1 - \epsilon$ von $\lambda_2 = 1$ mit $0 < |\epsilon| \ll 1$ gilt

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

Also ist $A - \mu I$ fast singulär. Da sich der Faktor $\frac{1}{\epsilon^2+\epsilon}$ bei der Normierung $x^{(k+1)} = \frac{a^{(k)}}{\|a^{(k)}\|}$ in der Wielandt-Iteration aber heraus kürzt, ist die Berechnung der Richtung einer Lösung

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

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

Tabelle 5.3: Konvergenz des Wielandt-Verfahrens angewandt auf Beispiel 5.2.6 mit $\mu = 3.5$ (links) und mit $\mu_k = \lambda^{(k-1)}$ (rechts).

von $(A - \mu I)a^{(k)} = x^k$ gut konditioniert. ∎

5.4 QR-Verfahren

Die in den vorherigen Abschnitten vorgestellten Methoden der Vektoriteration und der inversen Iteration nach Wielandt haben den Nachteil, dass man mit ihnen nur bestimmte einzelne Eigenwerte bestimmen kann. In diesem Abschnitt werden wir uns nun einem effizienteren Verfahren zuwenden, mit dessen Hilfe man nicht nur einen, sondern **gleichzeitig alle Eigenwerte** einer Matrix $A \in \mathbb{R}^{n \times n}$ approximieren kann.

Bemerkung 5.4.1 Nach Korollar 4.6.2 kann man auch die Singulärwertzerlegung zur Berechnung der Eigenwerte verwenden, in dem man die SVD von $A^T A$ bestimmt und dann die Wurzeln der Singulärwerte berechnet. Dieses Vorgehen beinhaltet aber Konditions- und Genauigkeitsprobleme (aufgrund der Wurzelberechnung und des nicht eindeutigen Vorzeichens).

Könnte man die Schur-Zerlegung $Q^H AQ = R$ aus Satz A.3.8 einer Matrix $A \in \mathbb{R}^{n \times n}$ auf direkte Weise (also mit einer endlichen Anzahl an Operationen) berechnen, so hätte man das Eigenwertproblem gelöst, da die Eigenwerte $\lambda_i(A)$ dann gerade durch die Diagonalelemente r_{ii} der oberen Dreiecksmatrix R gegeben wären. Leider ist die direkte Bestimmung der Matrix Q für $n \geq 5$ nicht möglich – dies folgt aus dem *Abelschen Theorem* (vgl. [QSS07])[6]. Daher kann das Eigenwertproblem nur durch iterative Methoden lösen.

[6]Der Satz von Niles Henrik Abel aus den Jahr 1824 lautet: Für jedes $n \geq 5$ existiert ein Polynom p mit rationalen Koeffizienten vom Grad n, das eine reelle Nullstelle r besitzt mit der Eigenschaft, dass r nicht geschrieben werden kann als algebraischer Ausdruck, der rationale Zahlen, Additionen, Subtraktionen, Multiplikationen, Divisionen und k-te Wurzeln enthält. Anders ausgedrückt existiert keine Formel und damit kein endlicher algebraischer Algorithmus, der aus den Koeffizienten eines Polynoms vom Grad $n \geq 5$ die Nullstellen berechnet.

5.4.1 Der QR-Basisalgorithmus

Den Basisalgorithmus für das Verfahren, welches wir im Weiteren vorstellen und untersuchen möchten, liefert das *QR-Verfahren*.

Algorithmus 5.4.2 — QR-Basisiteration.

Input: $A \in \mathbb{R}^{n\times n}$
1: Setze $A^{(0)} = A$.
2: **for** $k = 1, 2, \dots$ **do**
3: $\quad A^{(k-1)} = Q_k R_k$ {Bestimme QR-Zerlegung}
4: $\quad A^{(k)} = R_k Q_k$
5: **end for**

Pro Iterationsschritt fällt also der Aufwand einer QR-Zerlegung $\mathcal{O}(n^3)$ und einer Matrixmultiplikation $\mathcal{O}(n^2)$ an. Insgesamt benötigt man also für dieses Verfahren $\mathcal{O}(k_{\max} n^3)$ Operationen, wobei $k_{\max}$ die Anzahl der Iterationen im Verfahren ist.

Bemerkung 5.4.3 Das QR-Verfahren ist eine sogenannte *Unterraum-Iteration*, die allgemein dafür konstruiert sind, mehrere Eigenwerte gleichzeitig zu bestimmen (bei QR sind dies alle). Man kann diese Verfahren auch als eine Verallgemeinerung der Potenzmethode für mehrere Startvektoren (zusammengefasst in einer Matrix) verstehen. Die allgemeine Form sieht dann so aus:

Input: $A \in \mathbb{R}^{n\times n}$, Startwerte $Q^{(0)} \in \mathbb{R}^{n\times m}$ mit orthonormalen Spalten
for $k = 1, 2, \dots$ **do**

$$
\begin{aligned}
&(1) \qquad & Z_k &= AQ^{(k-1)} \\
&(2) \qquad & Z_k &= X_k R_k \qquad \text{% Orthonomalisierung (QR)} \\
&(3) \qquad & Q^{(k)} &= X_k
\end{aligned}
$$

end
Dabei ist R_k eine obere Dreiecksmatrix und X_k besitzt orthonormale Spalten.

Lemma 5.4.4 Es sei $A \in \mathbb{R}^{n\times n}$, sowie $A^{(k)}$ für $k \in \mathbb{N}_0$ wie in Algorithmus 5.4.2 definiert. Dann gilt:

i) Die Matrizen $A^{(k)}$ sind orthogonal ähnlich zu A.
ii) Ist A symmetrisch, so sind es auch die Matrizen $A^{(k)}$.
iii) Ist A symmetrisch und tridiagonal, so haben auch die Matrizen $A^{(k)}$ diese Gestalt.

Beweis. Ad i): Sei $A \in \mathbb{R}^{n\times n}$. Mit den Bezeichnungen aus Algorithmus 5.4.2 gilt dann

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

also i). Ad ii): Sei $A \in \mathbb{R}^{n\times n}$ nun zusätzlich symmetrisch. Mit i) folgt, dass es für alle $k \in \mathbb{N}_0$ eine orthogonale Matrix $P_k \in \mathbb{R}^{n\times n}$ gibt, sodass $A^{(k)} = P_k^T A \, P_k$ gilt. Daraus folgt

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

Für den Beweis von iii) sei auf [DH19, Lemma 5.9] verwiesen. ∎

Für die QR-Basisiteration gilt folgendes Konvergenzresultat, das auf Wilkinson[7] zurückgeht, [Wil88].

Satz 5.4.5 — Wilkinson. Es sei $A \in \mathbb{R}^{n\times n}$ symmetrisch mit Eigenwerten

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

und $A^{(k)}$, Q_k sowie R_k seien wie in Algorithmus 5.4.2 definiert. Dann gilt:

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

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

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

Beweis. (i) Wir zeigen zunächst induktiv, dass für $k \in \mathbb{N}$ gilt:

$$A^k = \underbrace{Q_1 \cdots Q_k}_{=:P_k} \underbrace{R_k \cdots R_1}_{=:U_k}. \tag{5.9}$$

Für $k = 1$ ist die Behauptung klar. Aus der Konstruktion der $A^{(k)}$ folgt wie im Beweis zu Lemma 5.4.4, dass

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

gilt und somit auch der Induktionsschritt (mit der Induktionsannahme (IA))

$$\begin{aligned} A^{k+1} &= AA^k \overset{\text{(IA)}}{=} AP_kU_k = P_k \underbrace{P_k^T A P_k}_{=Q_{k+1}R_{k+1}=A^{(k)}} U_k \\ &= P_k A^{(k)} U_k = P_k Q_{k+1} R_{k+1} U_k = P_{k+1} U_{k+1}, \end{aligned}$$

und damit (5.9).

(ii) Nach Voraussetzung ist A symmetrisch und somit diagonalisierbar. Es existiert also eine orthogonale Matrix Q mit

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

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

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

Für die unipotente untere Dreiecksmatrix $(\Lambda^k L \Lambda^{-k})$ gilt mit $L = (l_{ij})_{i,j=1,\dots,n}$

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

[7] James H. Wilkinson, 1919-1986.

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

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

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

$$A^k = Q^T (I + E_k)\Lambda^k R = (Q^T \widetilde{Q}_k)(\widetilde{R}_k \Lambda^k R)\,, \tag{5.10}$$

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

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

(iii) Wir haben in (5.10) neben (5.9) eine weitere QR-Zerlegung von A^k gefunden, daher gilt bis auf Vorzeichen der Diagonale

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

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

$$\begin{aligned} Q_k &= (Q_1 \cdots Q_{k-1})^T (Q_1 \cdots Q_k) = P_{k-1}^T P_k = \widetilde{Q}_{k-1}^T Q\, Q^T \widetilde{Q}_k = \widetilde{Q}_{k-1}^T \widetilde{Q}_k \xrightarrow{k\to\infty} I\,, \\ R_k &= (R_k \cdots R_1)(R_{k-1} \cdots R_1)^{-1} = U_k U_{k-1}^{-1} = \widetilde{R}_k \Lambda^k R\, R^{-1} \Lambda^{-(k-1)} \widetilde{R}_{k-1}^{-1} \\ &= \widetilde{R}_k \Lambda \widetilde{R}_{k-1}^{-1} \xrightarrow{k\to\infty} \Lambda \end{aligned}$$

und

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

was den Beweis komplettiert.

∎

Bemerkung 5.4.6 i) Sofern A nicht symmetrisch ist, aber die Eigenwerte von A immer noch getrennt sind, d.h., $|\lambda_1| > |\lambda_2| > \cdots > |\lambda_n|$, kann man zeigen, dass die Folge $A^{(k)}$ anstatt gegen eine Diagonal- gegen eine Dreiecksmatrix konvergiert:

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

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

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

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

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

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

5.4.2 Verfeinerungen und Beschleunigungen

Der QR-Algorithmus 5.4.2 in seiner Grundform ist sehr aufwendig. Pro Iterationsschritt $A^{(k)} \to A^{(k+1)}$ benötigt man bei vollbesetzten Matrizen $\mathcal{O}(n^3)$ Operationen. Zudem besitzt das Verfahren in dieser Form auch den Nachteil, den schon die inverse Iteration nach Wielandt aufwies: Sind einige Eigenwerte von A betragsmäßig nur schlecht getrennt, d.h., $|\lambda_j/\lambda_k| \approx 1$ für $j \neq k$, so ist die Konvergenz des Verfahrens nur sehr langsam, vgl. Satz 5.4.5 iii). Mögliche Auswege sind:

1. Transformation auf einfachere Gestalt (z.B. Hessenberg-Form, vgl. Definition 4.5.1). Man kann dann insbesondere zeigen, dass die Iterierten wiederum Hessenberg-Form haben und somit jeder Schritt in $\mathcal{O}(n^2)$ Operationen durchführbar ist, da eine QR-Zerlegung einer Hessenberg-Matrix mittels $n-1$ Givens-Rotationen effizient berechnet werden kann.
   Hinzu kommt die einmalige Transformation auf Hessenberg-Form mit (im Allgemeinen) $\mathcal{O}(n^3)$ Operationen. In §4.5 (Lemma 4.5.2) hatten wir diese Transformation mittels Givens-Rotationen gesehen, es gibt auch eine analoge Transformation mit Householder-Spiegelungen. Insbesondere für große und dünnbesetzte Matrizen werden wir später noch eine Alternative kennenlernen, das *Arnoldi-Verfahren*.
2. *Shift-Techniken* oder *Verschiebungen* können zu einer Verbesserung des Konvergenzverhaltens des Verfahrens führen. Beispielsweise kann man Algorithmus 5.4.2 ersetzen durch

$$\begin{aligned} A^{(k-1)} - \mu_k I &= Q_k R_k \qquad && \text{\% Bestimme QR-Zerlegung} \\ A^{(k)} &= R_k Q_k + \mu_k I \end{aligned}$$

   wobei man i.d.R. die Transformation auf Hessenberg-Form vorschaltet. In der Literatur sind diverse Shift-Strategien beschrieben, siehe z.B. [GV13, §7.5].

5.5 Krylov-Raum-Methoden

Bei der Potenzmethode haben wir die Vektoren $Ax^{(0)}, A^2x^{(0)}, A^3x^{(0)}, ..., A^{k-1}x^{(0)}$ ausgehend von einem Startwert $x^{(0)}$ (plus Normalisierung) berechnet. Wir haben in Satz 5.2.2 gesehen, dass die Iteration gegen den normierten Eigenvektor zum betragsgrößten Eigenwert konvergiert. Wir verwenden also nur die letzte Iterierte $A^{k-1}x^{(0)}$ als Näherung für diesen Eigenvektor. Die davor berechneten Vektoren werden nicht verwendet, was ggf. nicht effizient ist. Dies motiviert folgende Definition.

Definition 5.5.1 Sei $A \in \mathbb{R}^{n\times n}$, $v \in \mathbb{R}^n$ und $m \in \mathbb{N}$. Dann heißt

$$\mathcal{K}_m(A, v) := \operatorname{span}\{v, Av, ..., A^{m-1}v\}$$

Krylov-Raum[8] der Ordnung m und $K_m(A, v) := [v, Av, ..., A^{m-1}v]$ heißt *Krylov-Matrix*.

Wir wollen nun die Krylov-Räume für ein Unterraum-Verfahren nach Bemerkung 5.4.3 verwenden. Dazu benötigen wir ein Orthogonalisierungsverfahren (zur Durchführung von $Z_k = X_kR_k$ in Schritt (2)). Wir kennen aus der Linearen Algebra das Gram-Schmidt-Orthonormalisierungsverfahren (Satz A.4.1). Ähnlich gehen wir hier vor.

Algorithmus 5.5.2 — Arnoldi-Verfahren.

Input: $A \in \mathbb{R}^{n\times n}$, Vektor $v \in \mathbb{R}^n$

```
Setze q_1 := v / ||v||_2
for k = 1, 2, ... do
    for j = 1, 2, ..., k do
        h_jk := (A q_k)^T q_j
    end for
    q̂_{k+1} = A q_k − Σ_{j=1}^{k} h_jk q_j ∈ R^n
    if q̂_{k+1} = 0 then
        setze m := k, stop
    else
        h_{k+1,k} := ||q̂_{k+1}||_2
        q_{k+1} := q̂_{k+1} / ||q̂_{k+1}||_2
    end if
end for
```

Output: orthonormale Vektoren $q_1, ..., q_m$

Lemma 5.5.3 (i) Die durch das Arnoldi-Verfahren erzeugten Vektoren $q_1, .., q_m$ sind eine Orthonomalbasis vom $\mathcal{K}_m(A, v)$.

(ii) Ist A regulär, so gilt für die Lösung des linearen Gleichungssystems $A^{-1}b \in \mathcal{K}_m(A, b)$.

Beweis. Die Normierung der q_i folgt unmittelbar aus dem Algorithmus. Die Orthogonaltät zeigen wir per Induktion. Per Definition gilt für $k = 1, ..., m-1$

$$q_{k+1} = \frac{\hat{q}_{k+1}}{h_{k+1,k}} = \frac{1}{h_{k+1,k}}\Big(Aq_k - \sum_{j=1}^{k} h_{jk}q_j\Big). \tag{5.11}$$

[8] Nikolai M. Krylov, 1879-1955; Vladimir I. Krylov, 1902-1994; Krylov-Unterraumverfahren wurden unter die zehn wichtigsten numerischen Algorithmen des 20. Jahrhunderts eingestuft.

Damit gilt

$$\langle q_1, q_2\rangle = \tfrac{1}{h_{21}} q_1^T\Big(Aq_1 - h_{11}q_1\Big) = \tfrac{1}{h_{21}}(q_1^T Aq_1 - h_{11}\|q_1\|_2^2) = \tfrac{1}{h_{21}}(h_{11} - h_{11}) = 0.$$

Nehmen wir nun an, dass $q_1, ..., q_k$ für ein $2 \le k < m$ paarweise orthogonal sind (Induktionsannahme). Dann gilt für $j = 1, ..., k$ mit (5.11) und der Induktionsannahme (IA)

$$\begin{aligned}\langle q_{k+1}, q_j\rangle &= \tfrac{1}{h_{k+1,k}}\Big(Aq_k - \sum_{\ell=1}^{k} h_{\ell k}\, q_\ell\Big)^T q_j = \tfrac{1}{h_{k+1,k}}\Big((Aq_k)^T q_j - \sum_{\ell=1}^{k} h_{\ell k}\, \underbrace{\langle q_\ell, q_j\rangle}_{\overset{\text{(IA)}}{=}\delta_{\ell,j}}\Big)\\ &= \tfrac{1}{h_{k+1,k}}(h_{jk} - h_{jk}) = 0.\end{aligned}$$

Also sind $\{q_1, ..., q_m\}$ orthonormal, insbesondere linear unabhängig. Für $1 \le k \le m$ sind per Konstruktion im Algorithmus die Vektoren $\{q_1, ..., q_{k-1}, Aq_{k-1}\}$ linear unabhängig, also aus Dimensionsgründen

$$\operatorname{span}\{q_1, ..., q_k\} = \operatorname{span}\{q_1, ..., q_{k-1}, Aq_{k-1}\} \quad \text{für alle } 1 \le k \le m.$$

Wir wollen noch zeigen, dass $\operatorname{span}\{q_1, ..., q_{k-1}, Aq_{k-1}\} = \mathcal{K}_k(A, v)$ gilt. Wegen $Aq_{k-1} \in A(\mathcal{K}_{k-1}(A, v)) \subseteq \mathcal{K}_k(A, v)$ gilt „$\subseteq$“. Weiter gilt

$$k = \dim(\operatorname{span}\{q_1, ..., q_{k-1}, Aq_{k-1}\}) \le \dim(\mathcal{K}_k(A, v)) \le k,$$

also muss Gleichheit gelten und die beiden Räume sind identisch. Dies zeigt (i).

Nun zur zweiten Aussage (ii): Hier starten wir den Algorithmus mit $v := b$. Aufgrund der Abbruchbedingung gilt $Aq_m \in \operatorname{span}\{q_1, ..., q_m\}) = \mathcal{K}_m(A, b)$, wobei die letzte Identität aus (i) folgt. Per Konstruktion gilt $Aq_j \in \mathcal{K}_{j+1}(A, b) \subseteq \mathcal{K}_m(A, b)$ für alle $j = 1, .., m - 1$ und damit $A(\mathcal{K}_m(A, b)) \subseteq \mathcal{K}_m(A, b)$. Wie oben folgt aus Dimensionsgründen die Gleichheit, also $A(\mathcal{K}_m(A, b)) = \mathcal{K}_m(A, b)$. Damit ist die lineare Abbildung $A : \mathcal{K}_m(A, b) \to \mathcal{K}_m(A, b)$ bijektiv, also muss $A^{-1}b \in \mathcal{K}_m(A, b)$ gelten. ∎

Bemerkung 5.5.4 Im Beweis haben wir sogar eine stärkere Aussage bewiesen, nämlich

$$\operatorname{span}\{q_1, ..., q_k\} = \operatorname{span}\{q_1, ..., q_{k-1}, Aq_{k-1}\} = \mathcal{K}_k(A, v) \quad \text{für alle } 1 \le k \le m. \tag{5.12}$$

Satz 5.5.5 Sei $A \in \mathbb{R}^{n\times n}$, $0 \ne v \in \mathbb{R}^n$ und $Q_k := (q_1, ..., q_k) \in \mathbb{R}^{n\times k}$, $k = 1, ..., m$ mit den Größen aus der Arnoldi-Iteration. Dann gilt

$$\begin{aligned} AQ_k &= Q_{k+1}H_k, \quad \text{mit } H_k := (h_{ij})_{i=1,...,k+1, j=1,..,k} \in \mathbb{R}^{(k+1)\times k}, \quad k = 1, ..., m-1,\\ AQ_m &= Q_m H_m, \quad \text{mit } H_m := (h_{ij})_{i,j=1,..,m} \in \mathbb{R}^{m\times m}.\end{aligned}$$

Beweis. Wir brauchen nur die zweite Identität zu beweisen, da sich die erste dann aus den spaltenweisen Produkten ergibt. Es gilt aufgrund des Algorithmus für $k = 1, ..., m$

$$(Q_m H_m)_k = \sum_{j=1}^{k+1} h_{jk} q_j = h_{k+1,k} q_{k+1} + \sum_{j=1}^{k} h_{jk} q_j = \hat{q}_{k+1} + Aq_k - \hat{q}_{k+1} = Aq_k = (AQ_m)_k$$

und damit die Behauptung. ■

Bemerkung 5.5.6 (a) Ein bekannter Spezialfall für symmetrische Matrizen ist das so genannte *Lanczos-Verfahren*.

(b) Bricht der Algorithmus nicht ab, dann gilt $m = n$ und aufgrund von Satz 5.5.5 gilt dann $Q_n^T A Q_n = H_n$ und H_n ist eine obere Hessenberg-Matrix. Wir haben also eine Ähnlichkeitstransformation von A auf Hessenberg-Gestalt gefunden, eine weitere neben Givens-Rotationen und Householder-Spiegelungen. Insbesondere für sehr große und dünnbesetzte Matrizen A ist das Arnoldi-Verfahren effizienter.

Im Folgenden zeigen wir eine Matlab-Realisierung mittels ChatGBT.

`Schreibe mir bitte ein Matlab-Programm für die Arnoldi-Iteration`

`Die Arnoldi-Iteration ist ein Verfahren zur Approximation von Eigenwerten und Eigenvektoren großer dünn besetzter Matrizen. Es wird häufig in der numerischen linearen Algebra und der Berechnung von Eigenwerten verwendet. Hier ist ein einfaches MATLAB-Programm, das die Arnoldi-Iteration implementiert:`

MATLAB-Funktion per ChatGBT: arnoldi_iteration.m

```
function [H, V] = arnoldi_iteration(A, v0, k)
    % Arnoldi-Iteration zur Approximation von Eigenwerten und -vektoren
    % Input:
    %   A: Die quadratische Matrix, von der die Eigenwerte und -vektoren approximiert werden sollen.
    %   v0: Der Startvektor fuer die Iteration.
    %   k: Die Anzahl der Iterationsschritte / Groesse der Hessenberg-Matrix H.
    % Output:
    %   H: Die Hessenberg-Matrix.
    %   V: Die Matrix der Arnoldi-Basisvektoren.

    n = size(A, 1);
    V = zeros(n, k+1);
    H = zeros(k+1, k);

    V(:, 1) = v0 / norm(v0); % Normalisiere den Startvektor und setze als ersten Basisvektor

    for j = 1:k
        w = A * V(:, j);
        for i = 1:j
            H(i, j) = V(:, i)' * w;
            w = w - H(i, j) * V(:, i);
        end
        H(j+1, j) = norm(w);
        if H(j+1, j) == 0
```

```
            break; % Falls w=0, brich ab, da keine weiteren
                Eigenwerte approximiert werden koennen.
        end
        V(:, j+1) = w / H(j+1, j);
    end
end
```

Dieses MATLAB-Programm verwendet die Arnoldi-Iteration, um die Hessenberg-Matrix H und die Matrix der Arnoldi-Basisvektoren V zu berechnen. Die Hessenberg-Matrix ist eine obere Dreiecksmatrix, die häufig zur Approximation von Eigenwerten und -vektoren verwendet wird. Die Arnoldi-Basisvektoren sind eine Orthonormalbasis des Krylov-Unterraums, der durch die Matrix A und den Startvektor v0 erzeugt wird.
Anwendung des Codes: Um die Arnoldi-Iteration auf eine spezifische Matrix A mit einem Startvektor v0 und einer bestimmten Anzahl von Iterationsschritten k anzuwenden, rufen Sie die Funktion arnoldi_iteration wie folgt auf:

MATLAB-Funktion per ChatGBT: arnoldi_test.m

```
A = ...; % Gib deine quadratische Matrix A ein
v0 = ...; % Gib deinen Startvektor v0 ein
k = ...;  % Gib die Anzahl der Iterationsschritte ein

[H, V] = arnoldi_iteration(A, v0, k);

% Hier sind H und V die Ergebnisse der Arnoldi-Iteration.
% Du kannst die Eigenwerte und -vektoren aus H ableiten.
```

Bitte beachten Sie, dass die Arnoldi-Iteration ein iteratives Verfahren ist und dass die Genauigkeit der Eigenwertapproximation von der Anzahl der Iterationsschritte abhängt. Je mehr Iterationsschritte durchgeführt werden, desto genauer wird die Approximation sein, aber auch die Berechnungszeit wird zunehmen.

■ **Beispiel 5.5.7** Noch einmal betrachten wir das Eigenwertproblem aus Beispiel 5.1.1, vgl. auch Beispiel 5.2.5 mit $h = \frac{1}{30}$. In der nachfolgenden Tabelle sind die Fehler der betragsgrößten 10 Eigenwerte für die Anzahl `It` der Arnoldi-Iterationen angegeben. Für `It` $= 5$ erhalten wir nur Näherungen der ersten fünf Eigenwerte. Wir erkennen die schnelle Konvergenz. ■

It	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9	e_{10}
5	0.368	1.3280	2.502	3.423	3.732	–	–	–	–	–
10	0.087	0.338	0.725	1.207	1.731	2.236	2.665	2.965	3.096	3.047
15	0.031	0.121	0.267	0.460	0.691	0.947	1.215	1.479	1.726	1.939
20	0.031	0.071	0.070	0.196	0.371	0.445	0.456	0.683	0.921	1.159
25	0.012	0.000	0.037	0.000	0.107	0.132	0.140	0.320	0.530	0.543
30	0.000	0.000	0.000	0.000	0.012	0.000	0.001	0.000	0.021	0.000
35	0.000	0.000	0.000	0.000	0.012	0.000	0.001	0.000	0.021	0.000

Kapitel 6

Iterative Lösung linearer Gleichungssysteme

Die in Kapitel 3 beschriebenen direkten Verfahren gehen überwiegend von beliebigen vollbesetzten Matrizen aus. Viele praktischen Probleme führen aber auf sehr große lineare Gleichungssysteme (LGS) $Ax = b$, bei dem $A \in \mathbb{R}^{n\times n}$ dünnbesetzt ist, d.h. viele Nulleinträge besitzt[1]. Die bisher vorgestellten Verfahren nutzen diese spezielle Struktur nicht aus und führen beim Lösen des LGS teilweise sogar zu vollbesetzten Zwischenmatrizen. Man betrachte dazu folgendes Beispiel.

■ **Beispiel 6.0.1** Zu der Matrix

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & & & & \\ 1 & & 5 & & & \\ 1 & & & 10 & & \\ 1 & & & & 15 & \\ 1 & & & & & 10 \end{pmatrix}$$

lautet die LR-Zerlegung mit $A = L \cdot R$

$$L = \begin{pmatrix} 1 & & & & & \\ 1 & 1 & & & & \\ 1 & -1 & 1 & & & \\ 1 & -1 & -\frac{2}{3} & 1 & & \\ 1 & -1 & -\frac{2}{3} & -\frac{1}{2} & 1 & \\ 1 & -1 & -\frac{2}{3} & -\frac{1}{2} & -\frac{1}{2} & 1 \end{pmatrix}, \qquad R = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ & 1 & -1 & -1 & -1 & -1 \\ & & 3 & -2 & -2 & -2 \\ & & & \frac{20}{3} & -\frac{10}{3} & -\frac{10}{3} \\ & & & & 10 & -5 \\ & & & & & \frac{5}{2} \end{pmatrix}.$$

Obwohl A nur in der ersten Zeile und Spalte sowie auf der Diagonalen Nichtnulleinträge besitzt, sind L und R vollbesetzt. ■

■ **Beispiel 6.0.2 — 2D-Standardmatrix.** In Erweiterung zu Beispiel 3.0.1 betrachten wir nun

[1] Als „formale“ Definiton könnte man sagen, dass eine Matrix $A \in \mathbb{R}^{n\times n}$ *dünnbesetzt* heißt, wenn die Anzahl der von Null verschiedenen Einträge $\mathcal{O}(n)$ sind, mit „kleiner“ Konstante – natürlich ist dies keine scharfe Definition.

eine (zweidimensionale) quadratische Membran, die am Rand fest eingespannt ist und auf die eine äußere Kraft f wirkt. Die gesuchte Auslenkung $u : [0,1]^2 \to \mathbb{R}$ ergibt sich als Lösung des sogenannten *Dirichlet-Problems* auf dem Einheitsquadrat

$$\begin{aligned}
-\Delta u(x,y) &:= -\frac{\partial^2}{\partial x^2}u(x,y) - \frac{\partial^2}{\partial y^2}u(x,y) = f(x,y), \quad (x,y) \in \Omega := (0,1)^2, \qquad (6.1)\\
u(x,y) &= 0, \quad (x,y) \in \Gamma := \partial\Omega.
\end{aligned}$$

Den Differenzialoperator Δ nennt man auch *Laplace-Operator*.

Wir verwenden die gleiche Idee wie im eindimensionalen Fall (Beispiel 3.0.1) und definieren ein äquidistantes Gitter

$$\Omega_h := \{(x,y) \in \overline{\Omega} : x = kh, \quad y = \ell h, \quad 0 \le k, \ell \le n_h + 1\} \tag{6.2}$$

für $h := \frac{1}{n_h+1}$, $n_h \in \mathbb{N}$, wie in Beispiel 3.0.1. Der Rand besteht jetzt natürlich aus mehr als zwei Punkten, nämlich

$$\partial\Omega_h := \{(x,y) \in \Gamma : x = kh \ \text{ oder } \ y = \ell h, \quad 0 \le k, \ell \le n_h + 1\}, \tag{6.3}$$

vgl. Abbildung 6.1. Wir definieren $\mathring{\Omega}_h := \Omega_h \setminus \partial\Omega_h$.

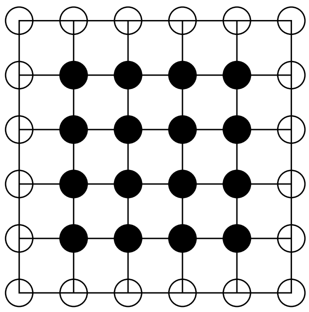


Abbildung 6.1: Äquidistantes Gitter auf $\overline{\Omega} = [0,1]^2$. Die inneren Punkte ($\mathring{\Omega}_h$) sind ausgefüllt dargestellt, $\partial\Omega_h$ besteht aus den nicht ausgefüllten Punkten ($\circ$).

Die partiellen Ableitungen zweiter Ordnung approximieren wir wieder durch den zentralen Differenzenquotienten, also

$$\begin{aligned}
\Delta u(x,y) &= \frac{\partial^2}{\partial x^2}u(x,y) + \frac{\partial^2}{\partial y^2}u(x,y)\\
&\approx \frac{1}{h^2}(u(x+h,y) - 2u(x,y) + u(x-h,y))\\
&\qquad + \frac{1}{h^2}(u(x,y+h) - 2u(x,y) + u(x,y-h))\\
&= \frac{1}{h^2}(u(x+h,y) + u(x-h,y) + u(x,y+h) + u(x,y-h) - 4u(x,y)). \qquad (6.4)
\end{aligned}$$

Wir erhalten wieder ein lineares Gleichungssystem $A_h u_h = f_h$. Die genaue Gestalt der Matrix $A_h \in \mathbb{R}^{n_h^2 \times n_h^2}$ hängt von der Nummerierung ab. Wir wählen die so genannte *lexiko-*

graphische Nummerierung (vgl. Abbildung 6.2)

$$z_k = (x_i, y_j), \quad x_i = ih, \quad y_j = jh, \quad k := (j-1)n_h + i.$$

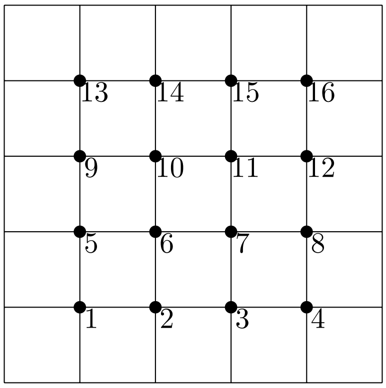


Abbildung 6.2: Lexikographische Nummerierung der Gitterpunkte für $\Omega = (0,1)^2$, $n_h = 5$.

Dann erhält man für das lineare Gleichungssystem $A_h u_h = f_h$ eine *Block-Tridiagonalmatrix*

$$A_h = \begin{bmatrix} B_h & C_h & & & 0 \\ C_h & B_h & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & B_h & C_h \\ 0 & & & C_h & B_h \end{bmatrix} \in \mathbb{R}^{n_h^2 \times n_h^2}$$

mit den Blöcken

$$B_h = \begin{bmatrix} 4 & -1 & & & 0 \\ -1 & 4 & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & 4 & -1 \\ 0 & & & -1 & 4 \end{bmatrix} \in \mathbb{R}^{n_h \times n_h} \quad \text{und} \quad C_h = \operatorname{diag}(-1) \in \mathbb{R}^{n_h \times n_h},$$

der rechten Seite $f_h = (h^2 f(z_k))_{k=1,\dots,n_h^2}$ mit $k = (j-1)n_h + i$, $1 \le i,j \le n_h$, sowie dem gesuchten Lösungsvektor $u_h = (u(z_k))_{k=1,\dots,n_h^2} \in \mathbb{R}^{n_h^2}$. Wir nennen diese Matrix auch *2D-Standardmatrix*, sie ist die direkte Verallgemeinerung der Standardmatrix aus Beispiel 3.0.1. Für die Matrix A_h kann man keine einfache Rekursionsformel für die Cholesky-Zerlegung wie in §3.8 herleiten. Allerdings ist A_h dünnbesetzt, symmetrisch und positiv definit. Mehr zu diesem Beispiel in [AU18; AU23]. ■

Bemerkung 6.0.3 Im obigen Beispiel ist A_h eine Bandmatrix mit Bandbreite n_h, also groß. Ist die Bandbreite einer Matrix (vgl. §3.8) groß und in jeder Zeile treten nur wenige Nichtnulleinträge auf, dann muss man mit fill-in rechnen und ein Bandlöser ist „teuer“. Die folgenden Verfahren liefern gute Alternativen.

Aus den oben genannten Gründen wurden schon früh iterative Verfahren zur Lösung von LGS herangezogen. Bei diesen Verfahren wird ausgehend von einem Startvektor $x^{(0)}$ eine Folge von Vektoren

$$x^{(0)} \to x^{(1)} \to x^{(2)} \to \cdots$$

mittels einer Iterationsvorschirft

$$x^{(i+1)} = \phi(x^{(i)}), \quad i = 0, 1, \ldots \tag{6.5}$$

erzeugt, die gegen die gesuchte Lösung x konvergiert. In den folgenden Abschnitten werden zunächst die klassischen Iterationsverfahren vorgestellt, die bereits Mitte des 19. Jahrhunderts entdeckt wurden und heute immer noch große Bedeutung als Teile hocheffizienter „Mehrgitter-Verfahren" besitzen. Des weiteren behandeln wir das Gradienteverfahren, das 1952 von Hestenes und Stiefel[2] entwickelte Verfahren der konjugierten Gradienten und das 1986 von Saad und Schultz[3] vorgestellte GMRES-Verfahren (Generalized Minimal RESidual method). Allen diesen Verfahren ist gemein, dass ein einzelner Iterationsschritt $x^{(i)} \to x^{(i+1)}$ einen Rechenaufwand erfordert, welcher vergleichbar ist mit der Multiplikation von A mit einem Vektor, d.h. insbesondere mit einem geringen (sprich linearem) Aufwand, sofern A dünnbesetzt ist.

6.1 Klassische Iterationsverfahren – Splitting-Methoden

Gegeben sei eine reguläre Matrix $A \in \mathbb{R}^{n \times n}$ und ein lineares Gleichungssystem $Ax = b$ mit der exakten Lösung $x^* := A^{-1}b$. Mit Hilfe einer beliebigen regulären Matrix $B \in \mathbb{R}^{n \times n}$ erhält man Iterationsvorschriften der Form (6.5) aus der Gleichung $Bx + (A - B)x = b$, indem man

$$Bx^{(i+1)} + (A - B)x^{(i)} = b \tag{6.6}$$

setzt und nach $x^{(i+1)}$ auflöst, also

$$x^{(i+1)} = x^{(i)} - B^{-1}(Ax^{(i)} - b) = (I - B^{-1}A)x^{(i)} + B^{-1}b =: Cx^{(i)} + B^{-1}b, \tag{6.7}$$

mit der *Iterationsmatrix* $C := I - B^{-1}A$. Jede Wahl einer nichtsingulären Matrix B führt zu einem möglichen **Iterationsverfahren**. Es wird umso brauchbarer, je besser B die folgenden (miteinander konkurrierende) Kriterien erfüllt:

i) B ist leicht zu invertieren (**einfache Realisierbarkeit**);
ii) B soll eine gute Approximation von A sein (**Konvergenzeigenschaft**).

In der Tat wäre das Optimum für i) die Wahl $B = I$, was für ii) schlecht wäre und umgekehrt wäre $B = A$ für ii) optimal, für i) aber schlecht. Wir geben nun einige Beispiele für eine mögliche Wahl von B an. Dazu verwenden wir folgende (additive) Standardzerlegung

$$A = L + D + R, \tag{6.8}$$

[2]Magnus Hestenes, 1906-1991; Eduard Stiefel, 1980-1978.
[3]Yousef Saad, 1950-; Martin H. Schultz.

wobei D eine **Diagonalmatrix**, L eine strikte **untere Dreiecksmatrix** und R eine strikte **obere Dreiecksmatrix** ist. Die Verwendung dieser Zerlegung ist auch der Grund für den Namen **Splitting-Methode**. Die Wahl

i) $B = \gamma I$ liefert das **Richardson**-Verfahren,
ii) $B = D$ liefert das **Jacobi**-Verfahren (**Gesamtschrittverfahren**),
iii) $B = L + D$ oder $B = D + R$ liefert das **Gauß-Seidel**-Verfahren (**Einzelschrittverfahren**).

Praktische Anwendung

In der praktischen Anwendung kann man bei dem **Gauß-Seidel-Verfahren** mit $B = L + D$ folgende Formulierung verwenden, um die Anzahl der Operationen zu reduzieren

$$x^{(i+1)} = (L + D)^{-1}(b - Rx^{(i)}), \tag{6.9}$$

da

$$\begin{aligned} x^{(i+1)} &= x^{(i)} - B^{-1}(Ax^{(i)} - b) = x^{(i)} - B^{-1}([(A - B) + B]\, x^{(i)} - b) \\ &= x^{(i)} - B^{-1}(A - B)x^{(i)} - B^{-1}Bx^{(i)} + B^{-1}b = B^{-1}(b - (A - B)x^{(i)}) \end{aligned}$$

gilt und mit $B = L + D$ sowie $A - B = R$ folgt (6.9). Ein Schritt des Gauß-Seidel-Verfahrens ist also etwa so aufwendig wie eine Matrix-Vektor-Multiplikation. Ähnlich kann man auch für $B = D + R$ vorgehen, d.h.

$$x^{(i+1)} = (D + R)^{-1}(b - Lx^{(i)}). \tag{6.10}$$

Das **Jacobi-Verfahren** vereinfacht man zu

$$x^{(i+1)} = D^{-1}(b - (L + R)x^{(i)}) \tag{6.11}$$

und das **Richardson-Verfahren** zu

$$x^{(i+1)} = x^{(i)} + \tfrac{1}{\gamma}(b - Ax^{(i)}). \tag{6.12}$$

6.1.1 Konvergenz iterativer Verfahren

Es sei wieder x^* Lösung von $Ax = b$. Mit (6.7) erhalten wir

$$\begin{aligned} x^* - x^{(k)} &= x^* - B^{-1}b - (I - B^{-1}A)x^{(k-1)} \\ &= Ix^* - B^{-1}Ax^* - (I - B^{-1}A)x^{(k-1)} \\ &= (I - B^{-1}A)(x^* - x^{(k-1)}) = \cdots \\ &= (I - B^{-1}A)^k\,(x^* - x^{(0)}) = C^k\,(x^* - x^{(0)}). \end{aligned} \tag{6.13}$$

Die Konvergenz des dargestellten Verfahrens hängt also nur von den Eigenschaften der Iterationsmatrix $C = I - B^{-1}A$ ab.

Definition 6.1.1 Es sei $C \in \mathbb{C}^{n\times n}$ mit Eigenwerten $\lambda_i := \lambda_i(C)$, $i = 1, \ldots, n$. Dann heißt

$$\varrho(C) := \max_{1\le i\le n}\{|\lambda_i(C)|\}$$

Spektralradius von C.

> **Satz 6.1.2** Ein iteratives Verfahren (6.7) konvergiert genau dann für alle Startwerte $x^{(0)} \in \mathbb{R}^n$ falls $\varrho(C) < 1$. Insbesondere gilt für jede Vektornorm
>
> $$\|x^* - x^{(k)}\| \le \|C\|^k \, \|x^* - x^{(0)}\|.$$

Beweis. Angenommen, es gelte $\varrho(C) \ge 1$. Dann gibt es einen Eigenwert λ mit $|\lambda| \ge 1$ und einen Vektor $x \neq 0$ mit $Cx = \lambda x$. Wegen $C^k x = \lambda^k x$ und $\lim\limits_{k\to\infty} \lambda^k \neq 0$ kann folglich $(C^k)_{k\in\mathbb{N}}$ keine Nullfolge sein. Die Bedingung $\varrho(C) < 1$ ist somit notwendig.

Sei nun $\varrho(C) < 1$. Weil $(TCT^{-1})^k = TC^kT^{-1}$ für jede Ähnlichkeitstransformation T gilt, reicht es, $\lim\limits_{k\to\infty} (TCT^{-1})^k = 0$ zu zeigen. Die Matrix C lässt sich durch Ähnlichkeitstransformation auf die Jordansche Normalform J transformieren, also $J = TCT^{-1}$ mit einem geeigneten T. Wir zeigen, dass $\lim\limits_{k\to\infty} J^k = 0$ gilt, wenn alle Eigenwerte $\lambda_1, \dots, \lambda_n$ dem Betrag nach kleiner Eins sind. Dazu sei

$$E_\mu = E_{n_\mu}(\lambda_\mu) = \begin{pmatrix} \lambda_\mu & 1 & & 0 \\ & \ddots & \ddots & \\ & & \ddots & 1 \\ 0 & & & \lambda_\mu \end{pmatrix} \in \mathbb{C}^{n_\mu \times n_\mu} \tag{6.14}$$

eine Elementarmatrix zum Eigenwert λ_μ der Jordanschen Normalform J von C (vgl. Satz A.3.7). Da offenbar

$$J^k = \begin{pmatrix} E_1^k & & & \\ & E_2^k & & \\ & & \ddots & \\ & & & E_\ell^k \end{pmatrix} \tag{6.15}$$

mit $1 \le \ell \le n$ gilt, genügt es, das Konvergenzverhalten einer Jordanmatrix E_μ zu untersuchen. Wir schreiben E_μ in der Form $E_\mu = \lambda_\mu I + S$ mit

$$S = \begin{pmatrix} 0 & 1 & & 0 \\ & \ddots & \ddots & \\ & & \ddots & 1 \\ 0 & & & 0 \end{pmatrix} \in \mathbb{C}^{n_\mu \times n_\mu} \tag{6.16}$$

und bilden $E_\mu^k = (\lambda_\mu I + S)^k$. Nach Anwendung der Binomialentwicklung und unter Beachtung von $S^{n_\mu} = 0$ erhält man die Beziehung

$$E_\mu^k = \sum_{\nu=0}^{\min\{k, n_\mu - 1\}} \binom{k}{\nu} \lambda_\mu^{k-\nu} S^\nu.$$

Für feste ν hat man mit

$$\binom{k}{\nu} = \frac{k!}{\nu!(k-\nu)!} = \frac{k(k-1)\cdots(k-\nu+1)}{1\cdots\nu} \le k^\nu$$

die Abschätzung

$$\left|\binom{k}{\nu}\lambda_\mu^{k-\nu}\right| \le |\lambda_\mu^{k-\nu}k^\nu| \,.$$

Da $|\lambda_\mu| < 1$ ist, gilt $k\log|\lambda_\mu| + \nu\log(k) \to -\infty$ mit $k \to \infty$ und mit

$$|\lambda_\mu^{k-\nu}k^\nu| \le \exp((k-\nu)\log|\lambda_\mu| + \nu\log k)$$

folgt $\lim\limits_{k\to\infty}\left|\binom{k}{\nu}\lambda_\mu^{k-\nu}\right| = 0$. Damit ist gezeigt, dass $(E_\mu^k)_{k\in\mathbb{N}}$ eine Nullfolge ist und somit auch die Konvergenz.

Mit (6.13) gilt mit der von der Vektornorm $\|\cdot\|$ induzierten Operatornorm

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

und damit die behauptete Fehlerabschätzung. ■

Um die Konvergenz der Verfahren zu untersuchen, muss der Spektralradius der Iterationsmatrix bestimmt oder zumindest abgeschätzt werden. Während man diesen in einigen Spezialfällen exakt angeben und somit Konvergenzaussagen treffen kann, so ist dies im allgemeinen Fall analytisch nicht möglich und numerisch sehr aufwendig. Es ist daher das Ziel der nächsten Abschnitte, aus einfachen algebraischen Eigenschaften der Matrix A auf die Konvergenz der Verfahren zu schließen.

6.1.2 Konvergenz des Richardson-Verfahrens

Satz 6.1.3 — Konvergenz des Richardson-Verfahrens. Es sei $A \in \mathbb{R}^{n\times n}$ regulär mit $\sigma(A) = [\lambda_{\min}, \lambda_{\max}] \subset \mathbb{R}^+$, dann gilt:

(a) Für die Iterationsmatrix $C_\mathsf{R}(\gamma) := I - \gamma^{-1}A$ des Richardson-Verfahrens gilt

$$\varrho(C_\mathsf{R}(\gamma)) = \max\left\{|1-\gamma^{-1}\lambda_{\max}|,\, |1-\gamma^{-1}\lambda_{\min}|\right\} \qquad \forall\,\gamma \in \mathbb{R}\setminus\{0\}\,.$$

(b) Das Richardson-Verfahren konvergiert genau dann, wenn $\gamma > \frac{\lambda_{\max}}{2}$ gilt.

(c) Der Spektralradius $\varrho(C_\mathsf{R}(\gamma))$ wird minimiert für $\gamma^* := \frac{\lambda_{\max}+\lambda_{\min}}{2}$ und es gilt $\varrho(C_\mathsf{R}(\gamma^*)) = \frac{\lambda_{\max}-\lambda_{\min}}{\lambda_{\max}+\lambda_{\min}}$.

Beweis. Für die Eigenwerte von A gilt nach Voraussetzung $0 < \lambda_{\min} = \lambda_1(A) \le \dots \le \lambda_n(A) = \lambda_{\max}$. Und damit $\lambda_i(\gamma) := \lambda_i(C_\mathsf{R}(\gamma)) = 1 - \gamma^{-1}\lambda_i(A)$ für $i = 1, \dots, n$. Damit gilt

$$\lambda_n(\gamma) = 1 - \gamma^{-1}\lambda_{\max} \le \lambda_i(\gamma) \le \lambda_1(\gamma) = 1 - \gamma^{-1}\lambda_{\min},$$

woraus (a) folgt. Zu (b): Das Richardson-Verfahren sei konvergent. Dann gilt nach (a) $-1 < -\varrho(C_\mathsf{R}(\gamma)) \le -|1-\gamma^{-1}\lambda_{\max}| \le 1-\gamma^{-1}\lambda_{\max}$, also $\gamma > \frac{\lambda_{\max}}{2}$, also $0 < \gamma < \frac{\lambda_{\max}}{2}$. Umgekehrt, sei $\gamma > \frac{\lambda_{\max}}{2}$, dann gilt $-1 < 1-\gamma^{-1}\lambda_{\max} \le 1-\gamma^{-1}\lambda_{\min} < 1$, da $\sigma(A) \subset \mathbb{R}^+$ nach Voraussetzung. Nach (a) folgt damit $\varrho(C_\mathsf{R}(\gamma)) < 1$.

Um (c) zu zeigen, gilt zunächst

$$\varrho(C_\mathsf{R}(\gamma^*)) = 1 - (\gamma^*)^{-1}\lambda_{\min} = (\gamma^*)^{-1}\lambda_{\max} - 1 = 1 - \frac{2\,\lambda_{\min}}{\lambda_{\max}+\lambda_{\min}} = \frac{\lambda_{\max}-\lambda_{\min}}{\lambda_{\max}+\lambda_{\min}} =: \varrho^*.$$

Der Spektralradius wird minimal, wenn $\gamma^{-1}\lambda_{\max} - 1 = 1 - \gamma^{-1}\lambda_{\min}$, was man wie folgt sieht (vgl. dazu auch Abbildung 6.3): Für $\frac{\lambda_{\max}}{2} < \gamma \le \gamma^*$ gilt $\max\left\{|1-\gamma^{-1}\lambda_{\max}|, |1-\gamma^{-1}\lambda_{\min}|\right\} = |1-\gamma^{-1}\lambda_{\max}| = \gamma^{-1}\lambda_{\min} - 1 \ge \varrho^*$. Für $\gamma \ge \gamma^*$ gilt $\max\left\{|1-\gamma^{-1}\lambda_{\max}|, |1-\gamma^{-1}\lambda_{\min}|\right\} = |1-\gamma^{-1}\lambda_{\min}| = 1-\gamma^{-1}\lambda_{\min} \ge \varrho^*$, woraus (c) folgt. ■

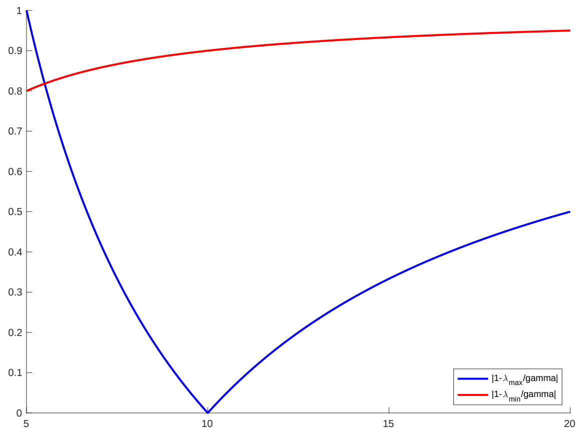


Abbildung 6.3: Spektralradius $\max\left\{|1-\gamma^{-1}\lambda_{\max}|, |1-\gamma^{-1}\lambda_{\min}|\right\}$ in Abhängigkeit von γ für $\lambda_{\min} = 1$ und $\lambda_{\max} = 10$.

Wir haben hier angenommen, dass $\sigma(A) \subset \mathbb{R}^+$, also dass A nur positive Eigenwerte besitzt; Symmetrie haben wir nicht vorausgesetzt.

6.1.3 Konvergenz des Jacobi-Verfahrens

Nach (6.11) lautet die Iterationsvorschrift des Jacobi-Verfahrens

$$x^{(i+1)} = D^{-1}(b - (L+R)x^{(i)}) = -D^{-1}(L+R)x^{(i)} + D^{-1}b,$$

also ist die Iterationsmatrix gegeben durch

$$C_{\mathrm{J}} := I - D^{-1}A = -D^{-1}(L+R).$$

Bei diesem Verfahren werden alle Komponenten des Lösungsvektors in einem Iterationsschritt gleichzeitig korrigiert, weswegen das Verfahren auch auch **Gesamtschrittverfahren** genannt wird.

Satz 6.1.4 — Konvergenz des Jacobi.Verfahrens. Es sei $A \in \mathbb{C}^{n\times n}$. Das Jacobi-Verfahren konvergiert für jeden Startvektor $x^{(0)} \in \mathbb{C}^n$, wenn die Matrix A das

i) starke Zeilensummenkriterium

$$|a_{ii}| > \sum_{\substack{k=1\\k\neq i}}^{n} |a_{ik}|, \quad \text{für } i = 1,2,\dots,n, \tag{6.17}$$

erfüllt, d.h., wenn A strikt diagonaldominant ist;

ii) oder das starke Spaltensummenkriterium

$$|a_{kk}| > \sum_{\substack{i=1\\i\neq k}}^{n} |a_{ik}|, \quad \text{für} \;\; k = 1, 2, \dots, n\,, \tag{6.18}$$

erfüllt, d.h., falls A^T strikt diagonaldominant ist.

Für den Beweis von Satz 6.1.4 benötigen wir das folgende Lemma.

Lemma 6.1.5 Es sei $A \in \mathbb{C}^{n\times n}$. Dann gilt für jede natürliche p-Matrixnorm $\varrho(A) \le \|A\|_p$.

Beweis. Jeder Eigenwert λ von A mit zugehörigem Eigenvektor x genügt für jede p-Matrixnorm $\|\cdot\|_p$ der Beziehung

$$\frac{\|Ax\|_p}{\|x\|_p} = |\lambda| \tag{6.19}$$

und damit der Abschätzung $\|A\|_p \ge |\lambda|$. ∎

Beweis von Satz 6.1.4. i) Wenn das starke Zeilensummenkriterium erfüllt ist, gilt für $C_\mathrm{J} = -D^{-1}(L+R)$ die Abschätzung

$$\|C_\mathrm{J}\|_\infty = \left\| \begin{pmatrix} 0 & -\frac{a_{1,2}}{a_{1,1}} & \dots & \dots & -\frac{a_{1,n}}{a_{1,1}} \\ -\frac{a_{2,1}}{a_{2,2}} & 0 & -\frac{a_{2,3}}{a_{2,2}} & & -\frac{a_{2,n}}{a_{2,2}} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \ddots & \vdots \\ -\frac{a_{n,1}}{a_{n,n}} & \dots & \dots & -\frac{a_{n,n-1}}{a_{nn}} & 0 \end{pmatrix} \right\|_\infty = \max_{1\le i\le n} \sum_{\substack{j=1\\j\neq i}}^{n} \frac{|a_{ij}|}{|a_{ii}|} < 1, \tag{6.20}$$

und Lemma 6.1.5 liefert dann die Behauptung.

ii) Ist für A das starke Spaltensummenkriterium (ii) erfüllt, so gilt i) für A^T. Also konvergiert das Jacobi-Verfahren für A^T und es ist daher wegen Satz 6.1.2 $\varrho(C_\mathrm{J}^T) < 1$ für $C_\mathrm{J}^T = I - D^{-1}A^T$. Nun hat C_J die gleichen Eigenwerte wie C_J^T. Also ist auch $\varrho(C_\mathrm{J}) < 1$, d.h., das Jacobi-Verfahren ist auch für die Matrix A konvergent. ∎

Die starken Zeilen- bzw. Spaltensummenkriterien sind oft zu einschränkend, sie sind z.B. für die Standardmatrix nicht erfüllt. Daher sucht man nach schwächeren Kriterien.

Definition 6.1.6 Eine Matrix $A \in \mathbb{R}^{n\times n}$ heißt **zerlegbar (reduzibel)**, wenn es nichtleere Teilmengen N_1 und N_2 der Indexmenge $N := \{1, 2, \dots, n\}$ gibt mit den Eigenschaften

i) $N_1 \cap N_2 = \varnothing$;
ii) $N_1 \cup N_2 = N$;
iii) $a_{ij} = 0$ für alle $i \in N_1$ und $j \in N_2$.

Eine Matrix, die nicht zerlegbar ist, heißt unzerlegbar (irreduzibel).

Bemerkung 6.1.7 Wir vergleichen Definition 6.1.6 mit Definition 5.1.11 (S. 71). Der Unterschied besteht darin, dass wir in Definition 5.1.11 noch eine Permutation der Indizes zugelassen haben. Wollte man das hier auch tun, müsste man das LGS $Ax = b$ transformieren, z.B. mittels Pivotisierung, zu $P^T AP\bar{x} = P^T b$ mit $\bar{x} = Px$.

■ **Beispiel 6.1.8** Wir geben zwei Beispiele.

i) Betrachte

$$A = \begin{pmatrix} a_{11} & \dots & a_{1k} & 0 & \dots & 0 \\ \vdots & & \vdots & \vdots & & \vdots \\ a_{N1} & \dots & a_{NN} & 0 & \dots & 0 \\ a_{N+1,1} & \dots & a_{N+1,N} & a_{N+1,N+1} & \dots & a_{N+1,2N} \\ \vdots & & \vdots & \vdots & & \vdots \\ a_{2N,1} & \dots & a_{2N,N} & a_{2N,N+1} & \dots & a_{2N,2N} \end{pmatrix}.$$

Die Teilmengen $N_1 = \{1, 2, ..., N\}, N_2 = \{N+1, ..., 2N\}$ haben alle in der Definition geforderten Eigenschaften. Somit ist A zerlegbar.

ii) Eine Tridiagonalmatrix mit nicht verschwindenden Nebendiagonal- und Diagonalelementen ist unzerlegbar. ■

Definition 6.1.9 — Schwaches Zeilen- und Spaltensummenkriterium. Eine Matrix $A \in \mathbb{R}^{n\times n}$ erfüllt das **schwache Zeilensummenkriterium**, falls

$$\sum_{\substack{\nu=1\\ \nu\neq\mu}}^{n} |a_{\mu\nu}| \le |a_{\mu\mu}|$$

für alle Zeilen $\mu = 1, ..., n$ gilt (d.h. falls A diagonaldominant ist) und

$$\sum_{\substack{\nu=1\\ \nu\neq\mu_0}}^{n} |a_{\mu_0\nu}| < |a_{\mu_0\mu_0}|$$

für mindestens einen Index $\mu_0 \in \{1, ..., n\}$ erfüllt ist.
Eine Matrix $A \in \mathbb{R}^{n\times n}$ erfüllt das **schwache Spaltensummenkriterium**, wenn A^T das schwache Zeilensummenkriterium erfüllt.

Satz 6.1.10 — Konvergenz des Jacobi-Verfahrens. Es sei $A \in \mathbb{R}^{n\times n}$ eine irreduzible Matrix, die das schwache Zeilensummenkriterium erfüllt. Dann ist das Jacobi-Verfahren konvergent.

Zum Beweis von Satz 6.1.10 werden wir den Spektralradius der Iterationsmatrix abschätzen. Die wesentliche Beobachtung dabei ist, dass jede irreduzible Matrix, die das schwache Zeilensummenkriterium erfüllt, bereits regulär ist.

Lemma 6.1.11 Jede irreduzible Matrix $M \in \mathbb{R}^{n\times n}$, die das schwache Zeilensummenkriterium erfüllt, ist regulär und für deren Diagonalelemente gilt $m_{jj} \neq 0$, $j = 1, ..., n$.

Beweis. Angenommen, M wäre nicht regulär, d.h., es existiert ein $x \in \mathbb{K}^n \setminus \{0\}$ mit $Mx = 0$. Insbesondere folgt dann aus der Dreiecksungleichung

$$|m_{jj}|\,|x_j| \le \underbrace{\left|\sum_{\ell=0}^{n} m_{j\ell}\, x_\ell\right|}_{=0} + \left|\sum_{\substack{\ell=1\\ \ell\neq j}}^{n} m_{j\ell}\, x_\ell\right| \le \sum_{\substack{\ell=1\\ \ell\neq j}}^{n} |m_{j\ell}|\,|x_\ell| \quad \text{für alle } \; j = 1, ..., n. \tag{6.21}$$

Wir definieren die Indexmengen $J := \{j = 1, ..., n : |x_j| = \|x\|_\infty\}$ und $K := \{k = 1, ..., n : |x^{(k)}| < \|x\|_\infty\}$. Offensichtlich gilt $J \cap K = \emptyset$, $J \cup K = \{1, ..., n\}$ und $J \neq \emptyset$. Wäre $K = \emptyset$, so könnte man in (6.21) die x_j- und x_ℓ-Terme heraus kürzen und erhielte einen Widerspruch zum schwachen Zeilensummenkriterium. Also gilt auch $K \neq \emptyset$, und aufgrund der Irreduzibilität von M existieren Indizes $j \in J$ und $k \in K$ mit $m_{jk} \neq 0$. Mit diesem ergibt sich

$$|m_{jj}| \leq \sum_{\substack{\ell=1\\ \ell\neq j}}^{n} |m_{j\ell}| \frac{|x_\ell|}{|x_j|} \leq \sum_{\substack{\ell=1\\ \ell\neq j}}^{n} |m_{j\ell}|,$$

denn der Quotient ist stets ≤ 1 wegen $|x_j| = \|x\|_\infty$ und er ist < 1 für $\ell = k \in K$. Also erhalten wir einen Widerspruch zum schwachen Zeilensummenkriterium von M, d.h. M ist regulär.

Gäbe es schließlich ein triviales Diagonalelement $m_{jj} = 0$, so folgte aus dem schwachen Zeilensummenkriterium, dass bereits die j-te Zeile die Nullzeile wäre. Da M regulär ist, folgt insbesondere $m_{jj} \neq 0$ für alle $j = 1, ..., n$. ■

Beweis von Satz 6.1.10. Wegen $a_{jj} \neq 0$ für alle $j = 1, ..., n$ ist $C_\mathrm{J} = -D^{-1}(A - D)$ wohldefiniert. Um $\varrho(C_\mathrm{J}) < 1$ zu zeigen, beweisen wir, dass $M := C_\mathrm{J} - \lambda I$ für $\lambda \in \mathbb{C}$ mit $|\lambda| \geq 1$ regulär ist. Da A irreduzibel ist, ist auch $A - D$ irreduzibel, denn es wird lediglich die Diagonale verändert. Die Matrix C_J entsteht durch zeilenweise Multiplikation von $A - D$ mit Werten $\neq 0$. Deshalb ist auch C_J irreduzibel. Da M und C_J sich nur auf der Diagonale unterscheiden, ist M irreduzibel. Aufgrund des schwachen Zeilensummenkriteriums von A gilt

$$\sum_{\substack{k=1\\ k\neq j}}^{n} |m_{jk}| = \sum_{\substack{k=1\\ k\neq j}}^{n} |c_{jk}^{(J)}| = \sum_{\substack{k=1\\ k\neq j}}^{n} \frac{|a_{jk}|}{|a_{jj}|} \leq 1 \leq |\lambda| = |m_{jj}| \quad \text{für alle} \quad j = 1, ..., n,$$

und für mindestens einen Index j gilt diese Ungleichung strikt. Also erfüllt M auch das schwache Zeilensummenkriterium und ist nach Lemma 6.1.11 insgesamt regulär. ■

Bemerkung 6.1.12 Das Jacobi-Verfahren ist auch konvergent unter der Annahme, dass $A \in \mathbb{R}^{n\times n}$ irreduzibel ist und das schwache Spaltensummenkriterium erfüllt.

6.1.4 Konvergenz des Gauß-Seidel-Verfahrens

Für $B = L + D$ lautet die Iterationsvorschrift des Gauß-Seidel-Verfahrens nach (6.10)

$$x^{(i+1)} = (L + D)^{-1}(b - Rx^{(i)}) = -(L + D)^{-1}Rx^{(i)} + (L + D)^{-1}b,$$

also gilt für die Iterationsmatrix $C_\mathrm{GS} = -(L + D)^{-1}R$; analog mit (6.11) für $B = D + R$

$$x^{(i+1)} = (D + R)^{-1}(b - Lx^{(i)}) = -(D + R)^{-1}Lx^{(i)} + (D + R)^{-1}b,$$

und damit $\widetilde{C}_\mathrm{GS} = -(D + R)^{-1}L$. Bei diesem Verfahren wird jede Komponente einzeln verändert, was der Grund für den Namen **Einzelschrittverfahren** ist. Das Verfahren ist besonders geeignet für die Parallelisierung, was z.B. in der Vorlesung *High Perfomance Computing II* näher beschrieben wird.

Satz 6.1.13 — Konvergenz des Gauß-Seidel-Verfahrens. Es sei $A \in \mathbb{R}^{n\times n}$ eine reguläre Matrix, die das starke Zeilensummenkriterium oder das starke Spaltensummenkriterium erfüllt. Dann sind beide Varianten des Gauß-Seidel-Verfahrens zur Lösung des linearen Gleichungssystems $Ax = b$ konvergent.

Beweis. Sei das starke Zeilensummenkriterium erfüllt. Dann gilt nach Satz 6.1.4

$$\|C_{\mathsf{J}}\|_\infty = \max_{1\le i\le n} \sum_{\substack{j=1\\ j\ne i}}^{n} \frac{|a_{ij}|}{|a_{ii}|} < 1\,. \tag{6.22}$$

Es sei jetzt $y \in \mathbb{R}^n$ beliebig und $z = C_{\mathsf{GS}}\, y$, also $-(L+D)z = Ry$, d.h. für alle $1 \le i \le n$ gilt

$$-\sum_{j=1}^{i} a_{ij}\, z_j = \sum_{j=i+1}^{n} a_{ij} y_j\,. \tag{6.23}$$

Durch vollständige Induktion beweisen wir nun die Abschätzung

$$|z_i| \le \|y\|_\infty \sum_{\substack{j=1\\ j\ne i}}^{n} \frac{|a_{ij}|}{|a_{ii}|}\,. \tag{6.24}$$

Für $i = 1$ gilt mit (6.23)

$$|z_1| \le \sum_{j=2}^{n} \frac{|a_{1j}|}{|a_{11}|} |y_j| \le \|y\|_\infty \sum_{j=2}^{n} \frac{|a_{1j}|}{|a_{11}|}, \tag{6.25}$$

also der Induktionsanfang. Nun nehmen wir an, dass (6.24) für $1 \le i-1 < n$ bewiesen ist (die Induktionshypothese (IH)) und schließen mit (6.23)

$$\begin{aligned}
|z_i| &\le \frac{1}{|a_{ii}|}\left(\sum_{j=1}^{i-1} |a_{ij}|\,|z_j| + \sum_{j=i+1}^{n} |a_{ij}|\,|y_j|\right)\\
&\overset{\text{(IH)}}{\le} \|y\|_\infty \underbrace{\left(\max_{1\le j\le i-1} \sum_{\substack{k=1\\ k\ne j}} \frac{|a_{jk}|}{|a_{jj}|}\right)}_{\le \|C_{\mathsf{J}}\|_\infty < 1} \sum_{j=1}^{i-1} \frac{|a_{ij}|}{|a_{ii}|} + \|y\|_\infty \sum_{j=i+1}^{n} \frac{|a_{ij}|}{|a_{ii}|}\\
&\le \|y\|_\infty \sum_{\substack{j=1\\ j\ne i}}^{n} \frac{|a_{ij}|}{|a_{ii}|},
\end{aligned} \tag{6.26}$$

also ist (6.24) bewiesen. Hiermit erhält man die Abschätzung $\|C_{\mathsf{GS}}\, y\|_\infty = \|z\|_\infty \le \|C_{\mathsf{J}}\|_\infty \|y\|_\infty$ für alle $y \in \mathbb{R}^n$ und somit

$$\|C_{\mathsf{GS}}\|_\infty \le \|C_{\mathsf{J}}\|_\infty < 1\,. \tag{6.27}$$

Da $\varrho(-(L+D)^{-1}R) = \varrho(C_{\mathsf{GS}}) \le \|C_{\mathsf{GS}}\|_\infty$, folgt daraus die Konvergenz des Gauß-Seidel-Verfahrens für $B = L + D$.

Um die Konvergenz des Gauß-Seidel-Verfahrens für $B = D + R$ mit $\widetilde{C}_{\mathsf{GS}} = -(D+R)^{-1}L$

nachzuweisen, betrachten wir zunächst folgende Permutationsmatrix

$$P = \begin{pmatrix} 0 & & 1 \\ & \iddots & \\ 1 & & 0 \end{pmatrix} = P^T \in \mathbb{R}^{n\times n}.$$

Die Matrix $\widetilde{A} = PAP^T$ geht aus A durch simultane Zeilen- und Spaltenvertauschungen hervor, sodass die Gültigkeit des starken Zeilensummenkriteriums auch für $\widetilde{A}$ vorliegt. Mit dem oben Bewiesenen gilt somit $\varrho(-(\widetilde{L} + \widetilde{D})^{-1}\widetilde{R}) < 1$, wobei

$$\widetilde{L} = PRP^T, \qquad \widetilde{D} = PDP^T, \qquad \widetilde{R} = PLP^T,$$

und daher

$$\begin{aligned} 1 &> \varrho(-(PRP^T + PDP^T)^{-1}PLP^T) = \varrho(-(P(R+D)P^T)^{-1}PLP^T) \\ &= \varrho(-P(R+D)^{-1}P^TPLP^T) = \varrho(-P(R+D)^{-1}LP^T) \\ &= \varrho(-(R+D)^{-1}L)\,. \end{aligned}$$

Also ist $\varrho(\widetilde{C}_{\mathrm{GS}}) = \varrho(-(D+R)^{-1}L) < 1$ und somit das Gauß-Seidel-Verfahren für $B = D + R$ konvergent.

Sei nun das starke Spaltensummenkriterium erfüllt. Dann erfüllt A^T das starke Zeilensummenkriterium und beide Varianten des Gauß-Seidel-Verfahrens konvergieren für A^T. Mit der Standardzerlegung von A^T

$$A^T = L_T + D_T + R_T\,,$$

wobei $L_T = R^T$, $D_T = D$ und $R_T = L^T$ ist, gilt somit

$$\begin{aligned} &\varrho(-(R^T + D)^{-1}L^T) = \varrho(-(L_T + D_T)^{-1}R_T) < 1\,, \\ &\varrho(-(L^T + D)^{-1}R^T) = \varrho(-(R_T + D_T)^{-1}L_T) < 1\,. \end{aligned}$$

Hieraus ergibt sich die Konvergenz des Gauß-Seidel-Verfahrens für $B = L + D$

$$\begin{aligned} \varrho(C_{\mathrm{GS}}) &= \varrho(-(L+D)^{-1}R) = \varrho(-(L+D)(L+D)^{-1}R\,(L+D)^{-1}) \\ &= \varrho(-R\,(L+D)^{-1}) = \varrho(-(L^T+D)^{-1}R^T) < 1 \end{aligned}$$

sowie für $B = D + R$

$$\begin{aligned} \varrho(\widetilde{C}_{\mathrm{GS}}) &= \varrho(-(D+R)^{-1}L) = \varrho(-(D+R)(D+R)^{-1}L\,(D+R)^{-1}) \\ &= \varrho(-L\,(D+R)^{-1}) = \varrho(-(D+R^T)^{-1}L^T) < 1\,. \end{aligned}$$

Damit sind alle Aussagen des Satzes bewiesen. ∎

Bemerkung 6.1.14 Häufig verleitet (6.27) zu der **falschen Schlussfolgerung**, dass das Gauß-Seidel-Verfahren schneller als das Jacobi-Verfahren konvergieren würde, wenn die Matrix strikt diagonaldominant ist. Dies ist nicht der Fall, wie das folgende Beispiel zeigt.

■ **Beispiel 6.1.15** Dass bei strikter Diagonaldominanz einer regulären Matrix $A \in \mathbb{R}^{n\times n}$ aus $\|C_{\mathsf{GS}}\|_\infty \le \|C_{\mathsf{J}}\|_\infty < 1$ **nicht** $\varrho(C_{\mathsf{GS}}) \le \varrho(C_J)$ folgen muss, sieht man, wenn man die Matrix A folgendermaßen wählt:

$$A = \begin{pmatrix} 50 & -10 & -20 \\ -20 & 49 & -20 \\ 20 & -10 & 49 \end{pmatrix}.$$

Dann besitzen die zugehörigen Iterationsmatrizen

$$C_{\mathsf{GS}} = -(L+D)^{-1}R = \frac{1}{12005}\begin{pmatrix} 0 & 2401 & 4802 \\ 0 & 980 & 6860 \\ 0 & -780 & -560 \end{pmatrix}, \qquad C_J = \frac{1}{245}\begin{pmatrix} 0 & 49 & 98 \\ 100 & 0 & 100 \\ -100 & 50 & 0 \end{pmatrix}$$

nämlich die Spektralradien $\varrho(C_{\mathsf{GS}}) = (4\sqrt{5})/49 \approx 0.1825$ und $\varrho(C_J) = 2/49 \approx 0.04082$ sowie die Maximumnormen $\|C_{\mathsf{GS}}\|_\infty = 32/49 \approx 0.6531$ und $\|C_J\|_\infty = 40/49 \approx 0.8163$. ■

Bemerkung 6.1.16 Ebenfalls falsch wäre die Schlussfolgerung aus Satz 6.1.13, dass eine Form des Gauß-Seidel-Verfahrens genau dann konvergent ist, wenn es die andere ist. Es gibt Beispiele regulärer Matrizen, für die $\varrho(C_{\mathsf{GS}}) < 1$, aber $\varrho(\widetilde{C}_{\mathsf{GS}}) \ge 1$ bzw. $\varrho(\widetilde{C}_{\mathsf{GS}}) < 1$, aber $\varrho(C_{\mathsf{GS}}) \ge 1$.

Man betrachte dazu folgendes Beispiel.

■ **Beispiel 6.1.17** Gegeben sei die reguläre Matrix

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

Die zugehörigen Iterationsmatrizen

$$C_{\mathsf{GS}} = -(L+D)^{-1}R = \begin{pmatrix} 0 & 0 & -1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \quad \widetilde{C}_{\mathsf{GS}} = -(D+R)^{-1}L = \begin{pmatrix} 0 & -2 & 0 \\ -1 & -2 & 0 \\ 0 & 2 & 0 \end{pmatrix}$$

besitzen die Spektralradien $\varrho(C_{\mathsf{GS}}) = 0 < 1$ sowie $\varrho(\widetilde{C}_{\mathsf{GS}}) = 1 + \sqrt{3} > 1$, d.h. das Gauß-Seidel-Verfahren für $B = L + D$ ist konvergent, für $B = D + R$ jedoch divergent. ■

Wiederum können wir Konvergenz auch unter schwächeren Voraussetzungen sichern.

Satz 6.1.18 — Konvergenz des Gauß-Seidel-Verfahrens. Ist $A \in \mathbb{R}^{n\times n}$ irreduzibel und erfüllt das schwache Zeilensummenkriterium, so sind beide Varianten des Gauß-Seidel-Verfahrens konvergent.

Beweis. Die Wohldefiniertheit von $C_{\mathsf{GS}} = -(L+D)^{-1}R$ und $\widetilde{C}_{\mathsf{GS}} = -(D+R)^{-1}L$ ist wieder klar. Wir betrachten $W := C_{\mathsf{GS}} - \lambda I$ sowie $\widetilde{W} := \widetilde{C}_{\mathsf{GS}} - \lambda I$ für $\lambda \in \mathbb{C}$ mit $|\lambda| \ge 1$. Durch

Multiplikation mit $-(L+D)$ sieht man, dass W genau dann regulär ist, wenn $M := R+\lambda L+\lambda D$ regulär ist. Analog folgt durch Multiplikation mit $-(D+R)$, dass $\widetilde{W}$ genau dann regulär ist, wenn $\widetilde{M} := L+\lambda D+\lambda R$ es ist. Offensichtlich erben M und $\widetilde{M}$ die Irreduzibilität von $A = D+L+R$. Ferner erfüllen M und $\widetilde{M}$ das schwache Zeilensummenkriterium, denn

$$\sum_{\substack{k=1\\k\neq j}}^{n} |m_{jk}| = |\lambda| \sum_{k=1}^{j-1} |a_{jk}| + \sum_{k=j+1}^{n} |a_{jk}| \leq |\lambda| \sum_{\substack{k=1\\k\neq j}}^{n} |a_{jk}| \leq |\lambda|\,|a_{jj}| = |m_{jj}|$$

sowie

$$\sum_{\substack{k=1\\k\neq j}}^{n} |\widetilde{m}_{jk}| = \sum_{k=1}^{j-1} |a_{jk}| + |\lambda| \sum_{k=j+1}^{n} |a_{jk}| \leq |\lambda| \sum_{\substack{k=1\\k\neq j}}^{n} |a_{jk}| \leq |\lambda|\,|a_{jj}| = |\widetilde{m}_{jj}|$$

für $j = 1, \ldots, n$ mit strikter Ungleichung jeweils für mindestens einen Index j. Nach Lemma 6.1.11 sind M und $\widetilde{M}$ regulär. Insgesamt erhalten wir $\varrho(C_{\mathrm{GS}}) < 1$ und $\varrho(\widetilde{C}_{\mathrm{GS}}) < 1$. ■

Bemerkung 6.1.19 (i) Man kann zeigen (Übung), dass beide Formen des Gauß-Seidel-Verfahrens auch konvergent sind unter der Annahme, dass $A \in \mathbb{R}^{n\times n}$ irreduzibel ist und das schwache Spaltensummenkriterium erfüllt ist.
(ii) Das Gauß-Seidel-Verfahren konvergiert für symmetrisch positiv definite Matrizen.

Bemerkung 6.1.20 i) Die obigen Iterationsverfahren ließen sich in der Form

$$x^{(k+1)} = B^{-1}(B-A)x^{(k)} + B^{-1}b = Cx^{(k)} + d$$

schreiben. Da die Iterationsmatrix C für alle k konstant ist, spricht man von **stationären Iterationsverfahren**.

ii) Das quantitative Verhalten solch stationärer Verfahren lässt sich durch die Einführung eines (Relaxations-)Parameters ω verbessern:

$$x^{(k+1)} = \omega(Cx^{(k)} + d) + (1-\omega)x^{(k)}\,.$$

Für $0 < \omega < 1$ spricht man von einem **Unterrelaxationsverfahren** und für $\omega > 1$ von einem **Überrelaxationsverfahren**. Man kann zeigen, dass der optimale Parameter für eine positiv definite Matrix A beim gedämpften Jacobi-Verfahren

$$\omega_{\mathrm{opt}} = \frac{2}{\lambda_{\min}(D^{-1}A) + \lambda_{\max}(D^{-1}A)}$$

lautet und für das überrelaxierte Gauß-Seidel-Verfahren (SOR = **s**uccessive **o**ver-**r**elaxation method) angewandt auf eine positiv definite Matrix $A = L+D+L^T$ ergibt sich der optimale Parameter zu

$$\omega_{\mathrm{opt}} = \frac{2}{1+\sqrt{\lambda_{\min}(D^{-1}A) + \lambda_{\max}((D+2L)D^{-1}(D+2L^T)A^{-1})}}\,.$$

iii) Die Bedeutung der oben genannten Iterationsverfahren liegt heute weniger in deren unmittelbarem Einsatz zur Lösung von $Ax = b$, sondern auf Grund deren „**Glättungseigenschaften**" als Beschleuniger anderer moderner Verfahren, insbesondere des *Mehrgitter-Verfahrens*. Mehr dazu in der Vorlesung *High Performance Computing II*.

6.1.5 Symmetrisches Gauß-Seidel-Verfahren

Besonders für die oben erwähnten Mehrgitter-Verfahren und als Vorkonditionierer im cg-Verfahren (siehe §6.5 und 6.6) sind Iterationsmatrizen B von Interesse, die symmetrisch sind.

Definition 6.1.21 Eine Splitting-Verfahren mit Iterationsmatrix $C = I - B^{-1}A$ für A s.p.d. heißt *symmetrisch*, falls B ebenfalls s.p.d. ist.

Dies ist für das Richardson- und das Jacobi-Verfahren der Fall, nicht aber für das Gauß-Seidel-Verfahren. Daher betrachtet man eine symmetrische Variante, die man durch folgende Iterationsvorschrift umsetzen kann:

$$\begin{aligned} x^{(k+1/2)} &= x^{(k)} - (D+L)^{-1}(Ax^{(k)} - b), \\ x^{(k+1)} &= x^{(k+1/2)} - (D+R)^{-1}(Ax^{(k+1/2)} - b). \end{aligned}$$

Hier sind also pro Iterationsschritt zwei gestaffelte Gleichungssysteme zu lösen. Die Iterationsmatrix lautet $C_{\text{SGS}} = (D+R)^{-1}L(D+L)^{-1}R = \widetilde{C}_{\text{GS}}\, C_{\text{GS}}$. Das Verfahren hat die Form $x^{(k+1)} = (I - B_{\text{SGS}}^{-1}A)x^{(k)} + B_{\text{SGS}}^{-1}b$.

Falls A s.p.d. ist, gilt $R = L^T$ und es ergibt sich $B_{\text{SGS}} = (D+L)^{-1}D(D+L)^{-T}$, so dass das symmetrische Gauß-Seidel-Verfahren im obigen Sinne symmetrisch ist. Man kann weiter zeigen, dass das symmetrische Gauß-Seidel-Verfahren für symmetrisch positiv definite Matrizen konvergiert.

6.1.6 Abbruch-Kriterien

Da ein Iterationsverfahren aufeinanderfolgende Näherungen der Lösung liefert, ist ein praktischer Test notwendig, um das Verfahren zu stoppen, wenn die gewonnene Approximation genau genug ist. Da es nicht möglich ist, den Fehler $x^* - x^{(k)}$, d.h. den Abstand zur eigentlichen (gesuchten) Lösung, zu bestimmen, müssen andere Größen gewonnen werden, die meist auf dem Residuum $r = b - Ax$ basieren.

Die hier vorgestellten Verfahren liefern eine Folge $(x^{(i)})_{i=1,2,\ldots}$ von Vektoren die (hoffentlich) gegen den Vektor x^* streben, welcher Lösung des linearen Gleichungssystems $Ax = b$ ist, also $x^* = A^{-1}b$. Damit das Verfahren effizient ist, müssen wir es geeignet abbrechen. Eine gute Methode sollte

i) feststellen, ob der Fehler $e^{(i)} := x^* - x^{(i)}$ klein genug ist,
ii) abbrechen, falls der Fehler nicht weiter kleiner wird oder nur noch sehr langsam, und
iii) den maximalen Aufwand, der zur Iteration verwendet wird, beschränken.

Das folgende Abbruchkriterium ist eine einfache, aber häufig genügende Variante. Man muss hierzu die Größen `maxit`, $\|b\|$, `tol` und wenn möglich auch $\|A\|$ (und $\|A^{-1}\|$) (oder zumindest gute Abschätzungen) zur Verfügung stellen. Dabei ist

- `maxit` $\in \mathbb{N}$ die a priori festgelegte maximale Anzahl an möglichen Iterationen des Verfahrens,

- $\|A\| \in \mathbb{R}$ eine Norm von A (jede einigermaßen „vernünftige“ Approximation des betragsgrößten Eintrags in A genügt schon),
- $\|b\| \in \mathbb{R}$ eine Norm von b (auch hier genügt eine einigermaßen „vernünftige“ Approximation des betragsgrößten Eintrags in b),
- `tol`$\in \mathbb{R}^+$ eine Schranke für die Größe des Residuums bzw. des Fehlers.

MATLAB-Beispiel: Beispiel eines Abbruchkriteriums

```
k = 0;
while 1
  k = k + 1;
  % Berechne die Approximation x^(k)
  % Berechne das Residuum r^(k) = b - A x^(k)
  % Berechne norm_ak = || A * x^(k) ||, norm_rk = || r^(k) ||
  % und norm_b = || b ||
  if (k >= maxit) | ( norm_rk <= tol * max( norm_ak, norm_b ) )
    break
  end
end
```

Da sich nach einigen Iterationen der Term $\|Ax^{(k)}\|$ oft nicht mehr groß ändert, braucht man diesen nicht immer neu zu bestimmen. Zu bestimmen ist

$$\|e^{(k)}\| = \|A^{-1}r^{(k)}\| \leq \|A^{-1}\|\,\|r^{(k)}\|.$$

Man beachte, dass man $\|A^{-1}B\|$ bei den bisherigen Verfahren mit Lemma 3.1.4 (Neumann-Reihe) abschätzen kann. Es gilt $x^{(k+1)} = B^{-1}(B-A)x^{(k)} + B^{-1}b = Cx^{(k)} + d$ und

$$B^{-1}A = I - B^{-1}(B-A) = I - C \quad \text{sowie} \quad \|A^{-1}B\| = \|(I-C)^{-1}\| \leq \frac{1}{1-\|C\|}.$$

Bemerkung 6.1.22 Man findet oft ein Abbruchkriterium der Art $\|x^{(k+1)} - x^{(k)}\| < \texttt{tol}$, man bricht also ab, wenn sich die Iterationen nicht mehr stark ändern. Es gibt zahlreiche Beispiele, die belegen, dass ein solches Kriterium in der Regel nicht zielführend ist – überlegen Sie sich, warum.

6.2 Gradientenverfahren

In diesem Abschnitt zeigen wir einen alternativen Ansatz, um iterative Methoden zur Lösung eines linearen Gleichungssystems $Ax = b$ zu konstruieren. Dabei nehmen wir an, dass A symmetrisch positiv definit (s.p.d.) ist, vgl. Definition 3.7.1. Wir betrachten die quadratische Funktion[4]

$$f : \mathbb{R}^n \to \mathbb{R}, \quad f(x) := \tfrac{1}{2}x^TAx - b^Tx\,. \tag{6.28}$$

[4] In der Sprache der Optimierung wird diese auch *Ziel-* oder *Kostenfunktion* genannt.

Der Gradient von f lautet $f'(x) = \nabla f(x) = \frac{1}{2}(A + A^T)x - b$. Da wir $A = A^T$ vorausgesetzt haben, gilt

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

Notwendig für ein Minimum x^* von f ist das Verschwinden des Gradienten $f'(x^*) = 0$,[5] also löst x^* das lineare Gleichungssystem $Ax = b$. Da die Hesse-Matrix $f''(x) = D^2 f(x) \equiv A$ positiv definit ist, liegt für die Lösung von $Ax = b$ tatsächlich ein Minimum vor. Damit haben wir folgende Aussagen bewiesen.

Lemma 6.2.1 Sei $A \in \mathbb{R}^{n\times n}$ s.p.d. Dann ist Lösung des Minimierungsproblems

$$f(x) = \min_{y\in\mathbb{R}^n} f(y)$$

äquivalent zur Lösung des linearen Gleichungssystems $Ax = b$. Mit anderen Worten: $x^* = A^{-1}b = \arg\min_{x\in\mathbb{R}^n} f(x)$. ■

Alternativer Beweis zu Lemma 6.2.1 . Ein zweiter Beweis folgt aus der Darstellung

$$f(x) = f(x^*) + \tfrac{1}{2}(x - x^*)^T A(x - x^*) \quad \text{mit } x^* := A^{-1}b. \tag{6.29}$$

Hieraus folgt $f(x) > f(x^*)$ für $x \neq x^*$, d.h. $x^* := A^{-1}b$ ist das eindeutige Minimum von f. ■

Man beachte, dass (6.29) ein Spezialfall der folgenden Entwicklung von f um einen beliebigen Wert $\tilde{x} \in \mathbb{R}^n$ ist, (welche sich leicht durch Ausmultiplizieren zeigen lässt)

$$f(x) = f(\tilde{x}) + \langle A\tilde{x} - b, x - \tilde{x}\rangle + \tfrac{1}{2}\langle A(x - \tilde{x}), x - \tilde{x}\rangle\,.$$

Zur Untersuchung der Konvergenz des Gradientenverfahren bietet sich die sogenannte *Energienorm* an.

Definition 6.2.2 Sei $A \in \mathbb{R}^{n\times n}$ s.p.d. Dann heißt $\langle x, y\rangle_A := x^T A y$, $x, y \in \mathbb{R}^n$, *Energie-Skalarprodukt* und $\|x\|_A := \sqrt{x^T A x}$ heißt *Energienorm*.

Lemma 6.2.3 Für die Funktion $g : \mathbb{R}^n \to \mathbb{R}$ definiert als $g(x) := f(x) + \frac{1}{2}\langle b, A^{-1}b\rangle$ gilt:

(i) Die Funktionen f und g besitzen die gleichen lokalen Minima.

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

Beweis. Teil (i) ist klar. Zu (ii):

$$\begin{aligned}
g(x) &= \tfrac{1}{2}\langle x, Ax\rangle - \langle b, x\rangle + \tfrac{1}{2}\langle b, A^{-1}b\rangle = \tfrac{1}{2}\langle x, Ax - b\rangle - \tfrac{1}{2}\langle b, x - A^{-1}b\rangle \\
&= \tfrac{1}{2}\langle x, A(x - x^*)\rangle - \tfrac{1}{2}\langle A^{-1}b, Ax - b\rangle = \tfrac{1}{2}\langle x, A(x - x^*)\rangle - \tfrac{1}{2}\langle x^*, A(x - x^*)\rangle \\
&= \tfrac{1}{2}\|x - x^*\|_A^2,
\end{aligned}$$

also die Behauptung. ■

[5] Man nennt dies auch einen *kritischen Punkt*.

Bemerkung 6.2.4 Da die Lösung eines linearen Gleichungssystems mit einer symmetrisch positiv definiten Matrizen äquivalent ist zur Lösung des zugehörigen Minimierungsproblems, kann man somit **Verfahren zur numerischen Optimierung** verwenden. Mehr dazu (auch für allgemeine Funktionen) in der Vorlesung *Numerische Optimierung*.

Das Gradientenverfahren gehört zur Klasse der Liniensuchverfahren. Diese bestehen aus zwei Schritten:

1. Bestimmung einer Such- oder Abstigesrichtung;
2. Minimierung entlang der Suchrichtung.

Da der **Gradient** die Richtung des **steilsten Anstiegs** ist, kann man $-\nabla f$ als Abstiegsrichtung wählen und entlang dieser Geraden ein Minimum suchen. Dies liefert eine erste Version eines allgemeinen Gradientenverfahrens, welche auch für beliebige Minimierungsprobleme (also auch solche, bei denen eine Nullstelle des Gradienten nicht durch ein LGS bestimmt ist) anwendbar ist (vgl. Vorlesung *Numerische Optimierung*).

Gradientenverfahren (allgemein):
Es sei $\Omega \subseteq \mathbb{R}^n$, $f : \Omega \to \mathbb{R}$, $x^{(0)} \in \Omega$ ein Startwert;.
For $k = 0, 1, 2, \ldots$ **do**
1) *Bestimmung der Abstiegsrichtung:* $d^{(k+1)} = -\nabla f(x^{(k)})$.
2) *Liniensuche:* suche auf der Geraden $\{x^{(k)} + \lambda d^{(k+1)} : \lambda \geq 0\} \cap \Omega$ ein Minimum, d.h. bestimme $\lambda^{(k+1)} \geq 0$ mit $f(x^{(k)} + \lambda^{(k+1)} d^{(k+1)}) \leq f(x^{(k)})$ und setze

$$x^{(k+1)} := x^{(k)} + \lambda^{(k+1)} d^{(k+1)} .$$

Daraus folgt $f(x^{(0)}) \geq f(x^{(1)}) \geq f(x^{(2)}) \geq \cdots$

Wie wir im Folgenden sehen werden, kann man im Falle der quadratischen Funktionen $f(x) = \frac{1}{2} x^T A x - b^T x$ und $\Omega = \mathbb{R}^n$ die jeweilige Lösung von 1) und 2) im o.g. Gradientenverfahren leicht exakt angeben. Zunächst zur Suchrichtung: Da $\nabla f(x) = Ax - b$, ergibt sich

$$d^{(k+1)} = -\nabla f(x^{(k)}) = b - Ax^{(k)}.$$

Nun zur Liniensuche, d.h. zur Bestimmung von λ_k: Sei $d \in \mathbb{R}^n \setminus \{0\}$ eine beliebige Suchrichtung und $F : \mathbb{R} \to \mathbb{R}$ mit $F(\lambda) := f(x + \lambda d)$, dann gilt

$$\begin{aligned}
F(\lambda) &= f(x + \lambda d) \\
&= \frac{1}{2} \langle x + \lambda d, A(x + \lambda d) \rangle - \langle b, x + \lambda d \rangle \\
&= \frac{1}{2} \langle x, Ax \rangle - \langle b, x \rangle + \lambda \langle d, Ax - b \rangle + \frac{1}{2} \lambda^2 \langle d, Ad \rangle \\
&= f(x) + \lambda \langle d, Ax - b \rangle + \frac{1}{2} \lambda^2 \langle d, Ad \rangle.
\end{aligned} \tag{6.30}$$

Da $d \neq 0$ und A positiv definit nach Voraussetzung ist, folgt $\langle d, Ad \rangle > 0$. Die Funktion F ist also quadratisch in λ mit positivem führenden Koeffizienten, es existiert also ein eindeutig bestimmtes Minimum. Damit folgt aus

$$0 \stackrel{!}{=} F'(\lambda) = \langle d, Ax - b\rangle + \lambda\langle d, Ad\rangle\,, \tag{6.31}$$

dass der Parameter

$$\lambda_{\text{opt}}(x,d) := \frac{\langle d, b - Ax\rangle}{\langle d, Ad\rangle}\,. \tag{6.32}$$

zu gegebenem Vektor $d \in \mathbb{R}^n \setminus \{0\}$ die Funktion $F(\lambda) := f(x + \lambda d)$ minimiert.

Bemerkung 6.2.5 Für beliebige Zielfunktionen f muss man den Gradienten entweder numerisch approximieren und die Liniensuche mittels sogenannter Schrittweitenregeln näherungsweise durchführen, vgl. Vorlesung *Numerische Optimierung*.

Nun formulieren wir nun das Gradientenverfahren zu quadratischen f mit der optimalen Schrittweite (6.32) in folgendem Algorithmus. Beachte dabei, dass für $k = 0, 1, \ldots$ gilt

$$r^{(k+1)} = b - Ax^{(k+1)} = b - A(x^{(k)} + \lambda_{\text{opt}}^{(k+1)}\, d^{(k+1)}) = r^{(k)} - \lambda_{\text{opt}}\, Ad^{(k+1)},$$

das Residuum kann also rekursiv ohne weitere Matrix-Vektor-Multiplikation berechnet werden.

Algorithmus 6.2.6 — Gradientenverfahren zur Lösung des LGS $Ax = b$.

Input: Startwert $x^{(0)} \in \mathbb{R}^n$

1: $r^{(0)} := b - Ax^{(0)}$
2: **for** $k = 0, 1, 2, \ldots$ **do**
3: $\quad a^{(k)} := Ar^{(k)}$
4: $\quad \lambda_{\text{opt}}^{(k+1)} := \langle r^{(k)}, r^{(k)}\rangle \,/\, \langle r^{(k)}, a^{(k)}\rangle$
5: $\quad x^{(k+1)} := x^{(k)} + \lambda_{\text{opt}}^{(k+1)}\, r^{(k)}$
6: $\quad r^{(k+1)} := r^{(k)} - \lambda_{\text{opt}}^{(k+1)}\, a^{(k)}$
7: **end for**

Bemerkung 6.2.7 Aus der Linearen Algebra (vgl. auch (A.6)) ist bekannt, dass alle Normen auf dem $\mathbb{R}^n$ äquivalent sind. Man könnte also meinen, dass es keine Rolle spielt, welche Norm wir hier verwenden. Beachte aber, dass die Äquivalenzkonstanten von der Raumdimension n abhängen, also von der Größe des LGS. Daher ist es in der Numerik sehr wohl relevant, in welcher Norm die Fehlerabschätzungen vorgenommen werden. Für die hier verwendeten Normen gilt

$$\lambda_{\min}(A)^{1/2}\,\|x\| \le \|x\|_A \le \lambda_{\max}(A)^{1/2}\,\|x\| \qquad \text{für alle } x \in \mathbb{R}^n. \tag{6.33}$$

Satz 6.2.8 Es sei $A \in \mathbb{R}^{n\times n}$ s.p.d.. dann gilt für das Gradientenverfahren

$$\|x^{(k)} - x^*\|_A \le \left(\frac{\kappa - 1}{\kappa + 1}\right)^k \|x^{(0)} - x^*\|_A$$

Beweis. (i) Zunächst halten wir fest, dass $x^{(k+1)} = \arg\min_{\lambda\in\mathbb{R}} f(x^{(k)} + \lambda r^{(k)})$ per Konstruktion des Verfahrens. Nach Lemma 6.2.3 sind die lokalen Minima von f und g mit $g(x) = \frac{1}{2}\|x - x^*\|_A^2$ identisch, daher gilt

$$\begin{aligned} x^{(k+1)} &= \arg \min_{x\in x^{(k)}+\operatorname{span}\{r^{(k)}\}} f(x) = \arg \min_{x\in x^{(k)}+\operatorname{span}\{r^{(k)}\}} g(x) \\ &= \arg \min_{x\in x^{(k)}+\operatorname{span}\{r^{(k)}\}} \tfrac{1}{2}\|x - x^*\|_A^2, \end{aligned}$$

also minimiert $x^{(k+1)}$ den Fehler in der Energienorm entlang der Suchrichtung.

(ii) Wir zeigen nun, dass wir das Gradientenverfahren als ein Richardson-Verfahren mit einem Iterations-abhängigen Dämpfungsparameter γ_k schreiben können: Es gilt nach Algorithmus 6.2.6

$$x^{(k+1)} = x^{(k)} + \lambda_{\text{opt}}^{(k)} r^{(k)} = x^{(k)} + \lambda_{\text{opt}}^{(k)} (b - Ax^{(k)}) = (I - \lambda_{\text{opt}}^{(k)} A)x^{(k)} + \lambda_{\text{opt}}^{(k)} b,$$

also $\gamma_k = \lambda_{\text{opt}}^{(k)} = \frac{\|r^{(k)}\|^2}{\|r^{(k)}\|_A^2}$. Da der Startwert $x^{(0)} \in \mathbb{R}^n$ beliebig war, genügt es also, den ersten Schritt $k = 1$ zu analysieren.

(iii) Wir betrachten das Richardson-Verfahren mit optimalem Dämpfungsparameter γ^*, vgl. Satz 6.1.3. Für die entsprechende erste Iterierte gilt für $k = 1$

$$x_{\text{R}}^{(1)} = (I - \gamma^* A)x^{(0)} + \gamma^* b = x^{(0)} + \gamma^* (b - Ax^{(0)}) \in x^{(0)} + \operatorname{span}\{r^{(0)}\}$$

und damit gilt

$$\|x^{(1)} - x^*\|_A = \min_{x\in x^{(0)}+\operatorname{span}\{r^{(0)}\}} \|x - x^*\|_A \le \|x_{\text{R}}^{(1)} - x^*\|_A,$$

d.h. das Gradientenverfahren ist lokal (für einen Iterationssschritt) besser als das Richardson-Verfahren mit optimaler Dämpfung. Wir können den Fehler des Gradientenverfahrens also mittels des Richardson-Verfahrens abschätzen.

(iv) Für den Fehler des optimalen Richardson-Verfahrens gilt

$$\|x_{\text{R}}^{(1)} - x^*\|_A \le \varrho(C_{\text{R}}(\gamma^*))\|x_{\text{R}}^{(0)} - x^*\|_A,$$

und mit Satz 6.1.3 (c) gilt weiter

$$\varrho(C_{\text{R}}(\gamma^*)) = \frac{\lambda_{\max}(A) - \lambda_{\min}(A)}{\lambda_{\max}(A) + \lambda_{\min}(A)} = \frac{\|A\| - \|A^{-1}\|^{-1}}{\|A\| + \|A^{-1}\|^{-1}} = \frac{\frac{\|A\|\,\|A^{-1}\|-1}{\|A^{-1}\|}}{\frac{\|A\|\,\|A^{-1}\|+1}{\|A^{-1}\|}} = \frac{\kappa - 1}{\kappa + 1}, \tag{6.34}$$

wobei wir verwendet haben, dass $\|A\| = \lambda_{\max}(A)$, $\lambda_{\min}(A) = \|A^{-1}\|^{-1}$ und $\kappa = \|A\|\,\|A^{-1}\|$ gelten.

Damit ist alles gezeigt. ∎

Bemerkung 6.2.9 i) Beachtet man

$$\frac{\kappa-1}{\kappa+1} = 1 - \frac{2}{\kappa} + \frac{2}{\kappa(\kappa+1)} = 1 - \frac{2}{\kappa} + \mathcal{O}(\kappa^{-2}) \quad (\kappa \to \infty),$$

so ergibt sich, dass für große κ der Term $\frac{\kappa-1}{\kappa+1}$ sehr nahe bei 1 liegt, d.h. man erhält eine sehr geringe Konvergenzgeschwindigkeit!

ii) Die geringe Konvergenzgeschwindigkeit tritt auch schon bei folgendem einfachen Beispiel auf:

$$A = \begin{pmatrix} 1 & 0 \\ 0 & a \end{pmatrix}, \; a \gg 1, \; b = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \text{ und } x^{(0)} = \begin{pmatrix} a \\ 1 \end{pmatrix}. \tag{6.35}$$

Daraus ergibt sich (Übung) $\begin{pmatrix} x_1^{(k+1)} \\ x_2^{(k+1)} \end{pmatrix} = \varrho \begin{pmatrix} x_1^{(k)} \\ -x_2^{(k)} \end{pmatrix}$ mit $\varrho = \dfrac{a-1}{a+1}$.

Da man $\kappa_2(A) = a$ hat, ist ϱ genau die Rate aus Satz 6.2.8!

iii) Wie in Abbildung 6.4 dargestellt, sieht man hier ein „Zick–Zack–Verhalten" der Iterierten. Dies erklärt sich dadurch, dass die Suchrichtungen aufeinander senkrecht stehen.

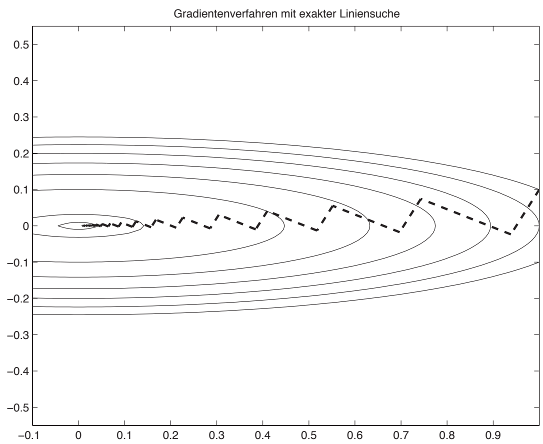


Abbildung 6.4: Dargestellt sind die Höhenlinien der quadratischen Kostenfunktion f zu den Daten aus Beispiel (6.35) und die Iterierten $x^{(k)} = (x_1^{(k)}, x_2^{(k)})^T$, welche durch die gestrichelte Linie verbunden sind. Es zeigt sich das Zick–Zack–Verhalten des Gradientenverfahrens mit exakter Liniensuche.

Bevor wir uns mit Verbesserungen des Gradientenverfahrens befassen, formulieren wir das Basis-Projektionsverfahren[6] zur Lösung von linearen Gleichungssystemen mit hier noch nicht näher spezifizierten Suchrichtungen $d^{(k)}$. Im Gradientenverfahren in Algorithmus 6.2.6 hatten wir die Residuen als Suchrichtungen verwendet, im Folgenden konstruieren wir Richtungen, die zu effizienteren Verfahren führen.

[6] Den Begriff *Projektionsmethode* definieren wir in Definition6.3.4.

Algorithmus 6.2.10 — Basis-Projektionsverfahren zur Lösung des LGS $Ax = b$.

Input: Startwert $x^{(0)} \in \mathbb{R}^n$

1: **for** $k = 0, 1, \dots$ **do**
2: $\quad r^{(k)} := b - Ax^{(k)}$
3: $\quad \lambda^{(k)} := \langle r^{(k)}, d^{(k)} \rangle / \langle d^{(k)}, Ad^{(k)} \rangle$
4: $\quad x^{(k+1)} := x^{(k)} + \lambda^{(k)} d^{(k)}$
5: **end for**

6.3 Projektionsmethoden und Krylov-Unterraum-Verfahren

Wir werden im Folgenden Verbesserungen für das Gradientenverfahren beschreiben. Dabei wird der Begriff der Optimalität eine wesentliche Rolle spielen.

Definition 6.3.1 Sei f wie in (6.28), dann heißt $x \in \mathbb{R}^n$

(a) *optimal bzgl. einer Richtung* $d \in \mathbb{R}^n$, falls $f(x) \le f(x + \lambda d)$ für alle $\lambda \in \mathbb{R}$;

(b) *optimal bzgl. eines Unterraums* $\mathcal{U} \subset \mathbb{R}^n$, falls $f(x) \le f(x + u)$ für alle $u \in \mathcal{U}$.

Offenbar sind die Iterierten des Gradientenverfahrens optimal bzgl. der Richtungen des steilsten Abstiegs. Es liegt also nahe, entweder diese Richtungen zu verbessern oder gleich nach einem Verfahren zu suchen, welche Optimalität bzgl. Unterräumen erlaubt. Um solche Verfahren konstruieren zu können, beschreiben wir einige Eigenschaften optimaler Punkte.

Lemma 6.3.2 Ein Punkt $x \in \mathbb{R}^n$ ist genau dann optimal bzgl. eines Unterraums $\mathcal{U} \subset \mathbb{R}^n$, wenn $r \perp \mathcal{U}$[7] für $r = b - Ax$.

Beweis. Wir betrachten die Funktion $F(\lambda) = f(x + \lambda u)$ definiert in (6.30) für $0 \ne u \in \mathcal{U}$ und die Ableitung in (6.31), also $F'(\lambda) = \langle u, Ax - b \rangle + \lambda \langle u, Au \rangle$ sowie $F''(\lambda) = \langle u, Au \rangle > 0$ für $u \ne 0$. Also ist $x \in \mathbb{R}^n$ genau dann optimal bzgl. $\mathcal{U}$ wenn $F'(0) = 0$ und dies gilt genau dann, wenn $\langle u, b - Ax \rangle = 0$. ■

Wenn der Unterraum $\mathcal{U} = \mathrm{span}\{d\}$ eindimensional ist, dann lautet die Optimalitätsbedingung offenbar $r \perp d$.

Satz 6.3.3 Die Iterierten $x^{(k)}$ des Gradientenverfahrens in Algorithmus 6.2.6 sind optimal bzgl. der Richtung $d^{(k)} = r^{(k-1)}$.

Beweis. Für $r^{(k-1)} = 0$ ist nichts zu zeigen, sei also $r^{(k-1)} \ne 0$. Wegen $\lambda^{(k)}_{\text{opt}} = \frac{\|r^{(k-1)}\|^2}{\|r^{(k-1)}\|_A^2}$ gilt

$$\langle r^{(k)}, r^{(k-1)} \rangle = \langle r^{(k-1)} - \lambda^{(k)}_{\text{opt}} A r^{(k-1)}, r^{(k-1)} \rangle = \|r^{(k-1)}\|^2 - \frac{\|r^{(k-1)}\|^2}{\|r^{(k-1)}\|_A^2} \|r^{(k-1)}\|_A^2 = 0,$$

also $r^{(k)} \perp r^{(k-1)}$ und damit folgt die Behauptung aus Lemma 6.3.2. ■

Wir beschreiben einen allgemeinen Rahmen für eine ganze Klasse von Verfahren. Dieser Zugang wird die spätere Konvergenzanalyse erheblich vereinfachen.

[7] Dabei bedeutet $r \perp \mathcal{U}$ wie üblich, dass $\langle r, u \rangle = 0$ für alle $u \in \mathcal{U}$.

Definition 6.3.4 Eine Folge $(x^{(k)})_{k\in\mathbb{N}}$ zur Lösung eines LGS $Ax = b$ heißt *Projektionsmethode*, wenn es k-dimensionale Unterräume $\mathcal{V}_k, \mathcal{W}_k \subset \mathbb{R}^n$ gibt mit

(i) $x^{(k)} \in x^{(0)} + \mathcal{V}_k$ und der Orthogonalitäts-Bedingung (ii) $(b - Ax^{(k)}) \perp \mathcal{W}_k$.

Bemerkung 6.3.5 (i) Wir nennen $\mathcal{W}_k$ *Testraum* und $x^{(0)} + \mathcal{V}_k$ (affinen) *Ansatzraum*.

(ii) Für $\mathcal{W}_k = \mathcal{V}_k$ steht das Residuum senkrecht auf $\mathcal{V}_k$, dies ist als *Galerkin-Bedingung* bekannt, die Methode heißt dann *orthogonal*. Falls $\mathcal{W}_k \neq \mathcal{V}_k$, nennt man dies auch *Petrov-Galerkin-Bedingung*. Die Bedeutung des Namens „(Petrov-)Galerkin-Orthogonalität" wird in den Vorlesungen *Numerik gewöhnlicher Differenzialgleichungen* bzw. *Numerik partieller Differenzialgleichungen* klarer.

(iii) Falls $\mathcal{V}_k = \mathcal{K}_k := \mathcal{K}_k(A, r^{(0)})$ mit dem in Definition 5.5.1 eingeführten Krylov-Raum, dann heißt die Methode *Krylov-Unterraum-Verfahren*.

(iv) Jeder Schritt des Gauß-Seidel-Verfahrens kann mit $\mathcal{V}_1 = \mathcal{W}_1 = \mathrm{span}\{e_i\}$ als Projektionsmethode interpretiert werden.

(v) Nach Satz 6.3.3 ist das Gradientenverfahren in jedem Schritt eine orthogonale Projektionsmethode mit $\mathcal{V}_1 = \mathcal{W}_1 = \mathrm{span}\{r^{(k-1)}\}$.

Lemma 6.3.6 Falls die Spalten der Matrizen V_k, $W_k \in \mathbb{R}^{n\times k}$ Basen von $\mathcal{V}_k$ und $\mathcal{W}_k$ bilden,[8] so dass $W_k^T A V_k \in \mathbb{R}^{k\times k}$ regulär ist, dann gilt für die Iterierten einer Projektiosmethode

$$x^{(k)} = x^{(0)} + V_k (W_k^T A V_k)^{-1} W_k^T r^{(0)}.$$

Beweis. Wegen $x^{(k)} \in x^{(0)} + \mathcal{V}_k$ gibt es einen Koeffizientenvektor $\alpha^{(k)} \in \mathbb{R}^k$ mit $x^{(k)} = x^{(0)} + V_k \alpha^{(k)}$. Die Orthogonalitäts-Bedingung lautet dann

$$0 = W_k^T (b - A(x^{(0)} + V_k \alpha^{(k)})), \quad \text{also folgt } W_k^T A V_k \alpha^{(k)} = W_k^T (b - Ax^{(0)}) = W_k^T r^{(0)}.$$

Da $W_k^T A V_k$ nach Voraussetzung regulär ist, gilt $\alpha^{(k)} = (W_k^T A V_k)^{-1} W_k^T r^{(0)}$. ■

Die Voraussetzung, dass $W_k^T A V_k$ invertierbar sein muss, lässt sich für spezielle Krylov-Unterraum-Methoden direkt nachweisen.

Lemma 6.3.7 Sei $A \in \mathbb{R}^{n\times n}$ s.p.d. Für $\mathcal{W}_k = \mathcal{V}_k$ ist $V_k^T A V_k$ invertierbar.

Beweis. Da A s.p.d. ist, existiert eine orthogonale Matrix $U \in \mathbb{R}^{n\times n}$ mit $U^T A U = D = \mathrm{diag}\{\lambda_1, \ldots, \lambda_n\}$ mit den reellen Eigenwerten $\lambda_i > 0$ von A. Mit $W_k = V_k$ gilt

$$V_k^T A V_k = (D^{1/2} U^T V_k)^T (D^{1/2} U^T V_k) \in \mathbb{R}^{k\times k}.$$

Da die Spalten von V_k eine Basis von $\mathcal{V}_k$ bilden, hat V_k vollen Rang k und daher ist $D^{1/2} U^T V_k$ injektiv. Damit gilt für alle $x \in \mathbb{R}^k \setminus \{0\}$

$$\langle x, V_k^T A V_k x \rangle = \| D^{1/2} U^T V_k x \|^2 > 0,$$

also ist $V_k^T A V_k$ regulär. ■

[8] Dies bedeutet, dass $\mathcal{V}_k = \mathrm{span}\{v_1, \ldots, v_k\}$ für die linear unabhängigen Spalten $v_i \in \mathbb{R}^n$ von V_k gilt.

Mit Lemma 6.3.6 können wir nun auch das Residuum ausdrücken:

$$\begin{aligned} r^{(k)} &= b - Ax^{(k)} = b - Ax^{(0)} - AV_k(W_k^T AV_k)^{-1}W_k^T r^{(0)} \\ &= (I - AV_k(W_k^T AV_k)^{-1}W_k^T)\, r^{(0)} =: P_k r^{(0)}. \end{aligned} \tag{6.36}$$

Definition 6.3.8 Es bezeichne $\mathbb{P}_k$ den Raum der matrixwertigen algebraischen Polynome $p_k : \mathbb{R}^{n\times n} \to \mathbb{R}^{n\times n}$ vom Grad maximal $k \in \mathbb{N}$.[a] Wir definieren dann

$$\mathbb{P}_k^1 := \{p_k \in \mathbb{P}_k : p(0) = I\},$$

wobei $0 \in \mathbb{R}^{n\times n}$ die Nullmatrix und $I \in \mathbb{R}^{n\times n}$ die Identität bezeichne.

[a] $\mathbb{P}_k = \{p_k(A) = a_0 I + a_1 A + a_2 A^2 + \dots + a_k A^k : a_0, \dots, a_k \in \mathbb{R},\ A \in \mathbb{R}^{n\times n}$; für $p_k \in \mathbb{P}_k^1$ muss $a_0 = 1$ gelten.

Satz 6.3.9 — Fehlerabschätzung für Krylov-Unterraummethode. Falls die Spalten der Matrizen V_k, $W_k \in \mathbb{R}^{n\times k}$ Basen von $\mathcal{V}_k$ und $\mathcal{W}_k$ bilden, so dass $W_k^T AV_k \in \mathbb{R}^{k\times k}$ regulär ist, dann gilt für jedes Krylov-Unterraum-Verfahren

$$\|r^{(k)}\| \le \|P_k\| \min_{p\in\mathbb{P}_k^1} \|p(A)r^{(0)}\| \qquad \text{und} \qquad \|x^* - x^{(k)}\| \le \|A^{-1}P_k\| \min_{p\in\mathbb{P}_k^1} \|p(A)r^{(0)}\|.$$

Beweis. Wegen $r^{(k)} = b - Ax^{(k)} = A(x^* - x^{(k)})$ brauchen wir nur die erste Ungleichung zu zeigen. Mit (6.36) gilt

$$P_k AV_k = AV_k - AV_k(W_k^T AV_k)^{-1}W_k^T AV_k = AV_k - AV_k = 0,$$

da $W_k^T AV_k$ nach Voraussetzung invertierbar ist. Damit gilt wiederum mit (6.36), dass $r^{(k)} = P_k r^{(0)} = P_k(r^{(0)} + AV_k\alpha)$ für alle $\alpha \in \mathbb{R}^n$. Nun gilt $AV_k\alpha \in A\mathcal{V}_k$ und für jede Krylov-Unterraum-Methode gilt $\mathcal{V}_k = \mathcal{K}_k(A, r^{(0)})$, also $r^{(0)} + AV_k\alpha = p(A)r^{(0)}$ mit $p \in \mathbb{P}_k^1$ und damit

$$r^{(k)} = P_k(p(A)r^{(0)}) \quad \text{für alle } p \in \mathbb{P}_k^1.$$

Damit können wir zur Bestimmung der Norm von $r^{(k)}$ das Minimum über alle Polynome betrachten, d.h.

$$\|r^{(k)}\| = \min_{p\in\mathbb{P}_k^1} \|P_k(p(A)r^{(0)})\| \le \|P_k\| \min_{p\in\mathbb{P}_k^1} \|p(A)r^{(0)}\|,$$

also die Behauptung. ∎

6.4 Das Verfahren der konjugierten Richtungen

Ein erster Schritt zu einer möglichen Verbesserung des Gradientenverfahrens besteht darin, die Optimalität aus Satz 6.3.3 bzgl. der Richtung $d^{(k)} = r^{(k-1)}$ auf den gesamten Unterraum $\mathcal{U}_k := \mathrm{span}\{d^{(1)}, \dots, d^{(k)}\}$ zu erweitern. Wir brauchen also Updates im Verfahren, welche die Optimalität zu den vorhergehenden Suchrichtungen erhalten. Dennoch bleiben hier zunächst die Suchrichtung noch offen, so dass das Verfahren der konjugierten

Richtungen eher ein abstraktes Hilfsmittel und kein konkret implementierbares Verfahren ist.

Satz 6.4.1 Sei $x^{(k)}$ optimal bzgl. des Unterraums $\mathcal{U}_k$ und $d \in \mathbb{R}^n$ eine Suchrichtung. Dann ist $\tilde{x}^{(k)} := x^{(k)} + d$ genau dann optimal bzgl. $\mathcal{U}_k$, wenn $Ad \perp \mathcal{U}_k$.

Beweis. Sei $\eta \in \mathcal{U}_k$ beliebig. Wir müssen nach Lemma 6.3.2 zeigen, dass $\langle b - A\tilde{x}^{(k)}, \eta\rangle = 0$ ist. Es gilt

$$\langle b - A\tilde{x}^{(k)}, \eta\rangle = \langle b - Ax^{(k)}, \eta\rangle - \langle Ad, \eta\rangle = -\langle Ad, \eta\rangle,$$

aufgrund der Optimalität von $x^{(k)}$ bzgl. $\mathcal{U}_k$, d.h., der Orthogonalität von $r^{(k)}$ zu $\mathcal{U}_k$. Also ist $\tilde{x}^{(k)}$ optimal bzgl. $\mathcal{U}_k$ genau dann wenn $\langle Ad, \eta\rangle = 0$ für alle $\eta \in \mathcal{U}_k$. ∎

Wir suchen also eine Suchrichtung $d^{(k+1)}$ mit $\langle d^{(k+1)}, d^{(j)}\rangle_A = 0$ für alle $j = 1, \dots, k$.

Definition 6.4.2 Zwei Vektoren $x, y \in \mathbb{R}^n$ heißen **konjugiert** oder **A-orthogonal**, falls $\langle x, y\rangle_A = 0$. Vektoren $\{d^{(1)}, \dots, d^{(k)}\}$ heißen *paarweise konjugiert*, wenn $d^{(i)} \neq 0$ und

$$\langle d^{(i)}, d^{(j)}\rangle_A = \delta_{ij} \|d^{(i)}\|_A^2, \quad \text{für alle } i, j \in \{1, \dots, k\}.$$

Wir können es auch so formulieren: Eine Idee besteht darin, dass Zick-Zack-Verhalten beim Gradientenverfahren dadurch zu vermeiden, dass man den Winkel zwischen den Suchrichtungen an die Zielfunktion f anpasst. Dies kann man dadurch erreichen, dass wir nicht mehr Orthogonalität bzgl. des Euklidischen Skalarproduktes verwenden, sondern bzgl. des Energie-Skalarproduktes. Das Skalarprodukt bzgl. dessen man Orthogonalität misst, wird also durch die Matrix A selber bestimmt.

Bemerkung 6.4.3 i) Paarweise konjugierte Vektoren $\{d^{(1)}, \dots, d^{(k)}\}$ sind linear unabhängig (Übung).

ii) Jeder Vektor $x \in \mathbb{R}^n$ besitzt eine eindeutige Entwicklung bezüglich konjugierter Richtungen $\{d^{(1)}, \dots, d^{(n)}\}$, d.h.

$$x = \sum_{k=1}^{n} \widehat{\alpha}_k(x)\, d^{(k)} \quad \text{mit} \quad \widehat{\alpha}_k(x) = \frac{\langle x, d^{(k)}\rangle_A}{\|d^{(k)}\|_A^2},$$

was man wie folgt sieht:

$$\langle x, d^{(k)}\rangle_A = \sum_{i=1}^{n} \widehat{\alpha}_i(x)\, \underbrace{\langle d^{(i)}, d^{(k)}\rangle_A}_{=\delta_{ik}\|d^{(k)}\|_A^2} = \widehat{\alpha}_k(x) \|d^{(k)}\|_A^2, \quad \text{also}$$

$$\widehat{\alpha}_k(x) = \frac{\langle d^{(k)}, Ax\rangle}{\langle d^{(k)}, Ad^{(k)}\rangle}, \quad k = 1, \dots, n. \tag{6.37}$$

iii) Für die Lösung x^* von $Ax = b$ gilt offenbar $\widehat{\alpha}_k(x^*) = \dfrac{\langle d^{(k)}, b\rangle}{\langle d^{(k)}, Ad^{(k)}\rangle}$, $k = 1, \dots, n$.

Nehmen wir nun an, dass $x^{(k)}$ optimal bzgl. des Raumes $\mathcal{U}_k = \operatorname{span}\{d^{(1)}, \dots, d^{(k)}\}$ ist und dass $d^{(k+1)}$ eine zu $\mathcal{U}_k$ konjugierte Richtung ist. Dann ist

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

für $\lambda \in \mathbb{R}$ optimal bzgl. des neuen Raumes $\mathcal{U}_{k+1}$, wenn für alle $j = 1, \ldots, k+1$ gilt

$$0 = \langle b - Ax^{(k+1)}, d^{(j)} \rangle = \langle b - Ax^{(k)}, d^{(j)} \rangle - \lambda \langle Ad^{(k+1)}, d^{(j)} \rangle$$

Für $j = 1, \ldots, k$ verschwinden beide Terme wegen der Optimalität (erster Summand) bzw. der Konjugiertheit (zweiter Summand). Es bleibt also die Forderung

$$0 = \langle b - Ax^{(k)}, d^{(k+1)} \rangle - \lambda \langle Ad^{(k+1)}, d^{(k+1)} \rangle,$$

und damit folgt

$$\lambda^{(k+1)} = \frac{\langle r^{(k)}, d^{(k+1)} \rangle}{\|d^{(k+1)}\|_A^2} \tag{6.38}$$

mit dem **Residuum** $r^{(k)} := b - Ax^{(k)}$. Das folgende Lemma besagt, dass auch das Verfahren der konjugierten Richtungen bei exakter Arithmetik ein direktes Verfahren ist.

Lemma 6.4.4 Seien $\{d^{(1)}, \ldots, d^{(n)}\}$ konjugierte Richtungen. Für jedes $x^{(0)} \in \mathbb{R}^n$ und

$$x^{(k+1)} := x^{(k)} + \lambda^{(k+1)} d^{(k+1)} \quad \text{mit } \lambda^{(k+1)} \text{ gemäß (6.38)}, \; k = 0, \ldots, n-1, \tag{6.39}$$

gilt nach (höchstens) n Schritten $x^{(n)} = x^* = A^{-1}b$.

Beweis. Aus (6.37) folgt für $x^* = A^{-1}b$, dass

$$x^* - x^{(0)} = \sum_{k=1}^{n} \tilde{\lambda}^{(k)} d^{(k)} \quad \text{mit} \quad \tilde{\lambda}^{(k)} = \frac{\langle d^{(k)}, A(x^* - x^{(0)}) \rangle}{\langle d^{(k)}, Ad^{(k)} \rangle} = \frac{\langle d^{(k)}, b - Ax^{(0)} \rangle}{\langle d^{(k)}, Ad^{(k)} \rangle}.$$

Aus (6.39) folgt

$$x^{(k-1)} - x^{(0)} = x^{(k-2)} + \lambda^{(k-1)} d^{(k-1)} - x^{(0)} = \cdots = \sum_{i=1}^{k-1} \lambda^{(i)} d^{(i)},$$

und da $d^{(k)}$ zu $d^{(i)}$, $i \neq k$, konjugiert ist, ergibt sich damit

$$\langle d^{(k)}, A(x^{(k-1)} - x^{(0)}) \rangle = \langle d^{(k)}, A \left(\sum_{i=1}^{k-1} \lambda^{(i)} d^{(i)} \right) \rangle = \sum_{i=1}^{k-1} \lambda^{(i)} \underbrace{\langle d^{(k)}, Ad^{(i)} \rangle}_{=0, \text{ da } i \neq k} = 0,$$

also

$$\begin{aligned} \langle d^{(k)}, A(x^* - x^{(0)}) \rangle &= \langle d^{(k)}, A(x^* - x^{(k-1)}) \rangle + \underbrace{\langle d^{(k)}, A(x^{(k-1)} - x^{(0)}) \rangle}_{=0} \\ &= \langle d^{(k)}, b - Ax^{(k-1)} \rangle = \langle d^{(k)}, r^{(k-1)} \rangle, \end{aligned}$$

und damit $\tilde{\lambda}^{(k)} = \lambda^{(k)}$, womit die Aussage bewiesen ist. ∎

Aus (6.39) ergibt sich folgende Rekursion für das Residuum

$$r^{(k)} = b - Ax^{(k)} = b - Ax^{(k-1)} - \lambda^{(k)} Ad^{(k)} = r^{(k-1)} - \lambda^{(k)} Ad^{(k)}. \tag{6.40}$$

Bemerkung 6.4.5 i) Lemma 6.4.4 besagt, dass das Verfahren ein direktes Verfahren ist, welches nach n Iterationen konvergiert. Da jeder Iterationsschritt einer Matrix-Vektor-Multiplikation entspricht, benötigt also ein Schritt für eine dünnbesetzte Matrix $\mathcal{O}(n)$ Operationen, insgesamt also $\mathcal{O}(n^2)$. Wir werden aber später sehen, dass dies *nicht* bestmöglich ist.

ii) Darüber hinaus hat sich in Experimenten gezeigt, dass der Fehler bei ungüstiger Wahl der Suchrichtungen immer noch sehr groß sein kann. Wir brauchen also eine geeignete Wahl der Suchrichtungen. Dies führt auf das cg-Verfahren in §6.5.

Satz 6.4.6 — Galerkin-Orthogonalität. Seien $\{d^{(1)}, \dots, d^{(n)}\}$ konjugierte Richtungen und $r^{(k)}$, $k = 0, \dots, n-1$, die Residuen. Dann gilt für alle $k = 0, \dots, n-1$

$$\langle r^{(k)}, d^{(j)} \rangle = 0 \text{ für } 1 \le j \le k, \qquad \text{bzw.} \qquad r^{(k)} \perp \mathcal{U}_k := \operatorname{span}\{d^{(1)}, \dots, d^{(k)}\}. \tag{6.41}$$

Beweis. Nach (6.40) gilt für $k \in \{0, \dots, n-1\}$

$$\begin{aligned} r^{(k)} &= r^{(k-1)} - \lambda^{(k)} Ad^{(k)} = r^{(k-2)} - \lambda^{(k-1)} Ad^{(k-1)} - \lambda^{(k)} Ad^{(k)} \\ &= \dots = r^{(0)} - \sum_{\ell=1}^{k} \lambda^{(\ell)} Ad^{(\ell)}. \end{aligned}$$

Daraus folgt nun für $1 \le j \le k$ aufgrund der A-Orthogonalität der $d^{(j)}$ und (6.38)

$$\begin{aligned} \langle r^{(k)}, d^{(j)} \rangle &= \langle r^{(0)}, d^{(j)} \rangle - \sum_{\ell=1}^{k} \lambda^{(\ell)} \langle d^{(j)}, Ad^{(\ell)} \rangle \\ &= \langle r^{(0)}, d^{(j)} \rangle - \lambda^{(j)} \langle d^{(j)}, Ad^{(j)} \rangle = \langle r^{(0)}, d^{(j)} \rangle - \frac{\langle r^{(j-1)}, d^{(j)} \rangle}{\langle d^{(j)}, Ad^{(j)} \rangle} \langle d^{(j)}, Ad^{(j)} \rangle \\ &= \langle r^{(0)} - r^{(j-1)}, d^{(j)} \rangle = \sum_{\ell=1}^{j-1} \lambda^{(\ell)} \langle d^{(\ell)}, Ad^{(j)} \rangle = 0, \end{aligned}$$

womit der Satz bewiesen ist. ∎

Bemerkung 6.4.7 Gleichung (6.41) ist in der Tat die *Galerkin-Orthogonalität* aus Bemerkung 6.3.5, $r^{(k)} \perp \mathcal{U}_k$. Also ist die Methode der konjugierten Richtungen eine orthogonale Projektionsmethode, so hatten wir sie ja konstruiert.

Wir formulieren den (abstrakten) Algorithmus der konjugierten Richtungen.

Algorithmus 6.4.8 — Verfahren der konjugierten Richtungen zur Lösung des LGS $Ax = b$.

Input: Startwert $x^{(0)} \in \mathbb{R}^n$, konjugierte Richtungen $\{d^{(1)}, \dots, d^{(n)}\}$[9]

1: $r^{(0)} := b - Ax^{(0)}$

2: **for** $k = 0, 1, \dots$ **do**

$\quad \lambda^{(k+1)} := \langle r^{(k)}, d^{(k+1)} \rangle / \langle d^{(k+1)}, Ad^{(k+1)} \rangle$
$\quad x^{(k+1)} := x^{(k)} + \lambda^{(k+1)} d^{(k+1)}$
$\quad r^{(k+1)} := r^{(k)} - \lambda^{(k+1)} Ad^{(k+1)}$
end for

Bislang haben wir die Suchrichtungen $d^{(k)}$ nicht spezifiziert. Man wählt z.B. $d^{(1)}$ beliebig und kann dann die folgenden Richtungen mit einem Orthogonalisierungsverfahren, z.B. mit Gram-Schmidt, bestimmen, vgl §A.4. Damit hätte man einen implementierbaren Algorithmus. Dieser ist jedoch nicht effizient und führt uns auf das cg-Verfahren.

6.5 Verfahren der konjugierten Gradienten (cg-Verfahren)

Wie bereits erwähnt, wurde das Verfahren der konjugierten Gradienten (engl. *conjugate gradient*, daher auch *cg-Verfahren* genannt) 1952 von von Hestenes und Stiefel entwickelt. Im Jahr 1971 gewann das cg-Verfahren durch die sogenannte *Vorkonditionierung* enorm an Bedeutung und heute gehören vorkonditionierte cg-Verfahren (pcg, siehe §6.6) zu den beweisbar optimalen Verfahren. Für das Beispiel der 2D-Standardmatrix (Beispiel 6.0.2) ist das cg-Verfahren ab einer Systemgröße von ca. 2000 Variablen deutlich effizienter als direkte Verfahren bei zusätzlich erheblich geringerem Speicherbedarf.

Das cg-Verfahren kombiniert das Gradientenverfahren mit dem Verfahren der konjugierten Richtungen: Es nutzt die Residuen zur Definition der konjugierten Suchrichtungen, wodurch problemangepasste Suchrichtungen berechnet werden und gleichzeitig die Optimalität des Verfahrens der konjugierten Richtungen erhalten wird. Wie bei der Methode der konjugierten Richtungen ist auch das cg-Verfahren -theoretisch- ein direktes Verfahren, vgl. Lemma 6.4.4.

Wir spezifizieren die Wahl der Suchrichtungen mittels der Residuen:

- Falls $r^{(0)} \neq 0$ gilt (sonst wäre $x^{(0)} = x^*$ die gesuchte Lösung) setzt man $d^{(1)} := r^{(0)}$.
- Für $k = 1, 2, 3, \ldots$ verfahren wir wie folgt: Falls $r^{(k)} \neq 0$ (sonst wäre $x^{(k)} = x^*$), gewinnt man *formal* $d^{(k+1)}$ mittels des Gram-Schmidtschen-Orthogonalisierungsverfahren (Satz A.4.1) bezüglich des Energie-Skalarproduktes $\langle \cdot, \cdot \rangle_A$ (vgl. Definition 6.2.2) aus $r^{(k)}$ und den bereits bestimmten konjugierten Richtungen $d^{(1)}, \ldots, d^{(k)}$, d.h.

$$d^{(k+1)} = r^{(k)} - \sum_{j=1}^{k} \frac{\langle r^{(k)}, d^{(j)} \rangle_A}{\|d^{(j)}\|_A^2} d^{(j)}. \tag{6.42}$$

Damit erhalten wir das cg-Verfahren. Wir geben zunächst einen „Basis-Algorithmus" an, den wir später hinsichtlich Effizienz und Stabilität umformulieren werden.

Algorithmus 6.5.1 — Basis-Verfahren der konjugierten Gradienten.

Input: Startwert $x^{(0)} \in \mathbb{R}^n$
$d^{(1)} := r^{(0)} := b - Ax^{(0)}$
for $k = 0, 1, 2, \ldots$ **do**
$\quad \lambda^{(k+1)} := \langle r^{(k)}, d^{(k+1)} \rangle / \langle d^{(k+1)}, Ad^{(k+1)} \rangle$
$\quad x^{(k+1)} := x^{(k)} + \lambda^{(k+1)} d^{(k+1)}$

[9] Die konjugierten Richtungen $\{d^{(1)}, \ldots, d^{(n)}\}$ können auch in der Schleife konstruiert werden.

5: $r^{(k+1)} := r^{(k)} - \lambda^{(k+1)} Ad^{(k+1)}$
6: $d^{(k+2)} := r^{(k)} - \sum_{j=1}^{k} \frac{\langle r^{(k)}, d^{(j)}\rangle_A}{\|d^{(j)}\|_A^2} d^{(j)}$
7: **end for**

Die Darstellung in (6.42) ist wegen der Summe von $j = 1, ..., k$ nicht effizient; wir werden diese reduzieren. Dazu:

Lemma 6.5.2 Für die A-orthogonalen Richtungen mit $\mathcal{U}_k$ definiert in (6.41) gelte $r^{(k)} \neq 0$, $k = 0, ..., n-1$. Dann gilt:

(i) $\mathcal{U}_k = \operatorname{span}\{r^{(0)}, ..., r^{(k-1)}\}$ mit $\dim \mathcal{U}_k = k$, $k = 1, ..., n$; insbesondere gilt $d^{(k+1)} \notin \mathcal{U}_k$;

(ii) $\mathcal{U}_k = \operatorname{span}\{r^{(0)}, Ar^{(0)}, ..., A^{k-1}r^{(0)}\} = \mathcal{K}_k(A, r^{(0)}) =: \mathcal{K}_k$.

Beweis. (i) Für $k = 1$ gilt $d^{(1)} = r^{(0)}$ und damit die Behauptung. Angenommen, die Behauptung gelte für alle $m = 1, ..., k$, $k < n$. Dann folgt aus (6.42), dass $d^{(k+1)}$ zu $\operatorname{span}\{d^{(1)}, ..., d^{(k)}\} = \mathcal{U}_k$ konjugiert ist. Also folgt nach Bemerkung 6.4.3 i), dass $\dim \mathcal{U}_{k+1} = k+1$. Weiterhin folgt aus (6.42), dass $d^{(k+1)} - r^{(k)} \in \mathcal{U}_k$ und damit $\mathcal{U}_{k+1} = \operatorname{span}\{\mathcal{U}_k, d^{(k+1)}\} = \operatorname{span}\{\mathcal{U}_k, r^{(k)}\} = \operatorname{span}\{r^{(0)}, ..., r^{(k)}\}$ mit der Induktionsannahme.
(ii): Für $k = 1$ ist die Aussage wiederum trivial. Angenommen, die Behauptung gelte für alle $m = 1, ..., k$ mit $k < n$. Dann folgt mit (i)

$$r^{(k-1)} \in \mathcal{U}_k = \operatorname{span}\{d^{(1)}, ..., d^{(k)}\} = \operatorname{span}\{r^{(0)}, Ar^{(0)}, ..., A^{k-1}r^{(0)}\} = \mathcal{K}_k$$

und $Ad^{(k)} \in A(\mathcal{U}_k) = \operatorname{span}\{Ar^{(0)}, A^2r^{(0)}, ..., A^k r^{(0)}\}$. Mit (6.40) gilt

$$r^{(k)} = r^{(k-1)} - \lambda^{(k)} Ad^{(k)} \in \operatorname{span}\{r^{(0)}, Ar^{(0)}, ..., A^k r^{(0)}\} = \mathcal{K}_{k+1},$$

also $\mathcal{U}_{k+1} = \operatorname{span}\{r^{(0)}, ..., r^{(k)}\} \subseteq \mathcal{K}_{k+1}$ und da beide nach (i) die gleiche Dimension besitzen, sind die Räume identisch. ∎

Korollar 6.5.3 Das cg-Verfahren ist ein orthogonales Krylov-Unterraum-Verfahren mit $\mathcal{V}_k = \mathcal{W}_k = \mathcal{K}_k(A, r^{(0)}) =: \mathcal{K}_k$ im Sinne von Definition 6.3.4.

Beweis. Es gilt $x^{(k)} = x^{(k-1)} + \lambda^{(k)} d^{(k)} = x^{(0)} + \sum_{j=1}^{k} \lambda^{(j)} d^{(j)} \in x^{(0)} + \mathcal{U}_k = x^{(0)} + \mathcal{K}_k$ nach Lemma 6.5.2 (ii), also $\mathcal{V}_k = \mathcal{K}_k$. Wegen der Galerkin Orthogonalität $r^{(k)} \perp \mathcal{U}_k$ und $\mathcal{U}_k = \mathcal{K}_k$ nach Satz 6.4.6 gilt $\mathcal{W}_k = \mathcal{K}_k$. ∎

Als Konsequenz halten wir fest:

Satz 6.5.4 Für jeden Schritt des cg-Verfahrens gilt

$$\|x^* - x^{(k)}\|_A = \min_{y \in \mathcal{K}_k(A, r^{(0)})} \|x^* - (x^{(0)} + y)\|_A,$$

mit anderen Worten: die cg-Approximation $x^{(k)}$ ist die beste Approximation an die exakte Lösung x^* aus dem affinen Raum $x^{(0)} + \mathcal{K}_k(A, r^{(0)})$ in der Energienorm. ∎

Nun können wir die Summe in (6.42) reduzieren.

Korollar 6.5.5 Sei $r^{(k)} \neq 0$, dann gilt $\langle r^{(k)}, d^{(j)} \rangle_A = 0$, $j = 1, \dots, k-1$ und (6.42) lautet

$$d^{(k+1)} = r^{(k)} - \beta^{(k+1)} d^{(k)} \quad \text{mit} \quad \beta^{(k+1)} := \frac{\langle r^{(k)}, d^{(k)} \rangle_A}{\|d^{(k)}\|_A^2}. \tag{6.43}$$

Beweis. Wir haben im Beweis von Lemma 6.5.2 (ii) gezeigt, dass $Ad^{(j)} \in \mathcal{U}_k$, $j = 1, \dots, k-1$. Aufgrund der Galerkin-Orthogonalität gilt aber $r^{(k)} \perp \mathcal{U}_k = \operatorname{span}\{d^{(1)}, \dots, d^{(k)}\}$ und damit $0 = \langle r^{(k)}, Ad^{(j)} \rangle = \langle r^{(k)}, d^{(j)} \rangle_A$ für $j = 1, \dots, k-1$. ∎

Für eine effiziente Umsetzung des Algorithmus schreiben wir nur noch die Koeffizienten um: nach (6.39) gilt $\lambda^{(k)} = \frac{\langle r^{(k-1)}, d^{(k)} \rangle}{\|d^{(k)}\|_A^2}$ und den Zähler schreiben wir um

$$\langle r^{(k-1)}, d^{(k)} \rangle = \langle r^{(k-1)}, r^{(k-1)} \rangle - \beta^{(k)} \underbrace{\langle r^{(k-1)}, d^{(k-1)} \rangle}_{=0} \quad \text{also}$$

$$\lambda^{(k)} = \frac{\langle r^{(k-1)}, r^{(k-1)} \rangle}{\langle d^{(k)}, Ad^{(k)} \rangle} = \frac{\|r^{(k-1)}\|^2}{\|d^{(k)}\|_A^2}. \tag{6.44}$$

Weiterhin gilt wegen $\lambda^{(k)} \langle r^{(k)}, Ad^{(k)} \rangle = \langle r^{(k-1)} - r^{(k)}, r^{(k)} \rangle = -\langle r^{(k)}, r^{(k)} \rangle$

$$\beta^{(k+1)} = \frac{\langle r^{(k)}, Ad^{(k)} \rangle}{\langle d^{(k)}, Ad^{(k)} \rangle} = -\frac{\langle r^{(k)}, r^{(k)} \rangle}{\lambda^{(k)} \langle d^{(k)}, Ad^{(k)} \rangle} = -\frac{\langle r^{(k)}, r^{(k)} \rangle}{\langle r^{(k-1)}, r^{(k-1)} \rangle} = -\frac{\|r^{(k)}\|^2}{\|r^{(k-1)}\|^2}. \tag{6.45}$$

Bemerkung 6.5.6 Die Ausdrücke (6.44), (6.45) haben sich als numerisch stabiler und Speicherplatz-optimal herausgestellt.

Wir können nun den Algorithmus formulieren.

Algorithmus 6.5.7 — Konjugiertes Gradientenverfahren (cg-Verfahren) für $Ax = b$.

Input: $A \in \mathbb{R}^{n \times n}$ s.p.d., $b \in \mathbb{R}^n$; Startwert $x^{(0)} \in \mathbb{R}^n$

```
r^(0) := b - A x^(0)
ϱ^(0) := ⟨r^(0), r^(0)⟩
d^(1) := r^(0)
for k = 0, 1, 2, ... do
    if ‖r^(k)‖ < tol then
        stop
    end if
    a^(k+1) := A d^(k+1)
    λ^(k+1) := ϱ^(k) / ⟨d^(k+1), a^(k+1)⟩
    x^(k+1) := x^(k) + λ^(k+1) d^(k+1)
    r^(k+1) := r^(k) - λ^(k+1) a^(k+1)
    ϱ^(k+1) := ⟨r^(k+1), r^(k+1)⟩
    d^(k+2) := r^(k+1) + (ϱ^(k+1) / ϱ^(k)) d^(k+1)
end for
```

Nun zur Fehlerabschätzung.

Satz 6.5.8 Es sei A s.p.d. Für das cg-Verfahren und $x^* = A^{-1}b$ gilt folgende Abschätzung

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

Beweis. Der Beweis gliedert sich in 4 Schritte. Es sei $e^{(k)} := x^* - x^{(k)}$ (also $Ae^{(k)} = r^{(k)}$).

(1) Sei $k \in \{1, \dots, n\}$. Da $x^{(k)} \in x^{(0)} + \mathcal{K}_k$, gibt es Koeffizienten $c_1, \dots, c_k \in \mathbb{R}$ mit

$$e^{(k)} = x^* - x^{(k)} = x^* - x^{(0)} - \sum_{i=1}^{k} c_i\, A^{i-1} r^{(0)} = e^{(0)} - \sum_{i=1}^{k} c_i\, A^i e^{(0)} = p_k(A) e^{(0)} \tag{6.46}$$

mit $p_k \in \mathbb{P}_k^1$, vgl. Definition 6.3.8.

(2) Nun benötigen wir die Energienorm von $e^{(k)}$. Nach Voraussetzung ist $A \in \mathbb{R}^{n\times n}$ s.p.d., also existieren orthonormale Eigenvektoren v_j, $j = 1, \dots, n$, zu Eigenwerten $0 < \lambda_1 \le \dots \le \lambda_n$, so dass $e^{(0)} = \sum_{j=1}^n \gamma_j v_j$ mit Koeffizienten $\gamma_j = \langle e^{(0)}, v_j\rangle \in \mathbb{R}$ gilt und

$$\|e^{(0)}\|_A^2 = \langle e^{(0)}, Ae^{(0)}\rangle = \sum_{j=1}^{n} \lambda_j\, \gamma_j^2 \quad \text{und analog}$$

$$\|p_k(A)e^{(0)}\|_A^2 = \sum_{j=1}^{n} \lambda_j\, p_k(\lambda_j)^2\, \gamma_j^2, \tag{6.47}$$

was man wie folgt sieht: Setze zunächst $\tilde{c}_0 := 1$ und $\tilde{c}_i := -c_i$ für $i = 1, \dots, k$. Dann gilt $p_k(x) = \sum_{i=0}^k \tilde{c}_i x^i$ und $p_k(A)e^{(0)} = \sum_{i=0}^k \tilde{c}_i A^i e^{(0)}$. Nun setzen wir die Eigenvektorbasis $\{v_1, \dots, v_n\}$ ein und erhalten

$$p_k(A)e^{(0)} = \sum_{i=0}^{k} \tilde{c}_i A^i e^{(0)} = \sum_{i=0}^{k} \tilde{c}_i \sum_{j=1}^{n} \gamma_j A^i v_j = \sum_{j=1}^{n} \gamma_j \left(\sum_{i=0}^{k} \tilde{c}_i\, \lambda_j^i\right) v_j = \sum_{j=1}^{n} \gamma_j\, p_k(\lambda_j)\, v_j.$$

Daraus erhalten wir nun (6.47) gemäß

$$\|p_k(A)e^{(0)}\|_A^2 = \sum_{j,\ell=1}^{n} \gamma_j \gamma_\ell\, p_k(\lambda_j) p_k(\lambda_\ell)\, \underbrace{\langle v_j, Av_\ell\rangle}_{=\lambda_\ell\, \delta_{j,\ell}} = \sum_{j=1}^{n} \lambda_j \gamma_j^2\, p_k(\lambda_j)^2.$$

(3) Analog zu Satz 6.3.9 gilt nun

$$\begin{aligned}\|e^{(k)}\|_A &= \min_{p_k \in \mathbb{P}_k^1} \|p_k(A)e^{(0)}\|_A = \min_{p_k \in \mathbb{P}_k^1} \left(\sum_{j=1}^{n} \lambda_j\, p_k(\lambda_j)^2\, \gamma_j^2\right)^{1/2} \\ &\le \min_{p_k \in \mathbb{P}_k^1} \max_{j=1,\dots,n} |p_k(\lambda_j)| \left(\sum_{j=1}^{n} \lambda_j\, \gamma_j^2\right)^{1/2} = \left(\min_{p_k \in \mathbb{P}_k^1} \max_{\lambda \in [\lambda_1, \lambda_n]} |p_k(\lambda)|\right) \|e^{(0)}\|_A. \end{aligned} \tag{6.48}$$

(4) Man sucht nun Polynome p_k, welche die rechte Seite in (6.48) minimieren. Man kann zeigen, dass die *Tschebyscheff–Polynome* $T_k : \mathbb{R} \to \mathbb{R}$, $k = 0, 1, \dots$, definiert als

$$T_k(x) := \frac{1}{2}\left[\left(x + \sqrt{x^2-1}\right)^k + \left(x - \sqrt{x^2-1}\right)^k\right], \quad x \in \mathbb{R}, \tag{6.49}$$

auf dem Intervall $[-1,1]$ folgende Eigenschaften besitzen

$$T_k(1) = 1\,, \qquad \text{sowie} \qquad |T_k(x)| \le 1 \text{ für alle } -1 \le x \le 1,$$

und dass sie auf dem Intervall $[-1,1]$ minimal bzgl. $\|\cdot\|_\infty$ unter allen Polynomen $p \in \mathbb{P}_k^1$ sind (siehe Lemma 6.5.10 (vi)).
Wir transformieren das Intervall $[\lambda_{\min}, \lambda_{\max}]$ auf $[-1,1]$. Setze $\widetilde{T}_k(z) := T_k\left(\frac{\lambda_{\max}+\lambda_{\min}-2z}{\lambda_{\min}-\lambda_{\max}}\right)$ und

$$\widetilde{p}_k(z) := \frac{\widetilde{T}_k(z)}{\widetilde{T}_k(0)} = T_k\left(\frac{\lambda_{\max}+\lambda_{\min}-2z}{\lambda_{\min}-\lambda_{\max}}\right) \Big/ T_k\left(\frac{\lambda_{\max}+\lambda_{\min}}{\lambda_{\min}-\lambda_{\max}}\right)$$

Dann ist $\widetilde{p}_k(0) = 1$ und dieses Polynom ist minimal bzgl. der Supremumsnorm auf dem Intervall $[\lambda_{\min}, \lambda_{\max}] = [\lambda_1, \lambda_n]$ bzgl. $\mathbb{P}_k^1$. Daraus folgt mit $|T_k(z)| \le 1$

$$\min_{p_k \in \mathbb{P}_k^1} \max_{\lambda \in [\lambda_1,\lambda_n]} |p_k(\lambda)| = \max_{\lambda \in [\lambda_1,\lambda_n]} |\widetilde{p}_k(\lambda)| = \max_{\lambda \in [\lambda_1,\lambda_n]} \left|\frac{\widetilde{T}_k(z)}{\widetilde{T}_k(0)}\right| \le \frac{1}{\left|T_k\left(\frac{\lambda_{\max}+\lambda_{\min}}{\lambda_{\min}-\lambda_{\max}}\right)\right|}$$

und mit (6.34) sowie der Definition (6.49) der Tschebyscheff–Polynome und $|T_k(-z)| = |T_k(z)|$, dass

$$\left|T_k\left(\frac{\lambda_{\max}+\lambda_{\min}}{\lambda_{\min}-\lambda_{\max}}\right)\right| = \left|T_k\left(\frac{\lambda_{\max}+\lambda_{\min}}{\lambda_{\max}-\lambda_{\min}}\right)\right| = \left|T_k\left(\frac{\kappa+1}{\kappa-1}\right)\right| \ge \frac{1}{2}\left(z + \sqrt{z^2-1}\right)^k$$

mit $z = \frac{\kappa+1}{\kappa-1}$, also

$$\begin{aligned} z + \sqrt{z^2-1} &= \frac{\kappa+1}{\kappa-1} + \sqrt{\frac{\kappa^2+2\kappa+1-\kappa^2+2\kappa-1}{(\kappa-1)^2}} \\ &= \frac{1}{\kappa-1}(\kappa+1+2\sqrt{\kappa}) = \frac{(\sqrt{\kappa}+1)^2}{(\sqrt{\kappa}+1)(\sqrt{\kappa}-1)} = \frac{\sqrt{\kappa}+1}{\sqrt{\kappa}-1}. \end{aligned}$$

Damit ist der Satz bewiesen. ■

Bemerkung 6.5.9 Man beachte, dass sich der Fehlerreduktionsfaktor im Vergleich zum Gradientenverfahren (vgl. Satz 6.2.8) durch die Wurzel reduziert.

Wir stellen noch die verwendeten Eigenschaften der Tschebyscheff-Ploynome[10] zusammen. Diese werden uns ausführlicher in der Vorlesung *Numerische Analysis* wieder begegnen.

Lemma 6.5.10 — Tschebyscheff–Polynome. Die *Tschebyscheff*–Polynome T_k in (6.49) haben folgende Eigenschaften:

(i) $T_k(0) = 1$, $T_k(1) = 1$, $T_k(-1) = (-1)^k$;
(ii) $T_k(-x) = (-1)^k T_k(x)$;
(iii) $|T_k(x)| \le 1$ für $x \in [-1,1]$;
(iv) $|T_k(x)|$ nimmt seinen maximalen Wert im Intervall $[-1,1]$ an den sogenannten

[10] Pafnuti Lwowitsch Tschebyschew (auch Čebyšëv), 1821-1894.

Tschebyscheff–Abszissen $\overline{x}_j := \cos(\frac{j\pi}{k})$ für $j = 0, ..., k$ an, d.h.

$$|T_k(x)| = 1 \Leftrightarrow x = \overline{x}_j \quad j = 0, \dots, k.$$

(v) Für $x \in \mathbb{R}$ erfüllen die Tschebyscheff-Polynome die folgende *Drei–Term–Rekursion*

$$\boxed{T_0(x) = 1,\ T_1(x) = x, \quad T_k(x) = 2x\,T_{k-1}(x) - T_{k-2}(x), \quad k \geq 2.}$$

(vi) Sie lösen das *MinMax–Problem*: $T_k = \arg\min\{\|p\|_\infty : p \in \mathbb{P}_k^1\}$[11] mit der Maximumsnorm $\|f\|_\infty = \max_{x\in[-1,1]} |f(x)|$.

Ergänzend seien die T_k für $k = 0, ..., 5$ explizit angegeben und diese grafisch dargestellt:

$$T_0(x) \equiv 1, \qquad T_1(x) = x, \qquad T_2(x) = 2x^2 - 1,$$
$$T_3(x) = 4x^3 - 3x, \qquad T_4(x) = 8x^4 - 8x^2 + 1, \qquad T_5(x) = 16x^5 - 20x^3 + 5x,$$

vgl. Abbildung 6.5.

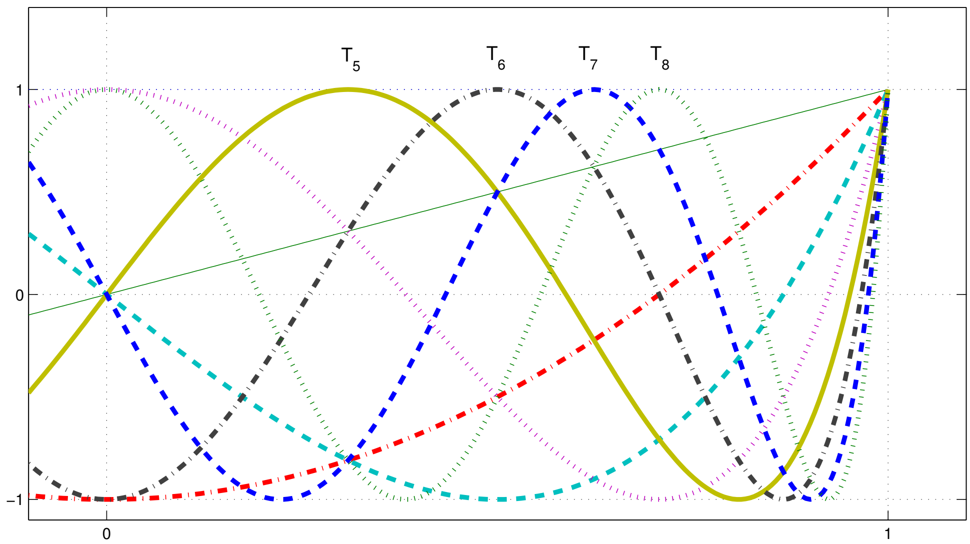


Abbildung 6.5: Tschebyscheff-Polynome $T_1, ..., T_8$.

6.6 Vorkonditionierung, das pcg-Verfahren

Die Abschätzung der Konvergenzgeschwindigkeit des cg-Verfahrens hängt gemäß Satz 6.5.8 monoton von der Kondition κ der Matrix A ab. Ziel dieses Abschnittes ist es daher, das LGS $Ax = b$ so zu transformieren, dass die entstehende Matrix $\bar{A}$ möglichst „gut konditioniert" (d.h. $\kappa(\bar{A})$ möglichst klein) ist. Auf diese Matrix $\bar{A}$ (bzw. das transformierte LGS) wendet man dann das cg-Verfahren an. Daraus ergibt sich unmittelbar, dass $\bar{A}$ symmetrisch sein muss.

[11] vgl. Definition 6.3.8 für reelwertige Polynome.

Dazu sei $B \in \mathbb{R}^{n\times n}$ s.p.d., dann ist auch $B^{-\frac{1}{2}}$ s.p.d., vgl. Definition 4.6.7. Wir transformieren $Ax = b$ dann wie folgt

$$Ax = b \iff B^{-\frac{1}{2}}Ax = B^{-\frac{1}{2}}b \iff (B^{-\frac{1}{2}}AB^{-\frac{1}{2}})(B^{\frac{1}{2}}x) = B^{-\frac{1}{2}}b \iff \bar{A}\bar{x} = \bar{b}$$

mit $\bar{A} := B^{-\frac{1}{2}}AB^{-\frac{1}{2}}$, $\bar{x} := B^{\frac{1}{2}}x$ und $\bar{b} := B^{-\frac{1}{2}}b$. Offenbar ist $\bar{A}$ wieder s.p.d. Die Matrix B nennt man *Vorkonditionierer* (manchmal auch *Präkonditionierer*) für A. Dieser soll i.W. zwei (i.A. konkurrierende) Eigenschaften haben:

1. Die Anwendung von B^{-1} auf einen Vektor soll effizient sein, nach Möglichkeit in $\mathcal{O}(n)$ Operationen (B^{-1} wird dabei *nicht* berechnet);
2. $\kappa(\bar{A}) = \kappa(B^{-\frac{1}{2}}AB^{-\frac{1}{2}}) \ll \kappa(A)$.

Mit der Konstruktion geeigneter Vorkonditionierer werden wir uns später beschäftigen. Wir beginnen mit der Anwendung des cg-Verfahrens auf das transformierte LGS. Wir wenden also *formal* das cg-Verfahren auf das transformierte System $\bar{A}\bar{x} = \bar{b}$ an, möchten dies aber zurückrechnen auf eine Iteration für das ursprüngliche System $Ax = b$. Mit $\bar{x}^{(k)} := B^{\frac{1}{2}}x^{(k)}$ ergibt sich

$$\bar{r}^{(k)} = \bar{b} - \bar{A}\bar{x}^{(k)} = B^{-\frac{1}{2}}b - B^{-\frac{1}{2}}AB^{-\frac{1}{2}}B^{\frac{1}{2}}x^{(k)} = B^{-\frac{1}{2}}(b - Ax^{(k)}) = B^{-\frac{1}{2}}r^{(k)}.$$

Die neuen Suchrichtungen $\bar{d}^{(k)}$ sind nun so zu wählen, dass diese $\bar{A}$-konjugiert sind, also

$$\delta_{k,\ell} \overset{!}{=} \langle \bar{d}^{(k)}, \bar{d}^{(\ell)}\rangle_{\bar{A}} = \langle \bar{d}^{(k)}, B^{-\frac{1}{2}}AB^{-\frac{1}{2}}\bar{d}^{(\ell)}\rangle = \langle B^{-\frac{1}{2}}\bar{d}^{(k)}, AB^{-\frac{1}{2}}\bar{d}^{(\ell)}\rangle.$$

Dies motiviert die Definition $\bar{d}^{(k)} := B^{\frac{1}{2}}d^{(k)}$, denn dann erhalten wir die Konjugiertheit aus $\langle B^{-\frac{1}{2}}\bar{d}^{(k)}, AB^{-\frac{1}{2}}\bar{d}^{(\ell)}\rangle = \langle d^{(k)}, Ad^{(\ell)}\rangle$. Sind also die Richtungen $d^{(k)}$ konjugiert bzgl. A, so sind $\bar{d}^{(k)}$ konjugiert bzgl. $\bar{A}$. Nun rechnen wir die übrigen Größen aus dem cg-Algorithmus um. Für die Schrittweite gilt

$$\bar{\lambda}^{(k)} = \frac{\langle \bar{r}^{(k)}, \bar{d}^{(k)}\rangle}{\langle \bar{d}^{(k)}, \bar{A}\bar{d}^{(k)}\rangle} = \frac{\langle r^{(k)}, d^{(k)}\rangle}{\langle d^{(k)}, Ad^{(k)}\rangle} = \lambda^{(k)},$$

also bleibt die Schrittweite gleich: $\bar{\lambda}^{(k)} = \lambda^{(k)}$. Weiter gilt

$$B^{\frac{1}{2}}x^{(k+1)} = \bar{x}^{(k+1)} = \bar{x}^{(k)} + \bar{\lambda}^{(k)}\bar{d}^{(k)} = B^{\frac{1}{2}}x^{(k)} + \lambda^{(k)}B^{\frac{1}{2}}d^{(k)} = B^{\frac{1}{2}}(x^{(k)} + \lambda^{(k)}d^{(k)}),$$

und da $B^{\pm\frac{1}{2}}$ invertierbar ist, ergibt sich $x^{(k+1)} = x^{(k)} + \lambda^{(k)}d^{(k)}$ genau wie beim cg-Verfahren für A. Für das Residuum gilt:

$$\begin{aligned} B^{-\frac{1}{2}}r^{(k+1)} = \bar{r}^{(k+1)} = \bar{r}^{(k)} - \bar{\lambda}^{(k)}\bar{A}\bar{d}^{(k)} &= B^{-\frac{1}{2}}r^{(k)} - \lambda^{(k)}B^{-\frac{1}{2}}Ad^{(k)} \\ &= B^{-\frac{1}{2}}(r^{(k)} - \lambda^{(k)}Ad^{(k)}), \end{aligned}$$

also $r^{(k+1)} = r^{(k)} - \lambda^{(k)}Ad^{(k)}$, wiederum genau wie beim cg-Verfahren für A. Für die Suchrichtungen gilt

$$B^{\frac{1}{2}}d^{(k+1)} = \bar{d}^{(k+1)} = \bar{r}^{(k+1)} - \frac{\langle \bar{r}^{(k+1)}, \bar{A}\bar{d}^{(k)}\rangle}{\langle \bar{d}^{(k)}, \bar{A}\bar{d}^{(k)}\rangle}\,\bar{d}^{(k)}$$

$$
\begin{aligned}
&= B^{-\frac{1}{2}} r^{(k+1)} - \frac{\langle B^{-1} r^{(k+1)}, Ad^{(k)} \rangle}{\langle d^{(k)}, Ad^{(k)} \rangle} B^{\frac{1}{2}} d^{(k)} \\
&= B^{\frac{1}{2}} \left[B^{-1} r^{(k+1)} - \frac{\langle B^{-1} r^{(k+1)}, Ad^{(k)} \rangle}{\langle d^{(k)}, Ad^{(k)} \rangle} d^{(k)} \right]
\end{aligned}
$$

und wiederum aufgrund der Invertierbarkeit von $B^{\pm\frac{1}{2}}$

$$
d^{(k+1)} = B^{-1} r^{(k+1)} - \frac{\langle B^{-1} r^{(k+1)}, Ad^{(k)} \rangle}{\langle d^{(k)}, Ad^{(k)} \rangle} d^{(k)} .
$$

Insgesamt ergibt sich folgendes *pcg-Verfahren*, bei dem wir die Hilfsvariable $q^{(k)} := B^{-1} r^{(k)}$ einführen, um zu erreichen, dass der Vorkonditionierer nur einmal pro Iteration angewendet werden muss. Für die erste Suchrichtung gilt dann

$$
B^{\frac{1}{2}} d^{(1)} = \bar{d}^{(1)} = \bar{r}^{(0)} = B^{-\frac{1}{2}} r^{(0)} = B^{\frac{1}{2}} \big(B^{-1} r^{(0)} \big) = B^{\frac{1}{2}} q^{(0)},
$$

also $d^{(1)} = q^{(0)}$.

Algorithmus 6.6.1 — pcg-Verfahren.

Input: $A, B \in \mathbb{R}^{n\times n}$ s.p.d., $b \in \mathbb{R}^n$; Startwert $x^{(0)} \in \mathbb{R}^n$

$r^{(0)} := b - Ax^{(0)}$
$q^{(0)} := B^{-1} r^{(0)}$, $d^{(1)} := q^{(0)}$
$\varrho^{(0)} := \langle q^{(0)}, r^{(0)} \rangle$
for $k = 0, 1, 2, \dots$ **do**
  if $\|r^{(k)}\| <$ tol **then**
    stop
  end if
  $a^{(k+1)} := Ad^{(k+1)}$
  $\lambda^{(k+1)} := \frac{\varrho^{(k)}}{\langle d^{(k+1)}, a^{(k+1)} \rangle}$
  $x^{(k+1)} := x^{(k)} + \lambda^{(k+1)} d^{(k+1)}$
  $r^{(k+1)} := r^{(k)} - \lambda^{(k+1)} a^{(k+1)}$
  $q^{(k+1)} := B^{-1} r^{(k+1)}$
  $\varrho^{(k+1)} := \langle q^{(k+1)}, r^{(k+1)} \rangle$
  $d^{(k+2)} := q^{(k+1)} + \frac{\varrho^{(k+1)}}{\varrho^{(k)}} d^{(k+1)}$
end for

Pro Iterationsschritt benötigt dieser Algorithmus gegenüber dem cg-Verfahren nur eine zusätzliche Multiplikation mit der Matrix B^{-1} . Doch dieser zusätzliche Aufwand rentiert sich, sofern $\kappa(\bar{A}) \ll \kappa(A)$.

Beispiele von Vorkonditionierern

Die Forderung $\kappa(\bar{A}) \ll \kappa(A)$ bedeutet, dass B^{-1} eine gute Näherung von A^{-1} sein soll, insbesondere so, dass die Spektren nahe beieinander liegen. Es sei noch einmal betont, dass wir B^{-1} nicht als Matrix benötigen, sondern lediglich die Anwendung $B^{-1} r^{(k+1)}$ im Algorithmus zu realisieren haben.

Diagonalskalierung

■ **Beispiel 6.6.2** Eine sehr einfache, aber häufig schon wirkungsvolle Vorkonditionierung einer s.p.d. Matrix A mit nichtverschwindenden Diagonalelementen liefert die Wahl $B^{-1} := D^{-1}$ mit $D = \operatorname{diag}(A)$. Man spricht in diesem Fall von *diagonaler Vorkonditionierung* (oder auch *Diagonalskalierung*). ■

Iterative Verfahren

Wir erkennen in Algorithmus 6.6.1, dass wir den Vorkonditionierer stets auf das Residuum anwenden, also $q^{(k)} := B^{-1}r^{(k)}$. Nun soll B eine gute Approximation an A sein, also ist $q^{(k)}$ eine Approximation der Lösung des linearen Gleichungssystems $Ae^{(k)} = r^{(k)}$, dessen exakte Lösung der Fehler $e^{(k)} = x^* - x^{(k)}$ ist. Auf dieses Gleichungssystem $Ae = r$ (wir lassen den Iterationsindex k zu besseren Lesbarkeit weg) wenden wir ein klassisches Iterationsverfahren (Splitting-Verfahren) aus §6.1 an. Sei dazu B_{Split} die in (6.6) gewählte Matrix und $C_{\text{Split}} = I - B_{\text{Split}}^{-1}A$ die sich daraus ergebene Iterationsmatrix, dann erhalten wir die Iterierten für $\nu \in \mathbb{N}$

$$\tilde{e}^{(\nu)} = C_{\text{Split}}\,\tilde{e}^{(\nu-1)} + B_{\text{Split}}^{-1}\,r = \cdots = C_{\text{Split}}^{\nu}\,\tilde{e}^{(0)} + \left(\sum_{\ell=0}^{\nu-1} C_{\text{Split}}^{\ell} B_{\text{Split}}^{-1}\right) r =: C_{\text{Split}}^{\nu}\,\tilde{e}^{(0)} + B_{\text{Iterative}}^{-1}\,r.$$

Wählt man nun als Startvektor $\tilde{e}^{(0)} := 0$ (was sinnvoll erscheint, da man zumindest hoffen kann, dass der Fehler klein ist, der Nullvektor also eine gute Approximation darstellt[12]), dann ergibt sich formal der Vorkonditionierer zu

$$B_{\text{Iterative}} := \left(\sum_{\ell=0}^{\nu-1} C_{\text{Split}}^{\ell}\, B_{\text{Split}}^{-1}\right)^{-1}. \tag{6.50}$$

Wir erinnern daran, dass $B_{\text{Iterative}}$ s.p.d. sein muss. Dies werden wir für die einzelnen Splitting-Verfahren im Anschluss untersuchen.

Es sei betont, dass wir $B_{\text{Iterative}}$ *nicht als Matrix* realisieren. Vielmehr wählt man die Anzahl der Schritte ν fest (typischerweise klein, oft maximal $\nu = 5$) und verwendet das gewählte klassische Iterationsverfahren mit rechter Seite $r^{(k)}$ und Startwert Null.

■ **Beispiel 6.6.3 — Jacobi-Verfahren.** Für das Jacobi-Verfahren lautet die Iterationsmatrix $C_{\text{J}} := I - D^{-1}A = -D^{-1}(L+R)$ und $B_{\text{J}} = D$. Beide Matrizen sind symmetrisch (falls A symmetrisch ist), also ist der entstehende Vorkonditionierer in (6.50) ebenfalls symmetrisch und wir können das Jacobi-Verfahren als Vorkonditionierer verwenden. ■

■ **Beispiel 6.6.4 — Gauß-Seidel-Verfahren.** Beim Gauß-Seidel-Verfahren gilt $C_{\text{GS}} = -(L+D)^{-1}R$ bzw. $\widetilde{C}_{\text{GS}} = -(D+R)^{-1}L$. Beide Matrizen sind nicht symmetrisch. Wir können also nicht hoffen, dass (6.50) symmetrisch ist. Hingegen besitzt das symmetrische Gauß-Seidel-Verfahren aus §6.1.5 eine symmetrische Iterationsmatrix, nämlich

$$B_{\text{SGS}} = (D+R)^{-1}L(D+L)^{-1}R \qquad \text{sowie} \qquad C_{\text{SGS}} = I - B_{\text{SGS}}^{-1}A,$$

die beide symmetrisch sind. Also ist (6.50) symmetrisch. ■

[12] Man überlege sich, was geschieht, wenn man $q^{(k-1)}$ aus dem vorherigen Schritt als Startwert verwendet – probieren Sie es aus!

Beispiel 6.6.5 Für Matrizen A_h, die aus der Diskretisierung von elliptischen partiellen Differenzialgleichungen der Gitterweite $h > 0$ herrühren (wie unsere Standardmatrix in den Beispielen 3.0.1 und 6.0.2) ist bekannt, dass $\kappa(A_h) = \mathcal{O}(h^{-2d})$ für $h \to 0+$ und d ist die Raumdimension. Die Kondition wird also stark verstärkt, wenn die Schrittweite h verkleinert wird. Macht man also h klein, um eine höhere Genauigkeit der Approximation zu erreichen, dann wird die Kondition des Gleichungssystems immer schlechter. Dies wirkt sich dann sowohl auf die Anzahl der notwendigen Iterationen als auch auf die Fehlerverstärkung im Algorithmus (Beispiel 2.1.8) aus.

Für solche Probleme gibt es jedoch (asymptotisch) *optimale* Vorkonditionierer:

$$\kappa(B_h^{-1/2} A_h B_h^{-1/2}) = \mathcal{O}(1)\ ,\ h \to 0,$$

z.B. den sogenannten BPX-Vorkonditionierer (Oswald 1988 und Bramble, Pascial, Xu 1989), Wavelet-Vorkonditionierer (Dahmen, Kunoth 1992) als auch Mehrgitter-Verfahren (vgl. Vorlsung *High Performance Computing 2*). Die obige Aussage bedeutet, dass man einen Anfangsfehler um einen Faktor reduzieren kann mit einer Anzahl von pcg-Schritten, die *unabhänging* von der Gitterweite h (und damit der Matrixdimension) ist. ■

Bemerkung 6.6.6 Es ist klar, dass die Wahl des Vorkonditionierers B elementar von den Eigenschaften der Matrix A abhängt. Die Konstruktion eines geeigneten Vorkonditionierers ist daher oft nicht leicht. Hat man aber einen gefunden, so erhöht sich die Konvergenzgeschwindigkeit beträchtlich, was besonders für große lineare Gleichungssysteme von Bedeutung ist.

Numerische Vergleiche

Wir wollen die bislang beschriebenen Verfahren anhand eines numerischen Experiments vergleichen. Dazu wählen wir aus der MATLAB®-Gallery die Matrix `poisson(n)` aus, eine Block-Tridiagonalmatrix, die aus der Diskretisierung der Auslenkung der Membran auf dem Einheitsquadrat entsteht. Wir wählen $n = 10000$.

Wir vergleichen das Jacobi- und Gauß-Seidel-Vefahren mit dem cg-Verfahren sowie zwei Varianten des vorkonditionierten cg-Verfahrens, einmal mit dem Diagonal-Vorkonditiionierer und das andere Mal mit der unvollständigen LR-Zerlegung (vgl. Übungen). Iterationszahlen und Rechenzeiten sind in Tabelle 6.1 dargestellt, die Konvergenzhistorien in Abbildung 6.6. Die Ergebnisse belegen eindrucksvoll die Leistungsfähigkeit des pcg-Verfahrens.

Verfahren	pcg (ilu)	pcg (diag)	cg	Jac	GS
Iterationen	57	160	160	1000*	1000*
CPU [sec]	0.275	14.540	25.233	1322.365	1334.948

Tabelle 6.1: Iterationszahlen und Rechenzeiten für verschiedene Verfahren. Die Abbruchtoleranz ist auf `tol` $= 10^{-6}$ und die maximale Iterationszahl auf 1000 gesetzt. Jacobi- und Gauß-Seidel-Verfahren erreichen also die Abbruchtoleranz nicht für die vorgegebene Maximalanzahl von Iterationen.

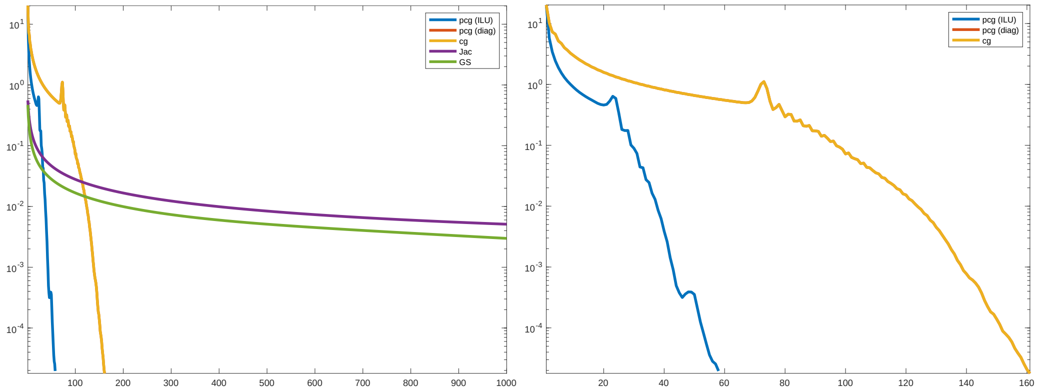


Abbildung 6.6: Konvergenzhistorien aller Verfahren (links) sowie der (p)cg-Varianten (rechts). Die Verläufe von cg und Diagonalskalierung sind gleich, da hier die Diagonale konstant ist.

6.7 Das GMRES-Verfahren

Die GMRES-Methode (engl. *generalized minimal residual*) überträgt die Idee des cg-Verfahrens auf allgemeine Matrizen, die nicht s.p.d. sein müssen. Wir erinnern an Satz 6.5.4, der besagt, dass die cg-Iterierten die jeweils besten Approximationen an die Lösung x^* in den affinen Krylov-Räumen $x^{(0)} + \mathcal{K}_k(A, r^{(0)})$ sind. Dies möchte man auf unsymmetrische Matrizen übertragen. Wir greifen dazu auf das Arnoldi-Verfahren aus §5.5 zurück, mit dessen Hilfe wir eine ON-Basis eines beliebigen Krylov-Raums $\mathcal{K}_k(A, v)$ berechnen können. Die gesuchte GMRES-Näherung wird dann mithilfe der Petrov-Galerkin-Orthogonalität (6.41) bestimmt:

$$\text{bestimme } x^{(k)} \in x^{(0)} + \mathcal{K}_k(A, r^{(0)}) \quad \text{mit} \quad \langle r^{(k)}, Ad^{(j)} \rangle = 0, \; j = 0, \dots, k-1, \tag{6.51}$$

also erhalten wir eine Projektionsmethode im Sinne von Definition 6.3.4 mit

$$\mathcal{V}_k = \mathcal{K}_k(A, r^{(0)}) =: \mathcal{K}_k \quad \text{und} \quad \mathcal{W}_k = A\mathcal{K}_k.$$

Definition 6.7.1 Ausgehend von einem Startwert $x^{(0)} \in \mathbb{R}^n$ sind die Iterierten $x^{(k)}$ des GMRES-Verfahrens definiert durch

$$x^{(k)} \in x^{(0)} + \mathcal{K}_k(A, r^{(0)}) \qquad \text{mit} \qquad x^{(k)} := \arg \min_{x \in x^{(0)} + \mathcal{K}_k(A, r^{(0)})} \|Ax - b\|_2.$$

Die Projektionsmethode ist identisch mit der Minimierung der Funktion $R(x) := \|b - Ax\|_2^2$ in affinen Krylov-Räumen.

Proposition 6.7.2 Es gilt $\tilde{x} = \arg \min_{x \in x^{(0)} + \mathcal{K}_k} R(x)$ genau dann, wenn $(b - A\tilde{x}) \perp A\mathcal{K}_k$.

Beweis. Es seien $x, \tilde{x} \in x^{(0)} + \mathcal{K}_k$ mit $x = x^{(0)} + z$ und $\tilde{x} = x^{(0)} + \tilde{z}$. Dann gilt

$$\begin{aligned} R(x) - R(\tilde{x}) &= \langle b - Ax, b - Ax \rangle - \langle b - A\tilde{x}, b - A\tilde{x} \rangle \\ &= \langle b - A(x^{(0)} + z), b - A(x^{(0)} + z) \rangle - \langle b - A(x^{(0)} + \tilde{z}), b - A(x^{(0)} + \tilde{z}) \rangle \end{aligned}$$

$$\begin{aligned}
&= \langle Az, Az\rangle + 2\langle Ax^{(0)} - b, Az\rangle + \langle Ax^{(0)} - b, Ax^{(0)} - b\rangle \\
&\qquad - \langle A\tilde{z}, A\tilde{z}\rangle - 2\langle Ax^{(0)} - b, A\tilde{z}\rangle - \langle Ax^{(0)} - b, Ax^{(0)} - b\rangle \\
&= \langle Az, Az\rangle - \langle A\tilde{z}, A\tilde{z}\rangle + 2\langle Ax^{(0)} - b, A(z-\tilde{z})\rangle \\
&= \langle A(z-\tilde{z}), A(z-\tilde{z})\rangle + 2\langle A\tilde{z}, A(z-\tilde{z})\rangle \\
&\qquad + 2\langle Ax^{(0)} - b, A(z-\tilde{z})\rangle \\
&= \langle A(z-\tilde{z}), A(z-\tilde{z})\rangle + 2\langle A\tilde{z} + Ax^{(0)} - b, A(z-\tilde{z})\rangle \\
&= \|A(z-\tilde{z})\|_2^2 + 2\langle A\tilde{x} - b, A(z-\tilde{z})\rangle. \qquad (6.52)
\end{aligned}$$

Angenommen, des gelte $(b - A\tilde{x}) \perp A\mathcal{K}_k$, dann verschwindet der letzte Term in (6.52), also $R(x) - R(\tilde{x}) = \|A(z-\tilde{z})\|_2^2 > 0$ falls $\tilde{x} \neq x$. Umgekehrt sei $\tilde{x} = \arg \min_{x \in x^{(0)} + \mathcal{K}_k} R(x)$, also $R(\tilde{x}) < R(x)$ für alle $x \in x^{(0)} + \mathcal{K}_k$. Nach (6.52) gilt aber $R(x) = R(\tilde{x}) + \|A(z-\tilde{z})\|_2^2 + 2\langle A\tilde{x} - b, A(z-\tilde{z})\rangle$. Würde nicht gelten, dass $(b - A\tilde{x}) \perp A\mathcal{K}_k$, dann könnte man ein z so konstruieren, dass $\|A(z-\tilde{z})\|_2^2 + 2\langle A\tilde{x} - b, A(z-\tilde{z})\rangle < 0$, was im Widerspruch zur Optimalität stehen würde. ∎

Im Folgenden verwenden wir die Notationen des Arnoldi-Verfahrens aus §5.5. Wir verwenden als Startvektor v für das Arnoldi-Verfahren in Algorithmus 5.5.2 das Residuum $r^{(0)} = b - Ax^{(0)}$ des LGS zu einem Startwert $x^{(0)}$. Wir suchen eine ON-Basis für den k-dimensionalen Krylov-Raum $\mathcal{K}_k = \mathcal{K}(A, r^{(0)})$. Das Arnoldi-Verfahren bricht nach m Schritten ab (siehe Zeile 8 in Algorithmus 5.5.2, es gilt dann $\mathcal{K}_m = \mathcal{K}_{m+1}$, also ist m die Maximalanzahl linear unabhängiger Krylov-Vektoren). Wäre $m < k$, dann bricht das GMRES-Verfahren ab (und man kann zeigen, dass die Iterierte dann bei exakter Rechnung die gesuchte Lösung $A^{-1}b$ ist). Also können wir stets $m \geq k$ annehmen.

Das Arnoldi-Verfahren in Algorithmus 5.5.2 liefert also eine ON-Basis $\{q^{(1)}, \dots, q^{(k)}\}$ für jeden Krylov-Raum $\mathcal{K}_k = \mathcal{K}_k(A, r^{(0)})$, $k = 1, \dots, m$. Wir setzen die so erhaltenen Vektoren zu Matrizen $Q_k := (q^{(1)}, \dots, q^{(k)}) \in \mathbb{R}^{n \times k}$, $k \leq m$, zusammen. Sei $H_k \in \mathbb{R}^{(k+1) \times k}$ die Hessenberg-Matrix aus Satz 5.5.5, dann gilt $AQ_k = Q_{k+1} H_k$ für $k = 1, \dots, m-1$. Insbesondere gilt $q^{(1)} = \frac{1}{\|r^{(0)}\|_2} r^{(0)}$. Mit $\beta := \|r^{(0)}\|_2$ gilt dann

$$Q_{k+1}^T r^{(0)} = Q_{k+1}^T (\beta q^{(1)}) = \beta\, e_1$$

mit dem ersten kanonischen Eigenvektor e_1.

Wählt man als Startwert $x^{(0)} := 0$, so ist $r^{(0)} = b$ und die Iterierten des GMRES-Verfahrens liegen wegen $x^{(0)} + \mathcal{K}_k = \mathcal{K}_k(A, b)$ im Krylov-Raum $\mathcal{K}_k(A, b)$. Fü das so entstehende GMRES-Verfahren gilt dann die folgende Aussage, [Pla04, §11].

Satz 6.7.3 Sei $x^{(0)} = 0$. Dann gilt für die Iterierten des GMRES-Verfahrens $x^{(k)} = Q_k z^{(k)}$, $k = 1, \dots, m$ genau dann, wenn

$$z^{(k)} = \arg \min_{z \in \mathbb{R}^n} \left\| H_k z - \beta\, e_1 \right\|_2, \qquad \beta := \|b\|_2$$

mit dem Einheitsvektor $e_1 = (1, 0, \dots, 0)^T \in \mathbb{R}^{\min\{n, k+1\}}$.

Beweis. Sei m die Maximalanzahl orthogonaler Vektoren aus dem Arnoldi-Verfahren in

Algorithmus 5.5.2. Dann gilt für $k \le m-1$ und beliebiges $z \in \mathbb{R}^n$ gilt mit Satz 5.5.5

$$\|AQ_k z - b\|_2 = \|Q_{k+1}H_k z - Q_{k+1}^T Q_{k+1} b\|_2 = \|H_k z - \beta\, e_1\|_2 \tag{6.53}$$

und dies gilt analog auch für $k = m$ mit Q_k anstelle von Q_{k+1}. ■

Nun kommen wir zur Situation mit einem beliebigen Startwert $x^{(0)}$.

Satz 6.7.4 Für die Iterierten des GMRES-Verfahrens gilt $x^{(k)} = x^{(0)} + \tilde{x}^{(k)}$ mit $\mathcal{K}_k \ni \tilde{x}^{(k)} = Q_k z^{(k)}$, $k = 1, \dots, m$ genau dann, wenn

$$z^{(k)} = \arg\min_{z \in \mathbb{R}^n} \left\| H_k z - \beta\, e_1 \right\|_2, \qquad \beta := \|r^{(0)}\|_2$$

mit dem Einheitsvektor $e_1 = (1, 0, \dots, 0)^T \in \mathbb{R}^{\min\{n,k+1\}}$.

Beweis. Zunächst gilt mit Satz 5.5.5 ($AQ_k = Q_{k+1}H_k$)

$$\begin{aligned} b - Ax^{(k)} &= b - Ax^{(0)} - A\tilde{x}^{(k)} = r^{(0)} - AQ_k z^{(k)} = \beta q^{(1)} - Q_{k+1}H_k z^{(k)} \\ &= Q_{k+1}(\beta e_1 - H_k z^{(k)}) \end{aligned}$$

Da die Spalten von Q_{k+1} orthonormal sind, gilt

$$\|b - Ax^{(k)}\|_2 = \|Q_{k+1}(\beta e_1 - H_k z^{(k)})\|_2 = \|\beta e_1 - H_k z^{(k)}\|_2 \tag{6.54}$$

und damit die Behauptung. ■

Bemerkung 6.7.5
(i) Die Verwendung des Arnoldi-Verfahrens hat den Vorteil, dass die Faktoren, die zur Orthonormalisierung im Laufe der Iteration berechnet wurden, wiederverwendet werden können.
(ii) Der Nachteil des Verfahrens besteht darin, dass der Aufwand mit wachsender Anzahl von Schritten steigt, da die Orthogonalisierung für alle Schritte vom Anfang an durchgeführt werden muss (vgl. Zeilen 9 bis 12 in Algorithmus 6.7.6). Daher wird in der Praxis oft ein *Restart* durchgeführt, s.u.
(iii) Das Verfahren kann leicht vorkonditioniert werden.
(iv) Andere Verfahren zur Lösung von LGS mit nicht-symmetrischen, dünnbesetzten Matrizen sind das *BiCGStab*- und das *CGS*-Verfahren, vgl. z.B. [Mei15].

Algorithmus 6.7.6 — Vorkonditioniertes GMRES-Verfahren (Basis-Version).

Input: $A, B \in \mathbb{R}^{n\times n}$ regulär, $b \in \mathbb{R}^n$; Startwert $x^{(0)} \in \mathbb{R}^n$

1: $r^{(0)} := b - Ax^{(0)}$
2: $\beta := \|r^{(0)}\|_2$
3: **if** $\beta = 0$ **then**
4:     **exit**;
5: **end if**
6: $q^{(1)} := \frac{1}{\beta} r^{(0)}$

7: **for** $k = 1, 2, \dots$ **do**
8: $\quad w^{(k)} := AB^{-1}q^{(k)}$
9: $\quad$ **for** $i = 1, \dots, k$ **do**
10: $\quad\quad h_{i,k} := \langle q^{(i)}, w^{(k)} \rangle$
11: $\quad\quad w^{(k)} := w^{(k)} - h_{i,k}q^{(i)}$
12: $\quad$ **end for**
13: $\quad h_{k+1,k} = \|w^{(k)}\|_2$;
14: $\quad$ **if** $h_{k+1,k} = 0$ **then**
15: $\quad\quad m := k$
16: $\quad\quad$ **break**;
17: $\quad$ **end if**
18: $\quad q^{(k+1)} = \frac{1}{h_{k+1,k}} w^{(k)}$
19: **end for**
20: $z^{(m)} := \arg\min_z \|H_m z - \beta e_1\|_2$;
21: $x^{(m)} := x^{(0)} + B^{-1}Q_m z^{(m)}$;

Bemerkung 6.7.7 (a) Das lineare Ausgleichsproblem in Zeile 20 hat niedrige Dimension und die Matrix H_m hat Hessenberg-Gestalt. Daher bieten sich Givens-Rotationen zur Lösung an. Dies kann man auch zu einer Modifikation des GMRES-Verfahrens nutzen, bei der das lineare Ausgleichsproblem in jedem Schritt aufdatiert wird, vgl. [Kan05, §6.2].

(b) Auch das GMRES-Verfahren ist bei exakter Arithmetik eine direkte Methode.

(c) Algorithmus 6.7.6 verwendet die modifizierte Gram-Schmidt-Orthogonalisierung im Arnoldi-Verfahren. Eine numerisch stabilere Variante verwendet eine Householde-Orthogonalisierung, vgl. [Saa03, §6.5.2].

(d) In der Praxis hat Algorithmus 6.7.6 aber entscheidende Nachteile:
   (i) es gibt keinen expliziten Zugang zu jeder Iterierten $x^{(k)}$, $1 \le k < m$;
   (ii) falls $A \in \mathbb{R}^{n \times n}$ für $n \gg 1$ dünn besetzt ist, würde man gerne einen optimalen Aufwand von $\mathcal{O}(n)$ zur Lösung des LGS erhalten. Die Matrix H_m in Zeile 20 ist aber eine (üblicherweise nicht mehr dünnbesetzte) obere Hessenbergmatrix, wodurch das Lösen des linearen Ausgleichsproblem zu aufwändig wird, um $\mathcal{O}(n)$ zu realisieren.
   (iii) Ein weiteres Problem ergibt sich durch die Matrix Q_m in Zeile 21. Ist $n \gg 1$, so passt eine dünn besetzte Systemmatrix $A \in \mathbb{R}^{n \times n}$ noch in den Speicher. Da aber Q_m üblicherweise vollbesetzt ist, ergeben sich Speicherprobleme.

(e) Ein Ausweg ist es, die Dimension des Krylov-Raumes und damit die Anzahl der erzeugten Basis-Vektoren im Arnoldi-Algorithmus zu beschränken. Ist ein gewähltes Konvergenzkriterum erfüllt, bricht man ab, falls nicht, beginnt man mit der aktuellen Iterierten als Startwert von vorne. Dies ist der sogenannte GMRES(m)-Algorithmus mit *Restart*. Wir geben diesen in Algorithmus 6.7.8 an.

Algorithmus 6.7.8 — Vorkonditioniertes GMRES(m)-Verfahren (Restart-Version).

Input: $A, B \in \mathbb{R}^{n \times n}$ regulär, $b \in \mathbb{R}^n$; Startvektor $x^{(0)} \in \mathbb{R}^n$; Dimension $m \in \mathbb{N}$;

`restart` $\in \mathbb{N}$
for $r = 1, \ldots,$`restart` **do**
  $r^{(0)} := b - Ax^{(0)}$;
  $\beta := \|r^{(0)}\|_2$;
  if $\beta = 0$ **then**
    exit;
  end if
  $q^{(1)} := \frac{1}{\beta} r^{(0)}$;
  for $k = 1, \ldots, m$ **do**
    $w^{(k)} := AB^{-1}q^{(k)}$;
    for $i = 1, \ldots, k$ **do**
      $h_{i,k} := \langle q^{(i)}, w^{(k)} \rangle$;
      $w^{(k)} := w^{(k)} - h_{i,k} q^{(i)}$;
    end for
    $h_{k+1,k} = \|w^{(k)}\|_2$;
    if $h_{k+1,k} = 0$ **then**
      $m := k$
      break;
    end if
    $q^{(k+1)} = \frac{1}{h_{k+1,k}} w^{(k)}$;
  end for
  $z^{(m)} := \arg\min_z \|H_m z - \beta e_1\|_2$;
  $x^{(m)} := x^{(0)} + B^{-1} Q_m z^{(m)}$;
  if convergence then
    exit;
  end if
  $x^{(0)} := x^{(m)}$
end for

Bemerkung 6.7.9 In [Kan05, S. 236] ist zu lesen: „Unter Verwendung des GMRES(m)-Verfahrens kann man zwar den maximal benötigten Speicherplatz kontrollieren, allerdings besitzt dieses Verfahren im Allgemeinen nicht mehr die Eigenschaft des eigentlichen GMRES-Verfahrens, nach spätestens n Schritten mit einer Lösung abzubrechen. Es ist noch nicht einmal garantiert, dass die durch das GMRES(m)-Verfahren erzeugte Folge gegen die Lösung x^* des linearen Gleichungssystems $Ax = b$ konvergiert.“

Der unten gezeigte (von ChatGBT erzeugte) Code realisiert das GMRES(m)-Verfahren mit Restart.

Zur Fehleranalyse wollen wir die Vorarbeiten zur Projektionsmethode verwenden. Dazu zeigen wir zunächst, dass die Voraussetzung von Lemma 6.3.6 erfüllt ist.

Lemma 6.7.10 Sei $A \in \mathbb{R}^{n\times n}$ regulär, $\mathcal{V}_k = \mathcal{K}_k$ und $\mathcal{W}_k = A\mathcal{K}_k$. Dann ist $W_k^T A V_k$ invertierbar.

Beweis. Nach Konstruktion gilt $W_k = AV_k$ und damit $W_k^T A V_k = V_k^T A^T A V_k = (AV_k)^T AV_k$. Da A regulär ist und V_k vollen Rang hat, ist AV_k injektiv. ∎

In diesem Lemma ist V_k eine beliebige Matrix, deren Spalten eine Basis für $\mathcal{K}_k$ bilden, es muss nicht unbedingt die Matrix Q_k aus dem Arnoldi-Verfahren sein. Man kann das GMRES-Verfahren auch herleiten über die Definition $W_k := AV_k$ und $V_k := Q_k$ aus dem Arnoldi-Verfahren.

Die Projektion aus (6.36) nimmt hier folgende Gestalt an:[13]

$$P_k = I - AV_k(W_k^T AV_k)^{-1} W_k^T = I - AV_k((AV_k)^T AV_k)^{-1}(AV_k)^T = I - \tilde{Q}_k \tilde{Q}_k^+$$

mit $\tilde{Q}_k := AV_k$. Man sieht leicht, dass $\tilde{Q}_k^2 = \tilde{Q}_k$, also ist $\tilde{Q}_k$ und damit auch P_k eine Projektionsmatrix. Somit gilt $\|P_k\|_2 \le 1$ in der Euklidischen Operatornorm und damit folgt aus Satz 6.3.9, dass

$$\|r^{(k)}\|_2 = \min_{p \in \mathbb{P}_k^1} \|p(A) r^{(0)}\|_2 \le \Big(\min_{p \in \mathbb{P}_k^1} \|p(A)\|_2 \Big) \|r^{(0)}\|_2 \tag{6.55}$$

Satz 6.7.11 Sei $A \in \mathbb{R}^{n \times n}$ positiv definit[14] und $r^{(k)}$ das Residuum aus dem GMRES-Verfahren im Schritt k. Dann gilt

$$\|r^{(k)}\|_2 \le \left(1 - \frac{\lambda_{\min}^2 \big(\frac{1}{2}(A^T + A)\big)}{\lambda_{\max}(A^T A)} \right)^{\frac{k}{2}} \|r^{(0)}\|_2. \tag{6.56}$$

Beweis. Für $\alpha > 0$ definiere $p_1(A) := I - \alpha A \in \mathbb{P}_1^1$ und $p_k(A) := (p_1(A))^k \in \mathbb{P}_k^1$, dann gilt

$$\min_{p \in \mathbb{P}_k^1} \|p(A)\|_2 \le \|p_k(A)\|_2 \le \|p_1(A)\|_2^{k}.$$

Wir schätzen die Operatornorm ab

$$\begin{aligned}
\|p_1(A)\|_2^2 &= \sup_{x \neq 0} \frac{\|(I - \alpha A)x\|_2^2}{\|x\|_2^2} = \sup_{x \neq 0} \left\{ 1 - 2\alpha \frac{\langle x, Ax \rangle}{\langle x, x \rangle} + \alpha^2 \frac{\langle Ax, Ax \rangle}{\langle x, x \rangle} \right\} \\
&\le 1 - 2\alpha \, \lambda_{\min}\Big(\tfrac{1}{2}(A^T + A)\Big) + \alpha^2 \, \lambda_{\max}(A^T A).
\end{aligned}$$

Der Ausdruck auf der rechten Seite wird minimal für $\alpha^* = \lambda_{\min}\big(\frac{1}{2}(A^T + A)\big) / \lambda_{\max}(A^T A) > 0$ und setzen wir α^* ein, erhalten wir

$$\|p_1(A)\|_2^2 \le 1 - \frac{\lambda_{\min}^2\big(\frac{1}{2}(A^T + A)\big)}{\lambda_{\max}(A^T A)},$$

und damit die Behauptung. ∎

```
Gib mir bitte einen Matlab-Code für GMRES mit Vorkonditionierung
```

[13] A^+ bezeichnet wieder die Pseudo-Inverse von A, vgl. Definition 4.6.3.

[14] Wir nehmen *nicht* an, dass A symmetrisch ist, vgl. Definition 3.7.1.

Gerne! Hier ist ein einfaches Beispiel für die Verwendung der GMRES-Methode (Generalized Minimal Residual) mit Vorkonditionierung in MATLAB: Dieses Skript definiert eine Funktion my_gmres, die die GMRES-Methode mit Vorkonditionierung implementiert. Sie benötigt die quadratische Matrix A, den Vektor b, die Anzahl der Iterationen vor einem Neustart (restart), die relative Toleranz für den Abbruch des Verfahrens (tol), die maximale Anzahl der Iterationen (maxit) und optional die Vorkonditionierungsmatrix M.
Es wird das Arnoldi-Verfahren verwendet, um die Orthonormalbasis des Krylov-Unterraums zu berechnen und eine Approximation der Lösung zu erhalten. Das Verfahren bricht ab, wenn der relative Fehler kleiner als die angegebene Toleranz ist oder die maximale Anzahl der Iterationen erreicht wird.

MATLAB-Funktion per ChatGBT: my_gmres.m

```
function [x, flag, relres, iter, resvec] = my_gmres(A, b, restart, tol, maxit, M)
    % GMRES mit Vorkonditionierung
    % Eingabe:
    %  A: Die quadratische Matrix A des linearen Gleichungssystems Ax = b
    %  b: Der Vektor b des linearen Gleichungssystems Ax = b
    %  restart: Die Anzahl der Iterationen vor einem Neustart (normalerweise 10-50)
    %  tol: Die relative Toleranz fuer den Abbruch des Verfahrens
    %  maxit: Die maximale Anzahl der Iterationen
    %  M: Die Vorkonditionierungsmatrix (optional)
    %
    % Ausgabe:
    %  x: Die Loesung des linearen Gleichungssystems
    %  flag: Ein Indikator fuer den Abbruchstatus (0: Erfolg, 1: Maximalzahl der Iterationen erreicht)
    %  relres: Das relative Residuum ||b - Ax|| / ||b||
    %  iter: Die Anzahl der durchgefuehrten Iterationen
    %  resvec: Ein Vektor, der die Residuen in jedem Iterationsschritt enthaelt

    % Groesse der Matrix A
    n = size(A, 1);

    % Initialisierung
    x = zeros(n, 1);
    resvec = zeros(maxit, 1);
    flag = 0;
    relres = 1;

    % Pruefen, ob eine Vorkonditionierungsmatrix M angegeben ist
```

```
    if nargin < 6
        M = eye(n); % Keine Vorkonditionierung
    end

    % Residuum berechnen
    r = b - A * x;

    % Vorkonditionierung des Residuums
    r = M \ r;

    % Norm des Vorkonditionierten Residuums berechnen
    beta = norm(r);

    % Pruefen, ob das Startresiduum bereits klein genug ist
    if beta < tol
        return;
    end

    % Schleife ueber die Iterationen
    for iter = 1:maxit
        % GMRES-Schritte
        [v, h] = arnoldi(A, M, r, restart);

        % Loesung des kleineren, unteren Hessenberg-Systems
        y = h(1:restart, 1:restart) \ (beta * eye(restart, 1));

        % Approximierte Loesung berechnen
        x = x + v(:, 1:restart) * y;

        % Berechnung des neuen Residuums
        r = b - A * x;

        % Vorkonditionierung des Residuums
        r = M \ r;

        % Norm des Vorkonditionierten Residuums berechnen
        beta = norm(r);

        % Relativer Fehler berechnen
        relres = beta / norm(b);

        % Residuum speichern
        resvec(iter) = relres;

        % Ueberpruefen der Abbruchbedingung
        if relres < tol
            break;
        end
    end

    % Flag setzen, wenn das Verfahren nicht konvergiert
```

```
    if relres >= tol
        flag = 1;
    end

    % Ausschneiden von ueberschuessigen Eintraegen im Residuen-Vektor
    resvec = resvec(1:iter);
end
```

Hierbei wir das Arnoldi-Verfahren mit Vorkonditionierer verwendet (allerdings wird hier der Vorkonditionierer explizit als Matrix übergeben, was nicht optimal ist):

MATLAB-Funktion per ChatGBT: arnoldi.m

```
function [V, H] = arnoldi(A, M, v, m)
    % Arnoldi-Verfahren
    % Eingabe:
    %   A: Die quadratische Matrix A des linearen Gleichungssystems Ax = b
    %   M: Die Vorkonditionierungsmatrix
    %   v: Der Startvektor fuer das Arnoldi-Verfahren
    %   m: Die Anzahl der Iterationsschritte
    %
    % Ausgabe:
    %   V: Die Orthonormalbasis des Krylov-Unterraums
    %   H: Die obere Hessenberg-Matrix

    % Groesse der Matrix A
    n = size(A, 1);

    % Initialisierung der Vektoren
    V = zeros(n, m + 1);
    H = zeros(m + 1, m);

    % Normalisierung des Startvektors
    V(:, 1) = v / norm(v);

    % Arnoldi-Verfahren Schleife
    for j = 1:m
        % Arnoldi-Schritt
        w = A * V(:, j);

        % Vorkonditionierung des Ergebnisses
        w = M \ w;

        for i = 1:j
            % Hessenberg-Eintraege berechnen
            H(i, j) = V(:, i)' * w;
            w = w - H(i, j) * V(:, i);
        end

        % Naechste Orthonormalbasis-Vektor berechnen
```

```
        H(j + 1, j) = norm(w);
        if H(j + 1, j) == 0
            break;
        end
        V(:, j + 1) = w / H(j + 1, j);
    end

    % Verkleinern der Hessenberg-Matrix auf die tatsaechliche Groesse
    H = H(1:j, :);
end
```

■ **Beispiel 6.7.12 — Konvektion-Diffusion.** Wir greifen noch einmal Beispiel 3.0.1 auf, die schwingende Saite. Wir fügen nun einen Term hinzu, der physikalisch einen Transport (Konvektion) modelliert:

$$-u''(x) + \beta u'(x) = f(x)\ ,\ x \in (0,1), \qquad u(0) = u(1) = 0 \tag{6.57}$$

wobei $\beta \in \mathbb{R}^+$ als Geschwindigkeit interpretiert werden kann.

Wir diskretisieren wieder mittels Finiter Differenzen auf einem äquidistanten Gitter und approximieren die erste Ableitung durch eine Vorwärtsdifferenz, d.h. $u'(x_i) \approx \frac{1}{h}(u_i - u_{i-1})$. Dann erhalten wir $A_h = D_h + N_h$ mit

$$D_h := \frac{1}{h^2}\begin{bmatrix} 2 & -1 & & 0 \\ -1 & 2 & \ddots & \\ & \ddots & \ddots & -1 \\ 0 & & -1 & 2 \end{bmatrix}, \qquad N_h := \frac{\beta}{h}\begin{bmatrix} 1 & -1 & & 0 \\ & 1 & \ddots & \\ & \ddots & \ddots & -1 \\ 0 & & & 1 \end{bmatrix},$$

also ist A_h unsymmetrisch. Mit wachsender Geschwindigkeit β wird die Unsymmetrie immer stärker, man nennt ein solches Problem *konvektionsdominant*. Die Konvergenzhistorie des GMRES-Verfahrens ist in Abbildung 6.7 dargestellt. Wir erkennen die nicht monotone Konvergenz. Es sei jedoch betont, dass hier nur die äußeren Iterationszahlen dargestellt sind und nicht die Gesamtanzahl (die ja auch die inneren Arnoldi-Iterationen beinhaltet). ■

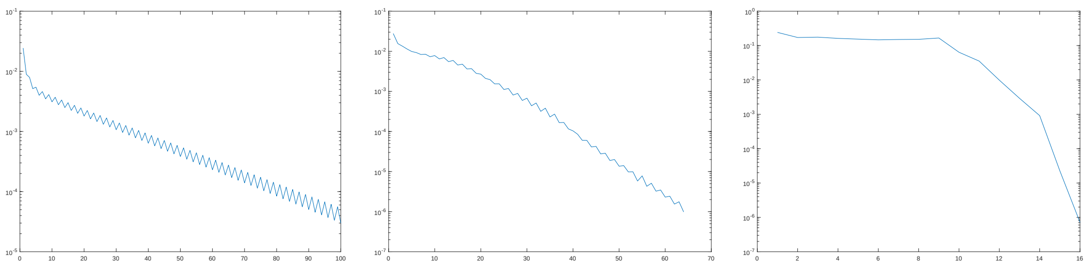


Abbildung 6.7: Konvergenzhistorie des GMRES-Verfahrens mit Restart nach `restart` $= 10$ Iterationen und diagonaler Vorkonditionierung für $n = 100$ und $\beta = 1, 10, 100$ (v.l.n.r.).

Numerische Vergleiche

Wir wollen den effekt der Vorkonditionierung anhand eines numerischen Experiments untersuchen. Dazu wählen wir aus der MATLAB®-Gallery die Matrizen `dorr(n)` (diagonaldominat, schlecht konditioniert, tridiagonal) und `toeppen(n)` (pentadiagonale Töplitz-Matrix) aus. Wir wählen $n = 5000$. Wir vergleichen das GMRES-Verfahren ohne Vorkonditionierung („plain") mit Diagonal- und ILU-Vorkonditionierung. Man beachte, dass in beiden Fällen die ILU exakt ist (es sind Bandmatrizen), also terminiert das vorkonditionierte GMRES-Verfahren nach einem Schritt. Bei der Töplitz-Matrix gibt es Null-Einträge auf der Diagonalen, so dass die Diagonalskalierung nicht funktioniert.

Problem	Dorr			Töplitz		
Vorkonditionierung	plain	diag	ilu	plain	diag	ilu
Iterationen	1708	1709	1	153	—	1
CPU [sec]	51.140	26.745	0.004	5.528	—	0.246

Tabelle 6.2: Iterationszahlen und Rechenzeiten für verschiedene Vorkonditionierer im GMRES-Verfahren für die Dorr- (links) und Töplitz-Matrix (rechts).

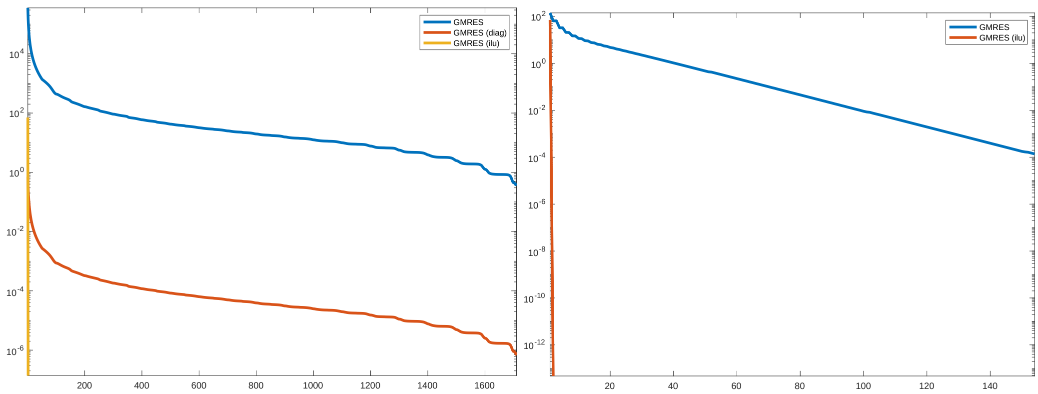


Abbildung 6.8: Konvergenzhistorien für die Dorr- (links) und Töplitz-Matrix (rechts).

Schließlich wollen wir noch eine Matrix testen, bei der die ILU-Zerlegung nicht konstant ist. Dazu wählen wir `wathen(nx,ny)`, eine dünnbesetzte, zufällige Finite Elemente-Matrix. Wir wählen $nx = ny = 60$ und erhalten eine Matrix der Dimension 11401×11401. Wir erkennen den positiven Effekt der Vorkonditionierung deutlich.

Vorkonditionierung	plain	diag	ilu
Iterationen	207	2	3
CPU [sec]	37.404	0.580	0.110

Tabelle 6.3: Iterationszahlen und Rechenzeiten für verschiedene Vorkonditionierer im GMRES-Verfahren für die Wathen-Matrix.

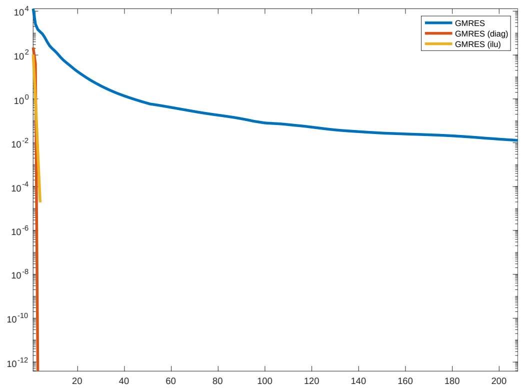


Abbildung 6.9: Konvergenzhistorien für die Wathen-Matrix.

Kapitel 7

Rechnerarithmetik

Eine der Aufgaben der Numerischen Mathematik ist es, Verfahren zur exakten oder näherungsweise Lösung von Problemen bereitzustellen, welche auf Computern realisiert werden. Dazu müssen Zahlen auf dem Rechner dargestellt werden, diese können dort jedoch nur mit endlicher Stellenzahl repräsentiert werden, egal wie leistungsstark der Rechner auch sein mag. Man muß also in geeigneter Weise Rundungen einführen, wodurch unvermeidbare Fehler auftreten. Weiterhin kann selbst bei einfachen Algorithmen eine Akkumulation von Fehlern (*Fehlerverstärkung*) auftreten.

Um ein Resultat einer numerischen Simulation im Hinblick auf seine Genauigkeit beurteilen zu können, ist es unerläßlich, eine Fehleranalyse durchzuführen. Dabei muß man zwischen verschiedenen Fehlertypen unterscheiden. Neben dem eben schon angesprochenen **Rundungsfehler** beeinflussen sowohl **Datenfehler** als auch **Verfahrensfehler** das Resultat einer Rechnung.

Datenfehler treten z.B. durch die Rundung der Eingabedaten auf (π oder $\sqrt{2}$ lassen sich auf einem Rechner mit endlich vielen Stellen nicht darstellen) oder der Input besteht aus gemessenen Werten, wobei Messfehler auftreten. Fehler in den Eingabedaten sind somit häufig unvermeidbar.

Verfahrensfehler können z.B. dadurch entstehen, dass ein numerisches Verfahren auf einer Näherung (Approximation) beruht. So werden wir in der Vorlesung *Numerische Analysis* z.B. die numerische Approximation von Ableitungen und Integralen behandeln. Solche Größen können nur in einfachen Fällen exakt berechnet werden und werden daher approximiert.

Es ist das Ziel dieses Kapitels, die Grundlagen der Rechnerarithmetik darzustellen, auf der das maschinelle Rechnen mit Zahlen beruht. Mit deren Kenntnis lassen sich Möglichkeiten und Grenzen im Arbeiten mit Computern realistisch beurteilen. Wir betonen, dass es sich hierbei um die Betrachtung von *Standardrechnern* auf der Basis des Binärsystems handelt – wollte man auf Quantencomputern arbeiten, wären gänzlich andere mathematische Grundlagen notwendig, [Sch19].

7.1 Einführung

Ein erstes Beispiel soll motivieren, wieso wir uns mit der Frage, wie Zahlen im Rechner dargestellt werden, beschäftigen sollten.

■ **Beispiel 7.1.1** Eine einfache Überlegung zeigt, dass das Ergebnis von

$$m := 4 \cdot 13860^4 - 19601^4 + 2 \cdot 19601^2 \in \mathbb{N}$$

ungerade ist und, dass an der letzten Stelle eine 1 oder 9 stehen muss

$$\begin{aligned} m &= 4(1386 \cdot 10)^4 - (1960 \cdot 10 + 1)^4 + 2(1960 \cdot 10 + 1)^2 \\ &= 4 \cdot 1386^4\ 10^4 - (1960^4\ 10^4 + 4 \cdot 1960^3\ 10^3 + 6 \cdot 1960^2\ 10^2 + 4 \cdot 1960\ 10 + 1) \\ &\quad + 2(1960^2\ 10^2 + 2 \cdot 1960 \cdot 10 + 1) \\ &= 10^4(\cdots) + 10^3(\cdots) + 10^2(\cdots) + 10^1(\cdots) + 10^0(-1+2). \end{aligned}$$

(dabei könnte sich die 9 z.B. durch $-1 \cdot 10 + 1$ ergeben). Rechnet man auch die anderen Stellen aus (per Hand oder mit MAPLE®) so ergibt sich

$$4 \cdot 13860^4 - 19601^4 + 2 \cdot 19601^2 = 1.$$

Überraschenderweise liefert MATLAB® aber ein anderes Resultat.

```
>> myfun = @(x,y) 4*x^4-y^4+2*y^2;
>> myfun(13860,19601)
ans =
     2
```

Auch die folgende Realisierung mittels quadratischer Ergänzung (setze $u := 4x^2 + 1$ und $v := y^2 - 1$, dann gilt $u - v^2 = 4x^4 - y^4 + 2y^2$) liefert ein (anderes) falsches Ergebnis.

```
>> x = 13860; y = 19601;
>> u = 4 * x * x * x * x + 1;
>> v = y * y - 1;
>> u - v * v
ans =
     0
```

Offenbar müssen wir bei der Umsetzung selbst einfacher Formel vorsichtig sein. Das geschilderte Phänomen liegt allerdings nicht an MATLAB®. Versucht man es mit MAPLE®, muss es auch nicht besser sein! Bei exakter Arithmetik funktioniert es:

```
> restart:
  4  *13860^4-19601^4+2*19601^2;                                    1
```

Verwendet man in MAPLE® nach einer Zahl noch einen Punkt (in diesem Fall rechnet MAPLE® nicht mehr mit exakter Arithmetik, sondern verwendet so genannte Gleitkommazahlen, vgl. später – die Anweisung `Digits:=18` führt dazu, dass Gleitkommazahlen mit 18 Stellen verwendet werden), so ergibt sich das folgende scheinbar überraschende Ergebnis.

```
> restart:
  4. *13860^4-19601^4+2*19601^2;                    68398402.
  Digits:=15:
  4. *13860^4-19601^4+2*19601^2;                    402.
  Digits:=16:
  4. *13860^4-19601^4+2*19601^2;                    2.
  Digits:=17:
  4. *13860^4-19601^4+2*19601^2;                    2.
  Digits:=18:
  4. *13860^4-19601^4+2*19601^2;                    1.
```

Wenn wir also die Anzahl der exakten Stellen erhöhen, stimmt das Ergebnis irgendwann. ∎

∎ **Beispiel 7.1.2** Die MATLAB®-Routine `floor` rundet auf die nächstkleinere ganze Zahl. Die Anwendung von `floor` auf die Eingabe 114 sollte also 114 ergeben. Tatsächlich passiert aber folgendes.

```
>> floor(114)
ans =
   114
>> floor(1.14*100)
ans =
   113
```

Es kommen leichte Zweifel an exakter Mathematik auf Computern auf. ∎

7.2 Zahlendarstellung

7.2.1 b-adische Brüche

Wir wollen den beiden überraschenden Beispielen auf den Grund gehen. Dazu betrachten wir die Darstellung von Zahlen auf dem Computer etwas genauer. Der Schlüssel hierfür ist folgende Definition.

Definition 7.2.1 — b-adischer Bruch. Es sei $b \geq 2$ eine natürliche Zahl, $n \in \mathbb{N}_0$, $a_k \in \mathbb{N}_0$ mit $0 \leq a_k < b$ und $a_n \neq 0$. Eine Reihe der Gestalt

$$\pm \sum_{k=-\infty}^{n} a_k b^k$$

wird als b-*adischer Bruch* bezeichnet, b heißt *Basis*.

Falls die Basis festgelegt ist, kann man einen b-adischen Bruch auch durch die Reihung der Ziffern a_k angeben:

$$\pm a_n a_{n-1} \ldots a_1 a_0 \,.\, a_{-1} a_{-2} a_{-3} \ldots . \tag{7.1}$$

Bemerkung 7.2.2 i) Für $b = 10$ spricht man von **Dezimalbrüchen**, für $b = 2$ von **dyadischen Brüchen** und die Babylonier verwendeten ein System zur Basis $b = 60$, was sich heute noch im Verhältnis von Sekunde, Minute und Stunde wieder findet.[1]

ii) Die Basis wird auch als *Grundzahl* bezeichnet, weswegen man auch die Bezeichnung g-adisch findet.

iii) Vorsicht, nicht b-adisch mit p-adisch verwechseln (dabei ist p eine Primzahl)!

iv) Der Null kommt immer eine Sonderrolle zu. Sie wird gesondert betrachtet.

Der nächste Satz besagt, dass sich jede reelle Zahl durch einen b-adischen Bruch darstellen lässt und umgekehrt.

Satz 7.2.3 (a) Jeder b-adische Bruch konvergiert gegen eine reelle Zahl.
(b) Jedes $0 \neq x \in \mathbb{R}$ lässt sich in einen b-adischen Bruch entwickeln.

Beweis. (a) Wir beschränken uns auf den Fall eines nicht-negativen b-adischen Bruchs $\sum_{k=-\infty}^{n} a_k b^k$. Wir definieren für $-\ell \leq n$ die Partialsummen

$$s_\ell := \sum_{k=-\ell}^{n} a_k b^k$$

und wollen zeigen, dass die Folge $(s_\ell)_{-\ell \leq n}$ eine Cauchy-Folge ist. Sei dazu $\varepsilon > 0$ und $L \in \mathbb{N}$ so groß, dass $b^{-L} < \varepsilon$ gelte. Dann gilt für $\ell \geq m \geq L$ wegen $a_k \leq b-1$ mit der geometrischen Reihe

$$\begin{aligned}
|s_\ell - s_m| &= \sum_{k=-\ell}^{n} a_k b^k - \sum_{k=-m}^{n} a_k b^k = \sum_{k=-\ell}^{-m-1} a_k b^k \\
&\leq \sum_{k=-\ell}^{-m-1} (b-1) b^k = (b-1) b^{-m-1} \sum_{k=-\ell+m+1}^{0} b^k = (b-1) b^{-m-1} \sum_{k=0}^{\ell-m-1} b^{-k} \\
&< (b-1) b^{-m-1} \frac{1}{1-b^{-1}} = b^{-m} \leq b^{-L} < \varepsilon
\end{aligned}$$

womit die Behauptung bewiesen ist.

(b) Es genügt, nur den Fall $x \geq 0$ zu betrachten. Zuerst zeigen wir, dass es zu jeder Basis $b \geq 2$ mindestens eine Zahl $k \in \mathbb{N}$ gibt mit $x < b^k$: Nach dem Archimedischen Axiom[2] gibt es ein $k \in \mathbb{N}$ mit $k > (x-1)/(b-1) \in \mathbb{R}$. Mit diesem k und der Bernoullischen Ungleichung[3] erhalten wir $b^k = (1+(b-1))^k \geq 1 + k(b-1) > 1 + x - 1 = x$. Sei

$$n := \min\{k \in \mathbb{N}_0 \,|\, 0 \leq x < b^{k+1}\}.$$

Durch vollständige Induktion konstruieren wir nun eine Folge $(a_\ell)_{-\infty < \ell \leq n}$ natürlicher Zahlen $0 \leq a_{-\ell} < b$, so dass für alle $m \geq -n$ gilt

$$x = \sum_{\ell=-m}^{n} a_\ell b^\ell + \nu_{-m} \quad \text{mit } 0 \leq \nu_{-m} < b^{-m}\,. \tag{7.2}$$

[1]**Sexagesimalsystem**: Zählt man mit den Fingern der linken Hand wie üblich von 1 bis 5 und registriert die an der linken Hand gezählten 5-er Gruppen durch das Drücken des Daumens der rechten Hand auf das passende Fingerglied an der rechten Hand, so kommt man zum $5 \times 12 = 60$-System. Diese Zählweise wurde schon von den Vorfahren der Sumerer im 4. Jahrtausend v. Chr. angewandt. Die Babylonier (1750 v. Chr.) verwendeten dann das Sexagesimalsystem in dem Sinne, dass sie 60 Herzschläge zu einer Minute zusammen fassten und 60 Minuten zu einer Stunde.

[2]Zu je zwei Größen $y > x > 0$ existiert eine natürliche Zahl $n \in \mathbb{N}$ mit $n\,x > y$.

[3]$(1+x)^n \geq 1 + nx$, $n \in \mathbb{N}_0$, $x \in \mathbb{R}$, $x > -1$.

Da notwendigerweise $\lim\limits_{m\to\infty} \nu_{-m} = 0$ gilt, folgt $x = \sum\limits_{\ell=-\infty}^{n} a_\ell b^\ell$ und somit die Behauptung. Nun zum Beweis von (7.2) per Induktion. **Induktionsanfang** $-m = n$**:** Es gilt $0 \le xb^{-n} < b$ nach Definition von n. Somit existiert ein $a_n \in \{0, 1, ..., b-1\}$ und ein $\delta \in \mathbb{R}$ mit $0 \le \delta < 1$, so dass $xb^{-n} = a_n + \delta$ gilt. Mit $\nu_n := \delta b^n$ gewinnt man $x = a_n b^n + \nu_n$ mit $0 \le \nu_n < b^n$.
Induktionsschritt $(-m) \to (-m-1)$**:** Es gilt $0 \le \nu_{-m} b^{m+1} < b$, also gibt es ein $a_{-m-1} \in \{0, 1, ..., b-1\}$ und ein $\delta \in \mathbb{R}$ mit $0 \le \delta < 1$, so dass $\nu_{-m} b^{m+1} = a_{-m-1} + \delta$ gilt. Mit $\nu_{-m-1} := \delta b^{-m-1}$ gewinnt man

$$x = \sum_{\ell=-m}^{n} a_\ell b^\ell + \nu_{-m} = \sum_{\ell=-m}^{n} a_\ell b^\ell + (a_{-m-1} + \delta) b^{-m-1} = \sum_{\ell=-m-1}^{n} a_\ell b^\ell + \nu_{-m-1}\,,$$

wobei $0 \le \nu_{-m-1} < b^{-m-1}$ gilt. Damit ist alles gezeigt. ■

Die b-adische Darstellung ist in der gewählten Form nicht eindeutig, siehe z. B. $1 = 0.\overline{9} = 0.999\ldots$, falls $b = 10$ gewählt ist. Dies behebt der folgende Satz.

Satz 7.2.4 Eine Basis $b \ge 2$ sei gegeben. Dann existieren zu jedem $x \in \mathbb{R}$ eindeutige $\sigma \in \{-1, 1\}$ (Vorzeichen), $n \in \mathbb{N}$ (höchste Potenz) und Koeffizienten $a_k \in \mathbb{N}_0, n \ge k \in \mathbb{Z}, a_n \ne 0$ mit $0 \le a_k < b$, $k \le n$, und $a_k < b - 1$ für unendlich viele $k \le n$, so dass

$$x = \sigma \sum_{k=-\infty}^{n} a_k\, b^k$$

Beweis. Übung. ■

Durch die zusätzliche Bedingung „$a_k < b - 1$ für unendlich viele $k \le n$" erhält man eine eindeutige Darstellung. Somit wäre z.B. für $b = 10$ die Darstellung $0.\overline{9} = 0.999\ldots$ für $x = 1$ nicht zugelassen.

■ **Beispiel 7.2.5** (i) Binärdarstellung ($b = 2$) von $x = (14.625)_{10} = (1110.101)_2$

$$\begin{array}{llllll}
14 = & 7 \cdot 2 & +0 & & & \Rightarrow a_0 = 0 \\
 & 7 & = & 3 \cdot 2 & +1 & \Rightarrow a_1 = 1 \\
 & & & 3 = & 1 \cdot 2 + 1 & \Rightarrow a_2 = 1 \\
 & & & & 1 = 0 \cdot 2 + 1 & \Rightarrow a_3 = 1
\end{array}$$

Im Folgenden bezeichnet $\lfloor\cdot\rfloor$ die untere Gaußklammer, d.h. $\lfloor x \rfloor := \max\{j \in \mathbb{Z} : j \le x\}$.

$$a_{-1} = \lfloor 2 * 0.625 \rfloor = \lfloor 1.25 \rfloor = 1$$
$$a_{-2} = \lfloor 2 * 0.25 \rfloor = \lfloor 0.5 \rfloor = 0$$
$$a_{-3} = \lfloor 2 * 0.5 \rfloor = \lfloor 1 \rfloor = 1$$

(ii) Binärdarstellung von $x = (0.2)_{10} = (0.\overline{0011})_2$

$$a_{-1} = \lfloor 2 * 0.2 \rfloor = \lfloor 0.4 \rfloor = 0$$
$$a_{-2} = \lfloor 2 * 0.4 \rfloor = \lfloor 0.8 \rfloor = 0$$
$$a_{-3} = \lfloor 2 * 0.8 \rfloor = \lfloor 1.6 \rfloor = 1$$
$$a_{-4} = \lfloor 2 * 0.6 \rfloor = \lfloor 1.2 \rfloor = 1$$

$$a_{-5} = \lfloor 2 * 0.2 \rfloor = \ldots \vdots$$

Im ersten Beispiel erhalten wir also eine endliche, im zweiten eine unendliche Binärdarstellung von Zahlen mit endlicher Dezimaldarstellung. ∎

7.2.2 Festpunktzahlen

Da es auf einer endlichen Maschine unmöglich ist, unendliche b-adische Brüche zu speichern, werden Zahlen auf Rechnern durch eine *endliche Entwicklung* approximiert. Damit ist die Idee einfach: wir schneiden die Entwicklung in (7.1) ab und approximieren reelle Zahlen der Form $x = \sigma \sum_{k=-\infty}^{n} a_k\, b^k$ durch

$$\sigma(a_n a_{n-1} \ldots a_1 a_0 \,.\, a_{-1} a_{-2} a_{-3} \ldots a_{-\ell})_b := \sigma \sum_{k=-\ell}^{n} a_k b^k \tag{7.3}$$

mit $\ell < \infty$.

Definition 7.2.6 — Festpunktzahl. Zu gegebener Basis b ist

$$F(\ell, n) := \left\{ \pm \sum_{k=-\ell}^{n} a_k b^k = \pm (a_n \ldots a_1 a_0 \,.\, a_{-1} \ldots a_{-\ell})_b \right\}$$

die Menge der *Festpunktzahlen* mit $(n+1)$ Vor- und ℓ Nachkommastellen.

Bemerkung 7.2.7 (i) Typischerweise werden ganze Zahlen (engl. integers) auf Computern dargestellt mit $\ell = 0$ und $b = 2$.

(ii) In modernen Rechnern werden negative Zahlen häufig als 2-er-Komplement[4] abgespeichert. Dabei ist x das Komplement zu y, wenn in der jeweiligen Arithmetik $x + y = 0$ gilt. Beispiel: Für $b = 2$ und $n = 3$ ergibt sich

$\lvert x \rvert$	x			$-x$		
0	0	0	0			
1	0	0	1	1	1	1
2	0	1	0	1	1	0
3	0	1	1	1	0	1
4	1	0	0	1	0	0

also lautet der darstellbare Bereich $\{-4, \ldots, 3\}$.

(iii) Für Zahlendarstellungen mit 8-, 16- und 32-Bit[5], bzw. 1-, 2- und 4-Byte[6]erhalten wir die Zahlenbereiche

8-Bit, $n = 7 : 0, \ldots, 255$ ohne Vorzeichen
bzw. $-128, \ldots, 127$ $(8$ Stellen$)$
16-Bit, $n = 15 : 0, \ldots, 65535$ ohne Vorzeichen
bzw. $-32768, \ldots, 32767$ $(16$ Stellen$)$
32-Bit, $n = 31 : 0, \ldots, 4.294.967.295$ ohne Vorzeichen
bzw. $-2.147.483.648, \ldots, 2.147.483.647$ $(32$ Stellen$)$

Wir sehen also, dass der darstellbare Zahlenbereich eingeschränkt ist, selbst wenn wir die Bit-Größe erhöhen. Dies liegt u.a. daran, dass die Verwendung eines Festpunktes die auf dem Rechner darstellbaren Werte stark einschränkt.

7.2.3 Gleitpunktdarstellung

Gibt man statt der festen Anzahl von Vor- und Nachkommastellen nur die *Gesamtanzahl* an Stellen vor und lässt die *Position des Punktes variabel*, skaliert also mit unterschiedlichen Potenzen der Basis b, so ist die Menge der darstellbaren Zahlen wesentlich größer. Dies führt zur Definition der Gleitpunktdarstellung.

Definition 7.2.8 Die **normalisierte Gleitpunktdarstellung** von $x = \sigma \sum_{k=-\ell}^{n} a_k b^k \in \mathbb{R}$ zur Basis $b \geq 2$, $b \in \mathbb{N}$, hat die Form $x = f \cdot b^{\ell}$, wobei

- die **Mantisse** f eine feste Zahl mit m Stellen ist, so dass $b^{-1} \leq |f| < 1$, falls $x \neq 0$ und $f = 0$, falls $x = 0$, d.h.,

$$f = \begin{cases} \sigma\,(0.d_1 \dots d_m)_b = \sigma \sum\limits_{j=1}^{m} d_j b^{-j}, & \text{falls } d_1 \neq 0, \\ 0, & \text{für } d_1 = 0; \end{cases}$$

- der **Exponent** ℓ eine ganze Zahl mit $r \leq \ell \leq R$ ist, welche sich in der Form

$$\ell = \pm\,(\ell_{n-1} \cdots \ell_0)_b = \pm \sum_{j=0}^{n-1} \ell_j b^j$$

  darstellen lässt. Hierbei braucht nicht $\ell_{n-1} \neq 0$ zu gelten.

Die **Menge der Maschinenzahlen** zur Basis b, mit m-stelliger Mantisse und n-stelligem Exponenten wird als $\mathbb{M}(b, m, n)$ bezeichnet.

■ **Beispiel 7.2.9** Die Menge $\mathbb{M}(2,3,1)$ besteht aus den Zahlen $x = \pm(d_{-1} \cdot 2^{-1} + d_{-2} \cdot 2^{-2} + d_{-3} \cdot 2^{-3}) \cdot 2^{\pm \ell_0}$, wobei entweder $d_{-1} \neq 0$ oder $x = d_{-1} = d_{-2} = d_{-3} = 0$ gilt. Somit erhalten wir

$$\mathbb{M}(2,3,1) = \left\{0, \pm\frac{1}{4}, \pm\frac{5}{16}, \pm\frac{3}{8}, \pm\frac{7}{16}, \pm\frac{1}{2}, \pm\frac{5}{8}, \pm\frac{3}{4}, \pm\frac{7}{8}, \pm 1, \pm\frac{5}{4}, \pm\frac{3}{2}, \pm\frac{7}{4}\right\}.$$

Abbildung 7.1 veranschaulicht die Menge $\mathbb{M}(2,3,1)$ auf dem Zahlenstrahl. Wir erkennen deutlich, dass die Zahlen sehr unterschiedlich verteilt sind. Ebenso fällt der nichtdarstellbare Bereich um den Nullpunkt auf. Will man nun eine Zahl $x \notin \mathbb{M}(2,3,1)$ darstellen, dann ist das Beste, was man tun kann, die nächstgelegene Zahl $\mathrm{fl}(x) \in \mathbb{M}(2,3,1)$ als

---

[4] **Zweierkomplement** ist eine Möglichkeit, negative Zahlen im Dualsystem darzustellen. Dabei werden keine zusätzlichen Symbole wie + und - benötigt. Da im Zweierkomplement der Wert 0 den positiven Zahlen zugerechnet wird, umfasst der Wertebereich bei n binären Stellen allgemein den Bereich $-2^{n-1}, \dots, 0, \dots, 2^{n-1} - 1$. Zum Beispiel bei 4 Bit von -8 bis 7. Negative Zahlen gewinnt man aus positiven Zahlen, indem sämtliche binären Stellen negiert werden und zu dem Ergebnis der Wert 1 addiert wird. Somit wird aus $(4)_{10} = (0100)_2$ durch Invertieren $\texttt{Not}(0100) = 1011$ und Addition der 1 $(1011)_2 + (0001)_2 = (1100)_2$. Addiert man nun 4 und -4, so gilt $(4)_{10} + (-4)_{10} = (0100)_2 + (1100)_2 = (0000)_2 = (0)_{10}$, wobei die vorderste Stelle verworfen wurde.

[5] **Bit** ~ (engl. binary digit, lat. digitus ~ Finger).

[6] **Byte** ~ Maßeinheit für eine Datenmenge von 8 Bit.

Approximation zu verwenden. Die Abbildung zeigt, dass der Rundungsfehler $|x - \mathrm{fl}(x)|$ von der Eingabe x abhängt! ∎

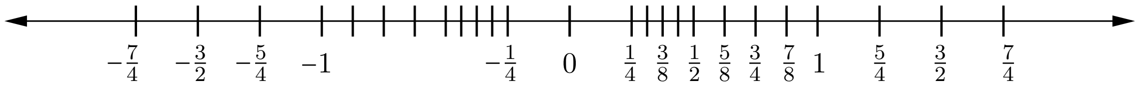


Abbildung 7.1: Graphische Darstellung von $\mathbb{M}(2,3,1)$.

Wie schon bei den Festpunktzahlen erkennen wir, dass $\mathbb{M}(b,m,n)$ eine *endliche* Menge ist. Zu $x \in \mathbb{R}$ bezeichnen wir wiederum $\mathrm{fl}(x) \in \mathbb{M}(b,m,n)$ eine geeignete Approximation (die genaue Definition geben wir etwas später).

Definition 7.2.10 Die betragsmäßig kleinste und größte Zahl in $\mathbb{M}(b,m,n)$ bezeichnen wir mit $x_{\min}$ bzw. $x_{\max}$,

$$x_{\min} := \min\{|x|, x \in \mathbb{M}(b,m,n)\}, \qquad |x| < x_{\min} \rightsquigarrow \mathrm{fl}(x) := 0,$$
$$x_{\max} := \max\{|x|, x \in \mathbb{M}(b,m,n)\}, \qquad |x| > x_{\max} \rightsquigarrow \textbf{overflow}.$$

Definition 7.2.11 Der **darstellbare Maschinenzahlenbereich** ist definiert durch

$$\mathbb{M}' \equiv \mathbb{M}'(b,m,n) := \{x \in \mathbb{R} : x_{\min} \le |x| \le x_{\max}\} \cup \{0\},$$

wobei die Abhängigkeit von b, m und n durch $x_{\min}$ und $x_{\max}$ kommt, die von diesen Parametern abhängen.

∎ **Beispiel 7.2.12** Wir geben einige Beispiele an. Dazu bestimmen wir zunächst die Binärdarstellung und danach die Rundung auf $\mathbb{M}(b,m,n)$.

$$\mathbb{M}(10,4,1): \quad x = -51.34 \qquad \rightsquigarrow -0.5134 \cdot 10^2 = \mathrm{fl}(x)$$
$$\mathbb{M}(2,4,2): \quad x = 3.25 = (11.01)_2 \qquad \rightsquigarrow (0.1101)_2 \cdot 2^{(10)_2} = \mathrm{fl}(x)$$

In beiden Fällen ist die Binärdarstellung exakt in $\mathbb{M}(b,m,n)$ darstellbar. Für den Fall

$$\mathbb{M}(2,4,2): \quad x = 14.625 = (1110.101)_2 \qquad \rightsquigarrow (0.1110101)_2 \cdot 2^{(100)_2} \notin \mathbb{M}(2,4,2)$$

ist der Exponent nicht mit $n = 2$ Stellen darstellbar. Damit wäre $\mathrm{fl}(x) = (0.1110101)_2 \cdot 2^{(11)_2} \in \mathbb{M}(2,4,2)$ die beste Approxmation, was im Dezimalsystem 7.3125 und damit einen relativen Fehler von 50% bedeuet. ∎

Bemerkung 7.2.13 Üblicherweise werden 8 Byte (64 Bit) Speicherplatz für eine (doppelt genaue) Gleitpunktzahl wie folgt aufgeteilt:

- 1 Bit für das Vorzeichen;
- 11 Bit für den Exponenten,
  also sind die Werte im Bereich $-1022, \ldots, 1023$ darstellbar; $(1, \ldots, 2046$ – Bias-Wert 1023[7]$)$. Die Werte -1023 und 1024, bzw. 0 und $2047 = 2^{11} - 1$ als Charakteristik sind reserviert für die speziellen Zahlenwerte „`Null`“, „`Unendlich`“ und „`NaN`“;
- 52 Bit für die Mantisse.

Dies sind eigentlich Werte zwischen $(0...01)_2$ und $(1...1)_2$, da die Null gesondert abgespeichert wird. In der normalisierten Darstellung ist die erste Null redundant und man gewinnt noch eine Stelle hinzu. Also muss auch noch gespeichert werden, ob eine normalisierte Darstellung vorliegt oder nicht.

Insgesamt ergibt sich $\mathbb{M}(2, 52, 11)$ und damit $r = -1023$ und $R = 1024$.

Bemerkung 7.2.14 Die Darstellung der 0, nicht-normalisierter Zahlen, ∞ und `NaN` kann durch die Kombination verschiedener Exponenten $e \in \{0, e_{max}\}$ und entsprechender Mantisse m kodiert werden.

Exponent e	Mantisse	Bedeutung	Bezeichnung
$e = 0$	$m = 0$	± 0	Null
$e = 0$	$m > 0$	$\pm 0, m \times 2^{1-Bias}$	denormalisierte Zahl
$0 < e < e_{max}$	$m \geq 0$	$\pm 1, m \times 2^{e-Bias}$	normalisierte Zahl
$e = e_{max}$	$m = 0$	$\pm\infty$	Unendlich (Inf)
$e = e_{max}$	$m > 0$	keine Zahl	**N**ot **a** **N**umber (NaN)

Die Werte von b, m, ℓ hängen in der Regel vom Typ des jeweiligen Computers ab, ebenso r, R. Der IEEE[8]-Standard versucht dies zu vereinheitlichen, vgl. Tabelle 7.1. Der Standardtyp in MATLAB®, wenn nichts Weiteres angegeben wird, ist von doppelter Genauigkeit, d.h. vom Typ `double`.

Tabelle 7.1: Zahlenformat nach IEEE-754

Typ	Größe	Exponent	Mantisse	Werte des Exponenten (e)
single double	32 Bit 64 Bit	8 Bit 11 Bit	23 Bit 52 Bit	-126 ≤ e ≤ 127 -1022 ≤ e ≤ 1023
	rel. Abstand zweier Zahlen	Dezimalstellen	betragsmäßig kleinste Zahl	größte Zahl
single double	$2^{-(23+1)}$ $2^{-(52+1)}$	7-8 15-16	$2^{-23} \cdot 2^{-126}$ $2^{-52} \cdot 2^{-1022}$	$(1-2^{-24})2^{128}$ $(1-2^{-53})2^{1024}$

7.3 Rundung und Maschinengenauigkeit

Wie bereits festgestellt, ist die Menge $\mathbb{M}(b, m, n)$ endlich und wir brauchen für eine Eingabe $x \in \mathbb{R} \setminus \mathbb{M}(b, m, n)$ eine Näherung $\mathrm{fl}(x) \in \mathbb{M}(b, m, n)$, fl steht hier für „float“. Dieser Prozess wird als *Rundung* bezeichnet, wobei natürlich Fehler entstehen, die wir zu analysieren haben. Bislang haben wir die Abbildung fl noch nicht definiert, was wir jetzt nachholen werden. Wir beschränken uns dazu auf die Gleitpunktdarstellung und wollen annehmen, dass bei allen auftretenden Rechnungen stets $x \in [-x_{\max}, x_{\max}]$ mit $-\infty < -x_{\max} \leq x_{\max} < \infty$ gilt.

[7] Hier zu verstehen als ein konstanter Wert, der zu etwas hinzuaddiert bzw. subtrahiert wird; wird auch als *Offset* bezeichnet.

[8] IEEE~ Institute of Electrical and Electronics Engineers.

7.3.1 Standardrundung, kaufmännische Rundung

Die Standardrundung entspricht dem „normalem Runden", also $1.4 \to 1 = \mathrm{fl}$, $1.5 \to 2$, $1.6 \to 2$. Wir definieren die entsprechende Funktion fl.

Definition 7.3.1 — Standardrundung. Sei $x \in \mathbb{R}$ mit $|x| \in [x_{\min}, x_{\max}]$ und b-adischer Darstellung

$$x = \sigma\, b^{\ell} \sum_{j=1}^{\infty} d_j\, b^{-j},$$

wobei ℓ ein n-stelliger Exponent sei. Nach Voraussetzung gelte $d_1 \neq 0$ sowie $d_j < b-1$ für unendlich viele $j \in \mathbb{N}$. Für eine vorgegebene Mantissenlänge m ist die **Standardrundung** fl_{std} definiert durch

$$\mathrm{fl}_{\text{std}}(x; b, m, n) = \mathrm{fl}_{\text{std}}(x) := \sigma \sum_{j=1}^{m} d_j\, b^{\ell-j} + \sigma \begin{cases} 0, & \text{falls } d_{m+1} < \frac{b}{2}, \\ b^{\ell-m}, & \text{falls } d_{m+1} \geq \frac{b}{2}, \end{cases}$$

und für die anderen Fälle gilt

$$|x| < x_{\min} \leadsto \mathrm{fl}_{\text{std}}(x) := 0, \qquad |x| > x_{\max} \leadsto \textbf{overflow}.$$

7.3.2 Banker's Rule – Round to even

Die Rundung im Falle $d_{m+1} = b/2$ (b gerade, $b/2$ ungerade) bei der Standardrundung ist (natürlich) willkürlich. Man kann sich auch überlegen, welche Rundung in solchen Fällen zu (statistisch) geringeren Rundungsfehlern führt. Die führt auf die „mathematisch unverzerrte Rundung" (auch „Banker's Rule" oder „Round to even" genannt).[9] Der einzige Unterschied zur Standardrundung tritt auf, wenn der zu rundende Rest genau zwischen zwei Zahlen mit der gewählten Anzahl von Ziffern liegt, also der Fall $d_{m+1} = b/2$ für b gerade und $b/2$ ungerade (was für die Fälle $b = 2$ und 10 erfüllt ist).

Die Standardrundung erzeugt statistische „Fehler", da das Aufrunden um $b/2$ vorkommt, das Abrunden um $b/2$ jedoch nie. Rundet man z.B. bei großen Mengen von Daten, bei denen der Trend untersucht werden soll, immer auf, falls der zu rundende Rest genau $b/2$ ist, so verschiebt sich der Trend aufwärts. Außerdem ist das Verhalten bei positiven und negativen Zahlen unterschiedlich, wenn die zu rundende Ziffer $b/2$ ist. Das „round to even" vermeidet dieses Verhalten. Es rundet von der genauen Mitte zwischen zwei Ziffern immer zur nächsten geraden Zahl auf oder ab. Ansonsten entspricht es der Standardrundung. Man könnte genauso gut auch zur nächsten ungeraden Zahl runden, dann würde man jedoch nie auf Null runden.

Definition 7.3.2 — Banker's Rule. Sei b gerade, $b/2$ ungerade und $x \in \mathbb{R}$ mit $|x| \leq x_{\max}$ und b-adischer Darstellung

$$x = \sigma\, b^{\ell} \sum_{j=1}^{\infty} d_j\, b^{-j},$$

[9] Diese Rundung ist auch in der IEEE-Norm P754 verankert.

wobei ℓ ein n-stelliger Exponent sei. Es gelte wiederum $d_1 \neq 0$ sowie $d_j < b-1$ für unendlich viele $j \in \mathbb{N}$. Für eine vorgegebene Mantissenlänge m ist die **Banker's Rule** $\mathrm{fl}_{\mathrm{br}} : \mathbb{R} \to \mathbb{M}(b,m,n)$ definiert durch

$$\mathrm{fl}_{\mathrm{br}}(x) := \sigma \sum_{j=1}^{m} d_j\, b^{\ell-j} + \sigma \begin{cases} 0, & \text{falls } d_{m+1} < b/2, \\ b^{\ell-m}, & \text{falls } d_{m+1} > b/2, \\ & \text{oder } d_{m+1} = b/2 \text{ und es ex. ein } k > m+1 \text{ mit } d_k \neq 0, \\ 0, & \text{falls } \sum_{k=m+1}^{\infty} d_k\, b^{\ell-k} = b^{\ell-m}/2 \text{ und } d_m \text{ gerade}, \\ b^{\ell-m}, & \text{falls } \sum_{k=m+1}^{\infty} d_k\, b^{\ell-k} = b^{\ell-m}/2 \text{ und } d_m \text{ ungerade}. \end{cases}$$

■ **Beispiel 7.3.3** Für $\mathbb{M}(10,2,1)$ erhält man mit der Banker's Rule folgende Ergebnisse:

$$\begin{aligned} \mathrm{fl}_{\mathrm{br}}(2.33) &= 2.3, & \mathrm{fl}_{\mathrm{br}}(-2.33) &= -2.3, \\ \mathrm{fl}_{\mathrm{br}}(2.35) &= 2.4, & \mathrm{fl}_{\mathrm{br}}(-2.45) &= -2.4, \\ \mathrm{fl}_{\mathrm{br}}(2.45) &= 2.4, & \mathrm{fl}_{\mathrm{br}}(-2.4500001) &= -2.5, \\ \mathrm{fl}_{\mathrm{br}}(2.4500001) &= 2.5, & \mathrm{fl}_{\mathrm{br}}(-2.449) &= -2.4. \end{aligned}$$

■

Zur Basis $b = 10$ lautet die Regel einfach ausgedrückt: Folgt auf die letzte beizubehaltende Ziffer lediglich eine 5 (also eine 5, auf die nur Nullen folgen), so wird derart gerundet, dass die letzte beizubehaltende Ziffer gerade wird.

■ **Beispiel 7.3.4 — Wiederholtes Runden.** Wir zeigen einige Beispiele, die verdeutlichen sollen, dass im Falle $b = 2$ oder $b = 10$ wiederholtes Runden bei unterschiedlichen Rundungsanweisungen zu unterschiedlichen Ergebnisse führen kann.

b	Standard	Fehler	Banker's Rule	Fehler
10	$2.445 \rightsquigarrow 2.45 \rightsquigarrow 2.5 \rightsquigarrow 3$	0.555	$2.445 \rightsquigarrow 2.44 \rightsquigarrow 2.4 \rightsquigarrow 2$	0.445
10	$4.545 \rightsquigarrow 4.55 \rightsquigarrow 4.6 \rightsquigarrow 5$	0.455	$4.545 \rightsquigarrow 4.54 \rightsquigarrow 4.5 \rightsquigarrow 4$	0.545
2	$0.101 \rightsquigarrow 0.11 \rightsquigarrow 1.0 \rightsquigarrow 1$	0.011	$0.101 \rightsquigarrow 0.10 \rightsquigarrow 0.1 \rightsquigarrow 0$	0.101
2	$0.011 \rightsquigarrow 0.10 \rightsquigarrow 0.1 \rightsquigarrow 1$	0.101	$0.011 \rightsquigarrow 0.10 \rightsquigarrow 0.1 \rightsquigarrow 0$	0.011

Es lassen sich sowohl bei der Standardrundung als auch bei der Banker's Rule Beispiele finden, bei denen durch wiederholtes Runden auf eine ganze Zahl der Fehler größer ist als der maximale Fehler b^{-1}, den man durch einmaliges Runden erhalten könnte. Insbesondere ist die Banker's rule nicht immer besser, die Rundungsfehler sind lediglich statistisch besser verteilt. ■

Wir werden nun die Rundungsfehler genauer untersuchen.

Definition 7.3.5 Es sei $x \in \mathbb{R}$ und $\tilde{x} \in \mathbb{R}$ eine Approximation.[10]
Die Größe $|x - \tilde{x}|$ heißt **absoluter** und $\dfrac{|x-\tilde{x}|}{|x|}$ **relativer Fehler**.

Der relative Fehler ist dimensionslos, während die Größe des absoluten Fehlers von der gewählten Maßeinheit abhängt. Daher ist man oft am relativen Fehler interessiert. Nun zur Abschätzung des Fehlers.

[10] Dies kann durch Rundung entstanden sein, muss aber nicht.

Lemma 7.3.6 Die Basis $b \geq 2$ sei gerade und $x = \sigma\, b^\ell \sum_{k=1}^{\infty} d_k\, b^{-k} \in \mathbb{R}$ mit $0 < d_1 < b$. Dann gilt für die Standardrundung auf m Stellen

$$|\,\mathrm{fl}_{\mathsf{std}}(x) - x| \leq \frac{1}{2} b^{\ell-m} \qquad \text{und} \qquad \frac{|\,\mathrm{fl}_{\mathsf{std}}(x) - x|}{|x|} \leq \frac{1}{2} b^{1-m}.$$

Beweis. O.B.d.A. sei x positiv. Sei $n \in \mathbb{N}$. Mit $b \geq 2$ und $0 \leq d_k \leq b-1$ $(k = n, n+1, \ldots)$ gilt

$$\sum_{k=n}^{\infty} d_k\, b^{-k} \leq \sum_{k=n}^{\infty} (b-1) b^{-k} = \sum_{k=n-1}^{\infty} b^{-k} - \sum_{k=n}^{\infty} b^{-k} = b^{-(n-1)}. \tag{7.4}$$

(i) Betrachten wir zuerst die Situation des Abrundens, also $d_{m+1} < b/2$. Da b nach Voraussetzung gerade ist, gilt $b/2 \in \mathbb{N}$. Somit folgt aus $d_{m+1} < b/2$, dass $d_{m+1} + 1 \leq b/2$ gilt. Da $0 < d_1 < b$ vorausgesetzt wurde,[11] erhalten wir $b^{-1} \leq \sum_{k=1}^{\infty} d_k\, b^{-k} \leq 1$ und somit $b^{\ell-1} \leq x \leq b^\ell$. Wegen $x - \mathrm{fl}_{\mathsf{std}}(x) = b^\ell \sum_{k=m+1}^{\infty} d_k\, b^{-k}$, schließen wir für $d_{m+1} < b/2$ mit (7.4)

$$\begin{aligned} |\,\mathrm{fl}_{\mathsf{std}}(x) - x| &= b^\ell \left[d_{m+1}\, b^{-(m+1)} + \sum_{k=m+2}^{\infty} d_k\, b^{-k} \right] \\ &\leq b^\ell \left[d_{m+1}\, b^{-(m+1)} + b^{-(m+1)} \right] = b^{\ell-(m+1)} \left(d_{m+1} + 1 \right) \\ &\leq b^{\ell-(m+1)} \cdot \frac{b}{2} = \frac{1}{2} b^{1-m} \cdot b^{\ell-1} = \frac{1}{2} b^{\ell-m}. \end{aligned} \tag{7.5}$$

(ii) Für das Aufrunden, d.h. für $d_{m+1} \geq \frac{b}{2}$ gilt nun $d_{m+1}\, b^{-1} \geq \frac{1}{2}$ und analog zu (7.5)

$$\begin{aligned} |\,\mathrm{fl}_{\mathsf{std}}(x) - x| &= b^\ell \left[b^{-m} - \sum_{k=m+1}^{\infty} d_k\, b^{-k} \right] \\ &\leq b^\ell \left[b^{-m} - d_{m+1}\, b^{-(m+1)} \right] = b^{\ell-m} \underbrace{|\,1 - \underbrace{d_{m+1}\, b^{-1}}_{\geq 1/2}\,|}_{\leq 1/2} \\ &\leq \frac{b^{1-m}}{2} \cdot b^{\ell-1} = \frac{1}{2} b^{\ell-m}. \end{aligned}$$

Damit ist die Abschätzung für den absoluten Fehler bewiesen und die Abschätzung für den relativen Fehler erhält man sofort mit $b^{\ell-1} \leq x$. ■

Definition 7.3.7 Die Zahl $\mathrm{eps} := \frac{b^{1-m}}{2}$ heißt (relative) **Maschinengenauigkeit.**

Lemma 7.3.8 (i) Es gilt $\mathrm{eps} = \inf\{\delta > 0 : \mathrm{fl}\,(1+\delta) > 1\}$, je nachdem welche Rundung fl verwendet wird.

(ii) Für alle $0 \neq x \in \mathbb{M}'$[12] existiert ein $0 < \varepsilon < \mathrm{eps}$ und $\frac{|\mathrm{fl}(x) - x|}{|x|} = \varepsilon$, d.h.

$$\mathrm{fl}\,(x) = x\,(1+\varepsilon). \tag{7.6}$$

Beweis. Übung. ■

[11] Dies entspricht der normierten Gleitpunktdarstellung in Definition 7.2.8.
[12] Vgl. Definition 7.2.11.

In MATLAB® liefert die Anweisung `eps` die Maschinengenauigkeit, die sich aber auch durch folgende Routine `mineps.m` bestimmen lässt, s.u. In anderen Programmiersprachen geht dies ähnlich, man muss sich jedoch vergewissern, dass der Compiler die Anweisung `while 1 < 1 + value` nicht zu `while 0 < value` optimiert!

MATLAB-Funktion: mineps.m

```
function value = mineps
value = 1;
while 1 < 1 + value
  value = value / 2;
end
value = value * 2;
```

Die Funktion `mineps` liefert das gleiche Ergebnis wie die MATLAB®-Konstante `eps`. Ergänzend sei noch die MATLAB®-Konstante `realmin` angegeben; sie ist die kleinste positive darstellbare Maschinenzahl.

```
>> eps
ans =
    2.220446049250313e-016
>> mineps
ans =
    2.220446049250313e-016
>> realmin
ans =
    2.225073858507201e-308
```

Wie schon angekündigt, wollen wir auch `ChatGBT` zur Erzeugung von MATLAB®-Programmen einsetzen. Hier ein erster Versuch:

> Schreibe mir ein Programm in Matlab zur Bestimmung der Maschinengenauigkeit

> Klar! Hier ist ein einfaches MATLAB-Programm, das die Maschinengenauigkeit bestimmt:

MATLAB-Funktion per ChatGBT: calculateMachineEpsilon.m

```
function machine_epsilon = calculateMachineEpsilon()
    epsilon = 1.0;

    while (1 + epsilon) > 1
        epsilon = epsilon / 2;
    end

    machine_epsilon = epsilon * 2;
end
```

```
Du kannst diese Funktion in MATLAB aufrufen, um die
Maschinengenauigkeit zu berechnen:
epsilon = calculateMachineEpsilon();
disp(['Die Maschinengenauigkeit beträgt: ', num2str(epsilon)]);
```

Ausgabe: `Die Maschinengenauigkeit beträgt: 2.2204e-16`

7.4 Gleitkommaarithmetik

Ein Algorithmus ist eine Folge von arithmetischen Operationen, mit denen Maschinenzahlen verknüpft werden. Ein Computer rechnet also mit Maschinenzahlen durch die Realisierung (Implementierung) eines Algorithmus. Bei dieser sogenannten „Gleitkommaarithmetik“ treten einige Effekte auf, die wir beschreiben und analysieren werden.

■ **Beispiel 7.4.1** Angenommen, wir wollen eine Addition in $\mathbb{M}(10,3,1)$ durchführen:

$$\underbrace{0.346 \cdot 10^3}_{\in \mathbb{M}(10,3,1)} + \underbrace{0.785 \cdot 10^3}_{\in \mathbb{M}(10,3,1)} = 0.1131 \cdot 10^4 \notin \mathbb{M}(10,3,1).$$

An diesem einfachen Beispiel sieht man, dass das Ergebnis einer arithmetischen Operation auf Maschinenzahlen *keine* Maschinenzahl sein muss! Die **Menge der Maschinenzahlen ist also nicht abgeschlossen** (selbst) unter (einfachen) arithmetischen Operationen. ■

Wir unterscheiden zwischen arithmetischen Operationen zwischen Zahlen und deren Realisierung auf dem Rechner.

Voraussetzung 7.4.2 Wir nehmen an, dass für die Addition $\oplus$ im Rechner gilt

$$x \oplus y = \mathrm{fl}(x+y) \quad \text{für alle } x,y \in \mathbb{M} \text{ mit } |x+y| \in \mathbb{M}'$$

und Entsprechendes auch für die anderen Grundrechenarten: $\circledast \in \{+,-,*,\div\}$ wird durch eine Gleitpunktoperation $\circledast$ ersetzt (also $\oplus$, $\ominus$, $\circledast$, $\oslash$) mit

$$x \circledast y = \mathrm{fl}(x \circledast y)\ .$$

Man sagt $\oplus$ (bzw. $\circledast$) ist **exakt gerundet**.

Korollar 7.4.3 Für alle $x,y \in \mathbb{M}$ mit $x+y \in \mathbb{M}'$ gilt

$$x \oplus y = (x+y)(1+\varepsilon) \tag{7.7}$$

für ein geeignetes $|\varepsilon| \le \mathrm{eps}$, und Entsprechendes auch für die anderen Grundrechenarten: $x \circledast y = (x \circledast y)(1+\varepsilon)$ mit $|\varepsilon| \le \mathrm{eps}$. Dies folgt aus $\mathrm{fl}(x) = x(1+\varepsilon)$.

Bemerkung 7.4.4 (a) Maschinenoperationen sind nicht assoziativ, wie folgendes Beispiel in $\mathbb{M}(10,3,1)$ zeigt. Für $x = 6590 = 0.659 \cdot 10^4$, $y = 1 = 0.100 \cdot 10^1$ und $z = 4 = 0.400 \cdot 10^1$ ergibt sich bei exakter Rechnung $(x+y)+z = x+(y+z) = 6595$.

Für die Maschinenoperationen gilt hingegen auf der einen Seite

$$x \oplus y = 0.659 \cdot 10^4 = 6590 \Rightarrow (x \oplus y) \oplus z = 6590,$$

aber

$$y \oplus z = 0.500 \cdot 10 = 5 \Rightarrow x \oplus (y \oplus z) = 6660.$$

Also ist die Reihenfolge der Verknüpfungen wesentlich! Als „Faustregel" (die aber auch nicht immer hilft, s.u.) kann man sagen, dass man die Summation in der Reihenfolge aufsteigender Beträge ausführen sollte, um Rundungsfehler zu minimieren.

(b) Ebenso gilt das Distributivgesetz nicht mehr.

(c) Besonders kritisch ist die Subtraktion, wie folgendes Beispiel zeigt: $0.73563 - 0.73441 = 0.00122$. Bei dreistelliger Rechnung: $0.736 - 0.734 = 0.002 = 0.2 \cdot 10^{-2}$, d.h., der absolute Fehler ist in der Größenordnung des Rundungsfehlers, für den relativen Fehler gilt jedoch

$$\frac{0.002 - 0.00122}{0 - 00122} = 0.64 \hat{=} 64\%$$

also sehr groß. Dieser unangenehme Effekt der Gleitpunktarithmetik heißt **Auslöschung** (genaueres später).

■ **Beispiel 7.4.5 — Auslöschung.** (a) Auslöschung kann aber auch bei der Addition auftreten: Für $b = 10$ und $m = 2$ gilt $(100 \oplus 4) \oplus 4 = 100 \neq 110 = 100 \oplus (4 \oplus 4)$. Man beachte dabei $100 \oplus 4 = 100$, da $100 + 4$ auf 2 Stellen gerundet 100 ergibt, und dass $100 + 8$ auf 2 Stellen gerundet 110 ergibt. Wir sehen also wieder den Effekt der Auslöschung.

(b) Wir betrachten die Rechnung $0.1236 + 1.234 - 1.356 = 0.0016 = 0.1600 \cdot 10^{-2}$. Gleitpunktrechnung mit $b = 10, m = 4$ liefert $(0.1236 \oplus 1.234) \ominus 1.356 = 1.358 \ominus 1.356 = 0.2000 \cdot 10^{-2}$, d.h. schon die erste Stelle des berechneten Ergebnisses ist falsch! Der Rundungsfehler der ersten Addition wird durch die nachfolgende Subtraktion also extrem verstärkt. ■

Bemerkung 7.4.6 In Beispiel 7.4.5 (a) hatten wir ja bereits gesehen, dass auch bei der Addition Auslöschung auftreten kann. Wir wollen nun sehen, ob die erwähnte „Faustregel" hilft. Sei wiederum $b = 10$ und $m = 4$. Wir wollen

$$\sum_{n=0}^{N} a_n \text{ mit } a_0 = 1 \tag{7.8}$$

berechnen. Falls $0 \leq a_n < 10^{-4}$ für alle $n > 0$, und falls wir der Reihe nach summieren, ist das berechnete Ergebnis 1, egal wie groß die Summe wirklich ist.

Nach der obigen Faustregel würden wir der Größe nach summieren (zuerst die kleinste Zahl). In unserem Beispiel (7.8) hilft das allerdings nicht wirklich, das berechnete Ergebnis ist dann höchstens 2.

Besser ist jedoch folgendes Vorgehen: Man ersetzt die zwei kleinsten Zahlen durch ihre Summe und wiederholt diesen Vorgang so lange, bis nur noch eine Zahl, nämlich die gesuchte Summe, übrig bleibt.

■ **Beispiel 7.4.7** Sei $\mathbb{M} = \mathbb{M}(10,3,1)$ und betrachte $\ominus$. Sei weiter $x = 0.215 \cdot 10^9$ und $y = 0.125 \cdot 10^{-9}$. Naive Realisierung von $x \ominus y = \mathrm{fl}(x - y)$ erfordert **schieben** von y auf den größeren Exponenten $y = 0.125 \cdot 10^{-18} \cdot 10^9$ und **subtrahieren** der Mantissen:

$$\begin{array}{rcl} x & = & 0.215000000000000000000 \cdot 10^9 \\ y & = & 0.000000000000000000125 \cdot 10^9 \\ \hline x - y & = & 0.214999999999999999875 \cdot 10^9 \end{array} \tag{7.9}$$

Runden auf drei Stellen liefert dann $x \ominus y = 0.215 \cdot 10^9$. Um das exakte Ergebnis zu erzielen, bräuchten wir einen Addierer mit $2(b^n - 1) + m = 21$ Stellen. In diesem Fall hätten wir das exakte Ergebnis auch durch die Abfolge **Schieben**, **Runde** y, **Subtrahiere** erhalten.

Im Allgemeinen ist das aber nicht gut wie folgendes Beispiel zeigt:

$$\begin{array}{rcl} x & = & 0.101 \cdot 10^1 \\ y & = & 0.993 \cdot 10^0 \end{array} \quad \longrightarrow \quad \begin{array}{rcl} x & = & 0.101 \quad \cdot 10^1 \\ y & = & 0.099 \quad \cdot 10^1 \\ \hline x \ominus y & = & 0.002 \quad \cdot 10^1 \end{array} .$$

Für den relativen Fehler im Ergebnis gilt dann

$$\frac{(x \ominus y) - (x - y)}{(x - y)} = \frac{0.02 - 0.017}{0.017} \approx 0.176 \approx 35 \text{ eps},$$

wobei $\text{eps} = 1/2 \cdot 10^{-3+1} = 0.005$ sei. Verwenden wir nun einen $m+1$-stelligen Addierer, so erhalten wir

$$\begin{array}{r} x = 0.1010 \cdot 10^1 \\ y = 0.0993 \cdot 10^1 \\ \hline x - y = 0.0017 \cdot 10^1. \end{array}$$

Das Ergebnis $x \ominus y = 1.7 \cdot 10^{-2}$ ist exakt! ■

Bemerkung 7.4.8 — Guard Digit. Allgemein kann man zeigen (siehe [Knu98]), dass mit einer zusätzlichen Stelle (der sogenannten **Schutzziffer** oder **Guard Digit**) gilt

$$\frac{(x \ominus y) - (x - y)}{(x - y)} \le 2 \text{ eps} .$$

Mit zwei Guard Digits erhält man sogar genau das gerundete Ergebnis der exakten Rechnung, d.h. das gerundete Ergebnis aus (7.9) ohne einen $2(b^n - 1) + m$-stelligen Addierer.

Völlig unkalkulierbar sind Gleitkommaoperationen aber nicht.

Satz 7.4.9 — Siehe [Knu98, 4.2.2, Theorem B, S. 236]. Sind u, v Gleitkommazahlen und

$$u' := (u \oplus v) \ominus v, \quad u'' := (u \oplus v) \ominus v', \qquad v' := (u \oplus v) \ominus u, \quad v'' := (u \oplus v) \ominus u',$$

dann folgt $u + v - (u \oplus v) = (u \ominus u') \oplus (v \ominus v'')$.

Dies erlaubt eine Berechnung des Fehlers mittels Gleitkommaarithmetik, da die rechte

Seite auswertbar ist.

7.5 Fehlerverstärkung bei elementaren Rechenoperationen

In einem Programm/Algorithmus werden *viele* Rechenoperationen durchgeführt. Es stellt sich also die Frage, was mit Fehlern im Programmverlauf passiert. Hier beschäftigen wir uns zunächst mit *exakten Operationen*, d.h. ohne weitere Rundung. Seien also x und y die exakten Eingaben sowie $\tilde{x}$ und $\tilde{y}$ die gestörten (realen) Eingaben, mit den relativen Fehlern

$$\delta_x := \frac{\tilde{x}-x}{x}\ ,\ \delta_y := \frac{\tilde{y}-y}{y}, \quad \text{d.h.} \quad \tilde{x} = x(1+\delta_x), \quad \tilde{y} = y(1+\delta_y), \tag{7.10}$$

und wir nehmen an, dass $|\delta_x|\ ,\ |\delta_y| \le \varepsilon < 1$ gilt, also, dass die Eingabefehler „klein" sind. Da ein Algorithmus aus den Grundrechenarten zusammengesetzt ist, betrachten wir die Fehlerverstärkung bei diesen elementaren Rechenoperationen.

Bemerkung 7.5.1 — Multiplikation. Offenbar gilt

$$\tilde{x} * \tilde{y} = (x(1+\delta_x)) * (y(1+\delta_y)) = (x*y)(1+\delta_x+\delta_y+\delta_x\delta_y) =: (x*y)(1+\delta)$$

mit $|\delta| = |\delta_x+\delta_y+\delta_x\delta_y| \le \varepsilon(2+\varepsilon)$, d.h. bei Vernachlässigung des quadratischen Terms ist dies von der Ordnung 2ε. Damit bleibt der relative Fehler im Rahmen der Maschinengenauigkeit (falls $|\delta_x|, |\delta_y| \le \mathrm{eps}$ und $\varepsilon < 1$):

$$\left|\frac{\tilde{x}*\tilde{y} - x*y}{x*y}\right| = |\delta| = 2\varepsilon + \varepsilon\cdot\varepsilon < 3\varepsilon \le 3\,\mathrm{eps}\,.$$

Bemerkung 7.5.2 — Division. Bei der Division gilt

$$\frac{\tilde{x}}{\tilde{y}} = \frac{x(1+\delta_x)}{y(1+\delta_y)}.$$

Mit Hilfe der geometrischen Reihe $\frac{1}{1-q} = \sum_{j=1}^{\infty} q^j$ für $q = -\delta_y$ gilt $\frac{1}{1+\delta_y} = 1 + \sum_{j=1}^{\infty}(-1)^j\delta_y^j$, und damit folgt

$$\frac{1+\delta_x}{1+\delta_y} = 1+\delta_x + (1+\delta_x)\sum_{j=1}^{\infty}(-1)^j\delta_y^j =: 1+\delta.$$

Für den Fehlerverstärkungsfaktor δ gilt dann unter der Annahme $|\delta_x|, |\delta_y| \le \varepsilon$

$$|\delta| \le \varepsilon + (1+\varepsilon)\varepsilon \underbrace{\sum_{j=0}^{\infty}\varepsilon^j}_{=\frac{1}{1-\varepsilon}} = \varepsilon\left(1+\frac{1+\varepsilon}{1-\varepsilon}\right).$$

Falls z.B. $\varepsilon \le \frac{1}{2}$ gilt, folgt $\frac{1+\varepsilon}{1-\varepsilon} \le 3$ und damit $|\delta| \le 4\,\varepsilon$, also zeigt die Division ein ähnliches Verhalten wie die Multiplikation, d.h. der relative Fehler bleibt im Rahmen der

Maschinengenauigkeit.

Bemerkung 7.5.3 — Addition. Im Fall der Addition gilt

$$\begin{aligned}\tilde{x}+\tilde{y} &= x(1+\delta_x)+y(1+\delta_y) = x+y+x\delta_x+y\delta_y\\ &= (x+y)\left(1+\frac{x}{x+y}\delta_x+\frac{y}{x+y}\delta_y\right) =: (x+y)(1+\delta)\end{aligned}$$

mit der folgenden Abschätzung für den Fehlerverstärkungsfaktor

$$|\delta| \le (|\delta_x|+|\delta_y|)\max\left\{\left|\frac{x}{x+y}\right|,\left|\frac{y}{x+y}\right|\right\} \le 2\varepsilon\max\left\{\left|\frac{x}{x+y}\right|,\left|\frac{y}{x+y}\right|\right\}$$

falls $|\delta_x|, |\delta_y| \le \varepsilon$. Man beachte, dass der Term

$$\max\left\{\left|\frac{x}{x+y}\right|,\left|\frac{y}{x+y}\right|\right\}$$

nicht gegen ε abgeschätzt werden kann. Insbesondere für $x \approx -y$ kann dieser Faktor beliebig groß werden, der Ausgabefehler ist also *nicht* notwendigerweise in der Größe des Eingabefehlers. Dies ist die **Erklärung der Auslöschung** und kann durch folgende Überlegung verdeutlicht werden. Nehmen wir an, wir wollten $x+y$ berechnen für zwei Eingaben mit folgender Darstellung

$$x = 0.d_1 \ldots d_r d_{r+1} \ldots d_m * b^e, \qquad y = -0.d_1 \ldots d_r \tilde{d}_{r+1} \ldots \tilde{d}_m * b^e,$$

d.h., die ersten r Stellen von x und y sind identisch. Wir nehmen der Einfachheit halber $d_j \ge \tilde{d}_j$ für $j \ge r+1$ und $d_{r+1} > \tilde{d}_{r+1}$ an. Daraus folgt

$$x+y = 0.0 \ldots 0\hat{d}_{r+1} \ldots \hat{d}_m * b^e = 0.\hat{d}_{r+1} \ldots \hat{d}_m * b^{e-r}$$

mit $\hat{d}_j \ge 0$ und damit für den relativen Fehler

$$\left|\frac{x}{x+y}\right| = \left|\frac{0.d_1 \ldots d_m * b^e}{0.\hat{d}_{r+1} \ldots \hat{d}_m * b^{e-r}}\right| \le \frac{b \cdot b^e}{b^{-1} b^{e-r}} = b^{r+2},$$

da $0.d \ldots d_m \le b$ und $0.\hat{d}_{r+1} \ldots \hat{d}_m * b^{-1}$. Man verliert also mindestens r Stellen an Genauigkeit, diese werden „**ausgelöscht**“. Intern (im Computer) werden diese Stellen zufällig aufgefüllt mit Information, die sich in den entsprechenden Speicherzellen befindet. Egal, was dort steht, diese Stellen sind im Prinzip immer falsch, da bezüglich x und y keine Information jenseits der m-ten Stelle vorliegt.

7.5.1 Vorwärtsanalyse

In der Vorwärtsanalyse untersucht man Rundungsfehler von der Eingabe zur Ausgabe, der Name suggeriert es schon. Dazu verwenden wir Lemma 7.3.8 und Korollar 7.4.3, insbesondere die Gleichungen $\mathrm{fl}(x) = x(1+\varepsilon)$ und $x \circledast y = (x * y)(1+\varepsilon)$. Diese werden für jeden Rechenschritt angewendet. Die entstehenden Schranken für die Rundungsfehler werden akkumuliert und ergeben so eine obere Schranke für den gesamten durch die

Rundungen bedingten Fehler. Falls Abschätzungen für die Konditionszahlen des Problems bekannt sind, kann man so (theoretisch wenigstens) feststellen, wie groß der Gesamtfehler werden kann.

Wir beginnen mit einem einfachen Beispiel.

Satz 7.5.4 — Vorwärtsanalyse für das Skalarprodukt. Sei $\tilde{\varphi} : \mathbb{R}^{2n} \to \mathbb{R}$ eine Approximation von $\varphi(x,y) := \langle x,y\rangle = x^T y$ durch folgenden Pseudocode gegeben:

```
Input: x,y
s:=0
for k:=1 to n
    s := s + x(k)*y(k)
end
Output s
```

Falls $0 < \mathrm{eps} < 1/n$, dann gilt

$$\frac{|\tilde{\varphi}(x,y) - \varphi(x,y)|}{|\varphi(x,y)|} = \frac{|s - \langle x,y\rangle|}{|\langle x,y\rangle|} \le \frac{n\,\mathrm{eps}}{1 - n\,\mathrm{eps}}. \tag{7.11}$$

Beweis. Sei $s_0 := 0$ und s_k der Wert der Variablen im k-ten Schleifendurchlauf. Mit Korollar 7.4.3 gilt dann

$$\begin{aligned} s_1 &= x_1 y_1 (1 + \delta_1), && \text{mit } |\delta_1| \le \mathrm{eps}, \\ s_k &= [s_{k-1} + x_k y_k (1 + \delta_k)]\,(1 + \varepsilon_k), && \text{mit } |\delta_k|, |\varepsilon_k| \le \mathrm{eps}, \text{ für } k = 2, \dots, n, \end{aligned}$$

wobei wir $x_i \odot y_i = x_i y_i (1 + \delta_i)$ sowie $s_{i-1} \oplus (x_i \odot y_i) = (s_{i-1} + (x_i \odot y_i))(1 + \varepsilon_i)$ verwendet haben. Also gilt für die Ausgabe $\tilde{\varphi}(x,y) = s = s_n$ die Abschätzung mit $\varepsilon_1 = 0$

$$\begin{aligned} s - \langle x,y\rangle &= \sum_{k=1}^{n} x_k y_k \left[(1 + \delta_k) \prod_{j=k}^{n} (1 + \varepsilon_j) - 1\right] \\ &\le \max_{k=1,\dots,n} \left[(1 + \delta_k) \prod_{j=k}^{n} (1 + \varepsilon_j) - 1\right] \cdot \left|\sum_{k=1}^{n} x_k y_k\right| \\ &\le |(1 + \mathrm{eps})^n - 1|\,|\langle x,y\rangle|. \end{aligned} \tag{7.12}$$

Den Fehlerverstärkungsfaktor in (7.12) können wir mit der Bernoullischen-Ungleichung[13] aufgrund der Voraussetzung $\mathrm{eps} < 1/n$ abschätzen

$$|(1 + \mathrm{eps})^n - 1| \le \left|\frac{1}{(1 - \mathrm{eps})^n} - 1\right| \le \left|\frac{1}{1 - n\,\mathrm{eps}} - 1\right| = \frac{n\,\mathrm{eps}}{1 - n\,\mathrm{eps}}. \tag{7.13}$$

Mit (7.12) folgt dann die Behauptung. ∎

Aus der Abschätzung (7.11) ergibt sich dass der relative Fehler beschränkt ist, selbst wenn x und y fast orthogonal sind, das Problem also schlecht konditioniert ist (vgl. Beispiel 2.1.7, (d)). Der obige Beweis zeigt das „Rezept“ für die Vorwärtsanalyse, wir müssen die möglichen Fehler in jedem Schritt akkumulieren und eine obere Schranke suchen. Damit ist klar, dass diese Methode relativ pessimistisch ist, die Schranken also ggf. zu groß sind.

[13] $(1 + x)^n \ge 1 + nx$.

7.5.2 Rückwärtsanalyse

In der Rückwärtsanalyse geht man folgendermaßen vor: Sei $\varphi : X \to Y$ das gegebene Problem und $\widetilde{\varphi} : X \to Y$ ein Algorithmus zur Approximation des Problems. Zu gegebener Eingabe $x \in X$ sucht man ein „gestörte Eingabe" $\widetilde{x} \in X$ mit

$$\widetilde{\varphi}(x) = \varphi(\widetilde{x}). \tag{7.14}$$

Gilt dann eine Abschätzung für den relativen Fehler der folgenden Form (hier sei $n = \dim(X)$, also z.B. $X = \mathbb{R}^n$)

$$\frac{\|\widetilde{x} - x\|}{\|x\|} \leq C\, n\, \mathrm{eps}, \tag{7.15}$$

mit einer Vektornorm $\|\cdot\|$ und einer kleinen Konstanten C, dann lässt sich das berechnete Ergebnis $\widetilde{\varphi}(x)$ interpretieren als das *Ergebnis einer exakten Rechnung mit kleinem Eingabe-Fehler* (Daten-Fehler). Man hat also den Ausgabefehler auf den Eingabefehler zurück gespielt, daher der Name „Rückwärtsanalyse". Dies kann man auch zur Definition der Stabilität heranziehen.

Definition 7.5.5 — Alternative Definition der numerischen Stabilität. Ein Algorithmus heißt numerisch stabil, falls er (7.14), (7.15) erfüllt mit einer „kleinen" Konstanten C.

Korollar 7.5.6 — Rückwärtsanalyse für das Skalarprodukt. Unter den Voraussetzungen von Satz 7.5.4 gilt $s = \langle \widetilde{x}, \widetilde{y} \rangle = \varphi(\widetilde{x}, \widetilde{y})$ für

$$\widetilde{x}_k := \eta_k\, x_k,\ \widetilde{y}_k := \eta_k\, y_k, \qquad \text{mit } \eta_k := (1+\delta_k)^{1/2} \prod_{j=k}^{n} (1+\varepsilon_k)^{1/2},\ k = 1, \ldots, n. \tag{7.16}$$

und damit folgt für $0 < \mathrm{eps} \leq 1/n$

$$\frac{\|(\widetilde{x}, \widetilde{y}) - (x, y)\|}{\|(x, y)\|} \leq \frac{n\, \mathrm{eps}}{1 - n\, \mathrm{eps}}. \tag{7.17}$$

Beweis. Die Behauptung (7.16) ergibt sich aus der ersten Formel in (7.12). Dann gilt weiter für die Euklidische Vektornorm

$$\begin{aligned}
\|(\widetilde{x}, \widetilde{y}) - (x, y)\|^2 &= \|\widetilde{x} - x\|^2 + \|\widetilde{y} - y\|^2 \\
&= \sum_{k=1}^{n} \left(|\widetilde{x}_k - x_k|^2 + |\widetilde{y}_k - y_k|^2 \right) = \sum_{k=1}^{n} (\eta_k - 1)^2 \left(|x_k|^2 + |y_k|^2 \right) \\
&\leq \|(x, y)\|^2 \max_{k=1,\ldots,n} (\eta_k - 1)^2.
\end{aligned}$$

Nun gilt wie im Beweis von Satz 7.5.4, dass

$$\max_{k=1,\ldots,n} (\eta_k - 1) \leq |(1 + \mathrm{eps})^n - 1| \leq \frac{n\, \mathrm{eps}}{1 - n\, \mathrm{eps}}$$

und damit die Behauptung. ■

Anhang A

Grundlagen aus der Linearen Algebra

In diesem Kapitel stellen wir einige der Grundlagen aus der Linearen Algebra zusammen, die wir im Rahmen dieser Vorlesung benötigen. Für weitere Informationen verweisen wir auf Lehrbücher der Linearen Algebra.

A.1 Normen

Mit Normen können wir *Längen* messen. Wir kennen das als Betrag einer reellen Zahl. Im mehrdimensionalen Raum $\mathbb{R}^n$ werden Längen durch Normen ausgedrückt. Man spricht dann auch von einer *Metrik*. Der Begriff einer Norm ist aber nicht auf endlich-dimensionale Räume beschränkt, was wir in der Numerik z.B. bei der numerischen Approximation von Lösungen von Differenzialgleichungen brauchen. Im Rahmen der Vorlesung *Numerische Lineare Algebra* benötigen wir auch insbesondere Normen für Matrizen.

Wir betrachten lineare Räume X über $\mathbb{R}$. Dies bedeutet, dass $\alpha x + \beta y \in X$ für alle $x, y \in X$ und $\alpha, \beta \in \mathbb{R}$. Man nennt diese auch $\mathbb{R}$-*Vektorraum*. Man kann Vektorräume über allgemeinen Körpern $\mathbb{K}$ definieren, also auch über $\mathbb{K} = \mathbb{C}$. Wir beschränken uns hier auf den reellen Fall.

Definition A.1.1 — Norm. Sei X ein linearer Raum über $\mathbb{R}$. Eine Abbildung $\|\cdot\| : \mathbb{X} \to \mathbb{R}$ heißt *Norm* auf X, falls

$$\|x\| \geq 0 \text{ für alle } x \in X \quad \text{und} \quad \|x\| = 0 \Leftrightarrow x = 0, \qquad \textit{(Definitheit)} \tag{A.1a}$$

$$\|x + y\| \leq \|x\| + \|y\| \text{ für alle } x, y \in X, \qquad \textit{(Dreiecksungleichung)} \tag{A.1b}$$

$$\|\alpha x\| = |\alpha| \, \|x\| \text{ für alle } \alpha \in \mathbb{R} \text{ und } x \in X. \qquad \textit{(Homgenität)} \tag{A.1c}$$

Wir nennen $x \in X$ mit $\|x\| = 1$ einen *Einheitsvektor* bzgl. der Norm $\|\cdot\|$.

Für $X = \mathbb{R}^n$ nennen wir eine Funktion $\|\cdot\| : \mathbb{R}^n \to \mathbb{R}$, die die Normaxiome (A.1) erfüllt, auch *Vektornorm*. Besonders häufig verwendete Vektornormen sind die p-Normen definiert für $x = (x_1, ..., x_n)^T$ durch

$$\|x\|_p := \begin{cases} \left(\sum\limits_{k=0}^{n} |x_k|^p\right)^{\frac{1}{p}}, & 1 \le p < \infty, \\ \max\limits_{1\le j\le n} |x_j| & p = \infty. \end{cases} \tag{A.2}$$

Für die Spezialfälle $p = 1$ und $p = 2$ ergibt sich:

$$\begin{aligned} \|x\|_1 &= |x_1| + \cdots + |x_n| \\ \|x\|_2 &= (|x_1|^2 + \cdots + |x_n|^2)^{\frac{1}{2}} = (x^T x)^{\frac{1}{2}} \end{aligned} \tag{A.3}$$

Wir stellen nun einige wichtige Eigenschaften von (Vektor-)normen zusammen. Ein klassisches Ergebnis bzgl. p-Normen ist die **Hölder-Ungleichung**, die für alle $x, y \in \mathbb{R}^n$ besagt, dass

$$|x^T y| \le \|x\|_p \, \|y\|_q \qquad \text{gilt mit } \frac{1}{p} + \frac{1}{q} = 1, \quad 1 \le p, q \le \infty. \tag{A.4}$$

Ein Spezialfall der Hölder-Ungleichung für $p = q = 2$ ist die **Cauchy-Schwarz-Ungleichung**, die wie folgt lautet

$$|x^T y| \le \|x\|_2 \, \|y\|_2. \tag{A.5}$$

Es ist ein zentraler Satz der Linearen Algebra, dass **alle Normen dem auf $\mathbb{R}^n$ äquivalent sind**: Seien $\|\cdot\|_p$ und $\|\cdot\|_{p'}$ beliebige Normen auf dem $\mathbb{R}^n$, dann existieren positive Konstanten $0 < c_1 \le c_2 < \infty$, so dass

$$c_1 \|x\|_p \le \|x\|_{p'} \le c_2 \|x\|_p, \quad x \in \mathbb{R}^n. \tag{A.6}$$

Für obige Fälle und $x \in \mathbb{R}^n$ gilt

$$\begin{aligned} \|x\|_2 &\le \|x\|_1 \le \sqrt{n}\, \|x\|_2, \\ \|x\|_\infty &\le \|x\|_2 \le \sqrt{n}\, \|x\|_\infty, \\ \|x\|_\infty &\le \|x\|_1 \le n\, \|x\|_\infty. \end{aligned} \tag{A.7}$$

Achtung: Wir erkennen, dass diese Konstanten von der Raumdimension n abhängen, insbesondere gilt im Allgemeinen, dass $c_1(n) \to 0$ und $c_2(n) \to \infty$ mit $n \to \infty$. In unendlich-dimensionalen Räumen sind *nicht* alle Normen äquivalent.

A.2 Matrixnormen

Wir betrachten nun Normen für Matrizen. Da der Raum $\mathbb{R}^{m\times n}$ der $m\times n$-Matrizen isomorph[1] zum R^{mn} könnten wir einfach eine Vektornorm für solche „langen" Vektoren verwenden, um eine Matrixnorm als eine Abbildung $\|\cdot\| : \mathbb{R}^{mn} \to \mathbb{R}$ mit den Eigenschaften (A.1) zu definieren.

[1] Dies bedeutet, dass eine bijektive Abbildung $\varphi : \mathbb{R}^{m\times n} \to \mathbb{R}^{mn}$ mit $\varphi(\lambda A + \mu B) = \lambda\varphi(A) + \mu\varphi(B)$ existiert.

▪ **Beispiel A.2.1** Sei $A = (a_{ij})_{i=1,\dots,m,j=1,\dots n} \in \mathbb{R}^{m\times n}$. Die mit am häufigsten verwendeten Normen in der Numerischen Linearen Algebra sind die Frobenius-Norm

$$\|A\|_F := \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|a_{ij}|^2} \tag{A.8}$$

und die p-Normen ($1 \le p \le \infty$)

$$\|A\|_p := \sup_{x\neq 0}\frac{\|Ax\|_p}{\|x\|_p}\,. \tag{A.9}$$

Man kann leicht zeigen, dass

$$\sup_{x\neq 0}\frac{\|Ax\|_p}{\|x\|_p} = \sup_{x\neq 0}\left\|A\,\frac{x}{\|x\|}\right\|_p = \sup_{\|x\|_p=1}\|Ax\|_p \tag{A.10}$$

gilt, so dass man auch eine der Identitäten in (A.10) als Definition von $\|A\|_p$ in (A.9) verwenden kann. ▪

Wir erkennen leicht, dass die Frobenius-Norm identisch mit der 2-Norm im R^{mn} ist.

Lemma A.2.2 — Submultiplikativität. Die Frobenius-Norm und die p-Normen erfüllen zusätzlich auch noch die Eigenschaft der **Submultiplikativität**, d.h.

$$\|A\cdot B\| \le \|A\|\,\|B\|. \tag{A.11}$$

Beweis. Wir zeigen die Behauptung nur für die p-Normen und lassen die Aussage für die Frobenius-Norm als Übung. Für $A = 0$ oder $B = 0$ gilt die Behauptung trivialerweise; seien also $A \neq 0$ und $B \neq 0$, insbesondere gilt dann $\max_{y\in\mathbb{R}^n}\|Ay\|_p > 0$ und $\max_{x\in\mathbb{R}^n}\|Bx\|_p > 0$. Dann gilt mit (A.9) und (A.10)

$$\begin{aligned}\|A\cdot B\|_p &= \max_{x\neq 0}\frac{\|ABx\|_p}{\|x\|_p} = \max_{x\neq 0}\frac{\|ABx\|_p\,\|Bx\|_p}{\|Bx\|_p\,\|x\|_p}\\ &\le \max_{x\neq 0}\frac{\|ABx\|_p}{\|Bx\|_p}\cdot\max_{x\neq 0}\frac{\|Bx\|_p}{\|x\|_p} = \max_{y\neq 0}\frac{\|Ay\|_p}{\|y\|_p}\cdot\max_{x\neq 0}\frac{\|Bx\|_p}{\|x\|_p} = \|A\|_p\,\|B\|_p,\end{aligned}$$

was die Behauptung beweist. ■

Bemerkung A.2.3 Nicht alle Matrix-Normen erfüllen diese Eigenschaft, z.B. gilt für $\|A\|_{\max} := \max_{i,j}|a_{ij}|$ (man zeige, dass dies eine Norm definiert) und $A = B = \begin{pmatrix}1 & 1\\ 1 & 1\end{pmatrix}$ die Ungleichung

$$\|A\cdot B\|_{\max} = 2 > \|A\|_{\max}\,\|B\|_{\max} = 1\,.$$

Die Frobenius und p-Normen (speziell $p = 1, 2, \infty$) erfüllen einige Ungleichungen, welche in der Analysis von Matrixberechungen verwendet werden.

Lemma A.2.4 Es sei $A \in \mathbb{R}^{m\times n}$, dann gelten folgende Ungleichungen

$$\|A\|_2 \le \|A\|_F \le \sqrt{n}\,\|A\|_2, \tag{A.12}$$

$$\|A\|_{\max} \le \|A\|_2 \le \sqrt{m\,n}\,\|A\|_{\max}, \tag{A.13}$$

$$\frac{1}{\sqrt{n}}\,\|A\|_\infty \le \|A\|_2 \le \sqrt{m}\,\|A\|_\infty, \tag{A.14}$$

$$\frac{1}{\sqrt{m}}\,\|A\|_1 \le \|A\|_2 \le \sqrt{n}\,\|A\|_1. \tag{A.15}$$

Beweis. Übung. ■

Zeilen- und Spaltensummennorm

Die p-Matrixnormen für $p = 1$ und $p = \infty$ spielen in der Analysis von numerischen Verfahren für lineare Gleichungssysteme eine besondere Rolle. Daher betrachten wir diese Normen etwas genauer und zeigen Darstellungen dieser.

Bemerkung A.2.5 Für $A \in \mathbb{R}^{m\times n}$ gilt

$$\begin{aligned}\|A\|_\infty &= \max_{\|x\|_\infty=1} \|Ax\|_\infty = \max_{\|x\|_\infty=1}\left\{\max_{1\le j\le m}\left|\sum_{k=1}^n a_{jk}\,x_k\right|\right\} = \max_{1\le j\le m}\left\{\max_{\|x\|_\infty=1}\left|\sum_{k=1}^n a_{jk}\,x_k\right|\right\}\\ &= \max_{1\le j\le m}\sum_{k=1}^n |a_{jk}|.\end{aligned} \tag{A.16}$$

Aufgrund dieser Gleichung bezeichnet man $\|\cdot\|_\infty$ auch als **Zeilensummennorm**. Die Gleichung (A.16) erlaubt eine einfache Berechnung der Norm.

Analog gilt für die 1-Norm:

$$\begin{aligned}\|A\|_1 &= \max_{\|x\|_1=1}\|Ax\|_1 = \max_{\|x\|_1=1}\sum_{j=1}^m\left|\sum_{k=1}^n a_{jk}\,x_k\right| = \max_{\|x\|_1=1}\sum_{j=1}^m\sum_{k=1}^n |a_{jk}|\,|x_k|\,\operatorname{sign}(a_{jk}x_k)\\ &= \max_{\|x\|_1=1}\sum_{k=1}^n |x_k|\sum_{j=1}^m |a_{jk}|\,\operatorname{sign}(a_{jk}x_k) = \max_{\|x\|_1=1}\sum_{k=1}^n |x_k|\sum_{j=1}^m |a_{jk}|\\ &= \max_{1\le k\le m}\sum_{j=1}^n |a_{jk}|\,.\end{aligned} \tag{A.17}$$

Diese Norm bezeichnet man auch als **Spaltensummennorm**.

Bemerkung A.2.6 Einfache Eselsbrücke: $\boxed{1}$-Spaltensummen, $\boxed{\infty}$-Zeilensummen.

Der Fall $p = 2$

Bei Vektornormen kennen wir $\|\cdot\|_2$ auch als *Euklidische Norm*. Man verwendet diesen Namen auch für die induzierte Matrixnorm definiert in (A.9). Wir charakterisieren diese Norm nun etwas genauer.

Satz A.2.7 Es sei $A \in \mathbb{R}^{m\times n}$. Dann existiert ein Vektor $z \in \mathbb{R}^n$ mit $\|z\|_2 = 1$, so dass

$$A^T A z = \mu^2\, z \quad \text{mit } \mu = \|A\|_2\,.$$

Bemerkung A.2.8 Der obige Satz besagt, dass z ein normierter Eigenvektor von A^TA zum Eigenwert $\mu^2 = \|A\|_2^2$ ist. Also ist $\|A\|_2^2$ eine Nullstelle des charakteristischen Polynoms von A^TA

$$\varrho(z) = \det(A^TA - \lambda I).$$

Man kann sogar zeigen, dass die 2-Norm von A die *Wurzel des größten Eigenwerts von* A^TA ist.

Beweis von Satz A.2.7. Es sei $z \in \mathbb{R}^n$ mit $\|z\|_2 = 1$ und $\|Az\|_2 = \|A\|_2$. Da z die Funktion

$$g : \mathbb{R}^n \to \mathbb{R} \qquad g(x) := \frac{1}{2}\frac{\|Ax\|_2^2}{\|x\|_2^2} = \frac{1}{2}\frac{x^TA^TAx}{x^Tx} \tag{A.18}$$

maximiert, folgt daraus, dass z ein kritischer Punkt von g sein muss, also $\nabla g(z) = 0$.[2] Die partiellen Ableitungen von g kann man leicht bestimmen, sie lauten für $i = 1, \ldots, n$

$$\frac{\partial}{\partial x_i} g(z) = \left[x^Tx \sum_{j=1}^{n} (A^TA)_{ij}\, x_j - (x^TA^TAx)x_i \right] \Big/ (x^Tx)^2. \tag{A.19}$$

Wegen $\nabla g(z) = 0$ folgt also

$$0 = z^Tz \sum_{j=1}^{n} (A^TA)_{ij}\, z_j - (z^TA^TAz)z_i = \left(z^Tz(A^TAz) - (z^TA^TAz)z \right)_i$$

für alle $i = 1, \ldots, n$, Da $1 = \|z\|_2^2 = z^Tz$, erhalten wir daraus die Gleichung

$$A^TAz = (z^TA^TAz)z = \|Az\|_2^2\, z = \mu^2\, z,$$

also die Behauptung. ∎

Operatornormen

In der Definition (A.9) haben wir schon gesehen, dass die p-Matrixnorm mithilfe der p-Vektornorm definiert wurde. Man sagt, dass die Matrixnorm von der Vektornorm *induziert* ist. Matrixnormen, die von einer Vektornorm induziert sind (man sagt auch, dass es eine *verträgliche Matrixnorm* ist), erlauben Ungleichungen, die für die Analysis numerischer Verfahren von Bedeutung sind. Man kennt solche Normen auch aus der Funktionalanalysis, wo sie *Operatornorm* genannt werden. Da jede Matrix eine lineare Abbildung von ihrem Urbild- in ihren Bildraum beschreibt, ist diese Bezeichnung naheliegend.

Definition A.2.9 — Operatornorm. Sei X ein $\mathbb{R}$-Vektorraum mit Norm $\|\cdot\|_X$ und $A : X \to X$ eine lineare Abbildung. Die Operatornorm ist definiert durch

$$\|A\|_X := \sup_{X \ni x \neq 0} \frac{\|Ax\|_X}{\|x\|_X}. \tag{A.20}$$

[2] ∇g ist der Gradient von g, also $\left(\nabla := \left(\frac{\partial}{\partial x_1}, \frac{\partial}{\partial x_2}, \ldots, \frac{\partial}{\partial x_n}\right)\right)^T$ (als Spaltenvektor).

Analog zu (A.10) kann man leicht zeigen, dass

$$\sup_{x\neq 0} \frac{\|Ax\|_X}{\|x\|_X} = \sup_{x\neq 0} \left\| A\frac{x}{\|x\|} \right\|_X = \sup_{\|x\|_X=1} \|Ax\|_X. \tag{A.21}$$

Falls X endlich-dimensional ist (also z.B. für $X = \mathbb{R}^n$), dann werden die Suprema zu Maxima. Nun zu der bereits angekündigten wichtigen Ungleichung.

Lemma A.2.10 Sei $\|\cdot\|_X$ eine Opertatornorm, dann gilt für alle $x \in X$

$$\|Ax\|_X \leq \|A\|_X \, \|x\|_X. \tag{A.22}$$

Beweis. Die Ungleichung gilt trivialerweise für $x = 0$. Sei also $x \neq 0$, dann gilt

$$\|Ax\|_X = \|x\|_X \frac{\|Ax\|_X}{\|x\|_X} \leq \|x\|_X \sup_{y\neq 0} \frac{\|Ay\|_X}{\|y\|_X} = \|A\|_X \, \|x\|_X$$

und damit die Behauptung. ∎

Die gerade bewiesene Eigenschaft findet man auch öfter als Definition in folgender Form: Eine Matrixnorm $\|\cdot\|$ auf $\mathbb{R}^{m\times n}$ heißt *kompatibel* oder *verträglich* mit der Vektornorm $\|\cdot\|$ auf $\mathbb{R}^n$, falls für alle $A \in \mathbb{R}^{m\times n}$ und alle $x \in \mathbb{R}^n$

$$\|Ax\| \leq \|A\| \, \|x\| \tag{A.23}$$

gilt. **Achtung**: Die Eigenschaft (A.23) bedeutet *nicht*, dass die Matrixnorm $\|\cdot\|$ eine Operatornorm ist!

Korollar A.2.11 Die p-Matrixnorm ist verträglich mit der p-Vektornorm, $1 \leq p \leq \infty$.

▪ **Beispiel A.2.12** Sei $A \in \mathbb{R}^{m\times n}$. Dann definiert

$$\|A\|_G := n \max_{i,j} |a_{ij}| = n \, \|A\|_{\max}.$$

eine Matrixnorm. ▪

Bemerkung A.2.13 Kombinationen von verträglichen Normen sind etwa

(i) Die Matrixnormen $\|\cdot\|_G, \|\cdot\|_\infty$ sind verträglich mit der Vektornorm $\|\cdot\|_\infty$;

(ii) Die Matrixnormen $\|\cdot\|_G, \|\cdot\|_1$ sind verträglich mit der Vektornorm $\|x\|_1$;

(iii) Die Matrixnormen $\|\cdot\|_G, \|\cdot\|_F, \|\cdot\|_2$ sind verträglich mit der Vektornorm $\|\cdot\|_2$.

Beweis. Übung. ∎

Die obigen Eigenschaften zeigen, dass die Matrixnormen $\|\cdot\|_G$ und $\|\cdot\|_F$ mit bestimmten Vektornormen kompatibel sind. Beide sind aber **keine Operatornormen**, d.h., es gibt keine Vektornorm, so dass man diese Matrixnormen in der Form (A.20) schreiben kann.

Mit diesen Eigenschaften können wir nun auch einen Zusammenhang der Euklidischen mit der Zeilen- und Spaltensummennorm beweisen.

Lemma A.2.14 Es sei $A \in \mathbb{R}^{m\times n}$. Dann gilt

$$\|A\|_2 \le \sqrt{\|A\|_1 \, \|A\|_\infty}\,. \tag{A.24}$$

Beweis. Nach Satz A.2.7 existiert ein Vektor $0 \neq z \in \mathbb{R}^n$ mit $A^T Az = \mu^2 z$ und $\mu = \|A\|_2$. Dann gilt wegen Bemerkung A.2.13 und Korollar A.2.11

$$\mu^2 \|z\|_1 = \|A^T Az\|_1 \le \|A^T\|_1 \, \|Az\|_1 \le \|A\|_\infty \, \|A\|_1 \, \|z\|_1, \tag{A.25}$$

wobei wir verwendet haben, dass die Spaltensummennorm von A^T gleich der Zeilensummennorm von A ist, also $\|A^T\|_1 = \|A\|_\infty$. Division durch $\|z\|_1$ und Wurzelziehen liefert das gewünschte Ergebnis. ■

A.3 Eigenwerte und -vektoren

Sei $A \in \mathbb{C}^{n\times n}$, $\lambda \in \mathbb{C}$ ein Eigenwert der Matrix A und $v \in \mathbb{C} \setminus \{0\}$ ein zugehöriger Eigenvektor:

$$Av = \lambda v. \tag{A.26}$$

Zusätzlich zur Definition (A.26) eines (rechtsseitigen) Eigenvektors $v \in \mathbb{C}^n \setminus \{0\}$ kann man auch linksseitige Eigenvektoren betrachten. $x \in \mathbb{C}^n \setminus \{0\}$ heißt linksseitiger Eigenvektor von A, falls $x^H A = \lambda x^H$ gilt, wobei x^H denjenigen Vektor bezeichnet, der aus x durch Bildung der konjugiert komplexen Einträge und Transponieren hervorgeht.

Satz A.3.1 Die Eigenwerte einer Matrix $A \in \mathbb{C}^{n\times n}$ sind gerade die n Nullstellen des zugehörigen charakteristischen Polynoms

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

Beweis. Die Eigenwertgleichung (A.26) besagt offensichtlich, daß $(A - \lambda I)v = 0$ gilt, also die Matrix $A - \lambda I$ singulär ist. Letzteres ist zu $\det(A - \lambda I) = 0$ äquivalent. ■

Definition A.3.2 Die Menge aller Eigenwerte einer Matrix $A \in \mathbb{C}^{n\times n}$ wird als *Spektrum* bezeichnet und man verwendet die Schreibweise

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

Satz A.3.3 Sei $\sigma(A) = \{\lambda_1, \dots, \lambda_n\}$. Es besteht folgender Zusammenhang zwischen der Determinante bzw. der Spur von A und den Eigenwerten λ_i:

$$\det(A) = \prod_{i=1}^{n} \lambda_i\,, \qquad \mathrm{Spur}(A) := \sum_{i=1}^{n} a_{ii} = \sum_{i=1}^{n} \lambda_i\,.$$

Beweis. Die zweite Identität erhält man z.B. dadurch, daß man den Koeffizienten vor λ^{n-1}

im charakteristischen Polynom untersucht.

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

was die Behauptung zeigt. ∎

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

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

wird als **invariant** bzgl. A bezeichnet. Das lineare Erzeugnis (Span) eines (rechtsseitigen) Eigenvektors bildet somit einen eindimensionalen Unterraum, der invariant bleibt unter der Multiplikation mit A. Es sei

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

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

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

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

Lemma A.3.4 Sei $A \in \mathbb{C}^{n\times n}$ beliebig sowie $X \in \mathbb{C}^{n\times n}$ eine reguläre Matrix. Dann gilt $\sigma(A) = \sigma(X^{-1}AX)$; ähnliche Matrizen haben also das gleiche Spektrum.

Beweis. Die Aussage folgt mit direkt mit Satz A.3.1 aus der Tatsache, daß ähnliche Matrizen das gleiche charakteristische Polynom haben:

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

und damit die Behauptung. ∎

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

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

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

Satz A.3.5 — Sylvester'scher Trägheitssatz. Es sei $A \in C^{n\times n}$ hermitesch und $X \in C^{n\times n}$ regulär. Dann haben A und $X^H AX$ den gleichen Rang sowie die gleiche Anzahl positiver bzw. negativer Eigenwerte.

Beweis. Siehe z.B. [Fis13]. ∎

Definition A.3.6 — Jordan-, bzw. Elementarmatrix. Eine Matrix $E_k(\lambda) \in \mathbb{C}^{k\times k}$ heißt Jordanmatrix (oder Elementarmatrix) zum Eigenwert λ, wenn

$$E_k(\lambda) = \begin{pmatrix} \lambda & 1 & & 0 \\ & \ddots & \ddots & \\ & & \ddots & 1 \\ 0 & & & \lambda \end{pmatrix}. \tag{A.27}$$

Satz A.3.7 — Jordansche Normalform (siehe z.B. [Fis13]). Zu jeder Matrix $A \in \mathbb{C}^{n\times n}$ existiert eine reguläre Matrix $T \in \mathbb{C}^{n\times n}$, so dass

$$A = T^{-1}JT\,,$$

wobei J, die durch die Paare $(\lambda_1, n_1), ..., (\lambda_k, n_k)$ mit $\lambda_i \in \mathbb{C}$, $n_i \geq 1$ (eindeutig bis auf die Reihenfolge) bestimmte Jordansche Normalform

$$J = \begin{pmatrix} E_{n_1}(\lambda_1) & & 0 \\ & \ddots & \\ 0 & & E_{n_k}(\lambda_k) \end{pmatrix}$$

von A ist.

Wir kommen nun zu einer wichtigen Matrixfaktorisierung, der Schur-Zerlegung. Im Gegensatz zur Jordan'schen Normalform führt ihre Berechnung auf numerisch stabile Algorithmen, da man hierfür nur orthogonale bzw. unitäre Transformationen benötigt und diese normerhaltend sind.

Satz A.3.8 — Satz von Schur. Sei $A \in C^{n\times n}$. Dann gibt es eine unitäre Matrix $Q \in C^{n\times n}$ ($QQ^H = Q^H Q = I$), so daß

$$Q^H AQ = R$$

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

Ist A reell, so reduziert sich die obige Aussage entsprechend.

Satz A.3.9 — Reelle Schur-Zerlegung. Für jedes $A \in \mathbb{R}^{n\times n}$, gibt es eine orthogonale

Matrix $Q \in \mathbb{R}^{n\times n}$, so daß $Q^T AQ$ in oberer Quasi-Dreiecksform

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

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

Wir betrachten noch Matrixfaktorisierungen für eine besondere Klasse von Matrizen.

Definition A.3.10 $A \in \mathbb{C}^{n\times n}$ heißt **normal**, wenn gilt $A^H A = AA^H$.

Bemerkung A.3.11 Alle hermiteschen, schiefhermiteschen ($A^H = -A$), Diagonal- und unitären Matrizen sind Beispiele für normale Matrizen.

Korollar A.3.12 — Schur-Zerlegung von normalen Martizen. Eine Matrix $A \in \mathbb{C}^{n\times n}$ ist genau dann normal, wenn eine unitäre Matrix $Q \in \mathbb{C}^{n\times n}$ existiert mit $Q^H AQ = D = \operatorname{diag}(\lambda_1, \ldots, \lambda_n)$.

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

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

Die Behauptung folgt nun aus der Tatsache, daß eine normale, obere Dreiecksmatrix eine Diagonalmatrix ist. ■

Korollar A.3.13 — Schur-Zerlegung von symmetrischen Martizen. Jede reelle, symmetrische Matrix $A \in \mathbb{R}^{n\times n}$ läßt sich mittels einer orthogonalen Matrix $Q \in \mathbb{R}^{n\times n}$ auf Diagonalgestalt transformieren

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

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

A.4 Gram-Schmidtsches Orthogonalisierungsverfahren

Dieses Verfahren erzeugt aus einem System linear unabhängiger Vektoren $\{w_1, \ldots, w_n\}$ aus einem Vektrorraum mit Skalarprodukt $\langle \cdot, \cdot \rangle$ (einem sogenannten *Prä-Hilbert-Raum*). Es ist nach Jørgen Pedersen Gram (1850-1916) und Erhard Schmidt (1876-1959) benannt, findet sich allerdings bereits in Arbeiten von Pierre-Simon Laplace (1749-1827) und Augustin-Louis Cauchy (1789-1857). Wir formulieren das Verfahren als Satz, vgl. [Fis13, §5.4.9]

Satz A.4.1 — Gram-Schmidt. Sei $\{w_1, ..., w_n\}$ linear unabhängig, dann ist $\{v_1, ..., v_n\}$ definiert durch $v_1 := w_1$ und

$$v_k := w_k - \sum_{i=1}^{k-1} \frac{\langle v_i, w_k \rangle}{\langle v_i, v_i \rangle} v_i, \quad k = 2, ..., n,$$

ein Orthonomalsystem. ■

Anhang B

Eine kurze Einführung in MATLAB®

Die Beschäftigung mit numerischen Methoden bringt es zwangsläufig mit sich, dass diese umgesetzt, also implementiert, werden. Letztlich können nicht alle Eigenschaften von numerischen Methoden bewiesen werden (z.B. wieviele Sekunden benötigt ein Verfahren auf einem Laptop?), so dass numerische Experimente unerläßlich sind. Ebenso lassen sich viele Größen (Konstanten, Laufzeiten, Speicherbedarf, u.s.w.) nicht, oder nur asymptotisch untersuchen. Daher sind auch hier numerische Experimente notwendig.

Die Implementierung bedingt immer die Wahl einer Programmiersprache. Dies kann eine höhere Programmiersprache sein, z.B. `C++`, `C#`, `Java` oder `Python`. Es gibt aber auch „mathematische Standardsoftware“ wie etwa MATLAB®oder auch der kostenlose Klon OCTAVE. Diese haben gegenüber einer höheren Programmiersprache oftmals den Vorteil einer sehr einfachen Bedienung, was jedoch in vielen Fällen auf Kosten der Effizienz geht. Will man „schnell“ ein Verfahren umsetzen, dann ist mathematische Standardsoftware bequem, braucht man hohe Effizienz, dann ist eine höhere Programmiersprache zu bevorzugen. Gerade weil es sich hier um eine Einführungsvorlesung handelt, geben wir eine kurze Einführung in MATLAB®. Im Internet finden Sie eine Vielzahl von Einführungen und es gibt auch entsprechende Lehrbücher, z.B. [QSG14].

B.1 Grundlegende MATLAB®-Befehle

Zunächst ist zu betonen, dass sich die Verwendung von MATLAB® vom Betriebssystem und der jeweiligen Version abhängig sein kann, Erscheinungsform, Fenster, Konsolen u.s.w. können sich unterscheiden. Ruft man das Programm MATLAB® mit dem Befehl `matlab` auf, so erscheinen auf dem Monitor einige Fenster. Auf den Linux-Rechnern des kiz müssen Sie vorher die notwendigen Pfade ergänzen. Geben Sie dazu in einem Konsole-Fenster den Befehl `option matlab` ein.

Von erscheinenden Fenstern ist das Befehl-Fenster der primäre Ort, um Befehle einzugeben und Fehler oder Ergebnisse abzulesen. Das Prompt-Zeichen

$$\gg$$

ist im Befehl-Fenster dargestellt und dort findet man üblicherweise einen blinkenden Cursor. Der blinkende Cursor und der MATLAB®-Prompt zeigen an, dass MATLAB® eine Eingabe erwartet.

B.1.1 Einfache mathematische Operationen

Genauso wie mit einem Taschenrechner kann man auch mit MATLAB® einfache mathematische Operationen ausführen, z.B. ergibt die Eingabe

```
≫ 3 + 4
```

die Ausgabe

```
ans =
      7
```

Man beachte, dass MATLAB® im Allgemeinen keine Leerzeichen benötigt, um Befehle eindeutig zu verstehen. Alternativ zu dem obigen Beispiel können auch Variablen verwendet werden:

```
≫ a = 3
a =
      3
≫ b = 4
b =
      4
≫ c = a + b
c =
      7
```

MATLAB® besitzt folgende einfache arithmetische Operationen:

Operation	**Symbol**	**Beispiel**
Addition, $a + b$	+	5 + 3
Subtraktion, $a - b$	–	23–12
Multiplikation, $a \cdot b$	*	13.3 * 63.13
Division, $a \div b$	/ or \	$17/4 = 4\backslash 17$
Potenz, a^b	⌢	3^4

Die Reihenfolge, in der eine Folge von Operationen abgearbeitet wird, lässt sich wie folgt beschreiben. Ausdrücke werden

- von links nach rechts ausgeführt,
- wobei die Potenzierung die höchste Priorität besitzt
- gefolgt von den Punktoperation Multiplikation und Division,
- die geringste Priorität haben Addition und Subtraktion.

Mit Hilfe von Klammern kann diese Vorgehensweise geändert werden, wobei die Ausdrücke in den inneren Klammern vor denjenigen in den äußeren Klammern ausgwertet werden.

B.1.2 Variablen

Wie in anderen Programmiersprachen hat auch MATLAB® Regeln für Variablennamen. Eine Variable repräsentiert ein Datenelement, dessen Wert während der Programmausführung – gegebenenfalls mehrfach – geändert werden kann. Variablen werden anhand ihrer „Namen“ identifiziert. Namen bestehen aus ein bis neunzehn Buchstaben, Ziffern oder Unterstrichen, *wobei das erste Zeichen ein Buchstabe sein muss.* Man beachte, dass

MATLAB® Groß- und Kleinschreibung unterscheidet. (Windows ist im Gegensatz zu Linux nicht so restriktiv, wenn man jedoch Programme austauschen möchte, sollten Windows-Benutzer besondere Aufmerksamkeit walten lassen.)

Einer Variablen ist Speicherplatz zugeordnet. Wenn man eine Variable verwendet, dann meint man damit entweder den zugeordneten Speicherplatz oder den Wert, der dort augenblicklich abgespeichert ist. Einen Überblick über alle Variablen erhält man mit dem Befehl `who` oder `whos`, wobei letzterer die Angabe des benutzten Speicherplatzes beinhaltet.

Zusätzlich zu selbstdefinierten Variablen gibt es in MATLAB® verschiedene spezielle Variablen. Diese lauten:

spezielle Variable	**Wert**
`ans`	standard Variablenname benutzt für Ergebnisse
`pi`	$3.1415\ldots$
`eps`	Maschinengenauigkeit
`flops`	Zähler für die Anzahl der Fließkommaoperationen
`inf`	steht für Unendlich (eng. infinity). z.B. $1/0$
`NaN`	englisch *Not a Number*, z.B. $0/0$
`i (und) j`	$i = j = \sqrt{-1}$

Der Speicherplatz, der durch Variablen belegt ist, durch den Befehl `clear` wieder freigegeben werden, z.B.

```
>> clear a b c
```

B.1.3 Kommentare und Punktion

Der Text, der nach einem Prozentzeichen „%“ folgt, wird in MATLAB® als Kommentar verstanden:

```
>> dummy = 4 % Wert von dummy
dummy =
      4
```

Mehrere Befehle können in eine Zeile geschrieben werden, wenn sie durch Kommata oder Semikola getrennt werden. Kommata veranlassen MATLAB®, die Ergebnisse anzuzeigen. Bei einem Semikolon wird die Ausgabe unterdrückt. Durch eine Sequenz von drei Punkten kann man einen Befehl in der folgenden Zeile fortsetzen.

B.1.4 Spezielle Funktionen

Eine unvollständige Liste von Funktionen, die MATLAB® bereitstellt, ist im folgenden dargestellt. Die meisten Funktionen sind selbsterklärend, z.B.

Funktion	Bedeutung
`abs(x)`	Absolutbetrag
`cos(x)`	Kosinus
`exp(x)`	Exponentialfunktion: e^x
`fix(x)`	rundet auf die nächste, vom Betrag her kleinere ganze Zahl
`floor(x)`	rundet auf die nächste, kleinere ganze Zahl
`gcd(x,y)`	größter gemeinsamer Teiler von x und y
`lcm(x,y)`	kleinstes gemeinsames Vielfaches von x und y
`log(x)`	natürlicher Logarithmus
`rem(x,y)`	Modulo (engl. remainder of division), z.B. rem(5,2)=1
`sign(x)`	Signum Funktion, z.B. sign(2.3) = 1, sign(0) = 0, sign(-.3) = -1
`sin(x)`	Sinus
`sqrt(x)`	Quadratwurzel
`tan(x)`	Tangens

Ein kleines Beispiel:

```
>> y = cos(pi)
y =
      -1
```

B.1.5 Skript-Dateien

Für einfache Probleme ist es schnell und effizient, die Befehle am MATLAB®-Prompt einzugeben. Für größere und umfangreichere Aufgabenstellungen bietet MATLAB® die Möglichkeit, sogenannte *Skript-Dateien* zu verwenden, in denen die Befehle in Form einer Textdatei eingegeben werden und die man am Prompt ausführt. MATLAB® öffnet dann diese Dateien und führt die Befehle so aus, als hätte man sie am Prompt eingegeben. Skript-Dateien werden auch *M-Dateien* genannt, wobei der Ausdruck „M-Datei" daher rührt, dass diese Dateien die Dateiendung (Suffix) `.m` haben, z.B. `newton.m`.

Um eine M-Datei zu erstellen, ruft man einen Editor auf und speichert die Datei in dem Verzeichnis, von dem aus man MATLAB® gestartet hat oder starten wird. Die Datei (z.B. `newton.m`), wird in MATLAB® dann durch Eingabe von `newton` am Prompt aufgerufen.

Für die Benutzung von Skript-Dateien hat MATLAB® unter anderem folgende hilfreichen Befehle:

M-Datei-Funktionen	
`disp(ans)`	zeigt den Wert der Variablen `ans`, ohne ihren Namen auszugeben
`input`	erwartet vom Benutzer eine Eingabe
`keyboard`	übergibt zeitweise die Kontrolle an die Tastatur
`pause`	hält das Programm an, bis eine Taste betätigt wird

Die folgende Skript-Datei `beispiel1.m`

```
% beispiel1.m
% Beispiel fuer eine Skript-Datei
tmp = input(' Geben Sie bitte eine Zahl an: ' );
3 * tmp;
```

führt zu der Ausgabe:

```
>> beispiel1

Geben Sie bitte eine Zahl an:  6

ans =

     18
```

B.1.6 Dateiverwaltung

MATLAB® unterstützt eine Vielzahl von Dateiverwaltungs–Befehlen, welche es ermöglichen, Dateien zu listen, Skript-Dateien anzusehen oder zu löschen und Verzeichnisse zu wechseln.

Datei-Managment-Funktionen	
`cd path`	wechselt in das Verzeichnis `path`
`delete beispiel`	löscht die Datei `beispiel.m`
`ls`	zeigt alle Dateien im aktuellen Verzeichnis an
`pwd`	zeigt den aktuellen Verzeichnispfad an
`type beispiel`	zeigt den Inhalt der Datei `beispiel.m` im Befehl-Fenster
`what`	zeigt alle M- und MAT-Dateien im aktuellen Verzeichnis an

B.1.7 Hilfe

MATLAB® bietet verschiedene Hilfsoptionen an. Die Online-Hilfe ruft man über den `help` auf, zum Beispiel:

```
>> help sqrt

 SQRT   Square root.
        SQRT(X) is the square root of the elements of X. Complex
        results are produced if X is not positive.

        See also SQRTM.
```

Wir erhalten also zu einem Befehl (hier `sqrt`) sowohl die Beschreibung als auch die Syntax. Wir müssen hierzu also den Befehl kennen. Wenn dies nicht der Fall ist, bietet `lookfor` eine Stichwortsuche. Hier werden alle ersten Zeilen der MATLAB® Hilfe-Kennwörter und Skript-Dateien, die im MATLAB® Suchpfad zu finden sind, durchsucht. Das Bemerkenswerte dabei ist, dass dieser Begriff kein Befehl zu sein braucht, wie folgendes Beispiel aus der Vorlesung zeigt:

```
>> lookfor cholesky

  CHOL    Cholesky factorization

>>  CHOL   Cholesky factorization.

  CHOL(X) uses only the diagonal and upper triangle of X.
  The lower triangular is assumed to be the (complex conjugate)
  transpose of the upper.  If X is positive definite, then
  R = CHOL(X) produces an upper triangular R so that R'*R =X.
  If X is not positive definite, an error message is printed.

  With two output arguments, [R,p] = CHOL(X) never produces an
  error message.  If X is positive definite, then p is 0 and R
  is the same as above.   But if X is not positive definite, then
  p is a positive integer and R is an upper triangular matrix of
  order q = p-1 so that R'*R = X(1:q,1:q).
```

Wir können dieses Beispiel sogar noch ein bisschen weiter führen und nach einem Oberbegriff suchen:

```
>> lookfor  factorization

  CHOL    Cholesky factorization.
  QRDELETE Delete a column from the QR factorization.
  QRINSERT Insert a column in the QR factorization.
  SYMBFACT Symbolic factorization analysis.
```

Eine weitere Möglichkeit, sich Hilfe zu verschaffen, besteht darin, den Helpdesk aufzurufen. Wenn Sie helpdesk am Prompt eingeben, öffnet sich eine Hilfsumgebung. Und natürlich sind Suchmaschinen oder auch `ChatGBT` oftmals gute Ratgeber*innen.

B.2 Mathematik mit Matrizen

MATLAB® ist eine Abkürzung für **Mat**rix **Lab**oratory. Es wurde ursprünglich speziell für Anwendungen in der Numerischen Linearen Algebra konzipiert. Es ist daher nicht erstaunlich, dass MATLAB® besonders im Umgang mit Matrizen besondere Stärken hat.

B.2.1 Matrixkonstruktion und Adressierung

Wir zeigen zunächst einige Möglichkeiten, Matrizen in MATLAB® zu definieren.

einfache Matrix Konstruktionen	
`x=[1 4 2*pi 4]`	erstelle einen Zeilenvektor `x` mit Einträgen
`x=anfang:ende`	erstelle einen Zeilenvektor `x` beginnend mit `anfang`, Inkrement 1 und endend mit `ende`
`x=anfang:inkrement:ende`	Ähnliches wie oben mit dem Inkrement `inkrement`
`x=linspace(anfang,ende,n)`	erzeugt einen Zeilenvektor der Dimension n mit $\mathtt{x}(i) = \frac{(n-i)\cdot\mathtt{anfang}+(i-1)\cdot\mathtt{ende}}{n-1}$

Im Folgenden zeigen wir einige typische Beispiele aufgeführt:

```
≫ B = [1 2 3 4; 5 6 7 8]
B =
      1 2 3 4
      5 6 7 8
```

Der Operator „'“ liefert für reelle Matrizen die Transponierte:

```
≫ C = B'
C =
      1 5
      2 6
      3 7
      4 8
```

Der Doppelpunkt „:“ in der zweiten Komponente spricht alle vorhandenen Spalten an, d.h., dies ist ein zu `1:4` äquivalenter Ausdruck:

```
≫ C = B(1,:)
C =
      1 2 3 4
≫C = B(:,3)'
C =
      3 7
```

Es lassen sich auch einzelne Komponenten neu definieren:

```
≫A = [1 2 3; 4 5 6; 7 8 9]
A =
      1 2 3
      4 5 6
      7 8 9
≫ A(1,3) = 9
A =
      1 2 9
      4 5 6
      7 8 9
```

Ist ein Eintrag noch nicht definiert, so verwendet MATLAB® die minimale Erweiterung dieser Matrix und setzt undefinierte Einträge zu Null:

```
≫ A(2,5) = 4
A =
      1 2 9 0 0
      4 5 6 0 4
      7 8 9 0 0
```

Im Folgenden werden die Vektoren `(3,2,1)` und `(2,1,3,1,5,2,4)` dazu verwendet, eine Matrix `C` (mit noch zu definierenden Einträgen `A(i,j)`) zu indizieren, d.h. `C` hat die (Index-)Struktur

```
A(3,2) A(3,1) A(3,3) A(3,1) A(3,5) A(3,2) A(3,4)
A(2,2) A(2,1) A(2,3) A(2,1) A(2,5) A(2,2) A(2,4)
A(1,2) A(1,1) A(1,3) A(1,1) A(1,5) A(1,2) A(1,4)
```

Damit erhält man nun:

```
>> C=A(3:-1:1,[2 1 3 1 5 2 4])
C =
      8 7 9 7 0 8 0
      5 4 6 4 4 5 0
      2 1 9 1 0 2 0
```

Ein weiteres Beispiel für Indizierung ist

```
>> C=C(1:2,2:3)
C =
      7 9
      4 6
```

dann würde sich `C(1,3)=9` ergeben. Im nächsten Beispiel wird ein Spaltenvektor dadurch konstruiert, dass alle Elemente aus der Matrix `C` hintereinander gehängt werden. Dabei wird spaltenweise vorgegangen:

```
>> b=C(:)'
b =
      7 4 9 6
```

Das Löschen einer ganzen Zeile oder Spalte kann durch das Umdefinieren in eine 0×0-Matrix geschehen, z.B.

```
>> C(2,:)=[ ]
C =
      7 9
```

B.2.2 Skalar-Matrix-Operationen

In MATLAB® sind Skalar-Matrix-Operationen in dem Sinne definiert, dass Addition, Subtraktion, Division und Multiplikation mit einem Skalar elementweise durchgeführt werden. Es folgen zwei (selbst-)erklärende Beispiele.

```
>> B - 1
ans =
      0 1 2 3
      4 5 6 7
>> 9 + 3 * B
ans =
      12 15 18 21
      24 27 30 33
```

B.2.3 Matrix-Matrix-Operationen

Die Operationen zwischen Matrizen sind nicht so kanonisch zu definieren wie die zwischen Skalar und Matrix, insbesondere sind Operationen zwischen Matrizen unterschiedlicher Dimension schwer zu definieren. Des Weiteren sind die Operationen $*$ und `.*`, bzw. / und `./` sowie \ und `.\` zu unterscheiden. In nachfolgender Tabelle sind die Matrixoperationen beschrieben:

komponentenweise Matrixoperationen	
Beispieldaten	$a = [a_1, a_2, \ldots, a_n]$, $b = [b_1, b_2, ..., b_n]$, c ein Skalar
komp. Addition	`a+c=` $[a_1 + c, a_2 + c, ..., a_n + c]$
komp. Multiplikation	`a*c` = $[a_1 \cdot c, a_2 \cdot c, ..., a_n \cdot c]$
Matrix–Addition	`a+b` = $[a_1 + b_1, a_2 + b_2, ..., a_n + b_n]$
komp. Matrix-Multiplikationen	`a.*b` = $[a_1 \cdot b_1, a_2 \cdot b_2, ..., a_n \cdot b_n]$
komp. Matrix-Div. von rechts	`a./b` = $[a_1/b_1, a_2/b_2, ..., a_n/b_n]$
komp. Matrix-Div. von links	`a.\b` = $[b_1/a_1, b_2/a_2, ..., b_n/a_n]$
komp. Matrix-Potenz	`a.^c` = $[a_1^c, a_2^c, ..., a_n^c]$
	`c.^a` = $[c^{a_1}\ c^{a_2}, ..., c^{a_n}]$
	`a.^b` = $[a_1^{b_1}\ a_2^{b_2}, ..., a_n^{b_n}]$

Es folgen nun einige Beispiele zu Matrixoperationen:

```
>> g=[1 2 3; 4 5 6]; % zwei neue Matrizen
>> h=[2 2 2; 3 3 3];
>> g+h   % addiere g und h komponentenweise
ans =
       3   4   5
       7   8   9
>> ans-g   % subtrahiere g von der vorherigen Antwort
ans =
       2   2   2
       3   3   3
>> h.*g   % multipliziere g mit h komponentenweise
ans =
        2    4    6
       12   15   18
>> g*h'   % multipliziere g mit h'
ans =
       12   18
       30   45
```

B.2.4 Matrix-Operationen und -Funktionen

Nun einige Operationen die sich auf Matrizen anwenden lassen:

Matrixfunktionen	
`reshape(A,m,n)`	erzeugt aus den Eintägen der Matrix `A` eine $m \times n$-Matrix, wobei die Einträge spaltenweise aus `A` gelesen werden.
`diag(A)`	ergibt die Diagonale von `A` als Spaltenvektor
`diag(v)`	erzeugt Diagonalmatrix mit dem Vektor `v` in der Diagonalen
`tril(A)`	extrahiert den unteren Dreiecksanteil der Matrix `A`
`triu(A)`	extrahiert den oberen Dreiecksanteil der Matrix `A`

Es folgen einige Beispiele:

```
>> g = linspace(1,9,9)   % ein neuer Zeilenvektor
g =
        1    2    3    4    5    6    7    8    9
>> B = reshape(g,3,3)   % macht aus g eine 3 x 3 Matrix
B =
        1    4    7
        2    5    8
        3    6    9
>> tril(B)
ans =
        1    0    0
        2    5    0
        3    6    9
```

Wir stellen noch einige Befehle zusammen:

Funktion	**Bedeutung**
`R=chol(A)`	Cholesky–Zerlegung
`cond(A)`	Konditionszahl der Matrix `A`
`d=eig(A)`	Eigenwerte und -vektoren
`[V,d]=eig(A)`	
`det(A)`	Determinante
`hess(A)`	Hessenberg–Form
`inv(A)`	Inverse
`[L,U]=lu(A)`	Zerlegung gegeben durch Gauss-Algorithmus
`norm(A)`	euklidische-Norm
`rank(A)`	Rang der Matrix `A`

Bemerkung B.2.1 Der \ Operator ist auch für Matrizen definiert und liefert in Kombination mit Vektoren für reguläre Matrizen ihre Inverse, d.h. `A\x=A^(-1)x`.

B.2.5 Spezielle Matrizen

Einige häufig auftretende spezielle Matrizen sind im folgenden aufgelistet:

	spezielle Matrizen
`eye(n)`	erzeugt eine Einheitsmatrix der Dimension `n`
`ones(m,n)`	erzeugt eine $m \times n$-Matrix mit den Einträgen `1`
`zeros(m,n)`	erzeugt eine $m \times n$-Matrix mit den Einträgen `0`

B.2.6 Spezielle Funktionen für schwachbesetzte Matrizen

Bei vielen numerischen Anwendungen treten schwachbesetzte Matrizen auf (ein Beispiel dafür haben wir in der Vorlesung vorgestellt). MATLAB® hat für solche Matrizen besondere „Sparse-Funktionen“, die dieser Eigenschaft Rechnung tragen:

Funktion	**Bedeutung**
`find(A)`	findet Indizes von Nichtnulleinträgen
`nnz(A)`	Anzahl an Nichtnulleinträgen
`spdiags(v)`	erzeugt eine Sparse-Diagonalmatrix mit Vektor `v` als Diagonale
`speye(n)`	erzeugt eine Sparse-Einheitsmatrix
`spy(A)`	visualisiert die Struktur der Matrix `A`

Wir geben eine kurze Illustration der Funktionsweise obiger Befehle anhand einiger Beispiele, die hoffentlich selbsterklärend sind:

```
>> E=eye(100);    % vollbesetzte 100 x 100 Einheitsmatrix
>> Es=sparse(E);  % Sparse-Version von E
>> whos
       Name      Size     Elements  Bytes  Density  Comple
          E   100 by 100     10000  80000     Full      No
         Es   100 by 100       100   1600   0.0100      No
 Grand total is 10100 elements using 81600 bytes

>> A=spdiags([7*ones(4,1),ones(4,1),2*ones(4,1)],[-1,0,1],4,4);
>> nnz(A)
ans =
      10
>> full(A)
ans =
       1   2   0   0
       7   1   2   0
       0   7   1   2
       0   0   7   1
```

Als Beispiel für `spy` sei hier die Besetzungstruktur einer 3D-FEM[1] Steifigkeitsmatrix in Abb. B.1 gezeigt. Die schwarzen Punkte bedeuten, dass in der Matrix der Dimension 256×256 ein Nichtnull-Eintrag steht, weiß hingegen, dass der entsprechende Eintrag Null ist. Die Matrix hat `nz=849` Nichtnull-Einträge (von insgesamt `256*256=65.536` Einträgen).

B.3 Datenverwaltung

Für die meisten Anwendungen genügt es, Daten (in Formen von Feldern, Arrays oder Matrizen) in einem geeigneten Format abzuspeichern und wieder laden zu können. Die Befehle `load` und `save` setzen voraus, dass die Daten in einem System unabhängigen, binären Format in einer Datei mit dem Suffix `.mat` gespeichert sind oder in einem einfachen ASCII-Format vorliegen. Wir beschreiben dies nun im Einzelnen.

B.3.1 Daten speichern

Im Folgenden wird eine 3×5-Matrix im binär-Format in der Datei `A.mat` gespeichert. Diese Daten sind sehr kompakt gespeichert.

[1] FEM: Finite Elemente Methode – eine Methode zur numerischen Lösung partieller Differenzialgleichungen. Die FEM führt auf ein lineares Gleichungssystem $Ax = b$, deren Systemmatrix A auch „Steifigkeitsmatrix“ genannt wird.

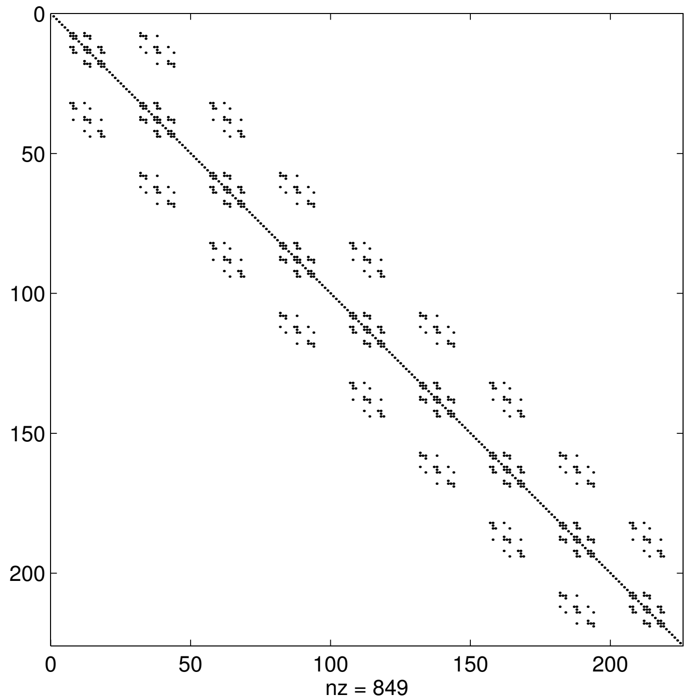


Abbildung B.1: Besetzungstruktur einer 3D-FEM Steifigskeitsmatrix.

```
>> A=zeros(3,5);
>> save A
```

Gibt man sich aber den Inhalt dieser Datei auf dem Bildschirm aus, so gibt er wenig Sinn. Möchte man sich also z.B. einen Lösungsvektor sichern um ihn später „per Hand zu analysieren“, so speichere man die Daten als ASCII-Datei, dabei kann man wählen zwischen einer 8-stelligen oder 16-stelligen Abspeicherung.

```
>> save mat1.dat A -ascii          %  8-stellige Speicherung
>> save mat2.dat A -ascii -double  % 16-stellige Speicherung
```

Hier wurde die Matrix mit 8-stelligem Format in der Datei `mat1.dat` gespeichert, bzw. 16-stellig in der Datei `mat2.dat`.

B.3.2 Daten laden

Mit dem Befehl `load A` versucht MATLAB®, die in `A.mat` gespeicherten Daten in ein Datenfeld `A` zu laden. Auch ASCII-Dateien kann MATLAB® lesen. Da es hier jedoch keine Standardendung gibt, ist die Datei inklusive Endung anzugeben. Es ist darauf zu achten, dass ein rechteckiges Feld an Daten vorliegt, d.h. dass m Zeilen mit jeweils n numerischen Werten vorliegen. MATLAB® erstellt dann eine $m \times n$-Matrix mit dem Namen der Datei ohne Suffix.

```
>> load  mat1.dat
>> whos

    Name        Size     Elements  Bytes  Density  Complex
    mat1       3 by 5          15    120     Full      No

 Grand total is 15 elements using 120 bytes
```

B.4 Ausgabe von Text

Mit dem Befehl `fprintf` lassen sich Strings (Zeichenfolgen) auf dem Bildschirm ausgeben.

```
>> fprintf('\n Hello world %12.3e\n',4);

 Hello world    4.000e+00
```

Man sieht, dass der auszugebende Text zusätzliche Zeichen enthält, die nicht mit ausgedruckt werden. Diese Zeichen nennt man „Escape-Sequenzen“. In obigem Beispiel ist die Sequenz `\n` eingebaut. `\n` steht für „newline“ und sorgt dafür, dass bei der Textausgabe an diesen Stellen eine neue Zeile begonnen wird. Der Ausdruck `%12.3e` dient als Platzhalter für einen reellen Wert, der durch Komma getrennt hinter der Zeichenkette folgt. Dabei sei die Zahl in Exponentialdarstellung auszugeben, wofür 12 Stellen mit 3 Nachkommastellen bereitgestellt. Im Beispiel ist dies der Wert 4. Anstatt eines expliziten Wertes können auch Variablen oder Ausdrücke, z.B. `3 * 4` stehen. Ein Platzhalter kann mehrfach in einem `printf`-Befehl vorkommen. In diesem Fall müssen hinter der Zeichenkette genau so viele Werte folgen, wie Platzhalter angegeben sind. Die Reihenfolge der Werte muss mit der Reihenfolge der Platzhalter übereinstimmen, da die Ausdrücke von links nach rechts bewertet werden. Die Escape-Sequenzen dürfen im Text an beliebiger Stelle stehen.

Dazu einige Beispiele:

Ausgabeformate

Befehl	**Ausgabe**
`fprintf('%.0e\n',1.234567)`	`1e00+`
`fprintf('%.2e\n',1.234567)`	`1.23e00+`
`fprintf('%.5e\n',1.234567)`	`1.23456e00+`
`fprintf('%10.0e\n',1.234567)`	`     1e00+`
`fprintf('%10.2e\n',1.234567)`	`  1.23e00+`
`fprintf('%10.5e\n',1.234567)`	`1.2346e00+`
`fprintf('%10.2f\n',1.234567)`	`      1.23`
`fprintf('%10.5f\n',1.234567)`	`   1.23457`

Beachte: MATLAB® rundet numerische Werte bei der Ausgabe, wenn nötig! ⚠

B.5 Kontrollbefehle

Man kann in MATLAB® analog zu einer höheren Programmiersprache programmieren. Wir zeigen einige Beispiele dazu.

B.5.1 For-Schleifen

1. Mit jeglicher gültigen Matrix-Darstellung lässt sich eine FOR-Schleife definieren, z.B.

```
>> data = [1 7 3 2; 5 4 7 2]
data =
      1 7 3 2
      5 4 7 2
>> for n=data
     x=n(1)-n(2)
   end
x =
      -4
x =
       3
x =
      -4
x =
       0
```

2. FOR-Schleifen können nach Belieben geschachtelt werden:

```
>> for k=3:5
     for l=4:-1:2
      A(k,l)=k^2-l;
     end
   end
>> A
A =
       0  0  0  0
       0  0  0  0
       0  7  6  5
       0 14 13 12
       0 23 22 21
```

3. FOR-Schleifen sollten vermieden werden, wann immer sie durch eine **äquivalente Matrix-Darstellung** ersetzt werden können. Der folgende Ausdruck ist darunter in optimierter Version aufgeführt.

```
>> for n=1:10
     x(n)=sin(n*pi/10);
   end
>>x
x =
      Columns 1 through 7
        0.3090  0.5878  0.8090  0.9511  1.0000  0.9511  0.8090
      Columns 8 through 10
        0.5878  0.3090  0.0000
```

Nun in optimierter Form:

```
>> n=1:10;
>> x=sin(n*pi/10);
>> x
x =
      Columns 1 through 7
        0.3090   0.5878   0.8090   0.9511   1.0000   0.9511   0.8090
      Columns 8 through 10
        0.5878   0.3090   0.0000
```

4. Um die Ausführungsgeschwindigkeit zu maximieren, **sollte benötigter Speicherplatz vor Ausführung der FOR-Schleife allokiert werden**!

```
>> x=zeros(1,10);
>> x(1)=0.5;
>> for n=2:10
     x(n)=x(n-1)*sin(n*pi/10);
   end
>> x
x =
      Columns 1 through 7
        0.5000   0.2939   0.2378   0.2261   0.2261   0.2151   0.1740
      Columns 8 through 10
        0.1023   0.0316   0.0000
```

B.5.2 WHILE-Schleifen

WHILE-Schleifen sind wie folgt aufgebaut.

```
while Aussage
      Anweisungen
end
```

Dabei werden die Anweisungen zwischen `while` und `end` so lange ausgeführt, wie `Aussage` wahr ist, z.B.

```
>> num=0; EPS=1;
>> while (1+EPS) > 1
     EPS=EPS/2;
     num=num+1;
   end
>> num
num =
      53
```

B.5.3 IF-ELSE-END Konstrukte

Eine IF-ELSE-END-Schleife enthält nach dem IF eine Aussage, die daraufhin überprüft wird, ob sie wahr oder falsch ist. Ist sie wahr, so werden die in den folgenden Zeilen stehenden Anweisungen ausgeführt und die Schleife beendet. Ist sie falsch, so erfolgen die Anweisungen, die dem ELSE folgen (ELSE ist optional). Solch ein Konstrukt kann erweitert werden um beliebig viele ELSIF Befehle, die dieselbe Funktion haben wie der am Anfang stehende IF Befehl, aber nur beachtet werden, falls alle vorher überprüften Aussagen falsch sind.

```
if Aussage1
      Anweisungen, wenn Aussage1 wahr
elseif Aussage2
      Anweisungen, wenn Aussage1 falsch und Aussage2 wahr
else
      Anweisungen, wenn Aussage1 und Aussage2 falsch
end
```

B.5.4 Relationen und logische Operatoren

Einige Relationen und logische Operatoren sind in den folgenden Tabellen gelistet.

	Relationen
<	kleiner als
<=	kleiner als oder gleich
>	größer als
>=	größer als oder gleich
==	gleich
~=	ungleich

	logische Operatoren
&	UND
|	ODER
~	NICHT

B.6 Graphische Darstellung

Ergebnisse von numerischen Simulationen müssen in vielen Fällen graphisch dargestellt werden. Hierzu sind in MATLAB® einige sehr nützliche Befehle realisiert.

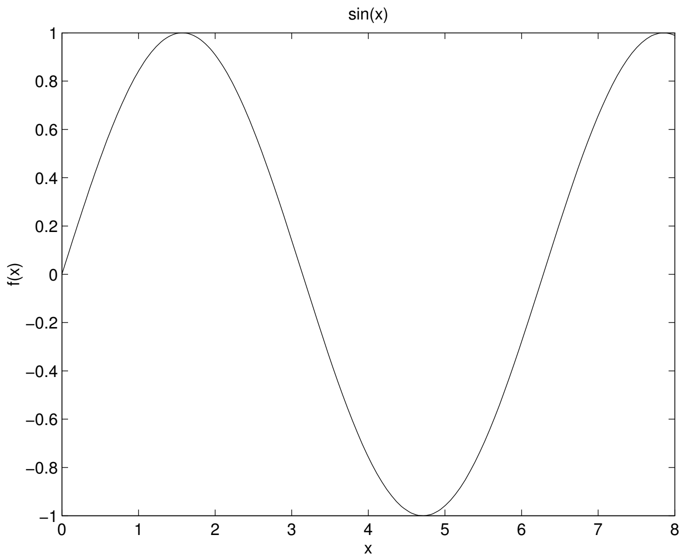
B.6.1 Zweidimensionale Graphiken

Wir beginnen mit einigen Befehlen zur Darstellung von Funktionen $f : \mathbb{R} \to \mathbb{R}$ in einem zweidimensionalen Graphen.

```
>> f='sin(x)';
>> fplot(f,[0 8]);
>> title(f),xlabel('x'),ylabel('f(x)');
```

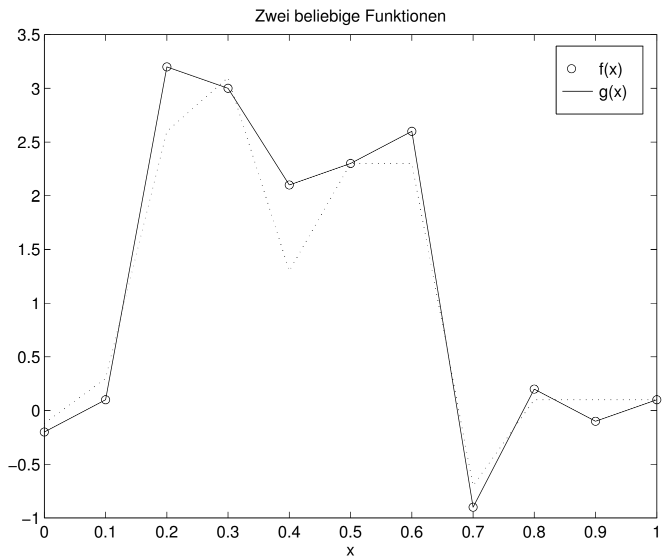
Das Ergebnis sieht dann wie folgt aus:

In obigem Beispiel war die zu plottende Funktion als Funktionsterm ($\sin(x)$) gegeben. Häufiger tritt es auf, dass ein Ergebnis numerischer Berechnungen nur an bestimmten Punkten gegeben ist. Diese werden dann wie folgt dargestellt:




```
>> x=0:0.1:1;
>> y=[-0.2 0.1  3.2  3  2.1  2.3  2.6 -0.9 0.2 -.1 .1];
>> z=[-0.12 0.3  2.6  3.1  1.3  2.3  2.3 -0.7 0.1 .1 .1];
>> plot(x,y,'o',x,y,x,z,':');
>> title('Zwei beliebige Funktionen'),xlabel('x');
>> legend('f(x)','g(x)')
```

Als Ergebnis erhalten wir:



Es gibt eine Vielzahl von Optionen zur Darstellung von Graphen. Wir stellen einige zusammen:

Linientypen und Farben

Symbol	Farbe	Symbol	Linientyp
`y`	gelb	$\cdot$	Punkt
`m`	magenta	$\circ$	Kreis
`c`	cyan	x	x-Markierung
`r`	rot	$+$	+-Markierung
`g`	grün	$*$	Sternchen
`b`	blau	$-$	durchgezogene Linie
`w`	weiß	:	gepunktete Linie
`k`	schwarz	$-.$	Strichpunkt-Linie
		$--$	gestrichelte Linie

Einige Befehle für Graphiken:

2-D Graphikanweisung

`axis`	modifiziert die Axen-Proportionen
`clf`	löscht die Graphik im Graphik-Fenster
`close`	schließt das Graphik-Fenster
`grid`	erzeugt ein achsenparalleles Gitter
`hold`	ermöglicht das Überlagern von Graphiken
`subplot`	erstellt mehrere Teilgraphiken in einem Fenster
`text`	gibt Text an vorgegebener Stelle aus
`title`	zeigt einen Titel an
`xlabel`	beschriftet die x-Achse
`ylabel`	beschriftet die y-Achse
`colormap(white)`	wechselt die Farbtabelle, für S/W-Monitore

B.6.2 Dreidimensionale Graphiken

Funktionen $f : \mathbb{R}^3 \to \mathbb{R}$ oder Punktwolken im $\mathbb{R}^3$ können ebenfalls dargestellt werden. Dazu gibt es in MATLAB® eine Reihe von Möglichkeiten und Optionen. Als Beispiel wollen wir hier eine Gauß-Kurve der Form

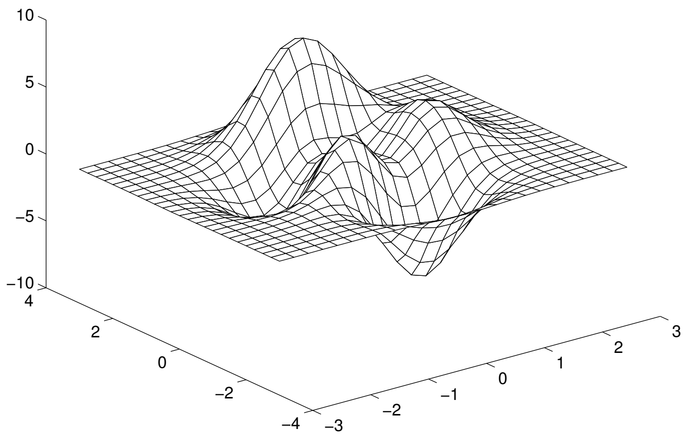
$$f(x,y) := 3(1-x^2)e^{-x^2-(y+1)^2} - 10\left(\frac{x}{5} - x^3 - y^5\right)e^{-x^2-y^2} - \frac{1}{3}e^{-(x+1)^2-y^2}$$

visualisieren. Diese Funktion ist durch den Befehl `peaks` in MATLAB® realisiert.

```
>> [X,Y,Z] = peaks(25)
>> mesh(X,Y,Z)
```

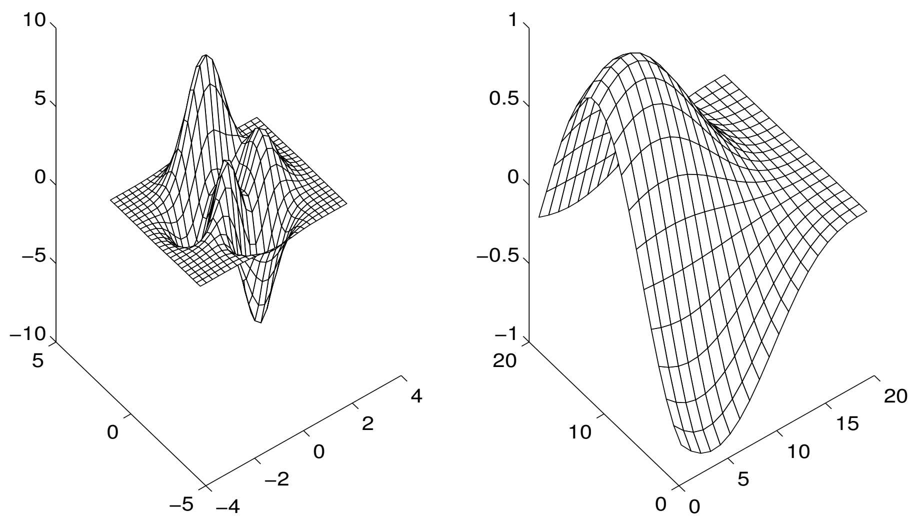
Als Ergebnis erhalten wir:

Wir können auch verschiedene Graphiken in einem Bild erzeugen:




```
>> subplot(1,2,1);
>> [X,Y,Z] = peaks(25);
>> mesh(X,Y,Z);
>> subplot(1,2,2);
>> X=1:20;
>> Y=1:20;
>> Z(X,Y)=(-(cos(X/4)))'*(sin((20-Y)/10).^3);
>> mesh(X,Y,Z);
```

mit dem entsprechenden Ergebnis in der folgenden Abbildung:



B.6.3 Graphiken drucken

Natürlich kann man Graphiken drucken oder in einer Graphikdatei speichern.

Graphik-Druckbefehl		
`print [-dAusgabetyp] [-Optionen] [Dateiname]`		
Ausgabetyp	`-dps`	Postscript für Schwarzweißdrucker
	`-dpsc`	Postscript für Farbdrucker
	`-deps`	Encapsulated Postcript
	`-depsc`	Encapsulated Color Postcript
Optionen	`-P<Drucker>`	Spezifiziert den zu benutzenden Drucker

Die Eingabe

```
>> print fig4 -deps
```

erzeugt die Datei `fig4.eps`.

B.7 Fortgeschrittenes

B.7.1 MATLAB®-Skripte

Wie schon in Abschnitt B.1 zu Skript-Dateien erwähnt, lässt sich eine Abfolge von MATLAB®-Befehlen auch in einer Datei speichern. Diese kann man dann im Befehl-Fenster durch einen einzigen Befehl aufrufen und die gespeicherte Folge von Befehlen wird ausgeführt. Solche Dateien werden als MATLAB®-Skripte oder M-Files bezeichnet. Alle MATLAB®-Befehle lassen sich in einer Datei z.B. mit dem Namen `abc.m` zusammenstellen. Für die Wahl des Dateinamens gelten dabei die folgenden Regeln:

- das erste Zeichen ist ein Buchstabe und
- die Datei hat die Endung „`.m`".

Um ein solches M-File zu erstellen, ruft man einen Editor[2] auf, der es erlaubt, Text einzugeben und diesen als Datei zu speichern. Wenn man den Editor gestartet hat, kann man beispielsweise die folgenden Befehle eingeben und die Datei unter dem Namen `abc.m` abspeichern:

```
a = 1;
b = 3;
c = -5;
x1 = (-b + sqrt(b^ 2 - 4*a*c))/(2*a)
x2 = (-b - sqrt(b^ 2 - 4*a*c))/(2*a)
```

Um nun die Befehle aus der Datei `abc.m` auszuführen, gibt man im MATLAB®-Befehlsfenster

```
>> abc
```

ein. MATLAB® sucht im aktuellen Pfad[3] nach der Datei `abc.m` und führt die darin enthaltenen Befehle aus. Das aktuelle Verzeichnis wird angezeigt, wenn man

```
>> pwd
```

eingibt (`pwd`: engl. `print working directory`).

[2] Dies kann, muss aber nicht, der MATLAB®-interne Editor sein.
[3] Deswegen muss man den entsprechenden Pfad vorher setzen!

B.7.2 Erstellen eigener Funktionen

Es ist vielleicht angenehmer und komfortabler als jedesmal erneut ein eigenes M-File anzufertigen, eine Funktion zu haben, der man *a,b* und *c* als Argument übergibt und dann nach Programmaufruf die beiden Wurzeln als Ergebnis auf dem Bildschirm erhält. Ein solches Programm könnte z.B. die folgende Datei `root.m` sein.

```
%-----------------
modified abc.m
%-----------------

a = input('Enter a: ');
b = input('Enter b: ');
c = input('Enter c: ');

x1 = (-b + sqrt(b^ 2 - 4*a*c))/(2*a)
x2 = (-b - sqrt(b^ 2 - 4*a*c))/(2*a)
```

Die Prozentzeichen in den ersten Zeile dienen der Dokumentation. Alles was einem solchen Zeichen folgt, wird von MATLAB® nicht ausgewertet, also nicht interpretiert. Die Argumente werden in dieser Funktion jedoch nur bedingt übergeben und ausgegeben, wir müssen die Eingaben immer noch eintippen. Eine Funktion, die dieses Manko behebt, ist die folgende.

```
%-----------------
modified abc.m
%-----------------

function [x1, x2] = quadroot(a,b,c)

radical = sqrt(b^2 - 4*a*c);

x1 = (-b + radical)/(2*a)
x2 = (-b - radical)/(2*a)
```

Wenn eine Funktion keine Rückgabewerte hat, können die eckigen Klammern mit den Variablen und das Gleichheitszeichen fehlen. Fehlen die Eingabeparameter, so kann auch der Klammerausdruck nach `quadroot` fehlen. Auf die in der Funktion verwendeten Variablen, hier `radical`, kann man vom Befehlsfenster nicht zugreifen. Ist die Funktion in der Datei `quadroot.m` gespeichert, so erhält man die Wurzeln, indem man

```
[x1, x2] = quadroot(1,3,-5);
```

eingibt.

Es kann vom Befehlsfenster oder einer anderen Datei immer nur die **erste Funktion in einer Datei aufgerufen** werden. Dies bedeutet, in einer Datei können mehrere Funktionen stehen, aber **nur die erste Funktion kann extern aufgerufen werden**. Alle weiteren Funktionen können nur von Funktionen in der gleichen Datei aufgerufen werden. Für den Anfang ist es einfacher, wenn jede Datei nur eine Funktion enthält. Entscheidend für den Funktionsnamen ist der Name der Datei.

Literatur

[AU18] W. Arendt und K. Urban. *Partielle Differenzialgleichungen: Eine Einführung in analytische und numerische Methoden*. 2. Aufl. Springer, Heidelberg, 2018 (siehe S. 16, 91).

[AU23] W. Arendt und K. Urban. *Partial differential equations: An Introduction to Analytical and Numerical Methods*. Springer Graduate Texts in Mathematics, 2023 (siehe S. 91).

[Bor16] F. Bornemann. *Numerische lineare Algebra. Eine konzise Einführung mit MATLAB und Julia*. Springer, 2016 (siehe S. iii).

[DR08] W. Dahmen und A. Reusken. *Numerik für Ingenieure und Naturwissenschaftler*. 2. Aufl. Berlin: Springer, 2008 (siehe S. 3).

[DH19] P. Deuflhard und A. Hohmann. *Numerische Mathematik 1*. 5. Aufl. De Gruyter Studium. Eine algorithmisch orientierte Einführung. De Gruyter, Berlin, 2019 (siehe S. iii, 80).

[Fis13] G. Fischer. *Lineare Algebra*. 18. Aufl. Grundkurs Mathematik [Foundational Course in Mathematics]. Vieweg-Verlag, Braunschweig, 2013 (siehe S. A-9, A-10).

[FH07] R. Freund und R. Hoppe. *Stoer/Bulirsch: Numerische Mathematik 1*. 10. Auflage. Springer, 2007 (siehe S. iii).

[GK96] G. Golub und W. Kahan. „Calculating the singular values and pseudo-inverse of a matrix“. In: *SIAM J. Numer. Anal.* Ser. B 2 (196), S. 205–224 (siehe S. 65).

[GR70] G. Golub und C. Reinsch. „Singular Value Decomposition and Least Squares Solutions“. In: *Numer. Math.* 14 (1970), S. 403–420 (siehe S. 65).

[GV13] G. H. Golub und C. F. Van Loan. *Matrix computations*. 4. Aufl. Johns Hopkins University Press, Baltimore, MD, 2013 (siehe S. iii, 65, 83).

[Hac16] W. Hackbusch. *Iterative solution of large sparse systems of equations*. 2. Aufl. Bd. 95. Applied Mathematical Sciences. Springer, Cham], 2016 (siehe S. iii).

[HH94] G. Hämmerlin und K.-H. Hoffmann. *Numerische Mathematik*. 4. Aufl. Springer-Lehrbuch. Grundwissen Mathematik. Springer-Verlag, Berlin, 1994 (siehe S. iii).

[Han09] M. Hanke-Bourgeois. *Grundlagen der numerischen Mathematik und des wissenschaftlichen Rechnens*. 3. Aufl. Vieweg + Teubner, Wiesbaden, 2009 (siehe S. iii).

[Har22] H. Harbrecht. *Algorithmische Mathematik. Graphen, Numerik und Probabilistik*. Springer Spektrum, 2022 (siehe S. iv).

[HJ01] R. A. Horn und C. R. Johnson. *Matrix Analysis*. 2. Aufl. Cambridge University Press, 2001 (siehe S. 70, 71).

[Kan05] C. Kanzow. *Numerik linearer Gleichungssysteme. Direkte und iterative Verfahren*. 1. Aufl. Springer, 2005 (siehe S. iii, 130, 131).

[Knu98] D. E. Knuth. *The art of computer programming. Vol. 2*. 3. Aufl. Addison-Wesley, Reading, MA, 1998, S. xiv+762 (siehe S. 154).

[Mei15] A. Meister. *Numerik linearer Gleichungssysteme: Eine Einführung in moderne Verfahren*. Mit MATLAB®-Implementierungen von C. Vömel. Wiesbaden: Springer Spektrum, 2015 (siehe S. iii, 129).

[Pla04] R. Plato. *Numerische Mathematik kompakt*. 2. Aufl. Grundlagenwissen für Studium und Praxis. Vieweg-Verlag, Wiesbaden, 2004 (siehe S. iii, 128).

[QSS07] A. Quarteroni, R. Sacco und F. Saleri. *Numerical mathematics*. 2. Aufl. Bd. 37. Texts in Applied Mathematics. Springer-Verlag, Berlin, 2007 (siehe S. iii, 79).

[QSG14] A. Quarteroni, F. Saleri und P. Gervasio. *Scientific computing with MATLAB and Octave*. 4. Aufl. Bd. 2. Texts in Computational Science and Engineering. Springer, Heidelberg, 2014 (siehe S. iii, B-1).

[RW17] T. Richter und T. Wick. *Einführung in die Numerische Mathematik - Begriffe, Konzepte und zahlreiche Anwendungsbeispiele*. ISBN 978-9-662-54177-7, Springer, 2017 (siehe S. iii).

[Saa03] Y. Saad. *Iterative Methods for Sparse Linear Systems*. 2. Aufl. SIAM, 2003 (siehe S. iii, 130).

[Sch19] W. Scherer. *Mathematics of Quantum Computing. An Introduction*. Cham: Springer, 2019 (siehe S. 139).

[Sch72] H. R. Schwarz. *Numerik symmetrischer Matrizen.* Unter Mitwirkung von H. Rutishauser und E. Stiefel., Zweite, durchgesehene und erweiterte Auflage. B. G. Teubner, Stuttgart,, 1972 (siehe S. iii).

[Sch97] H. R. Schwarz. *Numerische Mathematik*. 4. Aufl. B. G. Teubner, Stuttgart, 1997 (siehe S. iii).

[Sto94] J. Stoer. *Numerische Mathematik. 1*. 7. Aufl. Springer-Lehrbuch. Springer-Verlag, Berlin, 1994 (siehe S. iii).

[SB90] J. Stoer und R. Bulirsch. *Numerische Mathematik. 2*. 3. Aufl. Springer-Lehrbuch. Springer-Verlag, Berlin, 1990 (siehe S. iii).

[Wil88] J. H. Wilkinson. *The algebraic eigenvalue problem*. Monographs on Numerical Analysis. Oxford Science Publications. The Clarendon Press, Oxford University Press, New York, 1988 (siehe S. 81).