



Angewandte Numerik

Stefan Funken

Karsten Urban

Universität Ulm

Wintersemester 2024/25

Copyright © 2024 Stefan A. Funken, Karsten Urban
Dieser Text darf ausschließlich für Zwecke der Lehrveranstaltung verwendet werden.

Version 17. Oktober 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:

- Deuffhard 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 Vorlesungsmanskript ohne Druckfehler, Flüchtigkeitsfehler, Unzulänglichkeiten oder sonstige Inkorrektheiten. Wir freuen uns über entsprechende Hinweise, die dazu beitragen können, das vorliegende Manuskript zu verbessern.

Ulm, im Oktober 2024

Stefan A. Funken, Karsten Urban

(`stefan.funken@uni-ulm.de`, `karsten.urban@uni-ulm.de`)

Inhaltsverzeichnis

Vorwort	iii
1 Einleitung	1
2 Kondition und Stabilität	7
2.1 Kondition eines Problems	7
2.2 Stabilität eines Algorithmus	12
3 Direkte Löser Linearer Gleichungssysteme	15
3.1 Kondition eines linearen Gleichungssystems	17
3.2 Determinanten-basierte Methoden	21
3.3 Gestaffelte Systeme	24
3.4 Gaußsche Eliminationsmethode	28
3.5 Pivotisierung	32
3.6 Cholesky-Verfahren	37
3.7 Bandmatrizen	41
4 Lineare Ausgleichsprobleme	45
4.1 Die Methode der kleinsten Quadrate	45
4.2 Die QR -Zerlegung	48
4.3 Givens-Rotationen	51
4.4 Householder-Spiegelungen	53
4.5 Update einer QR -Zerlegung	59
5 Nichtlineare Gleichungen	63
5.1 Skalare Probleme	67
5.2 Systeme nichtlinearer Gleichungen	76
6 Interpolation und Approximation	91
6.1 Klassische Polynom-Interpolation	91
6.2 Das Neville-Aitken-Verfahren	93
6.3 Hermite-Interpolation, Newton-Darstellung und dividierte Differenzen	96
6.4 Tschebyscheff-Interpolation und Interpolationsfehler	103
7 Orthogonale Polynome und Drei-Term-Rekursionen^(*)	109
7.1 Einige Grundlagen	109
7.2 Drei-Term-Rekursionen	111

7.3	Nullstellen von Orthogonalpolynomen	118
8	Numerische Integration	121
8.1	Mittelpunkts- und Trapezregel	121
8.2	Newton-Cotes-Formeln	125
8.3	Konvergenz von Quadraturformeln	128
8.4	Gauß-Quadratur	130
8.5	Extrapolation und Romberg-Integration	137
8.6	Numerische Differentiation	145
A	Einige Grundlagen aus der Linearen Algebra	A-1
A.1	Normen	A-1
A.2	Matrixnormen	A-2
A.3	Eigenwerte und -vektoren	A-7
B	Einige Grundlagen aus der Analysis	B-1
B.1	Der Satz von Rolle	B-1
B.2	Der Satz von Taylor	B-1
C	Eine kurze Einführung in MATLAB[®]	C-1
C.1	Grundlegende MATLAB [®] -Befehle	C-1
C.2	Mathematik mit Matrizen	C-6
C.3	Datenverwaltung	C-11
C.4	Ausgabe von Text	C-13
C.5	Kontrollbefehle	C-13
C.6	Graphische Darstellung	C-16
C.7	Fortgeschrittenes	C-20
Literatur		

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 $u : [0, 1] \rightarrow \mathbb{R}$ bei gegebenem $f : [0, 1] \rightarrow \mathbb{R}$ 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.

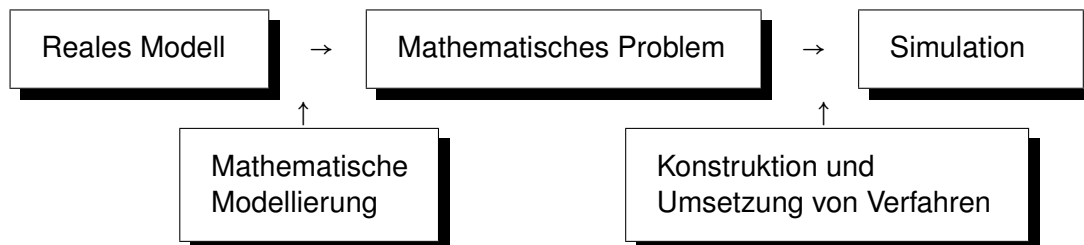


Abbildung 1.1: Modellierung und Simulation.

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 *Modellierungszirkel* 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 C 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

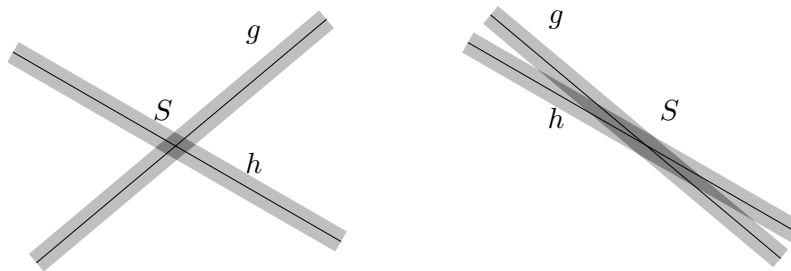


Abbildung 2.1: Schnittpunkt S zweier Geraden g und h . Links gut konditioniert, rechts schlecht konditioniert.

■ **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 $\angle(g, h)$ 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 $\angle(g, h)$ 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 $\varphi : X \rightarrow Y$. Wir betrachten das Problem:

Gegeben sei $x \in X$, 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 $h : \mathbb{R}^2 \rightarrow \mathbb{R}$, 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

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

¹ $\mathcal{R}(\mathbb{R})$ bezeichne die Menge der Riemann-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 $\tilde{x} \in \mathbb{R}^n$ zwei Eingaben. Der relative Fehler von $\varphi(\tilde{x})$ bezüglich $\varphi(x)$ lässt sich folgendermaßen abschätzen (falls $\varphi(x) \neq 0$ und $x_j \neq 0$ für alle j):

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

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

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

der **unvermeidbaren Fehler** (selbst bei exakter Rechnung), der bei der Berechnung von φ auftritt. Sei nun $\varphi : \mathbb{R}^n \rightarrow \mathbb{R}$ differenzierbar in x . Dann gilt mit der Taylor-Entwicklung²

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

und (2.1) wird zu

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

Somit gibt der Faktor $\frac{|x_j|}{|\varphi(x)|} \left| \frac{\partial \varphi}{\partial x_j}(x) \right|$ (der unabhängig von \tilde{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 $\varphi : \mathbb{R}^n \rightarrow \mathbb{R}^m$, so können wir die bisherigen Überlegungen auf jede Komponente von φ einzeln anwenden. ■

Dies motiviert den folgenden Begriff.

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

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

heißen *Konditionszahlen* von φ in x .

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

$$\kappa_1(x) = \frac{|x_1|}{|x_1 x_2|} \cdot \left| \frac{\partial \varphi}{\partial x_1}(x) \right| = 1 \quad \text{und} \quad \kappa_2(x) = 1.$$

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

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

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

²Die Landau-Symbole o und O werden hier als bekannt vorausgesetzt.

Im Falle der Auslöschung, 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“:

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

Eine einfache Rechnung ergibt

$$\kappa_p = \frac{p}{\sqrt{p^2 + q}} \quad \text{und} \quad \kappa_q = \frac{p + \sqrt{p^2 + q}}{2\sqrt{p^2 + q}}, \quad (2.4)$$

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

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

für $j = 1, \dots, n$. 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 $\varphi: \mathbb{R}^{n \times n} \rightarrow \mathbb{R}^n$ 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

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

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 einzelne 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 $\tilde{x} \in X$ eine Approximation von $x \in X$. Als **Fehler** bezeichnet

man $\Delta x := x - \tilde{x}$ und zu einer gegebenen Norm $\|\cdot\|_X$ ist

$$\|x - \tilde{x}\|_X \quad (2.7)$$

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

$$\frac{\|x - \tilde{x}\|_X}{\|x\|_X} = \frac{\|\Delta x\|_X}{\|x\|_X} \quad (2.8)$$

den **relativen Fehler** von \tilde{x} .

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

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

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

$$\kappa_\varphi := \frac{\delta_y}{\delta_x} \quad (2.9)$$

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

$$\kappa_{\varphi, \text{abs}} := \frac{\|\Delta y\|_Y}{\|\Delta x\|_X} \quad (2.10)$$

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

(c) Ein Problem heißt **gut konditioniert**, wenn „kleine“ Schranken für κ_φ für $\delta_x \rightarrow 0$ 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

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

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

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

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

Bemerkung 2.1.11 Relative Fehler in der ∞ -Norm können durch eine Aussage über die

Anzahl korrekter Stellen von \tilde{x} ausgedrückt werden: die Abschätzung

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

bedeutet, dass die betragsgrößte Komponente von \tilde{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 $\tilde{\varphi} : X \rightarrow Y$, 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 $\tilde{\varphi} : \mathbb{R}^n \rightarrow \mathbb{R}^m$ sollte also eine Abschätzung der Form

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

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

Definition 2.2.1 — Numerische Stabilität eines Algorithmus. Ein Algorithmus definiert durch die Eingabe-Ausgabe-Funktion $\tilde{\varphi} : \mathbb{R}^n \rightarrow \mathbb{R}^m$ zur Lösung eines Problems $\varphi : \mathbb{R}^n \rightarrow \mathbb{R}^m$ 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 $\tilde{\varphi}(x)$ tatsächlich eine gute Approximation der exakten Lösung $\varphi(x)$ für gegebenes $x \in X$ ist. Dies sieht man an einem einfachen Beispiel: $\tilde{\varphi}(x) \equiv 0$ 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.

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

Dazu untersuchen zwei numerische Verfahren:

Algorithmus 2.2.3

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

Algorithmus 2.2.4 — Satz von Viëta.

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

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 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. ■

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 Suánsù) 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. Euler¹ 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. Gauß² 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.

¹Leonard Euler, 1707-1783.

²Carl 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 $u : [0, 1] \rightarrow \mathbb{R}$ ist Lösung des folgenden linearen Randwertproblems zweiter Ordnung

$$-u''(x) + \lambda(x)u(x) = f(x), \quad x \in (0, 1), \quad u(0) = u(1) = 0 \quad (3.1)$$

mit gegebenen $f : (0, 1) \rightarrow \mathbb{R}$ und $\lambda : (0, 1) \rightarrow \mathbb{R}$ mit $\lambda(x) \geq 0$ für $x \in (0, 1)$. Genauer 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 $x_i := ih$, $i = 0, \dots, n_h + 1$ ($x_0 = 0, x_{n_h+1} = 1$), 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

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

Es gilt bekanntlich $\|u'' - D_{c;h}^2 u\| = \mathcal{O}(h^2)$, 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 $\lambda_i = \lambda(x_i)$, $f_i = f(x_i)$):

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

also ein lineares Gleichungssystem der Form

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

d.h. $A_h u_h = f_h$. Für $n_h \rightarrow \infty$ ($h \rightarrow 0$) 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 \tilde{x} eine Näherungslösung. Dann heißt $e := x - \tilde{x}$ **Fehler** und $r := b - A\tilde{x}$ das **Residuum**.

Den Fehler können wir im Allgemeinen nicht berechnen, da wir x nicht kennen. Das Residuum hingegen ist für eine gegebene Approximation \tilde{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:

$$Ae = Ax - A\tilde{x} = b - A\tilde{x} = r. \quad (3.2)$$

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

$$\|e\| = \|A^{-1}r\| \leq \|A^{-1}\| \|r\| \quad \text{und} \quad \|b\| = \|Ax\| \leq \|A\| \|x\|. \quad (3.3)$$

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

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

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

$$\frac{\|x - \tilde{x}\|}{\|x\|} = \frac{\|e\|}{\|x\|} \leq \frac{\|A^{-1}\| \|r\|}{\|b\| \cdot \frac{1}{\|A\|}} = \|A^{-1}\| \|A\| \cdot \frac{\|r\|}{\|b\|} = \kappa(A) \frac{\|r\|}{\|b\|}. \quad (3.5)$$

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

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

³Dabei verwenden wir $\|b\| = \|AA^{-1}b\| \leq \|A\| \|A^{-1}b\|$, also $\|A^{-1}b\| \geq \|b\| \frac{1}{\|A\|}$.

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

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

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

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

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

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

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

Betrachten wir nun die folgende (speziell gewählte) Näherung

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

Diese Wahl erklärt sich wie folgt: Wir bezeichnen mit $0 \leq \lambda_1 < \dots < \lambda_5$ die Eigenwerte von A mit den zugehörigen normierten Eigenvektoren $\varphi_1, \dots, \varphi_5$, $\|\varphi_j\|_2 = 1$. Dann gilt $\tilde{x} - x \approx \varphi_1$ 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 Größenordnung, 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 $\delta x = (A + \delta A)^{-1}(\delta b - \delta Ax) = (I + A^{-1}\delta A)^{-1}A^{-1}(\delta b - \delta Ax)$. Aufgrund der Submultiplikativität gilt $\|\delta x\| \leq \|(I + A^{-1}\delta A)^{-1}\| \|A^{-1}\| (\|\delta b\| + \|\delta A\| \|x\|)$ und wir erhalten die Abschätzung

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

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

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

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

$$\sum_{k=0}^m \|B^k\| \leq \sum_{k=0}^m \|B\|^k \leq \sum_{k=0}^{\infty} \|B\|^k = \frac{1}{1 - \|B\|} < \infty. \quad (3.9)$$

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 $A^k \rightarrow 0$ wegen $\|B\| < 1$ aus $\|B^k\| \leq \|B\|^k \rightarrow 0$ für $k \rightarrow \infty$. Die folgende Gleichung ist eine „Teleskopsumme“

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

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

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

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

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

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

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

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

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

Damit erhalten wir schließlich aus (3.7)

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

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 $\tilde{x} := (A + \delta A)^{-1}(b + \delta b)$ die Fehlerabschätzung

$$\frac{\|x - \tilde{x}\|}{\|x\|} \leq \frac{\kappa(A)}{1 - \kappa(A) \frac{\|\delta A\|}{\|A\|}} \left(\frac{\|\delta b\|}{\|b\|} + \frac{\|\delta A\|}{\|A\|} \right). \quad (3.13)$$

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

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

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

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

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 Determinante 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.

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

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 $\simeq 1 \text{ FLOP}$ 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 $1, 2, \dots, n$ zu berechnen ist. Der Wert des Ausdrucks $(-1)^{j(\pi)}$ ergibt sich aus der Anzahl $j(\pi)$ der Inversionen der Permutation $\pi = \begin{pmatrix} 1 & 2 & \dots & n \\ i_1 & i_2 & \dots & i_n \end{pmatrix}$.

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

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

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

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

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

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

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 $\det(A) =$

⁴FLOP=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.

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

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

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

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 $1.1 \text{ ExaFLOPs} = 1.1 \cdot 10^{18} \text{ FLOPs}$ 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 $180 \text{ TFLOPs} = 180 \cdot 10^{12} \text{ FLOPs}$.

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}, \dots, a_{nn} auch einfach

$$A = \text{diag}(a_{11}, \dots, a_{nn}).$$

Falls $a_{ii} \neq 0$, kann man die Inverse sofort hinschreiben $A^{-1} = \text{diag}(1/a_{11}, \dots, 1/a_{nn})$ 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 $a_{ii} \neq 0$, $b \in \mathbb{R}^n$

for $i = 1, \dots, n$

$x_i = b_i/a_{ii}$

end

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

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 $A = (a_{ij})_{i,j=1,\dots,n} \in \mathbb{R}^{n \times n}$. 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

⁵Diagonalmatrizen 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 Form⁷

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

Für die obere Dreiecksmatrix R gilt, dass

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

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:

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

Dieser Algorithmus wird 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

⁶L: links/lower; R: rechts, U: upper.

⁷Es wird später klar, warum die rechte Seite hier z und nicht b heißt.

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

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

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 $\text{FLOP}(A^{-1}b)$ **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 $\text{FLOP}(A^{-1}) = n \cdot \text{FLOP}(A^{-1}b)$, 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

```

1 function x = Rinvb(R,z)
2 % compute solution of R * x = z with upper triangular matrix R
3 n = size(R,1);
4 x = zeros(n,1);
5 for j = n:-1:1
6     for k=j+1:n
7         z(j) = z(j) - R(j,k) * x(k);
8     end
9     x(j) = z(j) / R(j,j);
10 end

```

MATLAB-Beispiel:

Man löse $Rx = b$ mit

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

```
>> 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

```
1 function x = Rinvb2(R,z)
2 % compute solution of R * x = z with upper triangular matrix R
3 n = size(R,1);
4 x = zeros(n,1);
5 for j = n:-1:1
6     x(j) = z(j) / R(j,j);
7     for k=1:j-1
8         z(k) = z(k) - R(k,j) * x(j);
9     end
10 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

```
1 function x = solve_upper_triangular(A, b)
2     % A ist die obere Dreiecksmatrix
3     % b ist der Vektor auf der rechten Seite des
4         Gleichungssystems
5
6     n = length(b);
7     x = zeros(n, 1);
8
9     % Rueckwaertssubstitution
10    for i = n:-1:1
11        x(i) = b(i) / A(i, i);
12        for j = i-1:-1:1
13            b(j) = b(j) - A(j, i) * x(i);
14        end
15    end
```

Vergleichen Sie nun die drei Algorithmen.

3.4 Gaußsche Eliminationsmethode

Im Folgenden werden wir uns mit der Gaußschen⁸ 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 Lagrange⁹ 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

$$A = L \cdot R, \quad (3.17)$$

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 $Lz = b$,
- 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

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

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,

⁸Carl Friedrich Gauß, 1777 - 1855.

⁹Joseph-Louis Lagrange, 1736 - 1813.

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

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

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

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

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

sind die Elemente in (3.19) gegeben durch

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

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

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

von (Rest-)Matrizen der speziellen Gestalt

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

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

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

Da jeder Eliminationsschritt eine lineare Operation auf den Zeilen von A ist, lässt sich

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

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.

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

Die entsprechende Matrix entsteht aus den Multiplikatoren

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

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.

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

Die Matrix R ergibt sich dann

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

Damit haben wir alle drei Schritte des Gaußschen Eliminationsverfahrens 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

$$\text{FLOP}(\text{LR-Zerlegung}) = \frac{1}{6}(4n^3 - 3n^2 - n),$$

$$\text{FLOP}(A^{-1}b) = \frac{1}{6}(4n^3 + 9n^2 - n),$$

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

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

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

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

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. Wir müssen die Zwischenmatrizen $A^{(k)}$ aus (3.21) speichern, beginnend mit $A^{(0)} := A$, wofür wir einen Speicheraufwand von (im Allgemeinen) $\mathcal{O}(n^3)$ benötigen.

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} hzu 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**:

$$A \rightarrow \left(\begin{array}{cccc} a_{11}^{(1)} & \cdots & \cdots & a_{1n}^{(1)} \\ \hline \ell_{21} & a_{22}^{(2)} & \cdots & a_{2n}^{(2)} \\ \vdots & \vdots & & \\ \hline \ell_{n1} & a_{n2}^{(2)} & & a_{nn}^{(2)} \end{array} \right) \rightarrow \dots \rightarrow \left(\begin{array}{cccc} a_{11}^{(1)} & \cdots & & a_{1n}^{(1)} \\ \hline \ell_{21} & a_{22}^{(2)} & & a_{2n}^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ \hline \ell_{n1} & \dots & \dots & \overline{\ell_{n,n-1}} \mid a_{nn}^{(n)} \end{array} \right)$$

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

Wir können also durch dieses geschickte Überschreiben der Einträge von A alle Informationen dort ablegen und haben insgesamt einen Speicheraufwand von $\mathcal{O}(n^3)$.

3.5 Pivottisierung

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 $\varepsilon > 0$ betrachten wir die Lösung des folgenden linearen Gleichungssystems

$$\varepsilon x_1 + x_2 = 1 \tag{3.27a}$$

$$x_1 + x_2 = 2. \tag{3.27b}$$

Bei exakter Arithmetik erhalten wir

$$\frac{1}{\varepsilon-1} \begin{pmatrix} 1 & -1 \\ -1 & \varepsilon \end{pmatrix} \begin{pmatrix} 1 \\ 2 \end{pmatrix} = \frac{1}{1-\varepsilon} \begin{pmatrix} 1 \\ 1-2\varepsilon \end{pmatrix}$$

also lautet die exakte Lösung $x_1^* = \frac{1}{1-\varepsilon}$ und $x_2^* = \frac{1-2\varepsilon}{1-\varepsilon}$. Nun betrachten wir den Einfluss der Rechnerarithmetik. Für $2\varepsilon < \text{eps}$ (also werden $1 \pm 2\varepsilon$ und 1 durch dieselbe Zahl dargestellt) erhalten wir ein gerundetes Ergebnis von

$$x_1 = x_2 = 1,$$

dies ist also das beste Ergebnis in Rechnerarithmetik.

Nun führen wir die Gauß-Elimination durch: wenn wir den Koeffizienten vor x_1 in (3.27b) eliminieren, d.h. das $1/\varepsilon$ -fache von (3.27a) subtrahieren, erhalten wir

$$\begin{aligned}\varepsilon x_1 + x_2 &= 1, \\ \left(1 - \frac{1}{\varepsilon}\right) x_2 &= 2 - \frac{1}{\varepsilon}.\end{aligned}$$

Somit ergibt sich für $2\varepsilon < \text{eps}$ aus der letzten Gleichung $x_2 = 1$ und damit aus der ersten Gleichung $x_1 = 0$!

Vertauschen align wir jedoch die beiden Zeilen (bzw. eliminieren den Koeffizienten vor x_1 in (3.27a)) so erhalten wir

$$\begin{aligned}x_1 + x_2 &= 2 \\ (1 - \varepsilon) x_2 &= 1 - 2\varepsilon,\end{aligned}$$

und dies liefert bei $2\varepsilon < \text{eps}$ das (optimale) Ergebnis $x_1 = x_2 = 1$.

In diesem Beispiel liefert eine **Permutation höhere Genauigkeit!** ■

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
 - höhere Genauigkeit.

MATLAB-Beispiel:

Dieses scheinbar theoretische Ergebnis aus Beispiel 3.5.1 können wir direkt in MATLAB® umsetzen. Da die Recheneinheiten heutiger Rechner meist zwei Stellen genauer rechnen als die bisherigen theoretischen Untersuchungen vermuten lassen, muss hier $\epsilon \leq \text{eps}/8$ gelten.

```
>> e = eps/8; % eps/2 genuegt nicht!
>> x(2) = (2 - 1/e) / (1 - 1/e);
>> x(1) = (1 - x(2)) / e
x =
    0    1
>> x(2) = (1 - 2*e) / (1 - e);
>> x(1) = 2 - x(2)
x =
    1    1
```

Das Vertauschen von Zeilen (oder Spalten) kann folglich nötig sein. Zum einen, damit eine LR-Zerlegung überhaupt existiert, zum anderen, um den „Rechenfehler“ bedingt durch die Rechnerarithmetik möglichst klein zu halten. Daraus leiten sich besondere Strategien. Betrachten wir zunächst die Gauß-Elimination mit **Spaltenpivottisierung**.

Algorithmus 3.5.2 — Gauß-Elimination mit Spaltenpivottisierung. a) Wähle im Eliminationsschritt $A^{(k)} \rightarrow A^{(k+1)}$ einen Index $p \in \{k, \dots, n\}$, so dass

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

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

b) Vertausche die Zeilen p und k

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

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

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

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

```

1 function [L,R,P] = mylu(A)
2 n = size(A,1); % get leading dimension
   of A
3 p = 1 : n; % pivot element vector
4 for k = 1 : n-1 % consider k-th column
5     [m,mptr] = max(abs(A(p(k:end),k))); % find pivot element
6     tmp = p(k); % interchange in vector p
7     p(k) = p(k-1+mptr);
8     p(k-1+mptr) = tmp;
9
10    for j = k+1 : n % modify entries in
11        A(p(j),k) = A(p(j),k)/A(p(k),k); % compute l_jk, store in
           A
12    for i = k+1 : n % (n-k-1)*(n-k-1)
           submatrix
13        A(p(j),i) = A(p(j),i) ...
14            - A(p(j),k)*A(p(k),i);
15    end
16 end
17 end
18 L = tril(A(p,:),-1)+eye(n); % these lines could be
19 R = triu(A(p,:)); % neglected, all
           information
20 P = eye(n); % already in A and p
21 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.

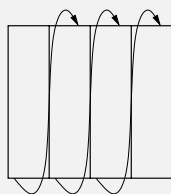
Beweis. Nach Voraussetzung gilt $\det A \neq 0$. Angenommen, es sei $a_{11} = 0$. Dann existiert eine Zeilenvertauschung, so dass das erste Pivotelement $a_{11}^{(1)}$ von Null verschieden und das betragsgrößte Element in der Spalte ist, d.h.

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

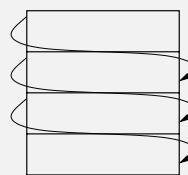
denn andernfalls wäre die erste Spalte eine Nullspalte und $\det(A) = 0$, im Widerspruch zur Voraussetzung. Die Zeilenvertauschung hat einen Vorzeichenwechsel in der Determinante zur Folge, ändert aber $|\det(A)|$ nicht, insbesondere bleibt diese von Null verschieden, die so entstandene Matrix ist wiederum 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 Speicher- methode 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.



Spaltenpivotisierung, bzw.



Zeilenpivotisierung

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) überprüfbar 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

$$|a_{ii}| \geq \sum_{\substack{k=1 \\ k \neq i}}^n |a_{ik}| \quad \forall i = 1, \dots, n, \quad (3.28)$$

gilt; sie heißt **strikt diagonaldominant**, falls in (3.28) statt „ \geq “ für alle $i = 1, \dots, n$ stets „ $>$ “ gilt; A heißt **diagonaldominant**, falls zusätzlich zu (3.28) „ $>$ “ für *einen* Index $i \in \{1, \dots, n\}$ 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

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

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

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

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

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

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

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 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.

3.6.1 Symmetrisch positiv definite Matrizen

Definition 3.6.1 Eine Matrix $A \in \mathbb{R}^{n \times n}$ heißt

- (i) *positiv definit*, wenn $x^T A x > 0$ 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 3.6.2 — 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) $\max_{i,j=1,\dots,n} |a_{ij}| = \max_{i=1,\dots,n} a_{ii}$,
- 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 $x^T A x = 0$, was im Widerspruch zu Definition 3.6.1 ist. Damit ist i) gezeigt.

Für den kanonischen Einheitsvektor e_i gilt $a_{ii} = e_i^T A e_i > 0$ wegen Definition 3.6.1 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 $z = (a_{12}, \dots, a_{1n})^T$ sei. Nach einem Eliminationsschritt ergibt sich

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

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.

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

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 $x := L_1^T(0, y_1, \dots, y_{n-1})^T \neq 0$. Nach Voraussetzung ist A symmetrisch positiv definit, also gilt

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

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

3.6.2 Konstruktion der Cholesky-Zerlegung

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

Satz 3.6.3 — 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 3.6.2 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 3.6.4 — Cholesky-Zerlegung. Sei $A \in \mathbb{R}^{n \times n}$ s.p.d. mit rationaler Cholesky-Zerlegung $A = LDL^T$. Dann heißt

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

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

Bemerkung 3.6.5 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 $\bar{L} = LD^{1/2}$ nicht eindeutig ist.

3.6.3 Cholesky-Algorithmus

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

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

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

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

Diese Identitten werden nun in einer geschickten Reihenfolge ausgewertet, was folgenden Algorithmus ergibt.

Cholesky-Verfahren zur Berechnung von L mit $A = LL^T$

Spaltenweise berechnet man fr $k = 1, 2, \dots, n$

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

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

Bemerkung 3.6.6 i) Aus der Cholesky-Zerlegung $A = LL^T$ ergibt sich fr $1 \leq k \leq n$ die

Abschätzung

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

Folglich sind alle Elemente der Matrix L betragsweise durch $\max_{1 \leq j \leq n} \sqrt{|a_{jj}|}$ 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 3.6.7 — Rechenaufwand Cholesky-Zerlegung. Sei $A \in \mathbb{R}^{n \times n}$ symmetrisch positiv definit, dann gilt

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

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

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

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

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

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. ■

3.7 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 3.7.1 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 3.7.2 Sei A eine s.p.d. Bandmatrix mit Bandbreite m und Cholesky-Zerlegung $A = LL^T$. Dann besitzt L eine 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.

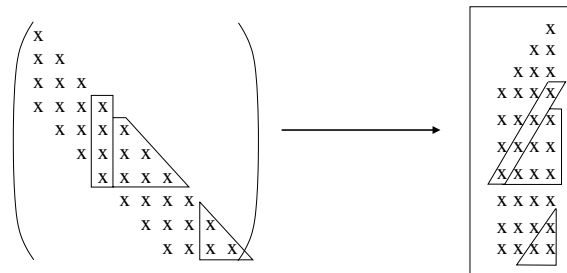


Abbildung 3.1: Reduktion und Speicherung einer symmetrisch positiv definiten Bandmatrix.

Algorithmus

Analog zur Herleitung des Cholesky-Verfahrens auf Seite 39 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 $k = 1, 2, \dots, n$

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

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

Aufwand

Satz 3.7.3 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

$$\text{FLOP}(\text{Cholesky-Zerl. von } A) = n(m^2 + 2m) - \frac{4m^3 + 9m^2 + 5m}{6} + n \text{ Wurzelberechnungen} = \mathcal{O}(nm^2).$$

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 3.6.7 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 3.7.4 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 $g(t) = mt + b$, deren y -Werte den kleinsten Quadratsummenabstand (mittlerer quadratischer Abstand) zu den vorgegebenen Daten $(t_i, f(x_i))$ haben, die sogenannte *Ausgleichsgerade*. Die geometrische Veranschaulichung dazu ist in der nachfolgenden Abbildung dargestellt. ■

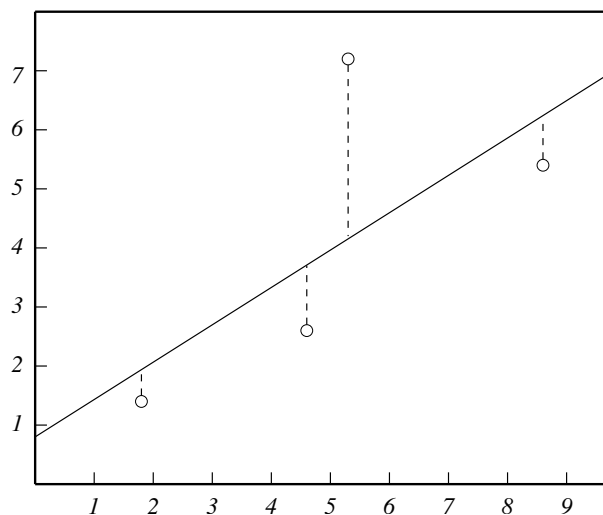


Abbildung 4.1: Die Ausgleichsgerade, die die Quadratsumme der Abstände minimiert

4.1 Die Methode der kleinsten Quadrate

Das Beispiel 4.0.1 können wir wie folgt mathematisch modellieren: zu gegebenen Punktpaaren $(t_1, f_1), (t_2, f_2), \dots, (t_n, f_n)$ sind die in der Geradengleichung $g(t) = mt + b$ freien

Parameter $x_1 := b$ (y -Achsenabschnitt) und $x_2 := m$ (Steigung) so zu bestimmen, dass gilt

$$F(x) := \sum_{i=1}^n (g(t_i) - f_i)^2 = \sum_{i=1}^n (x_2(t_i) + x_1 - f_i)^2 \rightarrow \min_{x \in \mathbb{R}^2},$$

oder in alternativer Formulierung

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

wobei wir nachfolgende Definitionen verwenden

$$A = \begin{pmatrix} 1 & x_1 \\ \vdots & \vdots \\ 1 & x_n \end{pmatrix} \in \mathbb{R}^{n \times 2}, \quad b = \begin{pmatrix} f_1 \\ \vdots \\ f_n \end{pmatrix} \in \mathbb{R}^n.$$

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

$$\|Ax - b\|_2^2 \rightarrow \min_{x \in \mathbb{R}^m}. \quad (4.1)$$

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 Ausgleichsproblem eine Verallgemeinerung linearer Gleichungssysteme für überbestimmte Systeme.

(ii) Offensichtlich ist (4.1) äquivalent zu $\|Ax - b\|_2 \rightarrow \min_{x \in \mathbb{R}^m}$. 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

$$\|Ax - b\| \rightarrow \min_{x \in \mathbb{R}^m}.$$

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

$$\begin{aligned} F(x) &:= \|Ax - b\|_2^2 = (Ax - b, Ax - b)_2 = (Ax - b)^T (Ax - b) \\ &= (Ax)^T Ax - (Ax)^T b - b^T Ax + b^T b = x^T A^T Ax - 2x^T A^T b + b^T b \end{aligned}$$

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 äquivalent zu

$$A^T Ax \stackrel{!}{=} A^T b, \quad (4.2)$$

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

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

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

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

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

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 Normalgleichung (4.2) anwenden und erhalten folgende Abschätzung:

$$\begin{aligned} \kappa_{\|\cdot\|}(A^T A) &= \frac{\|A^T Ax\|}{\|x\|} \|(A^T A)^{-1}\| = \frac{\|A^T Ax\|}{\|x\|} \max_{y \neq 0} \frac{\|(A^T A)^{-1}y\|}{\|y\|} \\ &\leq \max_{x \neq 0} \frac{\|A^T Ax\|}{\|x\|} \max_{y \neq 0} \frac{\|(A^T A)^{-1}y\|}{\|y\|} = \frac{\max_{\|x\|=1} \|A^T Ax\|}{\min_{\|x\|=1} \|A^T Ax\|}. \end{aligned} \quad (4.3)$$

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

$$\kappa(B) \equiv \kappa_{\|\cdot\|}(B) := \frac{\max_{\|x\|=1} \|Bx\|}{\min_{\|x\|=1} \|Bx\|}.$$

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

$$\frac{\max_{\|x\|=1} \|Ax\|}{\min_{\|x\|=1} \|Ax\|} = \|A\| \|A^{-1}\|,$$

d.h., die beiden Konditionsbegriffe stimmen überein.

ii) Falls A vollen Rang besitzt, gilt

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

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 Normalgleichungen 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 $Q \in \mathbb{R}^{n \times n}$ heißt *orthogonal*, falls gilt:

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

Orthogonale Abbildungen stets längen- und winkeltreu sind, dass also gilt

$$\|Qx\|_2^2 = (Qx)^T Qx = x^T Q^T Qx = x^T x = \|x\|_2^2, \quad \text{d.h. } \|Qx\|_2 = \|x\|_2,$$

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 $\text{rang}(A) = n < m$ 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 $\|Q^T x\|_2 = \|x\|_2$):

$$\|Ax - b\|_2^2 = \|Q^T(Ax - b)\|_2^2 = \|Q^T Ax - Q^T b\|_2^2 = \|Rx - Q^T b\|_2^2$$

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

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

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

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

Beweis. Der Kern des (konstruktiven) Beweises liegt in der Verwendung von orthogonalen Drehmatrizen der Form

$$\mathcal{U}(p, q; \varphi) := \begin{pmatrix} 1 & & & & & & 0 \\ & 1 & & & & & \\ & & \ddots & & & & \\ & & & \cos \varphi & \sin \varphi & & \\ & & & -\sin \varphi & \cos \varphi & & \\ & & & & & \ddots & \\ & & & & & & 1 \\ 0 & & & & & & 1 \end{pmatrix} \quad (4.4)$$

\downarrow p-te Spalte \downarrow q-te Spalte
 \leftarrow p-te Zeile \leftarrow q-te Zeile

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 $j = 1, \dots, n$

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

\downarrow p-te Spalte \downarrow q-te Spalte

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

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

$$(1, 2), (1, 3), \dots, (1, m), (2, 3), (2, 4), \dots, (2, m), (3, 4), \dots, (n, m)$$

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

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

Da die orthogonale Matrix Q regulär ist, ist der Rang von A und der Rang von R gleich n und folglich ist die Rechtecksdreiecksmatrix \hat{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 $x \in \mathbb{R}^n$ 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

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

Weiter ergibt sich für $a \neq 0$

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

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 $r = \text{sign}(a)\sqrt{a^2 + b^2}$, 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.

¹James Wallace Givens, 1910-1993.

²Alston Scott Householder, 1904-1993.

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

$$\begin{aligned} \text{Falls } a_{pj} \neq 0: \quad \cos \varphi &= \frac{|a_{pj}|}{\sqrt{a_{pj}^2 + a_{qj}^2}}, & \sin \varphi &= \frac{-\operatorname{sign}(a_{pj}) a_{qj}}{\sqrt{a_{pj}^2 + a_{qj}^2}}, \\ \text{falls } a_{pj} = 0: \quad \cos \varphi &= 0, & \sin \varphi &= 1. \end{aligned}$$

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:

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

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 $x \in \mathbb{R}^n$ mit der Drehmatrix $\mathcal{U}(p, q; \varphi)$ ist äquivalent zur Drehung von x um einen Winkel φ entgegen dem Uhrzeigersinn in der Koordinatenebene, vgl. Abbildung 4.2.

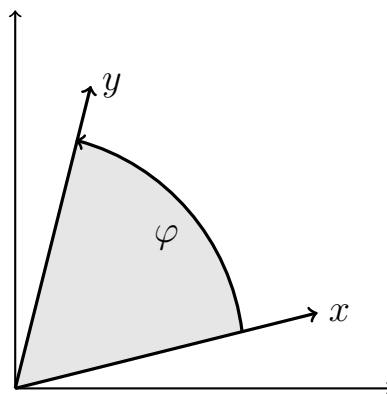


Abbildung 4.2: Drehung um einen Winkel φ in der Ebene.

Wir zeigen einige MATLAB[®]-Codes zur Realisierung.

MATLAB-Funktion: GivensRotMat.m

```

1 function rot = GivensRotMat(ap,aq)
2 if ap == 0
3     s = 1; c = 0;
4 else
5     dist2 = sqrt(ap^2+aq^2);
6     c = abs(ap)/dist2;
7     s = -sign(ap)*aq/dist2;
8 end
9 rot = [c,s;-s,c];

```

MATLAB-Funktion: Givens.m

```

1 function [Q,A] = Givens(A)
2 Q = eye(size(A,1));
3 for j=1:size(A,2)
4     for i=j+1:size(A,1)
5         rot = GivensRotMat(A(j,j),A(i,j));
6         A([j,i],j:end) = rot' * A([j,i],j:end);
7         Q(:, [j,i]) = Q(:, [j,i]) * rot;
8     end
9     A(j+1:end,j) = 0;
10 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 $\omega \in \mathbb{R}^n$. Die Matrix

$$P = I - 2 \frac{\omega \omega^T}{\omega^T \omega} \in \mathbb{R}^{n \times n} \quad (4.6)$$

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 ($(\omega\omega^T)^T = \omega\omega^T$), 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 $x \in \mathbb{R}^n$. Wie müsste ein $\omega \in \mathbb{R}^n$ aussehen, so dass $Px = \alpha e_1$ gelte ($\alpha \in \mathbb{R}$, e_1 erster kanonischer Einheitsvektor)? Mit $Px = x - 2 \frac{\omega\omega^T}{\omega^T\omega} x = x - \lambda\omega \stackrel{!}{=} \alpha e_1$ folgt $\omega \in \text{span}\{x - \alpha e_1\}$. Wir setzen nun $\omega = x - \alpha e_1$ in $Px = \alpha e_1$ ein und erhalten

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

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

$$1 = \frac{2(x - \alpha e_1)^T x}{\|x - \alpha e_1\|^2} \Leftrightarrow (x - \alpha e_1)^T (x - \alpha e_1) = 2x^T x - 2\alpha x_1 \Leftrightarrow \alpha = \pm \sqrt{x^T x}$$

gelten. Wie ist nun das Vorzeichen zu wählen, $\alpha = \pm \sqrt{x^T x}$? Die Wahl $\alpha = \|x\|_2$ hat die schöne Eigenschaft, dass Px ein positives Vielfaches von e_1 ist. Aber das Rezept ist gefährlich, wenn x annähernd ein positives Vielfaches von e_1 ist, da Auslöschungen auftreten können. Berechnet man ω_1 mittels $\omega_1 = x_1 - \|x\|_2$ treten für $x_1 \leq 0$ keine Auslöschungen auf und mit der Umformung

$$\omega_1 = x_1 - \|x\|_2 = \frac{x_1^2 - \|x\|_2^2}{x_1 + \|x\|_2} = \frac{-(x_2^2 + \dots + x_n^2)}{x_1 + \|x\|_2}$$

treten auch für $x_1 > 0$ keine Auslöschungen auf. Man beachte außerdem $(\omega_2, \dots, \omega_n) = (x_2, \dots, x_n)$. Normiert man den Vektor ω so, dass $\omega_1 = 1$ gilt, d.h.

$$\omega := \frac{x - \alpha e_1}{x_1 - \alpha} \quad \text{mit} \quad \alpha^2 = \|x\|^2 \quad (\omega^T e_1 = 1),$$

dann folgt

$$\omega^T \omega = \frac{(x - \alpha e_1)^T (x - \alpha e_1)}{(x_1 - \alpha)^2} = \frac{\|x\|^2 - 2\alpha x^T e_1 + \alpha^2}{(x_1 - \alpha)^2} = \frac{2\alpha(\alpha - x_1)}{(x_1 - \alpha)^2} = \frac{2\alpha}{\alpha - x_1}.$$

Bei der Speicherung der Matrix $P = I - 2(\omega\omega^T)/(\omega^T\omega)$, $P \in \mathbb{R}^{n \times n}$, auch des Vektors $(\omega_2, \dots, \omega_n)^T$ ist zu berücksichtigen, der sich auf den $n - 1$ freigewordenen Einträgen

speichern lässt.

Algorithmus 4.4.3 — Berechnung Householder-Vektor. Zu gegebenem $x \in \mathbb{R}^n$ berechnet die Funktion ein $\omega \in \mathbb{R}^n$ mit $\omega(1) = 1$ und $\beta \in \mathbb{R}$, so dass $P = I_n - \beta\omega\omega^T$ orthogonal ist und $Px = \|x\|_2 e_1$.

```
function: [ $\omega, \beta$ ] = housevector(x)
n = length(x)
 $\sigma = x(2:n)^T x(2:n)$ 
 $\omega = \begin{bmatrix} 1 \\ x(2:n) \end{bmatrix}$ 
if  $\sigma = 0$ 
     $\beta = 0$ 
else
     $\mu = \sqrt{x(1)^2 + \sigma}$ 
    if  $x(1) \leq 0$ 
         $\omega(1) = x(1) - \mu$ 
    else
         $\omega(1) = -\sigma / (x(1) + \mu)$ 
    end
     $\beta = 2\omega(1)^2 / (\sigma + \omega(1)^2)$ 
     $\omega = \omega / \omega(1)$ 
end
```

MATLAB-Funktion: HouseholderVektor.m

```
1 function [v,beta] = HouseholderVector(x)
2 n = length(x);
3 if n>1
4     sigma = x(2:end)'*x(2:end);
5     if sigma==0
6         beta = 0;
7     else
8         mu = sqrt(x(1)^2+sigma);
9         if x(1)<=0
10            tmp = x(1) - mu;
11        else
12            tmp = -sigma / (x(1) + mu);
13        end
14        beta = 2*tmp^2/(sigma + tmp^2);
15        x(2:end) = x(2:end)/tmp;
16    end
17    v = [1;x(2:end)];
18 else
19     beta = 0;
20     v = 1;
21 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:

$$H_1 A = \begin{pmatrix} * & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \\ 0 & * & * & * & * \end{pmatrix}$$

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

$$H_2 H_1 A = \begin{pmatrix} * & * & * & * & * \\ 0 & * & * & * & * \\ 0 & 0 & \boxplus & * & * \\ 0 & 0 & \boxplus & * & * \\ 0 & 0 & \boxplus & * & * \\ 0 & 0 & \boxplus & * & * \end{pmatrix}$$

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

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

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

$$H_3 H_2 H_1 A = \begin{pmatrix} * & * & * & * & * \\ 0 & * & * & * & * \\ 0 & 0 & * & * & * \\ 0 & 0 & 0 & * & * \\ 0 & 0 & 0 & * & * \\ 0 & 0 & 0 & * & * \end{pmatrix}$$

Mit dieser Vorgehensweise erhalten wir folgende Faktorisierung:

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

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

$$\omega^{(j)} = (\underbrace{0, \dots, 0}_{j-1}, 1, \omega_{j+1}^{(j)}, \dots, \omega_m^{(j)})^T$$

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 4.4.4 — Householder- QR : . Gegeben sei $A \in \mathbb{R}^{m \times n}$ mit $m \geq n$. Der folgende Algorithmus bestimmt die Householder-Matrizen H_1, \dots, H_n , so dass mit $Q = H_1 \cdots H_n$ die Matrix $R = Q^T A$ eine obere Dreiecksmatrix ist.

Der obere Dreiecksteil von A wird mit R überschrieben und unterhalb der Diagonalen werden die nichttrivialen Komponenten der Householder-Vektoren gespeichert.

```
function:  $A = \text{housematrix}(A)$ 
for  $j = 1 : n$ 
     $[\omega, \beta] = \text{housevector}(A(j : m, j))$ 
     $A(j : m, j : n) = (I_{m-j+1} - \beta \omega \omega^T) A(j : m, j : n)$ 
    if  $j < m$ 
         $A(j+1 : m, j) = \omega(2 : m - j + 1)$ 
    end
end
```

Bemerkung 4.4.5 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 $\text{span}\{\omega\}^\perp$. Hierzu Abbildung 4.3.

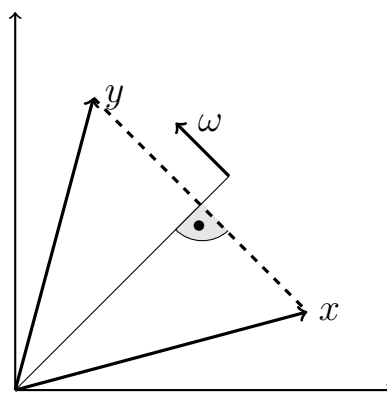


Abbildung 4.3: Spiegelung an einer Ebene mit Normalenvektor ω .

MATLAB-Funktion: Householder.m

```
1 function [A,R] = Householder(A)
```

```

2 for j = 1:size(A,2)
3     [v,beta(j)] = HouseholderVector(A(j:end,j));
4     A(j:end,j:end) = A(j:end,j:end) - v * (beta(j) * v'*A(j:end,j:
        end));
5     if j<size(A,1)
6         A(j+1:end,j) = v(2:end);
7     end
8 end
9 if nargin == 2
10    R = A;
11    A = eye(size(A,1));
12    for j = size(R,2):-1:1
13        v = [1;R(j+1:end,j)];
14        A(j:end,j:end) = A(j:end,j:end) - v*(beta(j)*v'*A(j:end,j:
            end));
15    end
16    R = triu(R);
17 end

```

MATLAB-Funktion: HouseholderMult.m

```

1 function y = prod_rx(A,x)
2 % extract R from A and multiply with x
3 y = triu(A) * x;
4
5 function y = prod_qx(A,x)
6 y = x;
7 for j = size(A,2)-1:-1:1
8     v = [1;A(j+1:end,j)];
9     beta = 2/(v'*v);
10    y(j:end) = y(j:end) - v*(beta*v'*y(j:end));
11 end
12
13 function y = prod_qtx(A,x)
14 y = x;
15 for j = 1:size(A,2)-1
16     v = [1;A(j+1:end,j)];
17     beta = 2/(v'*v);
18     y(j:end) = y(j:end) - v*(beta*v'*y(j:end));
19 end

```

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 4.4.6 — 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. Dazu sind Aufdatierungs-Strategien geeignet, von denen wir eine vorstellen.

Sei die Zerlegung $A_k = Q_k R_k$ gegeben und 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*) zu 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.

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

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

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. Multipliziert man R sukzessive von links mit Drehmatrizen $\mathcal{U}^T(p, q)$ (siehe (4.4)) zu Indexpaaren $(n, n-1), \dots, (4, 3), (3, 2)$ so hat QR obere Hessenberg-Gestalt, wobei $Q := \mathcal{U}_{32} \cdots \mathcal{U}_{n,n-1}$ ist. Dabei lassen sich die Drehmatrizen so wählen, dass $Qu = (*, *, 0, \dots, 0)^T$ gilt. Beachtet man nun, dass $(Qu)v^T$ und somit auch $Q(R+uv^T)$ obere Hessenberg-Form hat, so ist die Aussage bewiesen. ■

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 \tilde{Q} und eine Hessenberg-Matrix \tilde{H} mit $\tilde{Q}\tilde{H} = R+uv^T$ 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 Ansatz

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

liefert $v = t$ und $s = Q_k^T u$ 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

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

MATLAB-Funktion: GivensHessenberg.m

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

```

6   Q(:,[j+1,j]) = Q(:,[j+1,j]) * rot';
7   H(j+1,j) = 0;
8   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.

```

Kapitel 5

Nichtlineare Gleichungen

In der Vorlesung *Numerische Lineare Algebra* beschäftigen wir uns mit numerischen Lösungsverfahren für *lineare* Gleichungssysteme. Hier wenden wir uns nun *nichtlinearen* Gleichungen (in einer oder mehreren Variablen) zu. Dabei treten eine Reihe von Herausforderungen auf, wie folgende Beispiele zeigen.

5.0.1 Einführende Beispiele

■ **Beispiel 5.0.1** Gegeben seien zwei Funktionen $f : \mathbb{R}^2 \rightarrow \mathbb{R}$ und $g : \mathbb{R}^2 \rightarrow \mathbb{R}$ und wir betrachten das nichtlineare Gleichungssystem

$$f(x, y) = 0, \quad g(x, y) = 0$$

In Abbildung 5.1 ist die Situation für ein Beispiel von f und g gezeigt. Die durchgezogenen Linien sind Niveaulinien zu $f(x, y) = 0$, die gestrichelten Linien zu $g(x, y) = 0$. Wir erkennen sofort, dass es mehrere Lösungen gibt. Die Anzahl der Nullstellen ist im Allgemeinen a-priori nicht bekannt. Die gesuchten Lösungen sind die Schnittpunkte der völlig unabhängigen Nulllinien. Es ist klar, dass ein solches Problem für numerische Verfahren eine Herausforderung darstellt. ■

Bemerkung 5.0.2 Wir können jedes nichtlineare Gleichungssystem von n Gleichungen für n Unbekannte umschreiben in ein **Nullstellenproblem** $f(x) = 0$ für $f : \mathbb{R}^n \rightarrow \mathbb{R}^n$. Für $n = 1$ sprechen wir von einem **skalaren Problem**, ansonsten von einem **System**.

■ **Beispiel 5.0.3 — Zinssatz bei einem Kredit.** Wie hoch darf der Zinssatz sein, wenn man einen Kredit über €10.000 Euro in 10 Jahren abzahlen möchte und man höchstens €400 monatlich aufbringen kann? Oder allgemeiner: eine Kreditsumme K_0 soll in n Jahren mit monatlichen Raten R getilgt werden.

Der Einfachheit halber rechnen wir den jährlichen Zinssatz p in einen monatlichen Zinssatz m um, also $(1 + m)^{12} = 1 + p$. Für den Restbetrag K_i des Kredits nach $i \in \mathbb{N}$ Monaten gilt:

$$\begin{aligned} K_i &= K_{i-1}(1 + m) - R \\ K_{i+1} &= K_i(1 + m) - R = [K_{i-1}(1 + m) - R](1 + m) - R \\ &= K_{i-1}(1 + m)^2 - R[(1 + m) + 1] \end{aligned}$$

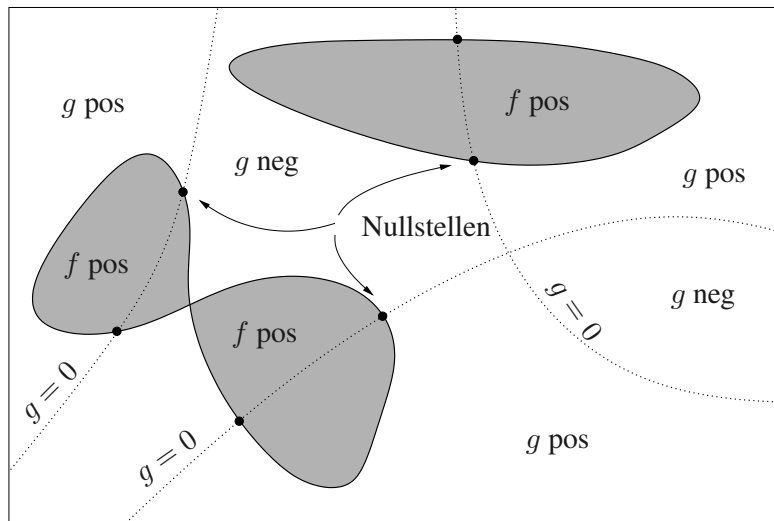


Abbildung 5.1: Lösung von zwei Gleichungen in zwei Unbekannten.

$$\begin{aligned}
 & \vdots \\
 K_n &= K_0(1+m)^n - R \left[(1+m)^{n-1} + \dots + (1+m) + 1 \right] \\
 &= K_0(1+m)^n - R \left[(1+m)^n - 1 \right] / m = (K_0 - R/m)(1+m)^n + R/m
 \end{aligned}$$

Zu gegebener Kreditsumme K_0 , monatlichem Zinssatz m und Tilgungsdauer (n Monate) ergibt sich die Rate R , um den Kredit vollständig zu tilgen, indem wir $K_n = 0$ setzen, d.h.

$$R = K_0 m (1+m)^n / [(1+m)^n - 1].$$

Auch die Tilgungsdauer lässt sich analytisch darstellen

$$n = \frac{\log(R / (R - m K_0))}{\log(1+m)}.$$

Zu gegebenen $K_0 > 0$, $n \in \mathbb{N}$, $K_0 > R > K_0/n$ suchen wir also m mit $(m K_0 - R)(1+m)^n + R = 0$, d.h.

$$f(m) = 0 \quad \text{für } f(m) := (m K_0 - R)(1+m)^n + R. \quad (5.1)$$

Man kann leicht nachweisen, dass f in $(0, 1)$ genau eine Nullstelle besitzt, die es numerisch zu bestimmen gilt, vgl. Abbildung 5.2. ■

■ **Beispiel 5.0.4 — Extremalstellen von skalaren Funktionen.** Die Rosenbrock¹-Funktion

$$f(x) := \sum_{i=1}^{n-1} \left[(1 - x_i)^2 + 100(x_{i+1} - x_i^2)^2 \right] \quad (x \in \mathbb{R}^n)$$

ist ein Standardbeispiel aus der numerischen Optimierung (vgl. die entsprechende Vorlesung). Sie hat für $n \geq 4$ mindestens ein lokales Minimum in der Umgebung von

¹Howard Harry Rosenbrock, 1920-2010.

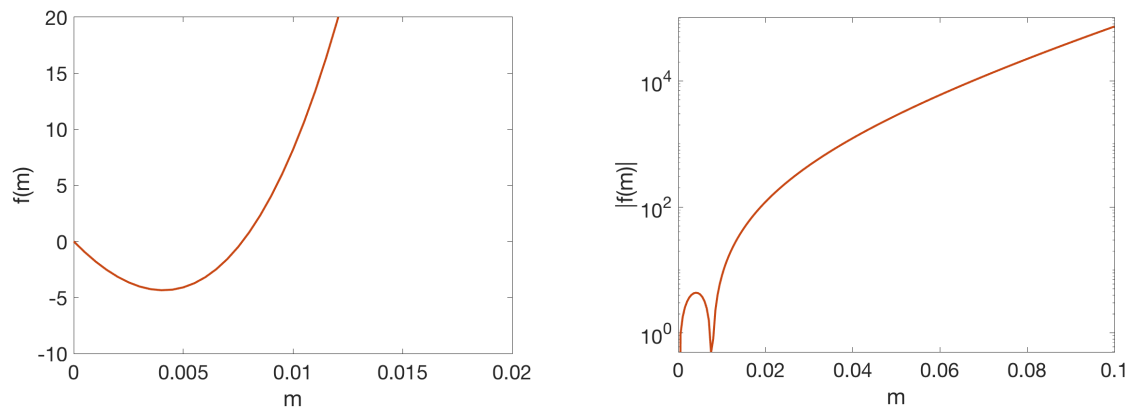


Abbildung 5.2: Der Graph von f aus (5.1) (links, y-Achse linear skaliert, $x \in [0, 0.02]$) bzw. $|f(m)|$ (rechts, logarithmisch-skaliert, $x \in [0, 0.1]$) für $K_0 = 10000$, $R = 250$ und $n = 48$. Die Funktion f hat eine Nullstelle bei 0 und bei etwa 0.008.

$(x_1, x_2, \dots, x_n) = (-1, 1, \dots, 1)$ neben dem globalen Minimum $(x_1, \dots, x_n) = (1, \dots, 1)$. Diese Extremalstellen erfüllen die notwendige Bedingung erster Ordnung $\nabla f = 0$. Wir suchen also ein $x \in \mathbb{R}^n \setminus \{(1, \dots, 1)\}$ mit

$$\begin{aligned} g(x) &= (g_1(x), \dots, g_n(x))^T = 0, \\ g_1(x) &:= \frac{\partial f}{\partial x_1} = -2(1 - x_1) - 400x_1(x_2 - x_1^2), \\ g_i(x) &:= \frac{\partial f}{\partial x_i} = -2(1 - x_i) - 400x_i(x_{i+1} - x_i^2) + 200(x_i - x_{i-1}^2), \quad i = 2, \dots, n-1, \\ g_n(x) &:= \frac{\partial f}{\partial x_n} = 200(x_n - x_{n-1}^2), \end{aligned}$$

also ein nichtlineares Gleichungssystem der Dimension $n + 1$. ■

5.0.2 Konvergenzgeschwindigkeit und Aufwand

Wir werden verschiedene numerische Verfahren zur näherungsweisen Lösung von nichtlinearen Problemen kennenlernen. Es stellt sich die Frage, wie wir die Qualität der verschiedenen Verfahren bewerten und vergleichen können. Wir werden ausschließlich **iterative Verfahren** vorstellen. Damit sind zwei naheliegende Kriterien:

- **Konvergenzgeschwindigkeit:** wie schnell konvergiert ein Verfahren gegen eine Nullstelle?
- **Aufwand:** wie viele Operationen werden pro Iteration benötigt?

Definition 5.0.5 — Konvergenzgeschwindigkeit. Eine konvergente Folge $(x^{(k)})_{k \in \mathbb{N}} \subset \mathbb{R}^n$ mit Grenzwert $x^* \in \mathbb{R}^n$ heißt

- (a) **linear konvergent**, wenn es ein $\varrho \in (0, 1)$ (den **Konvergenzfaktor** oder auch die **Kontraktionsrate**) gibt mit

$$\|x^* - x^{(k+1)}\| \leq \varrho \|x^* - x^{(k)}\|, \quad k \in \mathbb{N}_0;$$

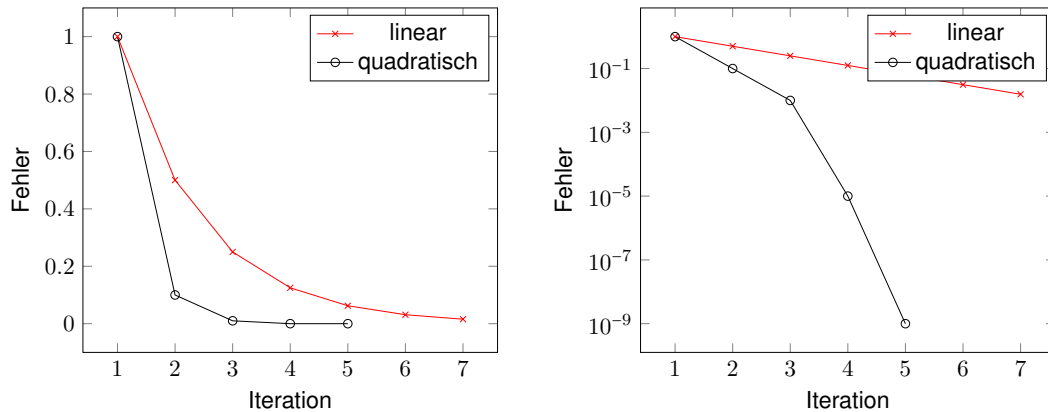


Abbildung 5.3: Darstellung linearer und quadratischer Konvergenz in normaler (links) und semilogarithmischer (rechts) Skala.

- (b) **superlinear konvergent**, wenn es eine gegen Null konvergente Folge $(\varrho_k)_{k \in \mathbb{N}} \in \mathbb{R}^+$ gibt mit

$$\|x^* - x^{(k+1)}\| \leq \varrho_k \|x^* - x^{(k)}\|, \quad k \in \mathbb{N}_0;$$

- (c) konvergent mit **Konvergenzordnung** q , wenn es ein $\varrho > 0$ und ein $1 < q \in \mathbb{R}$ gibt mit

$$\|x^* - x^{(k+1)}\| \leq \varrho \|x^* - x^{(k)}\|^q, \quad k \in \mathbb{N}_0.$$

Für $q = 2$ spricht man von **quadratischer Konvergenz**.

Wie kann man die Konvergenzordnung visualisieren, z.B. bei Ergebnissen von numerischen Experimenten? In Abbildung 5.3 haben wir eine linear (rot) und eine quadratisch (schwarz) konvergente Folge visualisiert. In der „normalen“ Darstellung (links) erkennt man i.W. nichts, während man in semilogarithmischer Skala (rechts) lineare Konvergenzordnung an der Steigung der (Ausgleichs-)Geraden durch den Fehlerverlauf ablesen kann und quadratische Konvergenz durch eine exponentiell verlaufende Kurve erkennbar ist. Stellt man den Logarithmus des Fehlers in semilogarithmischer Skala dar, erkennt man bei Konvergenzordnung zwei eine Gerade mit negativer Steigung von zwei.

Je höher die Konvergenzordnung, desto schneller ist das Verfahren. Was aber nutzt eine hohe Konvergenzordnung, wenn man diese mit vielen Operationen „erkauft“? Ein solches Verfahren wäre ineffizient. Die „teuersten“ Operationen sind i.d.R. bei der **Auswertung der Funktion** f versteckt (dies könnte sogar eine komplexe Simulation wie z.B. die Lösung einer partiellen Differenzialgleichung bedeuten). Daher verwendet man die Anzahl der Funktionsauswertungen als Synonym für den Aufwand bzw. die Anzahl der Operationen.

Definition 5.0.6 Sei m die Anzahl von Funktionsauswertungen pro Iterationsschritt. Damit definiert man den *Effizienzindex*

$$\text{ind} := p^{\frac{1}{m}}. \quad (5.2)$$

Wir untersuchen diese Größe für die verschiedenen Verfahren zur Lösung skalarer

Probleme am Ende des kommenden Abschnittes.

5.1 Skalare Probleme

Wir beginnen mit skalaren Problemen, d.h.

bestimme $x^* \in \mathbb{R}$ also Lösung von $f(x) = 0$ für $f : \mathbb{R} \rightarrow \mathbb{R}$. (5.3)

5.1.1 Bisektionsmethode

Diese Methode ist auch als **Intervallschachtelung** bekannt:

1. angenommen, es sei ein Intervall $I := [a, b]$ bekannt mit $f(a) \cdot f(b) < 0$. Wenn $f \in C([a, b])$ ist, liegt also in (a, b) mindestens eine Nullstelle;
2. betrachte den Mittelpunkt $m = \frac{1}{2}(a+b)$ von I und bestimme den Funktionswert $f(m)$;
3. ist $f(a) \cdot f(m) < 0$ setze mit $[a, m]$ fort, ansonsten mit $[m, b]$.

Der entsprechende Algorithmus ist leicht in MATLAB[®] umgesetzt.

MATLAB-Funktion: BisektionsMethode.m

```
1 function Nullstelle = BisektionsMethode(a,b,func,epsilon)
2 % Initialisierung
3 temp=[]
4 fa = func(a); fb = func(b);
5 while abs(a-b) > epsilon
6     m = (a+b)/2;
7     fm = func(m);
8     if fm == 0
9         Nullstelle = m;
10        return
11    elseif fa*fm < 0
12        b = m;
13        fb = fm;
14    else
15        a = m;
16        fa = fm;
17    end
18    temp = [temp;m];
19 end
20 % Loesung
21 Nullstelle = m;
```

Wir testen die Bisektionsmethode an Beispiel 5.0.3, also der Frage, wie hoch darf der Zinssatz sein, wenn man einen Kredit über €10.000 in 48 Monaten zurückzahlen möchte, aber nur €250 monatlich zur Tilgung aufbringen kann?

```
>> n = 48;
>> K0 = 10000;
>> R = 250;
>> f = @(m) (m*K0-R)*(1+m)^n+R;
>> m = Bisektionsmethode(eps,1,f,1e-7)
m =
    0.00770145654678
>> p = 100 * ((1+m)^12-1)
p =
    9.64343564476941
```

Nach der Umrechnung in den jährlichen Zinssatz sehen wir, dass wir uns den Kredit nur erlauben könnten, wenn der Zinssatz niedriger als 9.65% ist.

Lemma 5.1.1 — Konvergenz des Bisektionsverfahrens. Angenommen, $x^* \in (a, b)$ sei eine Nullstelle von f . Weiter sei $x^{(0)} := a$, $x^{(1)} := b$ und $x^{(k)}$ der Mittelpunkt im k -ten Schritt der Bisektionsmethode. Dann gilt

$$|x^{(k)} - x^*| \leq \frac{b-a}{2^{k+1}}, k \in \mathbb{N}_0.$$

Beweis. Die Schranke ergibt sich aus der Intervalllänge im k -ten Schritt. ■

Bemerkung 5.1.2 Man beachte, dass Lemma 5.1.1 *nicht* bedeutet, dass die Methode linear konvergiert, wie wir am folgendem Beispiel sehen: Wir suchen die Nullstelle der Funktion $f(x) := \cos((\pi+0.1)x)$ für $x \in [0, 1]$. Der Fehlerverlauf ist in semilogarithmischer Skala in Abbildung 5.4 dargestellt. Wir können keine Gerade erkennen, sondern einen Zick-Zack-Verlauf. Legt man durch die Daten eine Ausgleichsgerade, so würde diese lineare Konvergenz zeigen, das Bisektionsverfahren ist aber **nicht monoton** in dem Sinne, dass der Fehler in jeder Iteration monoton abnehmen würde.

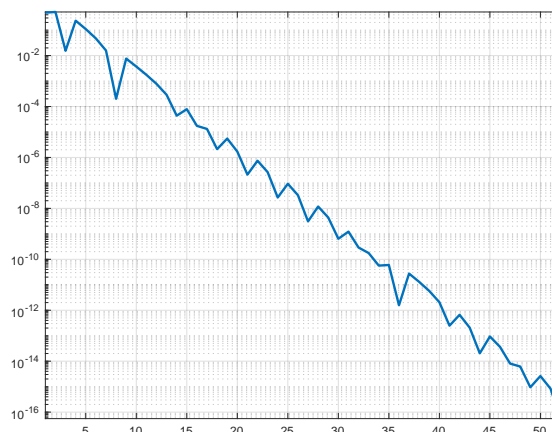


Abbildung 5.4: Konvergenzhistorie des Bisektionsverfahrens: das Verfahren ist nicht monoton.

5.1.2 Regula-Falsi

Bei der Regula-Falsi² gehen wir wieder davon aus, dass ein Intervall $I := [a, b]$ mit $f(a) \cdot f(b) < 0$ bekannt sei. Nun ersetzen wir f durch die Gerade durch die Punkte $(a, f(a))$ und $(b, f(b))$ (der „falsche Ansatz“) und verwenden die Nullstelle der Geraden

$$\xi = \frac{af(b) - bf(a)}{f(b) - f(a)} \quad (5.4)$$

als nächste Iteration. Mit den beiden Intervallen $[a, \xi]$, $[\xi, b]$ verfahren wir dann analog zur Methode der Intervallhalbierung, vgl. Abbildung 5.5.

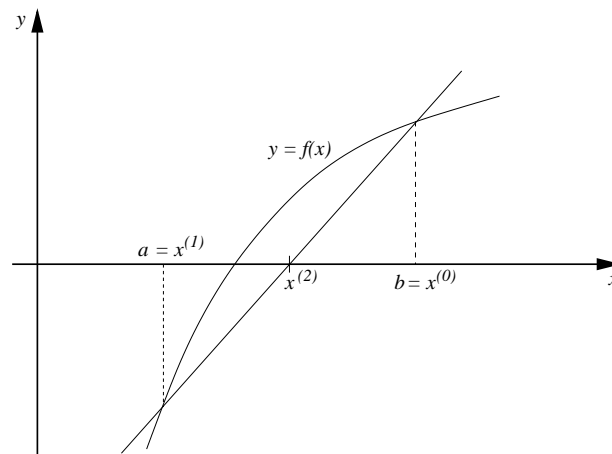


Abbildung 5.5: Regula-Falsi-Methode.

Als Algorithmus ergibt sich somit:

MATLAB-Funktion: RegulaFalsi.m

```
1 function Nullstelle = RegulaFalsi(a,b,func,epsilon)
2 fa = func(a); fb = func(b); % Initialisierung
3 m = (a*fb-b*fa)/(fb-fa); fm = func(m);
4 while abs(fm) > epsilon
5     if fa*fm < 0
6         b = m;
7         fb = fm;
8     else
9         a = m;
10        fa = fm;
11    end
12    m = (a*fb-b*fa)/(fb-fa);
13    fm = func(m);
14 end
15 Nullstelle = m; % Loesung
```

Satz 5.1.3 Es sei $f \in C^2(I)$, x^* sei einzige Nullstelle von f in $I := [a, b]$ und $f'(x^*) \neq 0$.

²lat. „Regel des falschen Ansatzes.“

$f''(x^*) \neq 0$. Dann beträgt die Konvergenzordnung der Regula-Falsi-Methode $p = 1$.

Beweis. Es bezeichne $(x^{(k)})_{k \in \mathbb{N}}$ die sich aus der Regula Falsi ergebende Folge von „Mittelpunkten“, d.h. $x^{(k)} \in (a^{(k)}, b^{(k)})$, $k \in \mathbb{N}$, mit den sich ergebenden Intervallen $(a^{(k)}, b^{(k)})$. Es seien

$$\varepsilon_k^a := a^{(k)} - x^*, \quad \varepsilon_k^b := b^{(k)} - x^*.$$

Aus (5.4) und der Voraussetzung $f(x^*) = 0$ folgt

$$\begin{aligned} \varepsilon_k &:= x^{(k)} - x^* = \frac{(a^{(k)} - x^*)f(b^{(k)}) - (b^{(k)} - x^*)f(a^{(k)})}{f(b^{(k)}) - f(a^{(k)})} \\ &= \frac{\varepsilon_k^a f(x^* + \varepsilon_k^b) - \varepsilon_k^b f(x^* + \varepsilon_k^a)}{f(x^* + \varepsilon_k^b) - f(x^* + \varepsilon_k^a)}. \end{aligned}$$

Da $f \in C^3(I)$ nach Voraussetzung gilt, liefert die Taylor-Entwicklung (Satz B.2.1)³:

$$\begin{aligned} \varepsilon_k &= \frac{\varepsilon_k^a \{ \varepsilon_k^b f'(x^*) + \frac{1}{2}(\varepsilon_k^b)^2 f''(x^*) + \dots \} - \varepsilon_k^b \{ \varepsilon_k^a f'(x^*) + \frac{1}{2}(\varepsilon_k^a)^2 f''(x^*) + \dots \}}{\{ \varepsilon_k^b f'(x^*) + \frac{1}{2}(\varepsilon_k^b)^2 f''(x^*) + \dots \} - \{ \varepsilon_k^a f'(x^*) + \frac{1}{2}(\varepsilon_k^a)^2 f''(x^*) + \dots \}} \\ &= \frac{\frac{1}{2} \varepsilon_k^a \varepsilon_k^b (\varepsilon_k^b - \varepsilon_k^a) f''(x^*) + \dots}{(\varepsilon_k^b - \varepsilon_k^a) \{ f'(x^*) + \frac{1}{2}(\varepsilon_k^b + \varepsilon_k^a) f''(x^*) + \dots \}} = \frac{\frac{1}{2} \varepsilon_k^a \varepsilon_k^b f''(x^*) + \dots}{f'(x^*) + \dots}. \end{aligned}$$

Nach der Herleitung des Algorithmus gilt $a^{(k)} < x^* < b^{(k)}$. Demzufolge besitzen ε_k^a und ε_k^b entgegengesetztes Vorzeichen und es gilt $\varepsilon_k^b - \varepsilon_k^a > 0$. Für hinreichend kleine $|\varepsilon_k^a|$ und $|\varepsilon_k^b|$ folgt damit

$$\varepsilon_k \approx \frac{f''(x^*)}{2f'(x^*)} \varepsilon_k^a \varepsilon_k^b. \quad (5.5)$$

Mit $f''(x^*) \neq 0$ gilt weiterhin, dass f in einer hinreichend kleinen Umgebung von x^* konvex oder konkav ist, und folglich bleibt ein Intervallende fest und wird im Verfahren nur umbenannt. Deshalb ist ε_k nur direkt proportional zu einem der beiden vorhergehenden ε_k^a oder ε_k^b . Die asymptotische Fehlerkonstante C ist für eine konkave Funktion gegeben durch

$$C = \left| \frac{f''(x^*)}{2f'(x^*)} \varepsilon_k^a \right|, \quad \varepsilon_k^a = \varepsilon_{k-1}^a = \dots = \varepsilon_1^a = x^{(1)} - x^*,$$

und damit konvergiert die Folge $(x^{(k)})$ linear. Analoges gilt für konvexe Funktionen mit ε_k^b anstatt ε_k^a . ■

³Brook Taylor, 1685-1731.

Testen wir das Regula-Falsi-Verfahren an Beispiel 5.0.3 $K_0 = 10000$, $n = 48$ und $R = 250$.

```
>> n = 48;
>> K0 = 10000;
>> R = 250;
>> f = @(m) (m*K0-R)*(1+m)^n+R;
>> m = RegulaFalsi(1e-5,0.1,f,1e-7)
m =
    0.00770147244890
```

Man beachte, dass wir hier andere Startwerte als bei der Bisektionsmethode verwendet haben.

5.1.3 Die Sekantenmethode

Als Modifikation des Regula-Falsi-Verfahrens verzichtet man bei der Sekantenmethode darauf, die Lösung durch zwei Näherungswerte einzuschließen. Zu zwei vorgegebenen Näherungswerten $x^{(0)}$ und $x^{(1)}$, welche die Lösung **nicht notwendigerweise einzuschließen** brauchen, bestimmt man x_2 als Nullstelle der Sekante zu $(x^{(0)}, f(x^{(0)}))$ und $(x^{(1)}, f(x^{(1)}))$. Ungeachtet der Vorzeichen bestimmt man aus $x^{(1)}$, $x^{(2)}$ eine nächste Näherung $x^{(3)}$, vgl. Abbildung 5.6.

Die Iterationsvorschrift der Sekantenmethode ergibt sich damit zu

$$x^{(k+1)} = x^{(k)} - f(x^{(k)}) \frac{x^{(k)} - x^{(k-1)}}{f(x^{(k)}) - f(x^{(k-1)})}, \quad k \in \mathbb{N}. \quad (5.6)$$

Dieses zweistufige Iterationsverfahren setzt natürlich $f(x^{(k)}) \neq f(x^{(k-1)})$ voraus, sonst bricht das Verfahren ab.

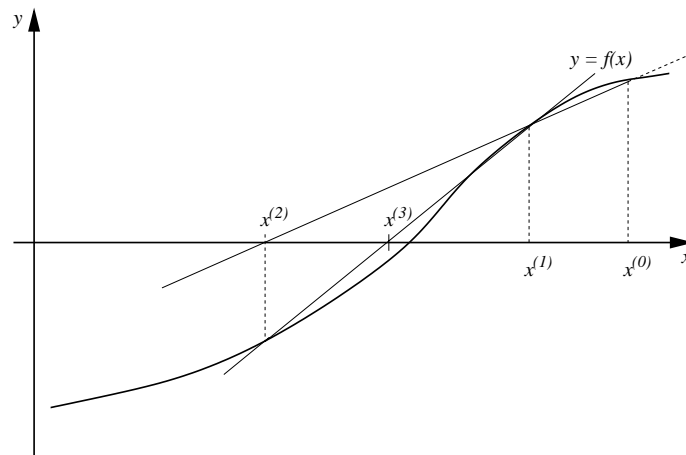


Abbildung 5.6: Geometrische Darstellung der Sekantenmethode.

Satz 5.1.4 Unter den Voraussetzungen von Satz 5.1.3 konvergiert die Sekantenmethode mit Ordnung $p = \frac{1}{2}(1 + \sqrt{5})$.⁴

Beweis. (Skizze, vgl. auch [Han09, Satz 18.4]) Wir betrachten die Sekantenmethode als Modifikation des Regula-Falsi-Verfahrens. Für hinreichend kleine Fehler $|\varepsilon_{k-1}|$ und $|\varepsilon_k|$

⁴Dies ist der goldene Schnitt.

bleibt (5.5) für die Sekantenmethode gültig, bzw. geht über in

$$\varepsilon_{k+1} \approx \frac{f''(x^*)}{2f'(x^*)} \varepsilon_k \varepsilon_{k-1}. \quad (5.7)$$

Es besteht jedoch der Unterschied, dass sich beide Werte ε_{k-1} und ε_k bei Erhöhung von k ändern. Mit der Konstanten $C := |f''(x^*)/(2f'(x^*))|$ gilt für hinreichend großes k

$$|\varepsilon_{k+1}| \approx C |\varepsilon_k| \cdot |\varepsilon_{k-1}|.$$

Nach Definition der Konvergenzordnung versuchen wir nun diese *Differenzengleichung* mit dem Ansatz

$$|\varepsilon_k| = \varrho |\varepsilon_{k-1}|^p, \quad \varrho > 0, \quad p \geq 1$$

zu lösen. Einsetzen ergibt

$$(|\varepsilon_{k+1}| =) \quad \varrho |\varepsilon_k|^p = \varrho \cdot \varrho^p |\varepsilon_{k-1}|^{p^2} \stackrel{!}{=} C \cdot \varrho |\varepsilon_{k-1}|^{p+1} \quad (= C |\varepsilon_k| \cdot |\varepsilon_{k-1}|).$$

Diese letzte Gleichung kann aber für alle (hinreichend großen) k nur dann gelten, wenn $\varrho^p = C$ und $p^2 = p + 1$ erfüllt sind. Die positive Lösung $p = \frac{1}{2}(1 + \sqrt{5})$ der quadratischen Gleichung ist deshalb die Konvergenzordnung der Sekantenmethode. Die asymptotische Fehlerkonstante ist $\varrho = C^{1/p} = C^{0.618}$. ■

MATLAB-Funktion: SekantenMethode.m

```

1 function x1 = SekantenMethode(x0,x1,func,epsilon)
2 % Initialisierung
3 fx0 = func(x0); fx1 = func(x1);
4 while abs(fx1) > epsilon && abs(fx0-fx1) > epsilon
5     tmp = x1;
6     x1 = x1 - fx1 * (x1-x0)/(fx1-fx0);
7     x0 = tmp;
8     fx0 = fx1;
9     fx1 = func(x1);
10 end

```

MATLAB-Beispiel:

Wir testen auch die Sekantenmethode an Beispiel 5.0.3 mit $K_0 = 10000$, $n = 48$ und $R = 250$.

```
>> n = 48;
>> K0 = 10000;
>> R = 250;
>> f = @(m) (m*K0-R)*(1+m)^n+R;
>> m = SekantenMethode(1e-2,0.1,f,1e-7)
m =
    0.00770147248822
```

Im Vergleich zu den oben gemachten Tests ist hier das Startintervall kleiner zu wählen.

5.1.4 Das Newton-Verfahren

Ist die Funktion f der zu lösenden nichtlinearen Gleichung $f(x) = 0$ stetig differenzierbar, so wird im Newton-Verfahren⁵ die Funktion f im Näherungswert $x^{(k)}$ linearisiert (durch die Tangente approximiert) und die nächste Iteration $x^{(k+1)}$ als Nullstelle dieser Tangente definiert, vgl. Abbildung 5.7.

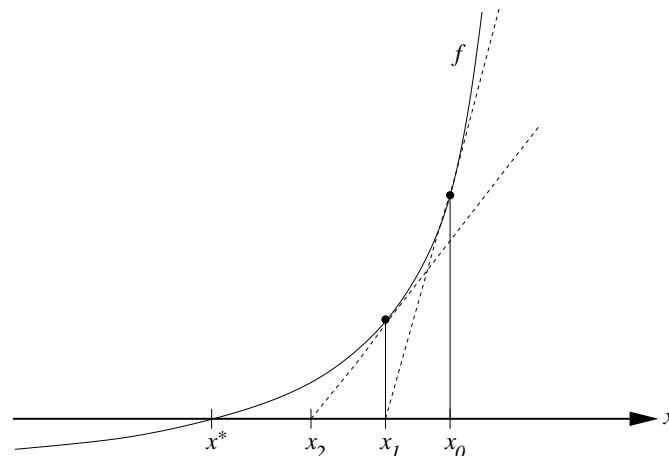


Abbildung 5.7: Die geometrische Interpretation des Newton-Verfahrens.

Aus der Tangentengleichung

$$y(x) = f(x^{(k)}) + (x - x^{(k)})f'(x^{(k)}) \stackrel{!}{=} 0$$

ergibt sich die Iterationsvorschrift

$$x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{f'(x^{(k)})}. \quad (5.8)$$

⁵Isaac Newton, 1642-1727.

Satz 5.1.5 Es sei $I := [a, b]$, $f \in C^3(I)$ und es existiere eine einfache Nullstelle $x^* \in (a, b)$ von f , d.h., $f'(x^*) \neq 0$. Dann existiert ein $\delta > 0$ so, dass das Newton-Verfahren für jeden Startwert $x^{(0)} \in \mathcal{U}_\delta(x^*)$ ⁶ mindestens quadratisch konvergiert, das Verfahren konvergiert **lokal quadratisch**.

Beweis. Wir stellen den Beweis zurück, da wir ihn später allgemeiner für das Newton-Verfahren für Systeme erbringen werden, vgl. Satz 5.2.11. ■

■ **Beispiel 5.1.6** Wir betrachten noch einmal das Problem aus Bsp. 5.0.3 mit $K_0 = 10000$, $n = 48$ und $R = 250$. Wir vergleichen die vorgestellten Verfahren mit $I = [a, b] = [0.005, 0.01]$ und wählen als Startwert für das Newton-Verfahren $x^{(0)} = b$.

Die Ergebnisse sind in Abbildung 5.8 dargestellt. In der linken Grafik sind die Fehler in semilogarithmischer Skala dargestellt, wobei eine Referenzlösung für x^* mittels eines Newton-Verfahrens mit hoher Genauigkeit berechnet wurde. Wir erkennen sowohl die nicht-monotone Konvergenz der Bisektionsmethode als auch die lineare Konvergenz von Regula Falsi. Sekanten- und Newton-Verfahren sind deutlich schneller, die Konvergenzordnung kann man nicht ablesen. Daher ist in der rechten Grafik der Logarithmus des Fehlers in semilogarithmischer Skala dargestellt. Wir erkennen, dass beide Verfahren sogar schneller konvergieren als es die Theorie vorhersagt, vgl. auch Beispiel 5.1.7. Für dieses Beispiel ist die Konvergenzgeschwindigkeit von Bisektion und Regula Falsi nahezu identisch. ■

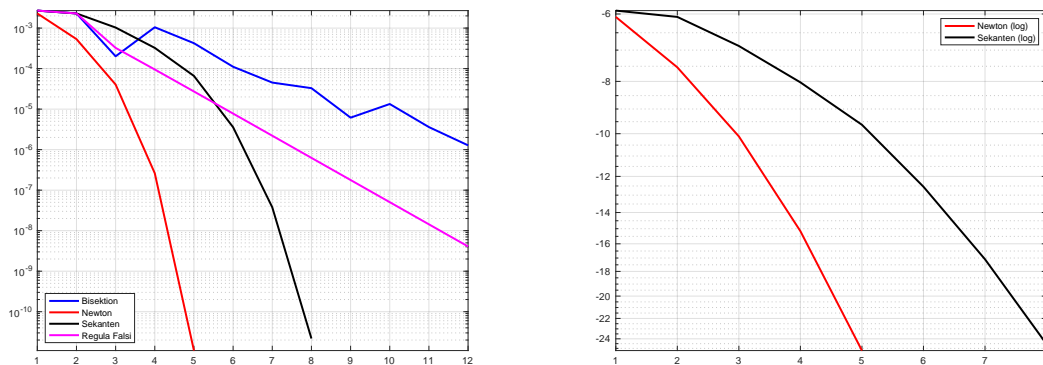


Abbildung 5.8: Konvergenz von Bisektions-, Regula Falsi-, Sekanten- und Newton-Verfahren für Beispiel 5.0.3 (links: semilogarithmisch, recht: Logarithmus des Fehlers von Newton- und Sekanten-Verfahren in semilogarithmischer Skala).

■ **Beispiel 5.1.7** Wir betrachten die Funktion $f(x) := \frac{x}{1-x}$ mit der Nullstelle $x^* = 0$. Wegen $f'(x) = \frac{1}{(1-x)^2}$ gilt dann für das Newton-Verfahren

$$x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{f'(x^{(k)})} = x^{(k)} - \frac{x^{(k)}}{1-x^{(k)}}(1-x^{(k)})^2 = x^{(k)} - x^{(k)}(1-x^{(k)}) = (x^{(k)})^2,$$

⁶ $\mathcal{U}_r(x) := \{y \in \mathbb{R} : |x - y| \leq r\}$ bezeichnet die Umgebung mit Radius $r > 0$ um $x \in \mathbb{R}$.

und damit für den Fehler

$$|x^{(k+1)} - x^*| = |(x^{(k)})^2 - 0| = (x^{(k)})^2 = |x^{(k)} - x^*|^2,$$

also konvergiert das Newton-Verfahren *genau* von Ordnung 2. Dies sehen wir auch in Abbildung 5.9, wo wir $I = [-0.45, 0.5] = [a, b]$ als Startwert für das Sekanten- und $x^{(0)} = a = -0.45$ als Startwert für das Newton-Verfahren verwendet haben. Aus der Darstellung in der rechten Grafik (Logarithmus des Fehlers in semilogarithmischer Skala) erkennen wir exakt die Ordnung $p = 2$ für das Newton- bzw. $p = \frac{1}{2}(1 + \sqrt{5})$ für das Sekanten-Verfahren. ■

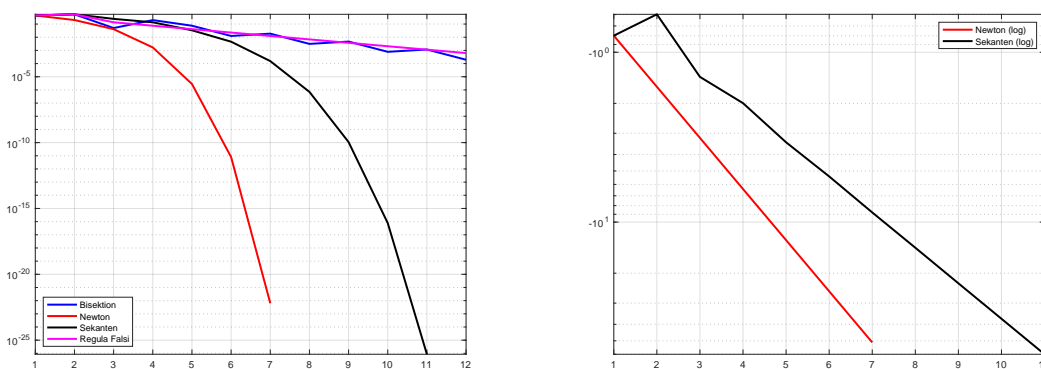


Abbildung 5.9: Konvergenz von Bisektions-, Regula Falsi-, Sekanten- und Newton-Verfahren für Beispiel 5.1.7 (links: semilogarithmisch, recht: Logarithmus des Fehlers von Newton- und Sekanten-Verfahren in semilogarithmischer Skala).

5.1.5 Effizienz

Wir vergleichen die Konvergenzordnung p und die Anzahl m der Funktionsauswertungen für die oben vorgestellten Verfahren zusammen. Daraus ergibt sich der Effizienzindex ind aus Definition 5.0.6. Beim Bisektionsverfahren und bei der Regula Falsi werden im ersten Schritt zwei Funktionsauswertungen benötigt, danach nur noch jeweils eine.

Verfahren	p	m	ind
Bisektion*	—	1	—
Regula Falsi*	1	1	1
Sekanten	$\frac{1+\sqrt{5}}{2}$	1	$\frac{1+\sqrt{5}}{2} \approx 1.6$
Newton	2	2^{**}	$\sqrt{2} \approx 1.4$

Tabelle 5.1: Konvergenzrate, Anzahl Funktionsauswertungen und Effizienzindex für verschiedene Verfahren. *: ab der zweiten Iteration; **: Unter der Annahme, dass die Auswertung von f' in etwa so aufwendig ist wie die von f .

Wir erkennen, dass das Sekantenverfahren für skalare Probleme den höchsten Effizienzindex besitzt. Dies bestätigt ein numerisches Experiment, bei dem wir die CPU-Zeiten

vom Sekanten- und Newton-Verfahren für Beispiel 5.1.7. Man sieht tatsächlich, dass das Sekanten-Verfahren pro Iteration schneller ist (vgl. Abbildung 5.10 links). Die Situation sieht etwas anders aus, wenn man die Zeit in Bezug zum Fehler setzt, vgl. Abbildung 5.10 rechts. Die hohe Konvergenzordnung des Newton-Verfahrens führt dazu, dass hohe Genauigkeit nur mit diesem Verfahren erreichbar ist.

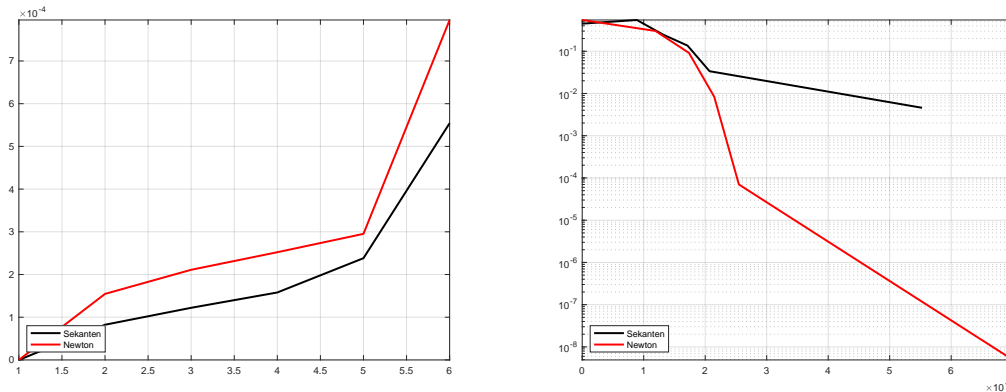


Abbildung 5.10: CPU-Zeiten von Sekanten- und Newton-Verfahren für Beispiel 5.1.7. Links: CPU-Zeit über Anzahl der Iterationen, rechts: CPU-Zeit über Fehler, semilogarithmisch.

5.2 Systeme nichtlinearer Gleichungen

Wir betrachten nun die allgemeinere Situation von Systemen von n nichtlinearen Gleichungen, d.h. $f : \mathbb{R}^n \rightarrow \mathbb{R}^n$ mit $n > 1$.⁷ Die geometrischen Überlegungen von Bisektions-, Regula Falsi- und Sekantenverfahren lassen sich nicht auf Systeme übertragen. Daher betrachtet man Fixpunktiterationen. Unterschiedliche Verfahren ergeben sich dann aus verschiedenen Varianten der Fixpunktfunktion.

5.2.1 Fixpunktiterationen

Wie bei klassischen Iterationsverfahren zur Lösung linearer Gleichungssysteme (vgl. Vorlesung *Numerische Lineare Algebra*) besteht die einfache Idee darin, das Nullstellen-Problem $f(x) = 0$ in ein *Fixpunkt-Problem* der folgenden Form umzuschreiben: Für

$$\phi : \mathbb{R}^n \rightarrow \mathbb{R}^n \quad \text{definiert durch} \quad \phi(x) := x - f(x). \quad (5.9)$$

gilt offensichtlich gilt $f(x^*) = 0$ genau dann, wenn $x^* = \phi(x^*)$, also wenn x^* Fixpunkt von ϕ ist. Natürlich sind auch andere Definitionen für ϕ denkbar, so dass wir eine ganze Klasse von Verfahren erhalten.

Die Konvergenz von Fixpunkt-Iterationen wird durch den Banach'schen Fixpunktsatz geklärt. Wir formulieren und beweisen diesen hier in einem etwas allgemeineren Rahmen als er aus der Analysis bekannt sein dürfte.

⁷Dabei muss f nicht auf ganz \mathbb{R}^n definiert sein, d.h. der Fall $f : \Omega \rightarrow \mathbb{R}^n$ mit $\Omega \subset \mathbb{R}^n$ offen ist ebenso zugelassen.

Satz 5.2.1 — Banach'scher Fixpunktsatz. Sei X ein linear normierter Raum mit Norm $\|\cdot\|$, $E \subseteq X$ sei eine vollständige Teilmenge von X . Die Abbildung $\phi : X \rightarrow X$ erfülle folgende Bedingungen:

- (i) **Selbstabbildung:** $\phi(E) \subseteq E$, also $\phi : E \rightarrow E$.
- (ii) **Kontraktion:** Es gelte $\|\phi(x) - \phi(y)\| \leq L\|x - y\|$ für alle $x, y \in E$ (Lipschitz-Stetigkeit) mit einer Konstanten $L < 1$.

Dann gilt:

- (a) Es existiert genau ein Fixpunkt x^* von ϕ in E .
- (b) Die Iteration

$$x^{(k+1)} := \phi(x^{(k)}), \quad k = 0, 1, 2, \dots \quad (5.10)$$

konvergiert für jedes $x^{(0)} \in E$ gegen den Fixpunkt x^* .

- (c) A-priori-Fehlerabschätzung:

$$\|x^{(k)} - x^*\| \leq \frac{L^k}{1-L} \|x^{(1)} - x^{(0)}\|. \quad (5.11)$$

- (d) A-posteriori-Fehlerabschätzung:

$$\|x^{(k)} - x^*\| \leq \frac{L}{1-L} \|x^{(k)} - x^{(k-1)}\|. \quad (5.12)$$

Beweis. 1) Wir zeigen, dass $(x^{(k)})_{k=0}^\infty$ eine Cauchy-Folge ist: Mit einer Teleskopsumme und der Dreiecks-Ungleichung gilt

$$\begin{aligned} \|x^{(k+m)} - x^{(k)}\| &= \|x^{(k+m)} - x^{(k+m-1)} + x^{(k+m-1)} - \dots - x^{(k+1)} + x^{(k+1)} - x^{(k)}\| \\ &\leq \|x^{(k+m)} - x^{(k+m-1)}\| + \dots + \|x^{(k+1)} - x^{(k)}\| \end{aligned}$$

sowie aufgrund der Kontraktivität:

$$\begin{aligned} \|x^{(k+1)} - x^{(k)}\| &= \|\phi(x^{(k)}) - \phi(x^{(k-1)})\| \\ &\leq L\|x^{(k)} - x^{(k-1)}\| \leq \dots \leq L^k \|x^{(1)} - x^{(0)}\|, \end{aligned} \quad (5.13)$$

also wegen $L < 1$ und der (endlichen) geometrischen Reihe

$$\begin{aligned} \|x^{(k+m)} - x^{(k)}\| &\leq (L^{k+m-1} + \dots + L^k) \|x^{(1)} - x^{(0)}\| = L^k \frac{1-L^m}{1-L} \|x^{(1)} - x^{(0)}\| \\ &\leq \frac{L^k}{1-L} \|x^{(1)} - x^{(0)}\| \xrightarrow{k \rightarrow \infty} 0. \end{aligned}$$

Also ist $(x^{(k)})_{k=0}^\infty$ eine Cauchy-Folge. Da E vollständig ist, existiert ein Grenzwert x^* , d.h. $\|x^{(k)} - x^*\| \rightarrow 0$ mit $k \rightarrow \infty$.

- 2) Zeige, dass x^* ein Fixpunkt ist: Mit der Dreiecks-Ungleichung gilt

$$\begin{aligned} \|x^* - \phi(x^*)\| &= \|x^* - x^{(k)} + x^{(k)} - \phi(x^*)\| \\ &\leq \|x^* - x^{(k)}\| + \|\phi(x^{(k-1)}) - \phi(x^*)\| \end{aligned}$$

$$\leq \underbrace{\|x^* - x^{(k)}\|}_{\xrightarrow{k \rightarrow \infty} 0} + L \underbrace{\|x^{(k-1)} - x^*\|}_{\xrightarrow{k \rightarrow \infty} 0} \xrightarrow{n \rightarrow \infty} 0,$$

also $x^* = \phi(x^*)$, da $k \in \mathbb{N}$ beliebig war.

- 3) **Eindeutigkeit:** Seien x^* und x^{**} zwei Fixpunkte, d.h. $x^* = \phi(x^*)$ und $x^{**} = \phi(x^{**})$. Dann gilt für $x^* \neq x^{**}$

$$\|\phi(x^*) - \phi(x^{**})\| \leq L\|x^* - x^{**}\| = L\|\phi(x^*) - \phi(x^{**})\| < \|\phi(x^*) - \phi(x^{**})\|,$$

da $L < 1$. Dies ist ein Widerspruch zu $x^* \neq x^{**}$, also folgt $x^* = x^{**}$.

- 4) **Fehlerabschätzung:** Für jedes $m > 0$ gilt wie in 1)

$$\begin{aligned} \|x^{(k)} - x^*\| &\leq \|x^{(k)} - x^{(k+1)}\| + \|x^{(k+1)} - x^{(k+2)}\| + \dots + \|x^{(k+m)} - x^*\| \\ &= \|\phi(x^{(k-1)}) - \phi(x^{(k)})\| + \|\phi(x^{(k)}) - \phi(x^{(k+1)})\| \\ &\quad + \dots + \|\phi(x^{(k+m-1)}) - \phi(x^*)\| \\ &\leq (L + L^2 + \dots + L^m) \|x^{(k-1)} - x^{(k)}\| + L\|x^{(k+m-1)} - x^*\| \\ &= L \left(\underbrace{\frac{1-L^m}{1-L}}_{\xrightarrow{m \rightarrow \infty} \frac{1}{1-L}} \|x^{(k-1)} - x^{(k)}\| + \underbrace{\|x^{(k+m-1)} - x^*\|}_{\xrightarrow{m \rightarrow \infty} 0} \right) \end{aligned}$$

also (d) und mit (5.13) folgt dann (c). ■

Bemerkung 5.2.2 (a) Für $X = \mathbb{R}$, $\|\cdot\| \equiv |\cdot|$ (Betrag) und $E = [a, b]$ oder $E = \mathbb{R}$ erhält man den Satz für skalare Gleichungen.

- (b) Der Satz sagt auch aus, wie viele Iterationen man machen muss, um eine gewünschte Genauigkeit zu erzielen. **Ziel:** $\|x^{(k)} - x^*\| \leq \varepsilon$ mit einer Toleranz $\varepsilon > 0$. Wegen (c) ist dies erfüllt, falls $\frac{L^k}{1-L} \|x^{(1)} - x^{(0)}\| \leq \varepsilon$, also wegen $L < 1$

$$L^{-k} \geq \frac{\|x^{(1)} - x^{(0)}\|}{\varepsilon(1-L)},$$

und damit

$$k \geq \log \left(\frac{\|x^{(1)} - x^{(0)}\|}{\varepsilon(1-L)} \right) / \log(L^{-1}). \quad (5.14)$$

- (c) Man beachte, dass der Satz insbesondere auch für unendlich-dimensionale Räume, wie z.B. Funktionenräume gilt.

Korollar 5.2.3 Sei $\phi \in C^1(\mathbb{R})$ mit $\phi : [a, b] \rightarrow [a, b]$ mit

$$\max_{x \in [a, b]} |\phi'(x)| =: L < 1, \quad (5.15)$$

ist Bedingung (ii) aus Satz 5.2.1 für $\|\cdot\| \equiv |\cdot|$ erfüllt.

Beweis. Mit dem Mittelwertsatz gilt für ein $\xi \in (a, b)$

$$|\phi(x) - \phi(y)| = |\phi'(\xi)(x - y)| \leq |x - y| \max_{\xi \in (a, b)} |\phi'(\xi)| = L|x - y|,$$

also ist ϕ eine Kontraktion. ■

Korollar 5.2.4 Sei $\Omega \subseteq \mathbb{R}^n$ abgeschlossen und konvex, $\phi : \Omega \rightarrow \mathbb{R}^n$ stetig differenzierbar mit $\phi : \Omega \rightarrow \Omega$. Falls für eine beliebige Vektornorm $\|\cdot\|$ auf \mathbb{R}^n und die zugehörige Matrixnorm für die Jacobi-Matrix ϕ' von ϕ

$$\max_{x \in \Omega} \|\phi'(x)\| = L < 1 \quad (5.16)$$

gilt, dann ist Bedingung (ii) aus Satz 5.2.1 erfüllt. ■

■ **Beispiel 5.2.5** Betrachte das System

$$6x = \cos x + 2y, \quad 8y = xy^2 + \sin y \quad (5.17)$$

auf $\Omega = [0, 1]^2$. Gesucht sind $x, y \in [0, 1]$ mit (5.17). Definiere also $\phi : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ durch

$$\phi(x, y) := \begin{pmatrix} \frac{1}{6} \cos x + \frac{1}{3}y \\ \frac{1}{8}xy^2 + \frac{1}{8} \sin y \end{pmatrix}.$$

Für diese Funktion untersuchen wir die Voraussetzungen des Banach'schen Fixpunktsatzes.

- Wegen $0 \leq \cos x \leq 1$, $0 \leq \sin y \leq 1$ für alle $x, y \in [0, 1]$ (da $1 < \frac{\pi}{2}$) gilt

$$0 \leq \frac{1}{6} \cos x + \frac{1}{3}y \leq \frac{1}{6} + \frac{1}{3} = \frac{1}{2} < 1 \quad \text{und} \quad 0 \leq \frac{1}{8}xy^2 + \frac{1}{8} \sin y \leq \frac{1}{8} + \frac{1}{8} = \frac{1}{4} < 1.$$

Also gilt $\phi : \Omega \rightarrow \Omega$, d.h. ϕ ist eine Selbstabbildung.

- Wähle $\|\cdot\|_\infty$ (die induzierte Matrixnorm ist die Zeilensummennorm). Wegen

$$\phi'(x, y) = \begin{pmatrix} -\frac{1}{6} \sin x & \frac{1}{3} \\ \frac{1}{8}y^2 & \frac{1}{4} + \frac{1}{8} \cos y \end{pmatrix}$$

gilt

$$\|\phi'\|_\infty = \max \left\{ \underbrace{\frac{1}{6}|\sin x| + \frac{1}{3}}_{\leq \frac{1}{2}}, \underbrace{\frac{1}{8}y^2 + \frac{1}{4}xy + \frac{1}{8} \cos y}_{\leq \frac{1}{2}} \right\} \leq \frac{1}{2} =: L.$$

- Wähle z.B. $(x_1^{(0)}, x_2^{(0)}) = (0, 0)$. Angenommen, wir wollten die Lösung bis auf einen Fehler von $\varepsilon = 0.1$ bestimmen: $(x_1^{(1)}, x_2^{(1)}) = \phi(x^{(0)}) = (\frac{1}{6}, 0)$

$$(x_1^{(2)}, x_2^{(2)}) = \phi(x^{(1)}) = \phi\left(\frac{1}{6}, 0\right) = \left(\frac{1}{6} \cos\left(\frac{1}{6}\right), 0\right) \doteq (0.164, 0).$$

Mit der Abschätzung (5.14) gilt mit $\varepsilon = 0.1$, $L = \frac{1}{2}$ und $\|(x^{(1)}, y_1) - (x^{(0)}, y_0)\|_\infty = \frac{1}{6}$

$$k \geq \log \left(\underbrace{\frac{\frac{1}{6}}{\frac{1}{10} \cdot \frac{1}{2}}}_{=\frac{20}{6}=\frac{10}{3}} \right) / \log(2) = \frac{\log 10 - \log 3}{\log 2} \doteq 1,74,$$

also reichen obige zwei Schritte aus, um die gewünschte Genauigkeit zu erzielen. ■

■ **Beispiel 5.2.6** Die Eigenschaften von ϕ (vor allem die Größe der Konstanten L) beeinflussen die Iteration maßgeblich. Daher ist u.U. eine Umformung sinnvoll. Wir betrachten die Funktion $f(x) := 2x - \tan x$, für die wir u.a. folgende Fixpunktfunktionen konstruieren können

$$\phi_1(x) := \frac{1}{2} \tan x, \quad \phi_2(x) := \arctan(2x).$$

Man beachte, dass ϕ_1 weder Selbstabbildung auf $\Omega = [0, \frac{\pi}{2}]$ noch Kontraktion ist! Untersuchen Sie dies für ϕ_2 als Übung. ■

Für Fixpunkt-Iterationen kann man ein einfach zu handhabendes Kriterium herleiten, um die Konvergenzordnung zu bestimmen.

Lemma 5.2.7 Sei $\phi : \mathbb{R} \rightarrow \mathbb{R}$ in einer Umgebung von $x^* = \phi(x^*)$ eine p -mal stetig differenzierbare Fixpunktfunktion und es gelte

$$\begin{aligned} |\phi'(x^*)| &< 1, & \text{falls } p = 1 \\ \phi'(x^*) = \phi''(x^*) = \dots = \phi^{(p-1)}(x^*) &= 0, & \text{falls } p \geq 2. \end{aligned}$$

Dann konvergiert $x^{(k+1)} := \phi(x^{(k)})$ *lokal* mindestens von der Ordnung p . Gilt für $p \geq 2$ zudem $\phi^{(p)}(x^*) \neq 0$, so konvergiert $x^{(k+1)} := \phi(x^{(k)})$ *lokal* genau von der Ordnung p .

Beweis. Für $p = 1$ folgt die Behauptung aus Satz 5.2.1 (Banach'scher Fixpunktsatz) und Folgerung 5.2.4. Sei im Folgenden also $p \geq 2$: Wir entwickeln $\phi(x)$ um x^* und erhalten mit einem $\xi_k \in (x^{(k)}, x^*)$

$$x^{(k+1)} = \phi(x^{(k)}) = \underbrace{\phi(x^*)}_{=x^*} + \sum_{j=1}^{p-1} \frac{(x^{(k)} - x^*)^j}{j!} \underbrace{\phi^{(j)}(x^*)}_{=0, j=1, \dots, p-1} + \frac{(x^{(k)} - x^*)^p}{(p)!} \phi^{(p)}(\xi_k),$$

also mit der Dreiecksungleichung

$$\frac{|x^{(k+1)} - x^*|}{|x^{(k)} - x^*|^p} \leq \frac{1}{p!} \underbrace{|\phi^{(p)}(\xi_k)|}_{\leq c},$$

weshalb $x^{(k+1)} := \phi(x^{(k)})$ *lokal* mindestens von der Ordnung p konvergiert. ■

Bemerkung 5.2.8 Lokale Konvergenz bedeutet, dass eine Konvergenz nur für jene Startwerte $x^{(0)} \in \mathbb{R}^n$ vorliegt, die **nahe genug** bei x^* liegen!

5.2.2 Das Newton-Verfahren für Systeme

Offenbar können wir die Newton-Iteration für skalare Gleichungen (5.8) nicht unmittelbar für Systeme übernehmen, weil $f'(x) = \nabla f(x) =: J_f(x)$ die *Jacobi-Matrix* $f = (f_1, \dots, f_n)^T : \mathbb{R}^n \rightarrow \mathbb{R}^n$ an der Stelle $x \in \mathbb{R}^n$ ist, d.h.

$$J_f(x) := \nabla f(x) = \begin{pmatrix} \nabla f_1 \\ \vdots \\ \nabla f_n \end{pmatrix} := \begin{pmatrix} \frac{\partial f_1}{\partial x^{(1)}} & \dots & \frac{\partial f_1}{\partial x_n} \\ \vdots & & \vdots \\ \frac{\partial f_n}{\partial x^{(1)}} & \dots & \frac{\partial f_n}{\partial x_n} \end{pmatrix} \in \mathbb{R}^{n \times n}.$$

Wir hatten ja das Newton-Verfahren dadurch erhalten, dass wir die Funktion f nahe $x^{(k)}$ durch die Tangente an f im Punkt $x^{(k)}$ ersetzt haben. Die Tangente ist nichts anderes als das *Taylor-Polynom erster Ordnung*. Also betrachten wir nun die Taylor-Entwicklung einer Funktion $f : \mathbb{R}^n \rightarrow \mathbb{R}^n$ (vgl. Satz B.2.1) in x um einen Punkt $x^{(k)}$

$$f(x) = \underbrace{f(x^{(k)}) + J_f(x^{(k)})(x - x^{(k)})}_{=: P_1[f](x)} + o(\|x - x^{(k)}\|) \quad \text{für } x \rightarrow x^{(k)}.$$

Die Nullstelle $x^{(k+1)}$ des linearen Taylor-Polynoms $P_1[f]$ ist gegeben durch

$$x^{(k+1)} = x^{(k)} - J_f(x^{(k)})^{-1} f(x^{(k)}),$$

falls $\det(J_f(x^{(k)})) \neq 0$, also die Jacobi-Matrix J_f im Punkt $x^{(k)}$ regulär ist. Daraus erhalten wir schon das Verfahren.

Newton-Iteration: Für einen Startwert $x^{(0)} \in \mathbb{R}^n$ berechne $x^{(k)}$, $k = 1, \dots$ durch

$$J_f(x^{(k)}) s^{(k)} = -f(x^{(k)}) \quad \text{Newton-Update,} \quad (5.18a)$$

$$x^{(k+1)} = x^{(k)} + s^{(k)} \quad \text{Newton-Korrektur.} \quad (5.18b)$$

Wir zeigen eine MATLAB®-Implementierung:

MATLAB-Funktion: NewtonSimple.m

```

1 function [x,nit] = NewtonSimple(x,f,Df,tol,maxit,param)
2 nit = 0;
3 fx = f(x,param);
4 while norm(fx) > tol && nit <= maxit
5     nit = nit+1;
6     x = x-Df(x,param)\fx;
7     fx = f(x,param);
8 end
9 nit
```

Bemerkung 5.2.9 Offenbar bedeutet die Bestimmung des Newton-Updates in (5.18a) die Lösung eines linearen Gleichungssystems mit Matrix $J_f(x^{(k)})$ und rechter Seite $-f(x^{(k)})$. Wir haben also die numerische Lösung eines **nichtlinearen** Gleichungssystems auf die numerische Lösung einer **Folge von linearen Gleichungssystemen** reduziert.

Bemerkung 5.2.10 — Invarianz. Sei $A \in \mathbb{R}^{n \times n}$ eine reguläre Matrix. Dann ist das Nullstellenproblem $f(x) = 0$ äquivalent zu $g(x) := Af(x) = 0$. Betrachten wir nun die Berechnung des Newton-Updates angewendet auf die Funktion g . Sei $s^{(k)}$ das Newton-Update zu f , also $J_f(x^{(k)})s^{(k)} = -f(x^{(k)})$, dann gilt

$$\begin{aligned} -J_g(x^{(k)})^{-1}g(x^{(k)}) &= -(AJ_f(x^{(k)}))^{-1}Af(x^{(k)}) = -J_f^{-1}(x^{(k)})A^{-1}Af(x^{(k)}) \\ &= -J_f^{-1}(x^{(k)})f(x^{(k)}) = s^{(k)}, \end{aligned}$$

also das Newton-Update zu f . Damit ist sowohl das ursprüngliche Problem $f(x) = 0$ als auch das Newton-Verfahren **affin-invariant**. Diese Eigenschaft kann z.B. zur Vorkonditionierung verwendet werden.

Satz 5.2.11 Sei $D \subset \mathbb{R}^n$ offen und konvex, $f : D \rightarrow \mathbb{R}^n$, $f \in C^1(D, \mathbb{R}^n)$ eine Funktion, deren Jacobi-Matrix $J_f(x) =: J$ für alle $x \in D$ regulär ist. Es gelte ferner für ein $L_J \geq 0$ die folgende Lipschitz-Bedingung:

$$\|J^{-1}(x)(J(x+sv) - J(x))v\| \leq s L_J \|v\|^2$$

für alle $s \in [0, 1]$, $x \in D$ und $v \in \mathbb{R}^n$ mit $x+v \in D$. Weiterhin existiere eine Lösung $x^* \in D$ und ein Startwert $x^{(0)} \in D$ derart, dass

$$\varrho := \|x^* - x^{(0)}\| < \frac{2}{L_J} \quad \text{und} \quad \mathcal{U}_\varrho(x^*) \subseteq D.^8$$

Dann gilt für die durch das Newton-Verfahren definierte Folge, dass $(x^{(k)})_{k \in \mathbb{N}} \subset \mathcal{U}_\varrho(x^*)$ und sie konvergiert gegen x^* , d.h.

$$\|x^{(k)} - x^*\| < \varrho \quad \text{für } k > 0 \quad \text{mit} \quad \lim_{k \rightarrow \infty} x^{(k)} = x^*.$$

Der Grenzwert x^* ist die eindeutige Nullstelle von f in $\mathcal{U}_{2/L_J}(x^*)$ und es gilt

$$\|x^{(k+1)} - x^*\| \leq \frac{L_J}{2} \|x^{(k)} - x^*\|^2 \quad \text{für } k = 0, 1, 2, \dots,$$

d.h. das Verfahren konvergiert lokal mindestens quadratisch.

Beweis. 1. Zunächst zeigen wir, dass

$$\|J^{-1}(x)(f(y) - f(x) - J(x)(y-x))\| \leq \frac{L_J}{2} \|y-x\|^2 \quad (5.19)$$

⁸ $\mathcal{U}_r(x) := \{y \in \mathbb{R}^n : \|x-y\| \leq r\}$ bezeichnet die Umgebung mit Radius $r > 0$ um $x \in \mathbb{R}^n$.

für alle $x, y \in D$ gilt: Wegen $\frac{d}{ds} f_j(x + s(y-x)) = \nabla f_j(x + s(y-x)) \cdot (y-x)$ gilt mit dem Hauptsatz der Differential- und Integralrechnung

$$\begin{aligned} \int_0^1 (J(x + s(y-x)) - J(x))(y-x) ds &= \int_0^1 (f'(x + s(y-x)) - f'(x))(y-x) ds \\ &= \int_0^1 f'(x + s(y-x)) (y-x) ds - f'(x) (y-x) \\ &= \int_0^1 \frac{d}{ds} f(x + s(y-x)) ds - f'(x) (y-x) = f(y) - f(x) - J(x)(y-x). \end{aligned}$$

Da $J^{-1}(x)$ regulär ist, erhalten wir für die linke Seite von (5.19)

$$\begin{aligned} &\|J^{-1}(x)(f(y) - f(x) - J(x)(y-x))\| \\ &= \left\| \int_0^1 J^{-1}(x)(J(x + s(y-x)) - J(x))(y-x) ds \right\| \\ &\leq \int_0^1 \|J^{-1}(x)(J(x + s(y-x)) - J(x))(y-x)\| ds \\ &\leq \int_0^1 s L_J \|y-x\|^2 ds = \frac{L_J}{2} \|y-x\|^2. \end{aligned}$$

2. Nun erhalten wir mit der Iterationsvorschrift $x^{(k+1)} = x^{(k)} - J^{-1}(x^{(k)})f(x^{(k)})$ und $f(x^*) = 0$, dass

$$\begin{aligned} x^{(k+1)} - x^* &= x^{(k)} - J^{-1}(x^{(k)})f(x^{(k)}) - x^* \\ &= x^{(k)} - x^* - J^{-1}(x^{(k)})(f(x^{(k)}) - f(x^*)) \\ &= J^{-1}(x^{(k)})[f(x^*) - f(x^{(k)}) - J(x^{(k)})(x^* - x^{(k)})]. \end{aligned}$$

Mit (5.19) erhalten wir damit

$$\|x^{(k+1)} - x^*\| \leq \frac{L_J}{2} \|x^{(k)} - x^*\|^2.$$

Da $\|x^{(0)} - x^*\| = \varrho$, folgt daraus, dass

$$\|x^{(1)} - x^*\| \leq \underbrace{\frac{L_J}{2} \|x^{(0)} - x^*\|}_{= \frac{L_J \varrho}{2} =: \alpha < 1} \|x^{(0)} - x^*\| < \|x^{(0)} - x^*\|$$

und somit induktiv für $k > 0$

$$\|x^{(k)} - x^*\| \leq \underbrace{\frac{L_J}{2} \|x^{(k-1)} - x^*\|}_{\leq \frac{L_J \varrho}{2} = \alpha < 1} \|x^{(k-1)} - x^*\| \leq \alpha^k \|x^{(0)} - x^*\| < \|x^{(0)} - x^*\|.$$

Daraus folgt $\|x^{(k)} - x^*\| < \varrho$ für alle $k > 0$ und die Konvergenz der Folge $(x^{(k)})_{k \in \mathbb{N}}$ gegen x^* .

3. Zum Beweis der Eindeutigkeit in der Umgebung $\mathcal{U}_{2/L_J}(x^*)$ um x^* mit Radius $2/L_J$ benutzen wir nochmals (5.19): Sei $x^{**} \in \mathcal{U}_{2/L_J}(x^*)$ eine weitere Lösung, also gilt

$f(x^{**}) = 0$ und $\|x^* - x^{**}\| < 2/L_J$. Einsetzen in (5.19) liefert

$$\|x^{**} - x^*\| = \left\| J^{-1}(x^*) \left(0 - 0 - J^{-1}(x^*)(x^{**} - x^*) \right) \right\| \leq \underbrace{\frac{L_J}{2}}_{<1} \|x^{**} - x^*\| \|x^{**} - x^*\|,$$

dies ist aber nur möglich, falls $x^{**} = x^*$.

Damit ist alles bewiesen. ■

Bemerkung 5.2.12 — Merkregel. Das *Newton*-Verfahren konvergiert lokal quadratisch.

■ **Beispiel 5.2.13 — Extremalstellen der Rosenbrock-Funktion.** Wir betrachten noch einmal die Rosenbrock-Funktion aus Beispiel 5.0.4, also $f(x) = \sum_{i=1}^{n-1} [(1 - x_i)^2 + 100(x_{i+1} - x_i^2)^2]$. Die Jacobi-Matrix $A := A(x) := J_f(x)$ enthält folgende Einträge

$$\begin{aligned} a_{11} &= 2 + 1200x_1^2 - 400x_2 \\ a_{jj} &= 202 + 1200x_j^2 - 400x_{j+1} \quad (j = 2, \dots, n-1) \\ a_{nn} &= 200 \\ a_{j,j+1} &= a_{j+1,j} = -400x_j \quad (j = 1, \dots, n-1) \\ a_{jk} &= 0 \quad (|j - k| \geq 2). \end{aligned}$$

Die Funktionsauswertung und die Jacobi-Matrix sind in den folgenden MATLAB®-Files umgesetzt. Danach zeigen wir die Anwendung des Newton-Verfahrens. ■

MATLAB-Funktion: rosenbrock.m

```
1 function value = rosenbrock(x,param)
2 n = size(x,1);
3 value(2:n,1) = 200 * (x(2:end)-x(1:end-1).^2);
4 value(1:n-1,1) = value(1:n-1,1) - 2 * (1-x(1:end-1)) ...
5                 - 400*x(1:end-1).*(x(2:end)-x(1:end-1).^2);
```

MATLAB-Funktion: D_rosenbrock.m

```
1 function D = D_rosenbrock(x,param)
2 n = size(x,1);
3 d0(2:n,1) = 200;
4 d0(1:n-1) = d0(1:n-1) + 2 + 1200*x(1:n-1).^2 - 400*x(2:n);
5 dm1 = -400*x;
6 dp1 = -400*x([1,1:n-1]);
7 D = spdiags([dm1,d0,dp1],[-1 0 1],n,n);
```

MATLAB-Beispiel:

Man kann (mit einem geeigneten Startwert) neben dem globalen Minimum $(x_1, \dots, x_n) = (1, \dots, 1)$ mit dem Newton-Verfahren ein weiteres lokales Minimum in der Umgebung von $(x_1, x_2, \dots, x_n) = (-1, 1, \dots, 1)$ von f in Bsp. 5.0.4 finden.

```
>> n = 6;
>> x0 = [-1; ones(n-1,1)];
>> [x,nit] = NewtonSimple(x0,@rosenbrock,
    @D_rosenbrock,1e-10,100,[])
x =
-0.98657497957099
 0.98339822883618
 0.97210667005309
 0.94743743682644
 0.89865118485173
 0.80757395203542
nit =
5
```

Konvergenztest

Das Newton-Verfahren konvergiert „nur“ lokal. Bei Wahl eines Startwertes wissen wir also nicht, ob das Verfahren konvergiert oder nicht. Wünschenswert wäre ein Konvergenztest während der Iteration, der anzeigt, ob das Verfahren konvergiert und – bei negativem Resultat – dazu führt, dass man mit einem neuen Startwert erneut startet.

Als Näherung an den Fehler $\|x - x^{(k)}\|$ verwenden wir den Term

$$\|J^{-1}(x^{(k)})f(x^{(k)})\| \quad (= \|J^{-1}(x^{(k)})(f(x) - f(x^{(k)}))\|).$$

Da man erwartet (bzw. hofft), dass der Fehler monoton fällt, d.h. $\|x - x^{(k+1)}\| \leq \|x - x^{(k)}\|$ gilt, testet man dieses für jedes k durch den **natürlichen Monotonietest**, d.h. man testet, ob es ein $\bar{\theta} < 1$ gibt mit

$$\|J^{-1}(x^{(k)})f(x^{(k+1)})\| \leq \bar{\theta} \|J^{-1}(x^{(k)})f(x^{(k)})\|. \quad (5.20)$$

Dies kann man sehr effizient umsetzen: Im Newton-Verfahren berechnen wir neben $x^{(k)}$ in jedem Schritt die Größen

$$J(x^{(k)})s^{(k)} = -f(x^{(k)}) \quad \text{und} \quad x^{(k+1)} = x^{(k)} + s^{(k)}.$$

Somit ist der Ausdruck $J^{-1}(x^{(k)})f(x^{(k)})$ auf der rechten Seite in (5.20) gleich dem Negativen der Newton-Korrektur $s^{(k)}$, die sowieso berechnet wird. Zusätzlich muss nur $\bar{s}^{(k)}$ durch Lösung eines linearen Gleichungssystems

$$J(x^{(k)})\bar{s}^{(k)} = f(x^{(k+1)}),$$

bestimmt werden. Hat man z.B. von $J(x^{(k)})$ bereits eine LR- oder QR-Zerlegung bestimmt (vgl. *Numerische Lineare Algebra*), so kann man den Zusatzaufwand vernachlässigen.

Theoretische Untersuchungen und numerische Experimente liefern $\bar{\theta} = 1/2$ als eine gute Wahl. Falls der **natürliche Monotonietest**, d.h.

$$\|\bar{s}^{(k)}\| \leq \frac{1}{2} \|s^{(k)}\|$$

für ein k verletzt ist, so ist das Newton-Verfahren abubrechen und es bleibt nichts Anderes

übrig, als einen (hoffentlich) besseren Startwert zu finden.

5.2.3 Das gedämpfte Newton-Verfahren

Ein weitere Strategie, um der lediglich lokalen Konvergenz zu begegnen, ist die Idee, zunächst ein langsames, aber sicher konvergentes Verfahren zu verwenden, bis der Konvergenztest anzeigt, dass man im Konvergenzradius des Newton-Verfahrens ist. Eine Strategie dazu ist das *gedämpfte Newton-Verfahren*. Für $n = 1$ ersetzt man die Tangente durch eine Gerade mit geringerer Steigung und hofft so, die Konvergenz zu „retten“, auf Kosten einer geringeren Konvergenzrate.

Man wählt häufig eine Dämpfung in der Form

$$x^{(k+1)} = x^{(k)} - \lambda_k s^{(k)} \quad \text{mit einem } \lambda_k \in (0, 1].$$

Für eine einfache Dämpfungsstrategie können wir den Dämpfungsparameter λ_k derart wählen, so dass der natürliche Monotonietest für $\bar{\theta} = 1 - \lambda_k/2$ erfüllt ist, d.h.

$$\|F'(x^{(k)})^{-1}F(x^{(k)} + \lambda_k s^{(k)})\| \leq \left(1 - \frac{\lambda_k}{2}\right) \|F'(x^{(k)})^{-1}F(x^{(k)})\|.$$

Dabei wählen wir λ_k aus einer endlichen Folge $\{1, \frac{1}{2}, \frac{1}{4}, \dots, \lambda_{\min}\}$ und brechen ggf. das Verfahren ab, falls $\lambda_k < \lambda_{\min}$ mit einer zuvor gewählten minimalen Dämpfung λ_{\min} notwendig wäre.

War λ_k erfolgreich, so zeigt die Praxis, dass es effizienter ist, mit $\lambda_{k+1} = \min\{1, 2\lambda_k\}$ fortzufahren anstatt wieder mit $\lambda_{k+1} = 1$ anzufangen. War der Monotonietest mit λ_k verletzt, so testet man erneut mit $\lambda_k/2$.

Das folgende MATLAB[®]-Programm zeigt eine entsprechende Umsetzung.

MATLAB-Funktion: Newton.m

```

1 function [u,nit] = newton(u,F,DF,tol,maxit,param)
2 Fu = F(u,param);
3 DFu = DF(u,param);
4 s = -DFu\Fu;
5 lam = 1;
6 tmp = max(tol,tol*norm(s));
7 nit = 0;
8 while norm(s) > tmp && nit <= maxit
9     nit = nit + 1;
10    u_old = u;
11    lam = min(1,2*lam);
12    for k=1:30
13        u = u_old + lam * s; % Daempfung mit Parameter lam
14        Fu = F(u,param);
15        if norm(DFu\Fu) <= (1-lam/2) * norm(s)
16            break           % Abbruch der for-Schleife, falls
17        end                 % Konvergenztest erfuehlt
18        lam = lam/2;        % lam noch zu gross--> halbieren
19    end
20    DFu = DF(u,param);
21    s = -DFu\Fu;
22 end

```

5.2.4 Das Broyden-Verfahren

Das Newton-Verfahren hat aufgrund seiner quadratischen Konvergenz große Bedeutung, besitzt aber auch einige Nachteile. Die Tatsache, dass die Konvergenz nur lokal ist, also vom Startwert abhängt, hatten wir bereits beschrieben. Einer weiterer Nachteil ist, dass in jeder Iteration die Jacobi-Matrix $J_f(x^{(k)})$ benötigt wird, also ggf. neu zu berechnen ist. Dies ist zum Einen aufwändig, zum Anderen sind in vielen Beispielen die analytischen Ableitungen (also Formeln für die Ableitungen) nicht bekannt. Wenn man die exakte Jacobi-Matrix durch eine Approximation ersetzt, spricht man von einem **Quasi-Newton-Verfahren**.

Das auf Broyden⁹ zurückgehende Quasi-Newton-Verfahren ist ein Kompromiss zwischen Neuberechnung der Jacobi-Matrix in jedem Iterationsschritt und der Verwendung einer festen Matrix (z.B. $J_f(x^{(0)})$ oder einer Approximation hiervon) im Lauf der gesamten Iteration. Ausgehend von einer Näherung an die Jacobi-Matrix $J_f(x^{(0)})$ am Startwert werden die Matrizen in jedem Schritt so aufdaetiert, dass sie möglichst ähnliche Abbildungseigenschaften aufweisen wie die exakte Jacobi-Matrix. Gleichzeitig achtet man darauf, dass diese Aktualisierung möglichst wenig zusätzlichen Rechenaufwand erfordert. Sie erfolgt daher mittels **Rang-1-Korrekturmatrizen**, die sich aus Termen berechnen lassen, die ohnehin während der Iteration bestimmt werden.

Die Näherungen an die Jacobi-Matrizen seien mit B_k bezeichnet. Dann wird im k -ten Schritt des Broyden-Verfahrens das lineare Gleichungssystem

$$B_k s^{(k)} = -f(x^{(k)}) \quad (5.21)$$

gelöst und dann $x^{(k+1)} = x^{(k)} + s^{(k)}$ gesetzt.

Im Eindimensionalen liefert das Sekantenverfahren (§5.1.3) zur Lösung des Nullstellenproblems $f(x) = 0$ zu zwei gegebenen Werten $x^{(-1)}$ und $x^{(0)}$ iterativ die Steigung der Sekante durch

$$\beta_k = \frac{f(x^{(k)}) - f(x^{(k-1)})}{x^{(k)} - x^{(k-1)}} \quad (k \geq 0) \quad (5.22)$$

und daraus eine weitere Näherung $x^{(k+1)}$ an x mittels

$$x^{(k+1)} = x^{(k)} - \beta_k^{-1} f(x^{(k)}).$$

Die formale Verallgemeinerung der Sekanten-Bedingung (5.22) auf Matrizen B_k lautet

$$B_k s^{(k-1)} = f(x^{(k)}) - f(x^{(k-1)}) =: \delta f_k.$$

Die Bedingung genügt jedoch nicht, um $B_k \in \mathbb{R}^{n \times n}$ eindeutig zu bestimmen. Daher versucht man für $k \geq m$ die letzten gewonnenen Informationen zu nutzen, so dass B_k ($k \geq m$) eine Lösung der folgenden Menge von m Systemen

$$B_k (x^{(k)} - x^{(k-j)}) = f(x^{(k)}) - f(x^{(k-j)}) \quad (j = 1, \dots, m) \quad (5.23)$$

⁹Charles George Broyden, 1933-2011.

ist. Da im Allgemeinen die Vektoren $x^{(k-j)}, \dots, x^{(k)}$ nicht linear unabhängig sind, fordern wir zusätzlich, dass die Differenz zwischen den linearen Approximationen von $f(x^{(k-1)})$ und $f(x^{(k)})$, nämlich

$$d^{(k)} := f(x^{(k)}) + B_k(x - x^{(k)}) - \left(f(x^{(k-1)}) + B_{k-1}(x - x^{(k-1)}) \right), \quad (5.24)$$

in der Euklidischen Norm minimiert wird. Setzen wir $j = 1$ in (5.23) so wird (5.24) zu

$$d^{(k)} = (B_k - B_{k-1})(x - x^{(k-1)}). \quad (5.25)$$

Zerlegt man den Vektor $x - x^{(k-1)}$ in der Form $x - x^{(k-1)} = \alpha s^{(k-1)} + r$ mit $\alpha \in \mathbb{R}$ und $\langle r, s^{(k-1)} \rangle = r^T s^{(k-1)} = 0$, dann erhält man aus (5.25)

$$d^{(k)} = \alpha(B_k - B_{k-1})s^{(k-1)} + (B_k - B_{k-1})r. \quad (5.26)$$

Da $(B_k - B_{k-1})s^{(k-1)} = \delta f_k - B_{k-1}s^{(k-1)}$ gilt, ist somit der erste Term in (5.26) unabhängig von B_k und es bleibt nur der zweite Term in (5.26) zu minimieren.

Die Matrix B_k , die $(B_k - B_{k-1})r$ minimiert für alle r , die orthogonal zu $s^{(k-1)}$ sind, unter der Restriktion, dass (5.23) gilt, kann rekursiv mittels des Rang-1-Updates von B_{k-1} , d.h.

$$B_k = B_{k-1} + \frac{(\delta f_k - B_{k-1}s^{(k-1)})(s^{(k-1)})^T}{(s^{(k-1)})^T s^{(k-1)}} \quad (5.27)$$

berechnet werden. Die Methode (5.21) mit der Wahl (5.27) wird als **Broyden-Verfahren** bezeichnet. Zur Initialisierung setzt man $B_0 = J_f(x^{(0)})$ oder eine geeignete Approximation z.B. mittels Differenzenquotienten, d.h. $(B_0)_{ij} \approx (f_i(x^{(0)} + h e_j) - f_i(x^{(0)}))/h$.

Bemerkung 5.2.14 Ist eine QR -Zerlegung¹⁰ von $B_0 \in \mathbb{R}^{n \times n}$ gegeben, so lässt sich die QR -Zerlegung von B_k ($k > 0$) aus der QR -Zerlegung von B_{k-1} mit $\mathcal{O}(n^2)$ bestimmen.

MATLAB-Funktion: Broyden.m

```

1 function [x,nit] = Broyden(x,f,B,tol,maxit,param)
2 fx = f(x,param);
3 fx1 = zeros(size(fx));
4 nit = 0; err = inf;
5 while nit < maxit && err > tol
6     s = - B \ fx;
7     x = x + s;
8     err = norm(s);
9     if err > tol
10         fx1 = f(x,param);
11         B = B + 1 / (s' * s) * fx1 * s';
12     end
13     fx = fx1;
14     nit = nit + 1;
15 end

```

Unter Verwendung des Broyden-Verfahrens lösen wir das nichtlineare Problem aus

¹⁰vgl. Vorlesung *Numerische Lineare Algebra*.

Beispiel 5.0.4 für $n = 6$. Diese Methode konvergiert in 18 Iterationen verglichen mit den 5 Iterationen, die das Newton-Verfahren erforderte bei gleichem Startwert $x^{(0)} = (-1, 1, \dots, 1)^T$. Die Matrix B_0 wurde gleich der Jacobi-Matrix im Punkt $x^{(0)}$ gesetzt. Abbildung 5.11 zeigt das Verhalten der Euklidischen Norm des Fehlers beider Methoden.

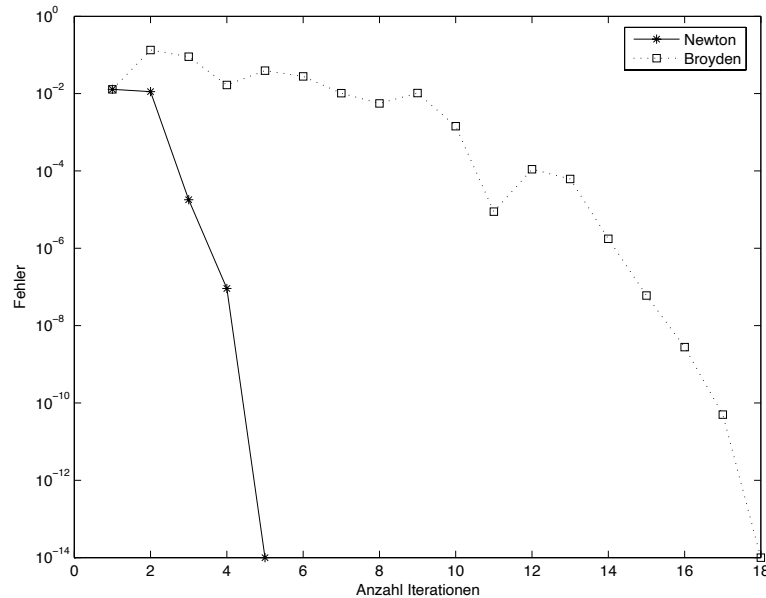


Abbildung 5.11: Euklidische Norm des Fehlers für das Broyden- und Newton-Verfahren im Fall des nichtlinearen Problems aus Bsp. 5.0.4 für $n = 6$.

Bemerkung 5.2.15 Wenn keine analytischen Formeln für die Ableitungen bekannt sind, sind folgende Vorgehensweisen verbreitet:

- Die Ableitungen werden durch Differenzenquotienten ersetzt, also z.B. für $n = 1$ mittels Vorwärtsdifferenz $f'(x) \approx \frac{1}{h}(f(x+h) - f(x))$. Nun muss aber einerseits h klein wählen, damit der Differenzenquotient eine gute Approximation an die Ableitung ist, andererseits ist dann aber $f(x+h) \approx f(x)$ für „vernünftige“ Funktionen f , so dass dann Stabilitätsprobleme und Auslöschung auftreten können.
- Man approximiert die Ableitung durch *Numerische Differentiation* analog zur numerischen Integration in Kapitel 8. Dazu approximiert man f durch ein Interpolationspolynom und berechnet von diesem die Ableitung exakt.
- In vielen Fällen ist die Funktion f durch ein (MATLAB®-)Programm gegeben. Man kennt ggf. den Funktionsterm nicht, hat aber eine Implementierung der Abbildung $x \mapsto f(x)$. Wenn dem so ist, dann können Methoden des *Automatischen* oder *Algorithmischen Differenzierens* ein (MATLAB®-)Programm erzeugen, welches die Abbildung $x \mapsto f'(x)$ realisiert. Dies ist *keine* Approximation. Ein Spezialfall dieser Methode ist unter dem Namen *Backtracking* (oder *Backpropagation*) beim Training von Neuronalen Netzen von enormer Bedeutung. Mehr dazu in der Vorlesung *Numerical Methods for Data Science*.

Kapitel 6

Interpolation und Approximation

Häufig sind z.B. durch Datenerhebungen oder technische Messungen (Daten-)Punkte bekannt. Diese sollen durch eine „einfache“ Funktion (z.B. durch ein Polynom) so angenähert werden, dass eine effiziente Auswertung schnell möglich ist. Eine andere Fragestellung (die aber auf die gleichen numerischen Methoden hinausläuft) ist, eine „komplexe“ Funktion durch „einfache“ Funktionen anzunähern (zu approximieren), indem man die komplexe Funktion nur an wenigen Stellen auswertet. Letzteres tritt z.B. dann auf, wenn Ergebnisse komplexer numerischer Simulationen auszuwerten sind oder auch wenn aus einer riesigen Datenflut relevante Informationen zu extrahieren sind (das so genannte *Compressed Sensing* bei *Big Data*).

6.1 Klassische Polynom-Interpolation

Wir beginnen mit der Interpolation von gegebenen Daten mittels algebraischer Polynome.

Gegeben seien $(n + 1)$ paarweise verschiedene **Stützstellen** $x_0, \dots, x_n \in \mathbb{R}$ und dazugehörige beliebige **Stützwerte** $f_0, \dots, f_n \in \mathbb{R}$. Gesucht ist ein Polynom $P \in \mathbb{P}_n$ vom Grad $\text{Grad } P \leq n$, also

$$P(x) = a_n x^n + \dots + a_1 x + a_0 \quad \text{mit } a_k \in \mathbb{R}, \quad k = 0, \dots, n,$$

welches die **Interpolationsbedingungen** erfüllt, d.h.

$$P(x_i) = f_i, \quad i = 0, \dots, n. \tag{6.1}$$

Zunächst zur Frage der Existenz und Eindeutigkeit eines solchen Polynoms.

Satz 6.1.1 — Existenz und Eindeutigkeit der Polynominterpolation. Zu beliebigen $(n + 1)$ Stützstellen (x_i, f_i) , $i = 0, \dots, n$ mit paarweise verschiedenen Stützstellen x_0, \dots, x_n existiert genau ein Polynom $P \in \mathbb{P}_n$, das (6.1) erfüllt.

Beweis. **Existenz:** Wir geben einen konstruktiven Beweis. Dazu betrachten wir die **La-**

grange-Polynome¹

$$L_{n,i}(x) := \prod_{\substack{j=0 \\ j \neq i}}^n \frac{x - x_j}{x_i - x_j}, \quad i = 0, \dots, n. \quad (6.2)$$

Diese Polynome sind vom echten Grad n und besitzen die sogenannte *Interpolationseigenschaft*

$$L_{n,i}(x_k) = \delta_{ik} = \begin{cases} 1, & \text{für } i = k, \\ 0, & \text{für } i \neq k. \end{cases} \quad (6.3)$$

Demzufolge erfüllt das Polynom

$$P(x) := \sum_{i=0}^n f_i L_{n,i}(x) \in \mathbb{P}_n \quad (6.4)$$

die geforderten Interpolationseigenschaften (6.1), denn wegen (6.3) gilt:

$$P(x_k) = \sum_{i=0}^n f_i L_{n,i}(x_k) = \sum_{i=0}^n f_i \delta_{ik} = f_k, \quad k = 0, 1, \dots, n.$$

Eindeutigkeit: Es seien $P, Q \in \mathbb{P}_n$ mit

$$P(x_k) = Q(x_k) = f_k \text{ für alle } k = 0, 1, \dots, n. \quad (6.5)$$

Für $D := P - Q \in \mathbb{P}_n$ gilt $D(x_i) = 0$ für alle $i = 0, \dots, n$ nach (6.5). Also hat $D \in \mathbb{P}_n$ mindestens $n + 1$ verschiedene Nullstellen. Nach dem Fundamentalsatz der Algebra muss nun aber $D \equiv 0$ gelten, also $P = Q$. ■

Definition 6.1.2 Das nach Satz 6.1.1 eindeutige $P \in \mathbb{P}_n$ heißt **Interpolationspolynom**; es wird bezeichnet mit

$$P = P(f|x_0, \dots, x_n).$$

Man sagt, dass P das „Interpolationspolynom an f zu den Stützstellen x_0, \dots, x_n “ ist.

Bemerkung 6.1.3 Der obige konstruktive Beweis legt bereits ein erstes numerisches Verfahren nahe, in dem man die Lagrange-Polynome (6.2) bestimmt. Allerdings ist die **Lagrange-Darstellung** (6.4) für numerische Zwecke meist zu rechenaufwendig und instabil (wir begründen das später), sie eignet sich jedoch oft sehr gut für theoretische Fragestellungen.

Offenbar haben wir in dem obigen Beweis auch gezeigt, dass $\mathbb{P}_n = \text{span}\{L_{n,0}, \dots, L_{n,n}\}$, die sogenannte **Lagrange-Basis**. Offenbar besitzt der Polynomraum \mathbb{P}_n zahlreiche Basen, die „einfachste“ ist die **monomiale Basis**, also $\mathbb{P}_n = \text{span}\{1, x, \dots, x^n\}$, d.h., $P(x) = a_0 + a_1x + \dots + a_nx^n$. In der monomialen Basis kann man der Interpolationsproblem leicht als lineares Gleichungssystem ausdrücken.

¹Joseph-Louis Lagrange, 1736-1813.

Bemerkung 6.1.4 — Vandermonde-Matrix. Die Interpolationsbedingungen $P(x_i) = f_i$, $i = 0, \dots, n$, lassen sich als lineares Gleichungssystem $V_n \mathbf{a}_n = \mathbf{f}_n$ auffassen:

$$\underbrace{\begin{pmatrix} 1 & x_0 & x_0^2 & \dots & x_0^n \\ 1 & x_1 & x_1^2 & \dots & x_1^n \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_n & x_n^2 & \dots & x_n^n \end{pmatrix}}_{=:V_n} \underbrace{\begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{pmatrix}}_{=: \mathbf{a}_n} = \underbrace{\begin{pmatrix} f_0 \\ f_1 \\ \vdots \\ f_n \end{pmatrix}}_{=: \mathbf{f}_n}.$$

Dabei wird V_n **Vandermonde-Matrix** genannt. Man kann einen Algorithmus konstruieren, der dieses LGS in $\mathcal{O}(n^2)$ löst (vgl. Vorlesung *Numerische Lineare Algebra*), was jedoch nicht optimal ist. Darüberhinaus ist die Kondition $\kappa(V_n)$ sehr schlecht, wenn einzelne Stützstellen nahe beieinander liegen, so dass wir mit enormer Fehlerverstärkung rechnen müssen.

Zur späteren Verwendung (im Rahmen von Extrapolationsverfahren zur numerischen Integration) notieren wir folgendes Resultat.

Lemma 6.1.5 Für die Lagrange-Funktionen $L_{n,i}$ zu Stützstellen x_0, \dots, x_n gilt

$$\sum_{j=0}^n L_{j,n}(0) x_j^m = \begin{cases} 1, & \text{für } m = 0, \\ 0, & \text{für } 1 \leq m \leq n, \\ (-1)^n x_0 \cdots x_n, & \text{für } m = n+1. \end{cases} \quad (6.6)$$

Beweis. Übung ■

6.2 Das Neville-Aitken-Verfahren

Ist man nur an der **Auswertung des Interpolationspolynoms P an wenigen Stellen** interessiert, so muss man dazu nicht erst das komplette Polynom P bestimmen und dieses dann auswerten. Für eine gegebene Stelle $x \in \mathbb{R}$ ist eine rekursive Berechnung von $P(x)$ deutlich effizienter. Dies beruht auf folgender Beobachtung.

Lemma 6.2.1 — von Aitken². Für $P = P(f|x_0, \dots, x_n)$ gilt die Rekursionsformel

$$P(f|x_0, \dots, x_n)(x) = \frac{(x_0 - x)P(f|x_1, \dots, x_n)(x) - (x_n - x)P(f|x_0, \dots, x_{n-1})(x)}{x_0 - x_n}, \quad (6.7)$$

mit $P(f|x_k) := f(x_k)$.

Beweis. Sei $\phi(x)$ definiert als der Term auf der rechten Seite von (6.7). Dann ist $\phi \in \mathbb{P}_n$ und es gilt:

$$\phi(x_i) = \frac{(x_0 - x_i)f(x_i) - (x_n - x_i)f(x_i)}{x_0 - x_n} = f(x_i), \quad i = 1, \dots, n-1.$$

Ebenso leicht folgt $\phi(x_0) = f(x_0)$ sowie $\phi(x_n) = f(x_n)$ und damit die Behauptung. ■

²Alexander Craig Aitken, 1895-1967.

Wir verwenden nun die obige Rekursionsformel zur Konstruktion des **Aitken-Neville-Algorithmus**³. Dazu verwenden wir die Abkürzung $f_i := f(x_i)$, $i = 0, \dots, n$. Für festes $x \in \mathbb{R}$ vereinfachen wir die Notation weiterhin durch

$$P_{ik} := P(f|x_{i-k}, \dots, x_i)(x), \quad i, k = 0, \dots, n \text{ mit } i \geq k.$$

Die Rekursion (6.7) können wir nun so ausdrücken

$$P_{nn} = \frac{(x_0 - x)P_{n,n-1} - (x_n - x)P_{n-1,n-1}}{x_0 - x_n},$$

oder allgemeiner für P_{ik} ($i \geq k$):

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

Damit lässt sich P_{nn} ausgehend von den Daten f_0, \dots, f_n wie folgt rekursiv berechnen:

$$\begin{array}{ccccccc} f_0 & = & P_{00} & & & & \\ & & \searrow & & & & \\ f_1 & = & P_{10} & \rightarrow & P_{11} & & \\ \vdots & & \vdots & & \vdots & & \\ \vdots & & \vdots & & \vdots & & \\ & & \searrow & & \searrow & & \\ \vdots & & \vdots & & \vdots & \rightarrow & P_{n-2,n-2} \\ & & \searrow & & \searrow & & \searrow \\ f_{n-1} & = & P_{n-1,0} & \rightarrow & \dots & \rightarrow & P_{n-1,n-2} \rightarrow P_{n-1,n-1} \\ & & \searrow & & \searrow & & \searrow \\ f_n & = & P_{n,0} & \rightarrow & P_{n,1} & \dots \rightarrow & P_{n,n-2} \rightarrow P_{n,n-1} \rightarrow \boxed{P_{nn}} \end{array}$$

Daraus gewinnen wir die folgende Rechenvorschrift, die auch als *Dreiecksschema* bekannt ist:

Algorithmus 6.2.2 — Aitken-Neville.

- 1) $P(j, 0) = f(j)$, $j = 0, \dots, n$;
- 2) $P(j, k) = P(j, k-1) + \frac{x-x(j)}{x(j)-x(j-k)} (P(j, k-1) - P(j-1, k-1))$, $k = 1, \dots, n$, $j = k, \dots, n$.

Das obige Dreiecksschema bietet auch den Vorteil, dass es sehr einfach aufzudatieren ist: Hat man ein neues Datum f_{n+1} , dann fügt man eine Zeile zum Schema dazu, um dann $P_{n+1,n+1}$ als bessere Approximation zu erhalten, vgl. folgende Abbildung 6.1.

Wenn wir dies in MATLAB® umsetzen, müssen wir den Index um eins verschieben (in MATLAB® beginnen Indizes immer mit 1).

MATLAB-Funktion: AitkenNeville.m

```
1 function value = AitkenNeville(x,fx,x0)
2 % evaluate the Interpolation polynomial given
```

³Eric Harold Neville, 1889-1961.

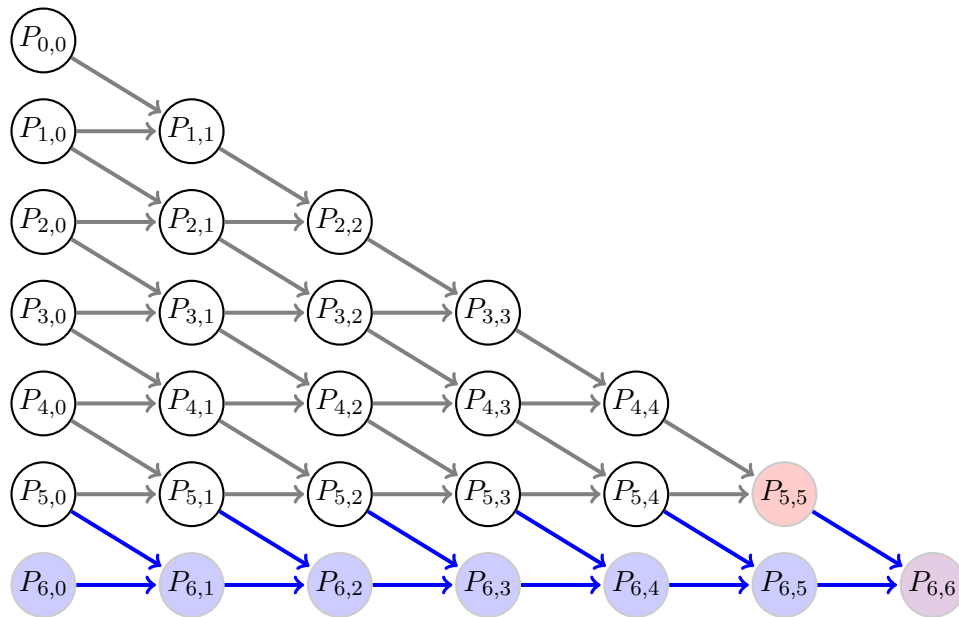


Abbildung 6.1: Aufdatierung eines Dreiecksschemas (in blau).

```

3 % by (x,fx) at the point x0
4 for k = 2:length(fx)
5     for j = length(fx):-1:k
6         fx(j) = fx(j) + (x0-x(j))/(x(j)-x(j-k+1))*(fx(j)-fx(j-1));
7     end
8 end
9 value = fx(end);

```

MATLAB-Beispiel:

Testen wir das Aitken-Neville-Verfahren anhand zweier Beispiele. Zum einen werten wir das Interpolationspolynom zu $f(x) := x^2$ und 3 Stützstellen an der Stelle 2 aus.

```

>> x = linspace(0,1,3);
>> AitkenNeville(x,x.^2,2)
ans =
    4

```

Zum anderen werten das Interpolationspolynom zu $f(x) := \sin(x)$ und 5 äquidistanten Stützstellen in $[0,1]$ an der Stelle $\pi/3$ aus. Man beachte, dass $\sin(\pi/3) = \sqrt{3}/2$ gilt.

```

>> x = linspace(0,1,5);
>> sqrt(3)/2-AitkenNeville(x,sin(x),pi/3)
ans =
    4.387286117690792e-005

```

6.3 Hermite-Interpolation, Newton-Darstellung und dividierte Differenzen

Bislang haben wir nur Funktionswerte interpoliert. Bei der **Hermite-Interpolation** berücksichtigt man auch Ableitungen der Funktion. Dies wird dadurch gewährleistet, dass Stützstellen mehrfach vorkommen. Um dies beschreiben zu können, benötigen wir einige Notationen.

Wir definieren die **Stützstellenfolge** $\Delta := \{x_j\}_{j=0,\dots,n}$ mit

$$a = x_0 \leq x_1 \leq \dots \leq x_n = b,$$

wobei Stützstellen nun auch mehrfach auftreten können. Sind an einer Stelle x_i der Funktionswert $f(x_i)$ und die Ableitungen $f'(x_i), \dots, f^{(k)}(x_i)$ bis zum Grad k gegeben, so soll die Stützstelle⁴ x_i in obiger Folge $(k+1)$ -mal auftreten. Gleiche Knoten nummerieren wir hierbei mit

$$d_i := \max\{j \in \mathbb{N} : x_i = x_{i-j}\}$$

von links nach rechts durch; z.B.

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

Führen wir nun mit diesen Abkürzungen nachfolgende lineare Abbildungen⁵

$$\mu_i : C^n[a, b] \rightarrow \mathbb{R}, \quad \mu_i(f) := f^{(d_i)}(x_i), \quad i = 0, \dots, n, \quad (6.8)$$

ein, so lautet die Aufgabe der

Hermite-Interpolation: Gegeben seien $\mu_i, i = 0, \dots, n$, gemäß (6.8). Gesucht ist $P \in \mathbb{P}_n$ mit

$$\mu_i(P) = \mu_i(f), \quad i = 0, \dots, n. \quad (6.9)$$

Die Lösung $P = P(f|x_0, \dots, x_n) \in \mathbb{P}_n$ von (6.9) heißt **Hermite-Interpolant**.

Wiederum untersuchen wir Existenz und Eindeutigkeit.

Satz 6.3.1 — Existenz und Eindeutigkeit. Zu jeder Funktion $f \in C^n([a, b])$ und jeder monotonen Stützstellenfolge $\Delta := \{x_j\}_{j=0,\dots,n} \subset [a, b]$ gibt es genau ein $P \in \mathbb{P}_n$, sodass

$$\mu_i(P) = \mu_i(f), \quad i = 0, \dots, n.$$

Beweis. Die Abbildung

$$\mu : \mathbb{P}_n \rightarrow \mathbb{R}^{n+1}, \quad P \mapsto (\mu_0(P), \dots, \mu_n(P))$$

ist offensichtlich eine lineare Abbildung zwischen den $(n+1)$ -dimensionalen reellen Vektorräumen \mathbb{P}_n und \mathbb{R}^{n+1} , sodass aus der Injektivität der Abbildung bereits die Surjektivität

⁴Stützstellen werden auch als *Knoten* bezeichnet.

⁵Man nennt dies auch ein *Funktional*.

folgen würde und damit der Satz bewiesen wäre. Somit reicht es die Injektivität der linearen Abbildung zu zeigen. Da $\mu(P) = 0$ gilt, folgt, dass P mindestens $(n+1)$ -Nullstellen inklusive Vielfachheiten besitzt, somit das Nullpolynom ist, also ist μ injektiv und damit bijektiv. ■

Wir hatten bereits die monomiale und die Lagrange-Basis von \mathbb{P}_n kennengelernt. Nun führen wir eine weitere Basis ein, die uns auch zu einem weiteren Algorithmus führen wird.

Definition 6.3.2 — Newton-Basis. Es seien $x_0, \dots, x_n \in \mathbb{R}$ und

$$\omega_0 := 1, \quad \omega_i(x) := \prod_{j=0}^{i-1} (x - x_j) \in \mathbb{P}_i, \quad i = 1, \dots, n.$$

Wir bezeichnen $\{\omega_0, \dots, \omega_n\}$ als **Newton-Basis** von \mathbb{P}_n mit Basispolynomen ω_i .

Bemerkung 6.3.3 Man beachte, dass in Definition 6.3.2 weder eine Ordnung der Knoten x_k noch „paarweise verschieden“ vorgeschrieben wurde. Je nach Nummerierung erhält man somit eine andere Newton-Basis. Dies ist bei der Stabilität der folgenden Verfahren zu berücksichtigen.

Wir leiten das numerische Verfahren basierend auf der Newton-Basis direkt für die allgemeinere Hermite-Interpolationsaufgabe her. Dazu drücken wir das Interpolationspolynom mittels der Linearfaktoren (also in der Newton-Basis) aus, also

$$P(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + \dots + a_n \prod_{j=0}^{n-1} (x - x_j) = \sum_{i=0}^n a_i \omega_i(x),$$

mit Koeffizienten $a_0, \dots, a_n \in \mathbb{R}$. Die gesuchten Koeffizienten können wir aus den Interpolationsbedingungen bestimmen:

$$\begin{array}{ll} P(x_0) = a_0 & = f(x_0) \\ P(x_1) = a_0 + a_1(x_1 - x_0) & = f(x_1) \\ P(x_2) = a_0 + a_1(x_2 - x_0) + a_2(x_2 - x_1)(x_2 - x_0) & = f(x_2) \\ \vdots & \vdots \end{array}$$

Dies führt offenbar auf ein lineares Gleichungssystem (LGS), wobei die Matrix eine untere Dreiecksmatrix ist. Das LGS kann also durch Vorwärtseinsetzen in $\mathcal{O}(n^2)$ Operationen gelöst werden (Details siehe Vorlesung *Numerische Lineare Algebra*).

Definition 6.3.4 Der führende Koeffizient a_n des Interpolationspolynoms

$$P(f|x_0, \dots, x_n)(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0$$

an f zu den Knoten $x_0 \leq x_1 \leq \dots \leq x_n$ heißt **n -te dividierte Differenz** von f bzgl. x_0, \dots, x_n und wird bezeichnet mit

$$[x_0, \dots, x_n]f := a_n.$$

Satz 6.3.5 — Newton-Darstellung. Für jede Funktion $f \in C^n(\mathbb{R})$ und Knoten $x_0 \leq \dots \leq x_n$ lautet das Interpolationspolynom $P(f|x_0, \dots, x_n)$ an f zu x_0, \dots, x_n

$$P(x) = \sum_{i=0}^n [x_0, \dots, x_i] f \cdot \omega_i(x). \quad (6.10)$$

Gilt darüber hinaus $f \in C^{n+1}(\mathbb{R})$, dann gilt

$$f(x) = P(x) + [x_0, \dots, x_n, x] f \cdot \omega_{n+1}(x). \quad (6.11)$$

Beweis. Wir zeigen die erste Behauptung durch Induktion nach $n \in \mathbb{N}$. Für $n = 0$ ist die Aussage trivialerweise erfüllt. Sei also im Folgenden $n > 0$ und

$$P_{n-1} := P(f|x_0, \dots, x_{n-1}) = \sum_{i=0}^{n-1} [x_0, \dots, x_i] f \cdot \omega_i$$

sei das Interpolationspolynom an f zu den Stützstellen x_0, \dots, x_{n-1} . Damit erhalten wir für das Interpolationspolynom $P_n := P(f|x_0, \dots, x_n)$ an f an x_0, \dots, x_n , dass

$$P_n(x) = [x_0, \dots, x_n] f \cdot x^n + a_{n-1}x^{n-1} + \dots + a_0 = [x_0, \dots, x_n] f \cdot \omega_n(x) + Q_{n-1}(x)$$

mit einem Polynom $Q_{n-1} \in \mathbb{P}_{n-1}$ gilt. Nun erfüllt aber $Q_{n-1} = P_n - [x_0, \dots, x_n] f \cdot \omega_n$ offensichtlich die Interpolationsaufgabe bzgl. der Knoten x_0, \dots, x_{n-1} , so dass wir nach Induktionsannahme

$$Q_{n-1} = P_{n-1} = \sum_{i=0}^{n-1} [x_0, \dots, x_i] f \cdot \omega_i$$

erhalten und damit (6.10). Insbesondere interpoliert $P_n + [x_0, \dots, x_n, x] f \cdot \omega_{n+1}$ die Funktion f an den Knoten x_0, \dots, x_n und x , also auch die Behauptung (6.11). ■

Aus den Eigenschaften der Hermite-Interpolation lassen sich sofort folgende Aussagen über die dividierten Differenzen zu f ableiten:

Lemma 6.3.6 i) Für $x_i \neq x_k$ gilt die Rekursionsformel

$$[x_0, \dots, x_n] f = \frac{[x_0, \dots, \widehat{x_i}, \dots, x_n] f - [x_0, \dots, \widehat{x_k}, \dots, x_n] f}{x_k - x_i},$$

wobei „ $\widehat{}$ “ anzeigt, dass die entsprechende Stützstelle weggelassen wird („seinen Hut nehmen muss“).

ii) Für zusammenfallende Knoten $x_0 = \dots = x_n$ gilt $[x_0, \dots, x_n] f = \frac{f^{(n)}(x_0)}{n!}$.

Beweis. Für das Hermite-Interpolationspolynom gilt für den Fall paarweise verschiedener Knoten, also $x_i \neq x_k$, dass

$$P(f|x_0, \dots, x_n) = \frac{\overbrace{[x_0, \dots, \widehat{x_i}, \dots, x_n] f}^{x^{n-1} + \mathbb{P}_{n-2}} - (x_k - x) P(f|x_0, \dots, \widehat{x_k}, \dots, x_n)}{x_i - x_k}, \quad (6.12)$$

was sich durch Überprüfen der Interpolationseigenschaft zeigen lässt mittels Einsetzen der Definitionen. Aus der Eindeutigkeit des führenden Koeffizienten folgt aus (6.12) unmittelbar Behauptung i) für paarweise verschiedener Knoten. Stimmen dagegen alle Knoten x_0, \dots, x_n überein, so lautet das Interpolationspolynom

$$P(f|x_0, \dots, x_n)(x) = \sum_{j=0}^n \frac{(x-x_0)^j}{j!} f^{(j)}(x_0),$$

wie man durch Einsetzen in μ_i leicht einsieht. Sei nun $0 \leq k \leq n$, dann folgt

$$\mu_k(P) = f^{(k)}(x_0) + \frac{k!(x-x_0)^1}{(k+1)!} f^{(k+1)}(x_0) + \dots + \frac{k!(x-x_0)^{n-k}}{n!} f^{(n)}(x_0),$$

und damit folgt ii). ■

Satz 6.3.7 Es sei $f \in C^n([a, b])$ und $f^{(n+1)}(x)$ existiere für alle $x \in (a, b)$; weiterhin sei $a \leq x_0 \leq x_1 \leq \dots \leq x_n \leq b$, dann gilt:

$$f(x) - P(f|x_0, \dots, x_n)(x) = \frac{(x-x_0)\cdots(x-x_n)}{(n+1)!} f^{(n+1)}(\xi), \quad (6.13)$$

mit einer Zwischenstelle $\xi \in (\min\{x, x_0, \dots, x_n\}, \max\{x, x_0, \dots, x_n\})$.

Beweis. Es gilt $P(f|x_0, \dots, x_n) = f(x_k)$ für alle $k = 0, 1, \dots, n$. Es sei $x \notin \{x_0, \dots, x_n\}$ und

$$K(x) := \frac{f(x) - P(f|x_0, \dots, x_n)(x)}{(x-x_0)\cdots(x-x_n)}, \quad K : [a, b] \rightarrow \mathbb{R}. \quad (6.14)$$

Nun betrachten wir die Funktion $W : [a, b] \rightarrow \mathbb{R}$

$$W(t) := f(t) - P(f|x_0, \dots, x_n)(t) - (t-x_0)\cdots(t-x_n)K(x), \quad t \in [a, b]. \quad (6.15)$$

Diese Funktion W verschwindet also an den Stellen $t \in \{x_0, \dots, x_n\}$ und aufgrund der Definition von K (6.14) auch an der Stelle $t = x$. Nach dem *verallgemeinerten Satz von Rolle*⁶ (Satz B.1.1) existiert ein $\xi \in (\min\{x, x_0, \dots, x_n\}, \max\{x, x_0, \dots, x_n\})$ mit $W^{(n+1)}(\xi) = 0$. Das $(n+1)$ -fache Differenzieren von (6.15) liefert

$$W^{(n+1)}(t) = f^{(n+1)}(t) - (n+1)! K(x),$$

so dass $0 = W^{(n+1)}(\xi) = f^{(n+1)}(\xi) - (n+1)! K(x)$. Damit erhalten wir aber unmittelbar

$$K(x) = \frac{1}{(n+1)!} f^{(n+1)}(\xi). \quad (6.16)$$

Einsetzen von (6.16) in (6.15) liefert die Behauptung für $t = x$ wegen $W(x) = 0$. ■

⁶Michel Rolle, 1652-1719.

Bemerkung 6.3.8 Im Beweis von Satz 6.3.7 haben wir gezeigt, dass

$$f(x) - P(f|x_0, \dots, x_n)(x) = \frac{\omega_{n+1}(x)}{(n+1)!} f^{(n+1)}(\xi), \quad (6.17)$$

mit einem $\xi \in (\min\{x_0, \dots, x_n, x\}, \max\{x_0, \dots, x_n, x\})$ und

$$\omega_{n+1}(x) := (x - x_0) \cdots (x - x_n) \in \mathbb{P}_{n+1} \quad (6.18)$$

gilt. Mit dem Satz 6.3.5 über die Newton-Darstellung gilt $f(x) - P(f|x_0, \dots, x_n)(x) = [x_0, \dots, x_n, x]f \cdot \omega_{n+1}(x)$. Somit folgern wir:

Für alle Knoten $x_0 \leq \dots \leq x_n$ existiert ein $\xi \in (x_0, x_n)$ mit

$$[x_0, \dots, x_n]f = \frac{f^{(n)}(\xi)}{n!}. \quad (6.19)$$

Die Auswertung der Rekursionsformel (6.12) erfolgt mit einem Dreiecksschema unter Verwendung der Startwerte $[x_i]f = f(x_i)$ für paarweise verschiedene Knoten

$$\begin{array}{lcl} x_0 & \rightarrow & [x_0]f \searrow \\ x_1 & \rightarrow & [x_1]f \swarrow \quad [x_0, x_1]f \searrow \\ x_2 & \rightarrow & [x_2]f \swarrow \quad [x_1, x_2]f \swarrow \quad [x_0, x_1, x_2]f \searrow \\ x_3 & \rightarrow & [x_3]f \swarrow \quad [x_2, x_3]f \swarrow \quad [x_1, x_2, x_3]f \swarrow \quad [x_0, x_1, x_2, x_3]f \searrow \\ x_4 & \rightarrow & [x_4]f \rightarrow \quad [x_3, x_4]f \rightarrow \quad [x_2, x_3, x_4]f \rightarrow \quad [x_1, x_2, x_3, x_4]f \rightarrow \quad [x_0, \dots, x_4]f \end{array}$$

Die gesuchten Koeffizienten a_k des Newton-Interpolationspolynoms findet man auf der oberen Diagonalen des Schemas (gelb hinterlegt).

Bemerkung 6.3.9 (a) Das obige Schema bietet wiederum die Möglichkeit der Aufdatierung beim Vorliegen zusätzlicher Daten.

(b) Im Gegensatz zum Neville-Aitken-Schema, bietet das Verfahren der dividierten Differenzen die Möglichkeit, das gesamte Polynom zu bestimmen. Will man das Interpolationspolynom nur an einer oder wenigen Stellen auswerten, dann ist das Neville-Aitken-Schema effizienter (weil man die anschließende Auswertung des Polynoms z.B. mit dem Horner-Schema nicht benötigt) – benötigt man viele Auswertungen oder die Darstellung des Polynoms, dann sind die dividierten Differenzen vorzuziehen.

■ **Beispiel 6.3.10** Wenn wir das Schema auf gegebene Daten (x_i, f_i) für $i = 0, \dots, 3$ anwenden, so erhalten wir

$$\begin{array}{lcl} x_0 = 0 & \rightarrow & f_0 = 1 \searrow \\ x_1 = \frac{3}{2} & \rightarrow & f_1 = 2 \swarrow \quad \frac{2}{3} \searrow \\ x_2 = \frac{5}{2} & \rightarrow & f_2 = 2 \swarrow \quad 0 \swarrow \quad -\frac{4}{15} \searrow \\ x_3 = \frac{9}{2} & \rightarrow & f_3 = 1 \rightarrow \quad -\frac{1}{2} \rightarrow \quad -\frac{1}{6} \rightarrow \quad \frac{1}{45} \end{array}$$

und damit ist

$$P(x) = 1 + \frac{2}{3}(x-0) - \frac{4}{15}(x-0)(x-\frac{3}{2}) + \frac{1}{45}(x-0)(x-\frac{3}{2})(x-\frac{5}{2}).$$

das Newtonsche Interpolationspolynom. ■

MATLAB-Funktion: NewtonInterpolation.m

```
1 function fx = NewtonInterpolation(x,fx)
2 for k = 2:length(fx)
3     for j = length(fx):-1:k
4         fx(j) = (fx(j) - fx(j-1))/(x(j)-x(j-k+1));
5     end
6 end
```

MATLAB-Funktion: EvalNewtonPoly.m

```
1 function value = HornerNewton(a,x0,x)
2 % evaluate Newton polynom
3 % p(x0) = a(1) + a(2)*(x0-x(1)) + a(3)*(x0-x(1))*(x0-x(2)) +
4 % ...
5 % = a(1) + ( (x0-x(1)) * ( a(2) + (x0-x(2)) * ( a(3) ....
6 value = a(end);
7 for k=length(a)-1:-1:1
8     value = a(k) + value.*(x0-x(k));
9 end
```

MATLAB-Beispiel:

Testen wir das Verfahren der dividierten Differenzen nochmals an den beiden Funktionen aus Beispiel 6.2. Zum einen werten wir das Interpolationspolynom zu $f(x) = x^2$ und 3 Stützstellen an der Stelle 2 aus.

```
>> x=linspace(0,2,3);
>> a = NewtonInterpolation(x,x.^2)
a =
    0    1    1
>> HornerNewton(a,2,x)
ans =
    4
```

Zum anderen werten das Interpolationspolynom zu $f(x) = \sin(x)$ und 5 äquidistanten Stützstellen in $[0,1]$ an der Stelle $\pi/3$ aus.

```
>> x=linspace(0,1,5);
>> a = NewtonInterpolation(x,sin(x));
>> sqrt(3)/2-HornerNewton(a,pi/3,x)
ans =
4.387286117690792e-005
```

■ **Beispiel 6.3.11** Betrachten wir das Schema für das Hermite-Interpolationsproblem (d.h. auch Ableitungen werden interpoliert). Unter Beachtung von Lemma 6.3.6, insbesondere

(6.12) ergibt sich:

$$\begin{aligned}
 x_0 &: \rightarrow [x_0]f \searrow \\
 x_0 &: \rightarrow [x_0]f \searrow [x_0, x_0]f = f'(x_0) \searrow \\
 x_0 &: \rightarrow [x_0]f \searrow [x_0, x_0]f = f'(x_0) \searrow [x_0, x_0, x_0]f = \frac{f''}{2}(x_0) \searrow \\
 x_1 &: \rightarrow [x_1]f \rightarrow [x_0, x_1]f \rightarrow [x_0, x_0, x_1]f \rightarrow [x_0, x_0, x_0, x_1]f
 \end{aligned}$$

Das resultierende Polynom P mit

$$P(x) = f(x_0) + (x - x_0)\left(f'(x_0) + (x - x_0)\left(f''(x_0) + (x - x_0)[x_0, x_0, x_0, x_1]f\right)\right)$$

erfüllt die Interpolationsbedingungen $P(x_0) = f(x_0)$, $P'(x_0) = f'(x_0)$, $P''(x_0) = f''(x_0)$ und $P(x_1) = f(x_1)$. ■

Für den speziellen Fall, dass an *allen* Knoten x_0, \dots, x_n sowohl f als auch f' interpoliert werden sollen, erhält man die folgende Darstellung des Interpolationspolynoms $P(x)$

Satz 6.3.12 Es sei $\omega(x) = (x - x_0) \cdots (x - x_n)$ und $L_{n,k}(x)$ seien die Lagrange-Polynome (6.2) zu den Knoten x_k , $k = 0, \dots, n$. Dann hat

$$P(x) = \sum_{k=1}^n f(x_k) \left(1 - \frac{\omega''(x_k)}{\omega'(x_k)}(x - x_k)\right) L_{n,k}^2(x) + \sum_{k=1}^n f'(x_k)(x - x_k) L_{n,k}^2(x)$$

die Interpolationseigenschaften

$$P(x_k) = f(x_k) \text{ und } P'(x_k) = f'(x_k) \text{ für alle } k = 0, \dots, n.$$

Beweis. Es sei x_ℓ einer der Knoten x_0, \dots, x_n , dann folgt sofort aus der Interpolationseigenschaft der Lagrange-Polynome $P(x_\ell) = f(x_\ell)$, da

$$\omega'(x) = \sum_{i=0}^n \prod_{\substack{k=0 \\ k \neq i}}^n (x - x_k) \text{ und somit } \omega'(x_\ell) = \prod_{\substack{k=0 \\ k \neq \ell}}^n (x_\ell - x_k) \neq 0$$

gilt. Für die Ableitung von P ergibt sich

$$\begin{aligned}
 P'(x) &= \sum_{k=1}^n f(x_k) \left[\left(1 - \frac{\omega''(x_k)}{\omega'(x_k)}(x - x_k)\right) 2L'_{n,k}(x) - \frac{\omega''(x_k)}{\omega'(x_k)} L_{n,k}(x) \right] L_{n,k}(x) \\
 &\quad + \sum_{k=1}^n f'(x_k) \left[(x - x_k) 2L'_{n,k}(x) + L_{n,k}(x) \right] L_{n,k}(x),
 \end{aligned}$$

sodass wir nun Folgendes erhalten:

$$P'(x_\ell) = f(x_\ell) \left(2L'_{n,\ell}(x_\ell) - \frac{\omega''(x_\ell)}{\omega'(x_\ell)} \right) + f'(x_\ell).$$

Nutzt man aus, dass gilt (Übung)

$$L_{n,\ell}(x) = \frac{\omega(x)}{(x - x_\ell) \omega'(x_\ell)},$$

so folgt aus $\omega(x) = L_{n,\ell}(x)(x - x_\ell)\omega'(x_\ell)$ nach zweimaligem Differenzieren

$$\omega''(x) = L''_{n,\ell}(x)(x - x_\ell)\omega'(x_\ell) + 2L'_{n,\ell}(x)\omega'(x_\ell).$$

Damit gilt an der Stelle x_ℓ

$$\frac{\omega''(x_\ell)}{\omega'(x_\ell)} = 2L'_{n,\ell}(x_\ell)$$

Einsetzen in die Ableitung von P schließt den Beweis ab. ■

6.4 Tschebyscheff-Interpolation und Interpolationsfehler

Wir beschäftigen uns in diesem Abschnitt mit dem Interpolationsfehler und geben Abschätzungen an. Bei der Analyse des Approximations- bzw. des Interpolationsfehlers haben wir in (6.17) und (6.18) gesehen, dass für den Interpolationsfehler R_n die Darstellung

$$R_n[f](x) := f(x) - P(f|x_0, \dots, x_n)(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!} \omega_{n+1}(x), \quad x \in [a, b], \quad (6.20)$$

für ein $\xi = \xi(x) \in (a, b)$ gilt. Damit können wir bereits eine erste Fehlerabschätzung angeben.

Satz 6.4.1 Seien $f \in C^\infty([a, b])$ und $\|f^{(n)}\|_\infty \leq M$ für alle $n \in \mathbb{N}_0$. Dann konvergiert der Interpolationsfehler $R_n[f]$ für $n \rightarrow \infty$ gleichmäßig auf $[a, b]$ gegen Null.

Beweis. Die Funktion ω_{n+1} können wir (grob) abschätzen durch $\|\omega_{n+1}\|_{\infty;[a,b]} \leq (b-a)^{n+1}$ und damit gilt nach Voraussetzung

$$\begin{aligned} \|R_n[f]\|_{\infty;[a,b]} &= \max_{x \in [a,b]} |f(x) - P(f|x_0, \dots, x_n)(x)| \leq \frac{\|f^{(n+1)}\|_{\infty;[a,b]}}{(n+1)!} \|\omega_{n+1}\|_{\infty;[a,b]} \\ &\leq M \frac{(b-a)^{n+1}}{(n+1)!} \xrightarrow{n \rightarrow \infty} 0, \end{aligned}$$

womit die Behauptung bewiesen ist. ■

Dieser Satz ist zwar für viele „Standardfunktionen“, jedoch in der Regel nicht auf praktisch interessierende Funktionen anwendbar. Insbesondere ist die starke Glattheitsforderung an die zu interpolierende Funktion sehr einschränkend. Natürlich ist die Abschätzung $\|\omega_{n+1}\|_{\infty;[a,b]} \leq (b-a)^{n+1}$ sehr grob und es liegt nahe, dies zu verbessern, indem wir die Stützstellen geschickt wählen. Wie das folgende Beispiel zeigen wird, hat die Verteilung der Stützstellen x_0, \dots, x_n über das Interpolationsintervall entscheidenden Einfluss auf die Güte der Approximation. Das klassische Beispiel hierfür stammt von Runge⁷.

■ **Beispiel 6.4.2 — Gegenbeispiel von Runge.** Die Interpolationspolynome $P(f|x_1, \dots, x_n)$ zur Funktion $f(x) = \frac{1}{1+x^2}$ im Intervall $I := [-5, 5]$ bei äquidistanten Stützstellen $x_k = -5 + \frac{10}{n}k$, $k = 0, \dots, n+1$, zeigen bei wachsendem n einen zunehmenden Interpolationsfehler durch stärker werdende Oszillationen, vgl. Abbildung 6.2. ■

⁷Carl Runge, 1856-1927.

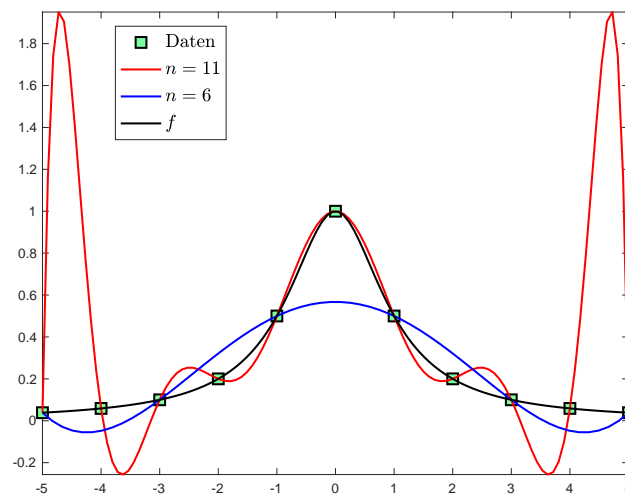


Abbildung 6.2: Beispiel von Runge: Funktion $f(x) = \frac{1}{1+x^2}$ (schwarz) im Intervall $I := [-5, 5]$ und Interpolationspolynom bei 6 (blau) und 11 (rot) äquidistanten Stützstellen.

MATLAB-Beispiel:

Das Interpolationspolynom zu 11 äquidistanten Stützstellen und Stützwerten zur Funktion

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

ist in Abb. 6.2 dargestellt.

```
n = 11;
f = @(x) 1./(x.^2+1);
x = linspace(-5,5,n);
fx = f(x);
s = linspace(x(1),x(end),10*n);
for j=1:length(s)
    ps(j) = AitkenNeville(x,fx,s(j));
end
plot(x,fx,'*',s,ps,'r-',s,f(s),'k')
```

Wie wir im weiteren zeigen werden, kann man bei geschickter, nichtäquidistanter Wahl der Stützstellen die obige Fehlerabschätzung deutlich verbessern. Genauer gesagt, wählen wir x_1, \dots, x_n als die Nullstellen der von $[-1, 1]$ auf $I = [a, b]$ transformierten *Tschebyscheff-Polynome*. Wir suchen Knoten $x_0, \dots, x_n \in [a, b]$, die das *MiniMax-Problem*

$$\max_{x \in [a,b]} |\omega_{n+1}(x)| = \max_{x \in [a,b]} |(x-x_0) \cdots (x-x_n)| = \min_{x_0, \dots, x_n} ! \quad (6.21)$$

lösen. Anders formuliert, wir wollen das monische Polynom $\omega_{n+1} \in \mathbb{P}_{n+1}$ mit reellen Nullstellen x_0, \dots, x_n so bestimmen, dass $\max_{x \in [a,b]} |\omega_{n+1}(x)|$ minimal für alle möglichen Wahlen von Stützstellen gilt. Dazu verwenden wir Satz 7.2.9, der zeigt, dass die Tschebyscheff-Polynome das MinMax-Problem auf $[-1, 1]$ für alle Polynome in \mathbb{P}_n mit führendem Koeffizienten 2^{n-1} lösen. Also müssen wir das Problem von $I = [a, b]$ auf das Intervall $[-1, 1]$

transformieren mit Hilfe der folgenden affinen Abbildung

$$\varphi : [a, b] \rightarrow [-1, 1], \quad \varphi(x) = \frac{2x - a - b}{b - a},$$

deren Umkehrabbildung

$$\psi : [-1, 1] \rightarrow [a, b], \quad \psi(\hat{x}) = \frac{b+a}{2} + \frac{b-a}{2}\hat{x} = \varphi^{-1}(\hat{x})$$

lautet. Seien also $\hat{x}_n^{(k)}$, $k = 1, \dots, n$, die Nullstellen des Tschebyscheff-Polynoms T_n auf $[-1, 1]$ nach Lemma 7.2.8 (vii), dann verwenden wir

$$x_n^{(k)} := \psi(\hat{x}_n^{(k)}), \quad k = 1, \dots, n, \quad (6.22)$$

als Knoten für die **Tschebyscheff-Interpolation** $P_n[f] := P(f|x_n^{(1)}, \dots, x_n^{(n)})$.

Satz 6.4.3 Für den Fehler der Tschebyscheff-Interpolation gilt

$$\|f - P_n[f]\|_{\infty; [a, b]} \leq \frac{2^{-n}}{(n+1)!} \|f^{(n+1)}\|_{\infty; [a, b]}.$$

Beweis. Wir transformieren das Polynom ω_{n+1} auf das Intervall $[-1, 1]$ mittels $\tilde{\omega}_{n+1} := \varphi(\omega_{n+1})$. Dieses Polynom hat den führenden Koeffizienten 2^{n+1} aufgrund der Transformation. Damit gilt mit Satz 7.2.9 und Lemma 7.2.8 (iii), dass⁸

$$\|\omega_{n+1}\|_{\infty; [a, b]} = 2^{-(n+1)} \|\tilde{\omega}_{n+1}\|_{\infty; [-1, 1]} \leq 2^{-n} \|T_{n+1}\|_{\infty; [-1, 1]} = 2^{-n}$$

und damit die Behauptung. ■

Zwar ist diese Abschätzung deutlich besser als diejenige in Satz 6.4.1 zu Beginn dieses Abschnittes, sie sichert aber nicht die gleichmäßige Konvergenz. Dazu macht der folgende Satz von Marcinkiewicz⁹ eine positive Aussage, [HW64; Mar39; Zyg59].

Satz 6.4.4 — Marcinkiewicz. Für jede Funktion $f \in C([a, b])$ gibt es eine Folge von Stützstellen $(x_k^{(n)})$, $k = 0, \dots, n$, $n \in \mathbb{N}_0$ derart, dass die entsprechende Folge $(P_n f)_{n \in \mathbb{N}_0}$ von Interpolationspolynomen $P_n f \in \mathbb{P}_n$ mit $(P_n f)(x_k^{(n)}) = f(x_k^{(n)})$, $k = 0, \dots, n$ auf $[a, b]$ gleichmäßig gegen f konvergiert. ■

Wenn also eine Funktion gegeben ist, dann gibt es eine Knotenfolge, die Konvergenz sichert – für diese Funktion. Auf der anderen Seite können wir gleichmäßige Konvergenz für eine gewählte Knotenfolge nicht erwarten, wie der folgende Satz von Faber¹⁰ zeigt. Für den Beweis verweisen wir auf die Originalarbeit [Fab14] sowie [WS78, S. 150-151].

Satz 6.4.5 — Faber. Zu jeder Folge $(\Delta^{(n)})_{n \in \mathbb{N}}$, $\Delta^{(n)} = \{x_0^{(n)}, \dots, x_n^{(n)}\}$, von Knoten in $[a, b]$ gibt es ein $f \in C([a, b])$, so dass der Fehler der zugehörigen Interpolationspolynome

⁸Beachte: Der führende Koeffizient von T_{n+1} ist 2^{-n} .

⁹Józef Marcinkiewicz, 1910-1940.

¹⁰Georg Faber, 1877-1966.

$P(f|x_0^{(n)}, \dots, x_n^{(n)})$ mit $n \rightarrow \infty$ nicht gleichmäßig gegen Null konvergiert. ■

- Bemerkung 6.4.6** (a) Für das Beispiel 6.4.2 von Runge konvergiert die Tschebyscheff-Interpolation sowohl punktweise als auch gleichmäßig, da für diesen Fall der obige Fehlerterm $\frac{2^{-n}}{(n+1)!} \|f^{(n+1)}\|_\infty$ gegen Null geht. Wir erkennen dies in Abbildung 6.3, in der wir auch erkennen, dass der Fehler für die Interpolation mit äquidistanten Knoten wächst.
- (b) Der Satz von Faber zeigt aber, dass auch die optimale Wahl der Tschebyscheff-Knoten das prinzipielle Problem im Gegenbeispiel von Runge nicht behebt in dem Sinne, dass der Interpolationsfehler nicht unbedingt gegen Null konvergieren muss, wenn man den Polynomgrad erhöht. Dies werden wir erst durch Splines erreichen können.

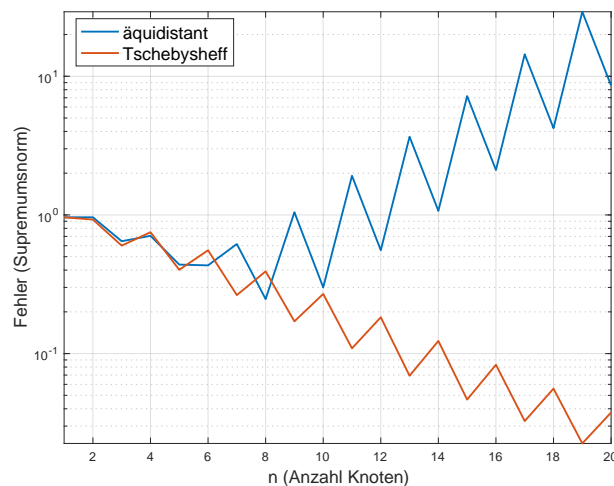


Abbildung 6.3: Fehler der Interpolation an die Runge-Funktion für äquidistante Knoten (blau) und Tschebyscheff-Knoten (rot).

Versuchen wir es mit ChatGBT, ein MATLAB®-Programm für die Tschebyscheff-Interpolation zu erzeugen.



Bitte schreibe mir ein Matlab-Programm für die Tschebyscheff-Interpolation



Natürlich, hier ist ein einfaches MATLAB-Programm, das die Tschebyscheff-Interpolation für gegebene Stützstellen durchführt. Beachte, dass dies eine grundlegende Implementierung ist und je nach deinen Anforderungen möglicherweise angepasst werden muss:

MATLAB-Funktion per ChatGBT: chebyshev_interpolation.m

```
1 function chebyshev_interpolation()
2
3     % Anzahl der Stuetzstellen
4     n = input('Anzahl der Stuetzstellen: ');
```

```

5
6     % Grenzen des Interpolationsintervalls
7     a = input('Untere Grenze des Intervalls: ');
8     b = input('Obere Grenze des Intervalls: ');
9
10    % Generiere Tschebyscheff-Stuetzstellen
11    x = cos(pi * (2*(n-1:-1:0) + 1) / (2*n));
12    x = 0.5 * (a + b) + 0.5 * (b - a) * x;
13
14    % Berechne Stuetzwerte der Funktion an den Stuetzstellen
15    y = f(x);
16
17    % Berechne Tschebyscheff-Koeffizienten
18    c = chebyshev_coefficients(x, y, n);
19
20    % Interpolationspolynom auswerten und plotten
21    xx = linspace(a, b, 1000);
22    yy = chebyshev_interpolant(xx, c, a, b);
23
24    plot(xx, yy, 'b', x, y, 'ro');
25    legend('Tschebyscheff-Interpolant', 'Stuetzwerte');
26    xlabel('x');
27    ylabel('y');
28    title('Tschebyscheff-Interpolation');
29 end
30
31 function y = f(x)
32     % Hier die Funktion definieren, die du interpolieren
33     % moechtest
34     y = cos(x);
35 end
36
37 function c = chebyshev_coefficients(x, y, n)
38     % Berechne Tschebyscheff-Koeffizienten mithilfe von DFT
39     c = zeros(1, n);
40     for k = 1:n
41         c(k) = sum(y .* cos((k-1) * acos((2*x - (a + b)) / (b -
42             a))));
43     end
44     c = c * 2 / n;
45 end
46
47 function p = chebyshev_interpolant(x, c, a, b)
48     % Evaluiere das Tschebyscheff-Interpolationspolynom
49     p = zeros(size(x));
50     for k = 1:length(c)
51         p = p + c(k) * cos((k-1) * acos((2*x - (a + b)) / (b -
52             a))));
53     end
54 end

```

Das Programm läuft nicht! ChatGBT ist kein Heilmittel – korrigieren Sie den Code und überprüfen Sie auch die Ergebnisse! Das Programm ist auch nicht sonderlich gut programmiert, es geht deutlich besser (und übersichtlicher).

Kapitel 7

Orthogonale Polynome und Drei-Term-Rekursionen^(*)

Orthogonale Polynome haben zahlreiche praktische Anwendungen in der Numerik und allgemeiner in der Mathematik, u.a.:

- In vielen Fällen lässt sich die Lösung eines Problems in eine Folge sogenannter „spezieller Funktionen“ entwickeln. Diese speziellen Funktionen sind in der Mathematik von großem Interesse, daher ist man an effizienten Berechnungsmethoden interessiert.
- In der Numerischen Linearen Algebra wird die Konvergenzgeschwindigkeit des cg- und des GMRES-Verfahrens durch spezielle Polynome dargestellt, bei cg waren dies die Tschebyscheff-Polynome.
- Bei der numerischen Integration und Differentiation (siehe später) spielen Nullstellen von orthogonalen Polynomen eine große Rolle. Dies trifft insbesondere dann zu, wenn die zu berechneten Integrale Erwartungswerte von Zufallsvariablen mit einer unbekannten Wahrscheinlichkeitsdichte sind. In diesem Fall kann man mit einer Drei-Term-Rekursion eine *empirische Wahrscheinlichkeitsdichte* schätzen.

7.1 Einige Grundlagen

Orthogonale Polynome sind eine für die Numerik besonders relevante Klasse spezieller Funktionen, die obige Aufzählung deutet es schon an. Wir müssen also zunächst klären, was wir unter „Orthogonalität“ von Funktionen verstehen. Dazu brauchen wir ein Skalarprodukt auf Funktionenräumen, die oftmals unendlich-dimensional sind.

Definition 7.1.1 — Skalarprodukt. Es sei V ein reeller Vektorraum. Ein **Skalarprodukt** (oder inneres Produkt) auf V ist eine *symmetrische positiv definite Bilinearform* $(\cdot, \cdot): V \times V \rightarrow \mathbb{R}$; dies bedeutet, dass für $x, y, z \in V$ und $\alpha, \beta \in \mathbb{R}$ die folgenden Bedingungen gelten:

- i) Positive Definitheit: $(x, x) \geq 0$ und $(x, x) = 0$ genau dann, wenn $x = 0$,
- ii) Symmetrie: $(x, y) = (y, x)$,
- iii) Bilinearität: $(\alpha x + \beta y, z) = \alpha(x, z) + \beta(y, z)$.

■ **Beispiel 7.1.2** (i) Für $V = \mathbb{R}^n$ erfüllt das Euklidische Skalarprodukt $\langle x, y \rangle := x^T y$ obige

Bedingungen.

- (ii) Sei $V = C([a, b])$ der Raum der stetigen Funktionen auf $[a, b] \subset \mathbb{R}$. Weiter sei $\omega \in C(a, b)$ mit $\omega(x) > 0$ für $x \in (a, b)$ eine positive *Gewichtsfunktion*. Dann ist

$$(f, g) := (f, g)_\omega := \int_a^b \omega(x) f(x) g(x) dx, \quad f, g \in C([a, b]),$$

ein Skalarprodukt, was man leicht nachweist. ■

Definition 7.1.3 (i) Elemente $x, y \in V$ heißen **orthogonal** (bzgl. (\cdot, \cdot)), falls $(x, y) = 0$.

(ii) $\|x\| := \sqrt{(x, x)}$, $x \in V$, heißt die durch das Skalarprodukt **induzierte Norm**.

Satz 7.1.4 — Cauchy-Schwarzsche-Ungleichung. Es sei V ein reeller Vektorraum, $(\cdot, \cdot): V \times V \rightarrow \mathbb{R}$ ein **Skalarprodukt** auf V , dann gilt $(x, y) \leq \sqrt{(x, x)} \sqrt{(y, y)}$.

Beweis. Der Fall $y = 0$ ist trivial. Sei also $y \neq 0$ und damit $(y, y) \neq 0$. Für jedes $\alpha \in \mathbb{R}$ gilt

$$0 \leq (x - \alpha y, x - \alpha y) = (x, x) - 2\alpha(x, y) + \alpha^2(y, y).$$

Wählt man nun speziell $\alpha := (x, y)/(y, y)$, so ergibt sich $0 \leq (x, x) - \frac{(x, y)^2}{(y, y)}$, also $(x, y)^2 \leq (x, x)(y, y)$ und damit die Behauptung. ■

Bemerkung 7.1.5 Sei ω eine positive Gewichtsfunktion auf (a, b) , so dass die durch $(\cdot, \cdot)_\omega$ induzierte Norm

$$\|p\| := \|p\|_\omega := \sqrt{(p, p)_\omega}$$

für alle Polynome $p \in \mathbb{P}_k$, $k \in \mathbb{N}$, wohldefiniert und endlich ist. Somit existieren auch die **Momente**

$$m_k := \int_a^b \omega(x) x^k dx < \infty,$$

da mit der Cauchy-Schwarzschen-Ungleichung $|m_k| = |(1, x^k)_\omega| \leq \|1\|_\omega \|x^k\|_\omega < \infty$ folgt.

■ **Beispiel 7.1.6** Einige Beispiele von Gewichtsfunktionen seien hier genannt, die auf den ersten Blick ungewöhnlich erscheinen und insbesondere spezielle numerische Techniken erfordern, die aber in praktischen Anwendungen verwendet werden.

- (i) $\omega(x) = x^\alpha \log(1/x)$ auf $[0, 1]$ mit $\alpha > 0$.

Die Momente $m_k = (k + \alpha + 1)^{-2}$ sind alle endlich und die zugehörigen Orthogonalpolynome werden verwendet, um Quadraturformeln zu konstruieren für Integrale über $[0, 1]$, deren Integranden zwei Singularitäten bei Null haben, eine logarithmische und eine algebraische (falls $\alpha \neq 0, 1, 2, \dots$).

- (ii) $\omega(x) = e^{-x}$ und $\omega(x) = e^{-x^2}$ auf $[0, c]$, $0 < c < \infty$.

Dies sind Laguerre bzw. Hermite-Gewichte auf einem endlichen Intervall. Die Momente m_k lassen sich durch die unvollständige Gamma-Funktion $\gamma(\alpha, x) = \int_0^x t^{\alpha-1} e^{-t} dt$

ausdrücken, nämlich $m_k = \gamma(k+1, c)$ bzw. $m_k = \frac{1}{2}\gamma(\frac{1}{2}(k-1), c^2)$. Beide Varianten finden Anwendung bei der Gauß-Quadratur von Integralen in der molekularen Quantenmechanik.

Wir werden noch weitere Beispiele kennenlernen. ■

Definition 7.1.7 — Orthogonalpolynome. Eine Folge $(p_n)_{n \in \mathbb{N}_0}$ von Polynomen $p_n \in \mathbb{P}_n$ exakt vom Grad n (also führendem Koeffizienten $a_n \neq 0$) heißt **Orthogonalpolynome** (bzgl. der positiven Gewichtsfunktion ω), falls

$$(p_i, p_j)_\omega = \delta_{ij} \|p_i\|_\omega^2 \geq 0 \quad \text{für alle } i, j \in \mathbb{N}_0. \quad (7.1)$$

Bemerkung 7.1.8 (i) Falls $(p_n)_{n \in \mathbb{N}_0}$ eine Folge von Orthogonalpolynomen ist, dann ist $\mathbb{P}_k = \text{span}\{p_0, \dots, p_k\}$, die ersten $k+1$ Orthogonalpolynome bilden also eine Basis von \mathbb{P}_k . Daraus folgt sofort

$$(p_k, q)_\omega = (p_k, \sum_{i=0}^{k-1} \alpha_i p_i)_\omega = \sum_{i=0}^{k-1} \alpha_i (p_k, p_i)_\omega = 0 \quad \text{für alle } q \in \mathbb{P}_{k-1}. \quad (7.2)$$

Man drückt dies durch „ $p_k \perp_\omega \mathbb{P}_{k-1}$ “ aus.

(ii) Die Definition der Orthogonalpolynome über (7.1) ist nicht eindeutig. Mögliche Standardisierungen sind z.B. dass der führende Koeffizient Eins ist, also

$$p_n(x) = x^n + a_{n-1}x^{n-1} + \dots + a_1x + a_0,$$

die Normierung $\|p_n\|_\infty = 1$ bzw. $\|p_n\|_\omega = 1$ (Orthonormalisierung) oder die Festlegung des führenden Koeffizienten a_n auf einen beliebigen Wert, den wir später mit k_n bezeichnen werden.

7.2 Drei-Term-Rekursionen

Definition 7.2.1 — Monisches Polynom. Ist der führende Koeffizient eines Polynoms Eins, so bezeichnet man dieses als **monisch** (oder auch normiert).

Satz 7.2.2 — Drei-Term-Rekursion. Sei ω eine positive Gewichtsfunktion. Dann gibt es eindeutig bestimmte monische Orthogonalpolynome; sie genügen der folgenden **Drei-Term-Rekursion**

$$p_{-1} := 0, p_0 := 1, \quad p_n(x) = (\alpha_n + x)p_{n-1}(x) + \gamma_n p_{n-2}(x), \quad n = 1, 2, \dots \quad (7.3)$$

$$\text{mit} \quad \alpha_n = -\frac{(xp_{n-1}, p_{n-1})_\omega}{(p_{n-1}, p_{n-1})_\omega}, \quad \gamma_n = -\frac{(p_{n-1}, p_{n-1})_\omega}{(p_{n-2}, p_{n-2})_\omega}. \quad (7.4)$$

Beweis. Wir zeigen die Behauptung per Induktion. Es gilt $p_0 \equiv 1 \in \mathbb{P}_0$ und dies ist das einzige monische Polynom in \mathbb{P}_0 , also eindeutig. Seien nun p_0, \dots, p_{n-1} die eindeutigen paarweise orthogonalen monischen Polynome mit $p_j \in \mathbb{P}_j$ vom Grad exakt j . Sei $p_n \in \mathbb{P}_n$ ein normiertes Polynom, dann folgt $p_n(x) - xp_{n-1}(x) \in \mathbb{P}_{n-1}$. Da $\{p_0, \dots, p_{n-1}\}$ eine Orthogonalbasis von \mathbb{P}_{n-1} bzgl. $(\cdot, \cdot)_\omega$ bildet, besitzt $p_n(x) - xp_{n-1}(x)$ eine Entwicklung in

dieser Basis, d.h.

$$p_n - xp_{n-1} = \sum_{j=0}^{n-1} \tilde{\gamma}_j p_j \quad \text{mit Entwicklungskoeffizienten} \quad \tilde{\gamma}_j := \frac{(p_n - xp_{n-1}, p_j)_\omega}{(p_j, p_j)_\omega}.$$

Außerdem folgt aus der Orthogonalität von p_n zu p_0, \dots, p_{n-1} bzgl. $(\cdot, \cdot)_\omega$

$$\tilde{\gamma}_j = -\frac{(xp_{n-1}, p_j)_\omega}{(p_j, p_j)_\omega} = -\frac{(p_{n-1}, xp_j)_\omega}{(p_j, p_j)_\omega}. \quad (7.5)$$

Da $x p_j \in \mathbb{P}_{j+1}$ und $p_{n-1} \perp_\omega \mathbb{P}_{n-2}$ folgt $(p_{n-1}, xp_j)_\omega = 0$ für $j = 0, \dots, n-3$, also $\tilde{\gamma}_0 = \dots = \tilde{\gamma}_{n-3} = 0$ und damit $p_n - xp_{n-1} = \tilde{\gamma}_{n-2} p_{n-2} + \tilde{\gamma}_{n-1} p_{n-1}$. Beachtet man $x p_{n-2} = p_{n-1} + \alpha_{n-2} p_{n-2} + \dots + \alpha_0 p_0$, so folgt mit (7.5), dass

$$\tilde{\gamma}_{n-2} = -\frac{(p_{n-1}, xp_{n-2})_\omega}{(p_{n-2}, p_{n-2})_\omega} = -\frac{(p_{n-1}, p_{n-1} + \alpha_{n-2} p_{n-2} + \dots + \alpha_0 p_0)_\omega}{(p_{n-2}, p_{n-2})_\omega} = -\frac{(p_{n-1}, p_{n-1})_\omega}{(p_{n-2}, p_{n-2})_\omega} = \gamma_n$$

und direkt aus (7.5)

$$\tilde{\gamma}_{n-1} = -\frac{(xp_{n-1}, p_{n-1})_\omega}{(p_{n-1}, p_{n-1})_\omega} = \alpha_n.$$

Damit gilt $p_n(x) = (\alpha_n + x) p_{n-1}(x) + \gamma_{n-2} p_{n-2}(x)$ wie in (7.3) und (7.4) behauptet. Dieses p_n ist eindeutig. ■

Wir übertragen die Drei-Term-Rekursion auf der Orthogonalpolynome in der Form

$$p_n(x) = k_n x^n + k'_n x^{n-1} + \mathcal{O}(x^{n-2}), \quad n \in \mathbb{N}_0, \quad (7.6)$$

also mit führendem Koeffizienten $k_n \in \mathbb{R}$.

Satz 7.2.3 Zu jedem Skalarprodukt $(\cdot, \cdot)_\omega$ gibt es eindeutig bestimmte Orthogonalpolynome $p_n \in \mathbb{P}_n$ mit führendem Koeffizienten k_n . Diese erfüllen die Drei-Term-Rekursion

$$p_{-1} = 0, \quad p_0 = k_0, \quad p_n(x) = (a_n + b_n x) p_{n-1} + c_n p_{n-2}, \quad n \in \mathbb{N}. \quad (7.7)$$

Die Koeffizienten lauten dabei

$$b_n = \frac{k_n}{k_{n-1}}, \quad a_n = b_n \left(\frac{k'_{n-1}}{k_{n-1}} - \frac{k'_n}{k_n} \right), \quad c_n = -\frac{k_n k_{n-2} h_{n-1}}{k_{n-1}^2 h_{n-2}}, \quad n = 2, 3, \dots, \quad (7.8)$$

wobei

$$h_n(p_n) := h_n := \int_a^b p_n^2(x) \omega(x) dx = \|p_n\|_\omega^2. \quad (7.9)$$

Beweis. Sei p_n wie in (7.6), dann ist $\bar{p}_n(x) := p_n(x)/k_n$ ein monisches Polynom und die Drei-Term-Rekursion aus Satz 7.2.2 liefert $\bar{p}_{-1} := 0$, $\bar{p}_0 := p_0/k_0 = 1$ und

$$\begin{aligned} p_n(x) &= k_n \bar{p}_n(x) = k_n (x - \beta_n) \bar{p}_{n-1}(x) - k_n \gamma_n \bar{p}_{n-2}(x) \\ &= \frac{k_n}{k_{n-1}} (x - \beta_n) p_{n-1}(x) - \frac{k_n}{k_{n-2}} \gamma_n p_{n-2}(x), \quad n = 1, 2, \dots \end{aligned} \quad (7.10)$$

mit den Koeffizienten

$$\beta_n = \frac{(x\bar{p}_{n-1}, \bar{p}_{n-1})_\omega}{(\bar{p}_{n-1}, \bar{p}_{n-1})_\omega}$$

und

$$\gamma_n = \frac{(\bar{p}_{n-1}, \bar{p}_{n-1})_\omega}{(\bar{p}_{n-2}, \bar{p}_{n-2})_\omega} = \frac{1/k_{n-1}^2 (p_{n-1}, p_{n-1})_\omega}{1/k_{n-2}^2 (p_{n-2}, p_{n-2})_\omega} = \frac{k_{n-2}^2}{k_{n-1}^2} \frac{h_{n-1}}{h_{n-2}}. \quad (7.11)$$

Man beachte

$$x\bar{p}_{n-1} = x \left(x^{n-1} + \frac{k'_{n-1}}{k_{n-1}} x^{n-2} + \mathcal{O}(x^{n-3}) \right) = \bar{p}_n + \left(\frac{k'_{n-1}}{k_{n-1}} - \frac{k'_n}{k_n} \right) \bar{p}_{n-1} + q(x),$$

mit einem $q \in \mathbb{P}_{n-2}$, also

$$\beta_n = \frac{(x\bar{p}_{n-1}, \bar{p}_{n-1})_\omega}{(\bar{p}_{n-1}, \bar{p}_{n-1})_\omega} = \frac{k'_{n-1}}{k_{n-1}} - \frac{k'_n}{k_n}. \quad (7.12)$$

Somit ergibt sich mit (7.11)

$$k_n \gamma_n \bar{p}_{n-2}(x) = k_n \frac{k_{n-2}^2}{k_{n-1}^2} \frac{h_{n-1}}{h_{n-2}} \frac{1}{k_{n-2}} p_{n-2}(x) = \frac{k_n k_{n-2}}{k_{n-1}^2} \frac{h_{n-1}}{h_{n-2}} p_{n-2}(x), \text{ bzw.}$$

$$k_n x \bar{p}_{n-1}(x) = \frac{k_n}{k_{n-1}} x p_{n-1}(x)$$

und mit (7.12) erhalten wir

$$k_n \beta_n \bar{p}_{n-1}(x) = \frac{k_n}{k_{n-1}} \left(\frac{k'_{n-1}}{k_{n-1}} - \frac{k'_n}{k_n} \right) p_{n-1}(x).$$

Die letzten drei Gleichungen mit (7.2) liefern dann die Behauptung. ■

■ **Beispiel 7.2.4** Die obige Sätze lassen sich **nicht übertragen** auf sogenannte *Kern-Funktionen* anstelle einer positiven Gewichtsfunktion. Eine Kernfunktion ist eine Abbildung $K : (a, b)^2 \rightarrow \mathbb{R}$. Mit ihrer Hilfe kann man ein Skalarprodukt wie folgt definieren

$$(f, g)_K := \int_a^b \int_a^b f(x) K(x, y) g(y) dy dx.$$

Solche Kernfunktionen sind z.B. in der Signal- und Bildverarbeitung und auch im Data Science von großer Bedeutung. Man spricht von einem *singulären Kern*, wenn $K(x, x) = \infty$. Das Beispiel $(a, b) = (-1, 1)$ mit dem singulären Kern $K(x, y) = \log|x - y|$ zeigt, dass die zugehörigen Orthogonalpolynome keine 3-Term-Rekursion erfüllen, was man wie folgt sieht: Symmetrieüberlegungen zeigen sofort, dass $(x^j, x^k)_K = 0$ für ungerade $j + k > 0$ gilt. Mit Hilfe von MAPLE® verifiziert man leicht durch mehrfaches Anwenden der Anweisungen

```
j:=0;
k:=1;
-int(int(x^j*y^k*log((x-y)^2)/2,x=-1..1),y=-1..1);
```

dass $(x^j, x^k)_K \neq 0$ für gerade $j + k \geq 0$ gilt. Definieren wir nun $p_0 := 1$, $p_1 = x$ und normieren

weitere p_k so, dass diese monisch sind. Damit $p_2(x) = x^2 + ax + b \in \mathbb{P}_2$ orthogonal zu x ist, muss $a = 0$ gelten und aus $(p_2, p_0)_K = 0$ folgt $b = -(x^2, 1)_K / (1, 1)_K$. Wenn die Drei-Term-Rekursion (7.3) gelten würde, hätten wir wegen $(x^2, x)_K = 0$, dass

$$\begin{aligned} p_2(x) &= (\alpha_2 + x)p_1(x) + \gamma_2 p_0(x) = \left(-\frac{(xp_1, p_1)_K}{(p_1, p_1)_K} + x \right) p_1(x) - \frac{(p_1, p_1)_K}{(p_0, p_0)_K} p_0(x) \\ &= \left(-\frac{(x^2, x)_K}{(x, x)_K} + x \right) x - \frac{(x, x)_K}{(1, 1)_K} = x^2 - \frac{(x, x)_K}{(1, 1)_K}. \end{aligned}$$

Dies müssen wir vergleichen mit $p_2(x) = x^2 + b$. Nun gilt aber $1 = (x, x)_K \neq (x^2, 1)_K = \frac{16}{9} - \frac{4}{3} \log(2)$ und somit gilt (7.3) nicht. ■

Anstelle der Normierung mithilfe des führenden Koeffizienten kann man auch orthonormale Polynome betrachten.

Korollar 7.2.5 Für die eindeutig bestimmten Orthonormalpolynome $(\tilde{p}_k)_{k \in \mathbb{N}_0}$ bzgl. $(\cdot, \cdot)_\omega$ gilt die Drei-Term-Rekursion

$$\sqrt{\gamma_{k+1}} \tilde{p}_{k+1}(x) = (\alpha_k + x) \tilde{p}_k(x) - \sqrt{\gamma_k} p_{k-1}(x) \quad k = 0, 1, 2, \dots, \quad (7.13)$$

$$p_{-1}(x) = 0, \quad p_0(x) = 1/\sqrt{\gamma_0}, \quad (7.14)$$

wobei α_k und γ_k durch (7.4) gegeben sind.

Natürlich gibt es auch weitere Möglichkeiten der Normierung, z.B. durch Festlegung des Wertes des führenden Koeffizienten $a_n \neq 1$.

Bemerkung 7.2.6 Es gibt auch eine explizite Formel für die Orthogonalpolynome, die sogenannte *Rodrigues-Formel*¹

$$p_n(x) = \frac{1}{e_n \omega(x)} \frac{d^n}{dx^n} \left\{ \omega(x) (g(x))^n \right\}, \quad n \in \mathbb{N}_0, \quad (7.15)$$

wobei g ein Polynom festen Grades ist und die Koeffizienten e_n geeignet zu wählen sind, vgl. Tabelle 7.1.²

p_n	Name	a	b	$\omega(x)$	Standard.	h_n	e_n	$g(x)$
P_n	<i>Legendre</i>	-1	1	1	$P_n(1) = 1$	$\frac{2^n}{2n+1}$	$(-2)^n n!$	$1 - x^2$
T_n	<i>Tschebyscheff</i>	-1	1	$1/\sqrt{1-x^2}$	$T_n(1) = 1$	$\left(\frac{\pi}{2}, n \neq 0\right)$	$(-2)^n n!$	$1 - x^2$
L_n	<i>Laguerre</i>	0	∞	e^{-x}	$k_n = (-1)^n / n!$	1	$1/n!$	x
H_n	<i>Hermite</i>	$-\infty$	∞	e^{-x^2}	$k_n = 2^n$	$\sqrt{\pi} 2^n n!$	$(-1)^n$	1
$P_n^{(\alpha, \beta)}$	<i>Jacobi</i> $(\alpha, \beta > -1)$	-1	1	$(1-x)^\alpha \cdot (1+x)^\beta$	$P_n^{(\alpha, \beta)}(1) = \binom{n+\alpha}{n}$	h_n^* ³	$(-2)^n n!$	$1 - x^2$

Tabelle 7.1: Beispiele von Orthogonalpolynomen.

¹Olinde Rodrigues, 1794-1851.

²Adrien-Marie Legendre, 1752-1833; Pafnuti Lwowitsch Tschebyschew (auch Čebyšëv), 1821-1894; Edmond Nicolas Laguerre, 1834-1886; Charles Hermite, 1822-1901.

7.2.1 Tschebyscheff-Polynome

Wir hatten die Tschebyscheff-Polynome schon in *Numerische Lineare Algebra* bei der Konvergenzanalyse des cg-Verfahrens gesehen. Wir werden ihnen auch in dieser Vorlesung später sowohl bei der Interpolation als auch bei der numerischen Integration noch einmal begegnen.

Definition 7.2.7 Die *Tschebyscheff-Polynome* $T_n : \mathbb{R} \rightarrow \mathbb{R}$, $n = 0, 1, \dots$, sind definiert als

$$T_n(x) := \frac{1}{2} \left[\left(x + \sqrt{x^2 - 1} \right)^n + \left(x - \sqrt{x^2 - 1} \right)^n \right], \quad x \in \mathbb{R}. \quad (7.16)$$

Lemma 7.2.8 — Tschebyscheff-Polynome. Die *Tschebyscheff-Polynome* T_n in (7.16) haben folgende Eigenschaften:

- (i) Sie sind orthogonal bezüglich $(\cdot, \cdot)_\omega$ auf $(-1, 1)$ mit $\omega(x) := \frac{1}{\sqrt{1-x^2}}$ und werden standardisiert durch $T_n(1) = 1$;
- (ii) Sie haben stets ganzzahlige Koeffizienten;
- (iii) Der höchste Koeffizient von T_n ist $a_n = 2^{n-1}$;
- (iv) T_n ist eine gerade Funktion, falls n gerade und eine ungerade, falls n ungerade ist;
- (v) $T_n(-1) = (-1)^n$;
- (vi) $|T_n(x)| \leq 1$ für $x \in [-1, 1]$;
- (vii) Die Nullstellen von $T_n(x)$ lauten $\hat{x}_n^{(j)} := \cos\left(\frac{2j-1}{2n}\pi\right)$, $j = 1, \dots, n$;
- (viii) Es gilt die Darstellung

$$T_n(x) = \begin{cases} \cos(n \cdot \arccos(x)), & -1 \leq x \leq 1; \\ \cosh(n \cdot \operatorname{arcosh}(x)), & x \geq 1; \\ (-1)^n \cosh(n \cdot \operatorname{arcosh}(-x)), & x \leq -1; \end{cases}$$

- (ix) $|T_n(x)|$ nimmt seinen maximalen Wert im Intervall $[-1, 1]$ an den sogenannten *Tschebyscheff-Abszissen* $\bar{x}_n^{(j)} := \cos\left(\frac{j\pi}{n}\right)$ für $j = 0, \dots, n$ an, d.h.

$$|T_n(x)| = 1 \Leftrightarrow x = \bar{x}_n^{(j)} \quad j = 0, \dots, n.$$

- (x) Für $x \in \mathbb{R}$ erfüllen die Tschebyscheff-Polynome die folgende *Drei-Term-Rekursion*

$$T_0(x) = 1, \quad T_1(x) = x, \quad T_n(x) = 2x T_{n-1}(x) - T_{n-2}(x), \quad n \geq 2.$$

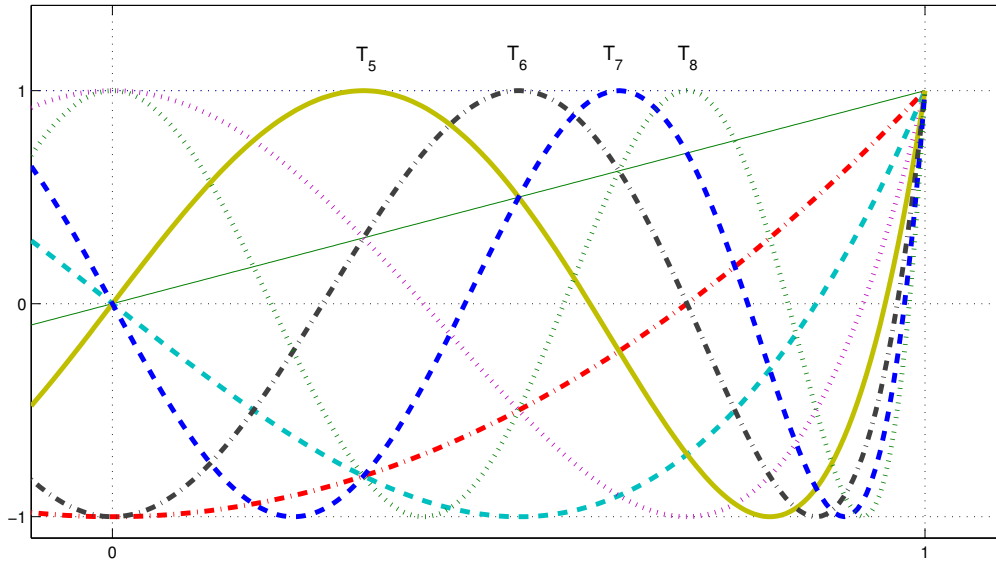
Beweis. Übung. ■

Wir zeigen nun, dass die Tschebyscheff-Polynome T_n das sogenannte *Minimax-Problem* (vgl. auch später (6.21)) lösen. Dies wird bei der Interpolation und numerischen Integration eine zentrale Rolle spielen.

Satz 7.2.9 Sei $p \in \mathbb{P}_n$ mit führendem Koeffizienten $a_n \neq 0$. Dann gilt

- (i) $|p(x)| \geq |a_n|/2^{n-1}$ für alle $x \in [-1, 1]$;

³Für die Jacobi-Polynome gilt $h_n^* = h_n(P_n^{(\alpha, \beta)}) = \frac{2^{\alpha+\beta+1}}{2n+\alpha+\beta+1} \frac{\Gamma(n+\alpha+1)\Gamma(n+\beta+1)}{n!\Gamma(n+\alpha+\beta+1)}$.

Abbildung 7.1: Tschebyscheff-Polynome T_1, \dots, T_8 .

- (ii) $T_n = \arg \min \{ \|p\|_\infty : p \in \mathbb{P}_n \text{ mit f\"uhrendem Koeffizienten } a_n = 2^{n-1} \}$ mit der Maximumsnorm $\|f\|_\infty = \max_{x \in [-1,1]} |f(x)|$.

Beweis. Wir zeigen zunächst (ii). Nach Lemma 7.2.8 (v) und (vi) gilt $\|T_n\|_\infty = 1$. Angenommen es gäbe ein $T_n \neq p_n \in \mathbb{P}_n$ mit führendem Koeffizienten $a_n = 2^{n-1}$ und $|p_n(x)| < 1$ für alle $x \in [-1, 1]$. Dann ist $T_n - p_n \in \mathbb{P}_{n-1}$, da beide den gleichen führenden Koeffizienten besitzen. An den Tschebyscheff-Abszissen $\bar{x}_n^{(j)} := \cos(\frac{j\pi}{n})$ gilt nach Lemma 7.2.8 (ix)

$$\begin{aligned} T_n(\bar{x}_n^{(2j)}) &= 1, \quad p_n(\bar{x}_n^{(2j)}) < 1 \Rightarrow p_n(\bar{x}_n^{(2j)}) - T_n(\bar{x}_n^{(2j)}) < 0, \\ T_n(\bar{x}_n^{(2j+1)}) &= -1, \quad p_n(\bar{x}_n^{(2j+1)}) > -1 \Rightarrow p_n(\bar{x}_n^{(2j+1)}) - T_n(\bar{x}_n^{(2j+1)}) > 0 \end{aligned}$$

Also ist $T_n - p_n$ an den $(n+1)$ Tschebyscheff-Abszissen abwechselnd positiv und negativ und besitzt damit mindestens n Nullstellen in $[-1, 1]$. Damit gilt $p_{n-1} \ni T_n - p_n \equiv 0$, also $p_n = T_n$ und damit ein Widerspruch. Damit ist zum Einen (ii) bewiesen und zum Anderen gezeigt, dass es zu jedem $p \in \mathbb{P}_n$ mit führendem Koeffizienten $a_n = 2^{n-1}$ ein $x \in [-1, 1]$ geben muss, so dass $|p(x)| > 1$.

Sei $p \in \mathbb{P}_n$ mit führendem Koeffizienten $a_n \neq 0$. Dann gilt $\tilde{p}_n := \frac{2^{n-1}}{a_n} p_n$ und \tilde{p}_n hat den führenden Koeffizienten $\tilde{a}_n = 2^{n-1}$. Nach (ii) gibt es ein $x \in [-1, 1]$ mit $|\tilde{p}_n(x)| > 1$ und damit folgt (i). ■

7.2.2 Legendre-Polynome

Die Legendre-Polynome $P_n \in \mathbb{P}_n$, $n \in \mathbb{N}_0$ sind orthogonal bezüglich des „klassischen“⁴ Skalarprodukts auf $[-1, 1]$ also für $\omega \equiv 1$,

$$(f, g) := \int_{-1}^1 f(x) g(x) dx,$$

⁴Das bedeutet $\omega \equiv 1$.

und sind durch $P_n(1) = 1$ standardisiert.

Lemma 7.2.10 Die Legendre-Polynome besitzen folgende Eigenschaften:

(i) Sie erfüllen die Drei-Term-Rekursion

$$P_0(x) = 1, P_1(x) = x, \quad (n+1)P_{n+1}(x) = (2n+1)xP_n(x) - nP_{n-1}(x), \quad n \in \mathbb{N}$$

(ii) $P_n(1) = 1, P_n(-1) = (-1)^n, n \in \mathbb{N}_0$.

(iii) $\|P_n\|^2 = \frac{2}{2n+1}, n \in \mathbb{N}_0$.

(iv) Für Ableitung und Stammfunktion gelten

$$P'_n(x) = \frac{n(n+1)}{2n+1} \frac{P_{n+1}(x) - P_{n-1}(x)}{x^2 - 1}, \quad n \geq 1, \quad (7.17)$$

$$\int_{-1}^x P_n(\xi) d\xi = \frac{1}{2n+1} \left(P_{n+1}(x) - P_{n-1}(x) \right), \quad n \geq 1. \quad (7.18)$$

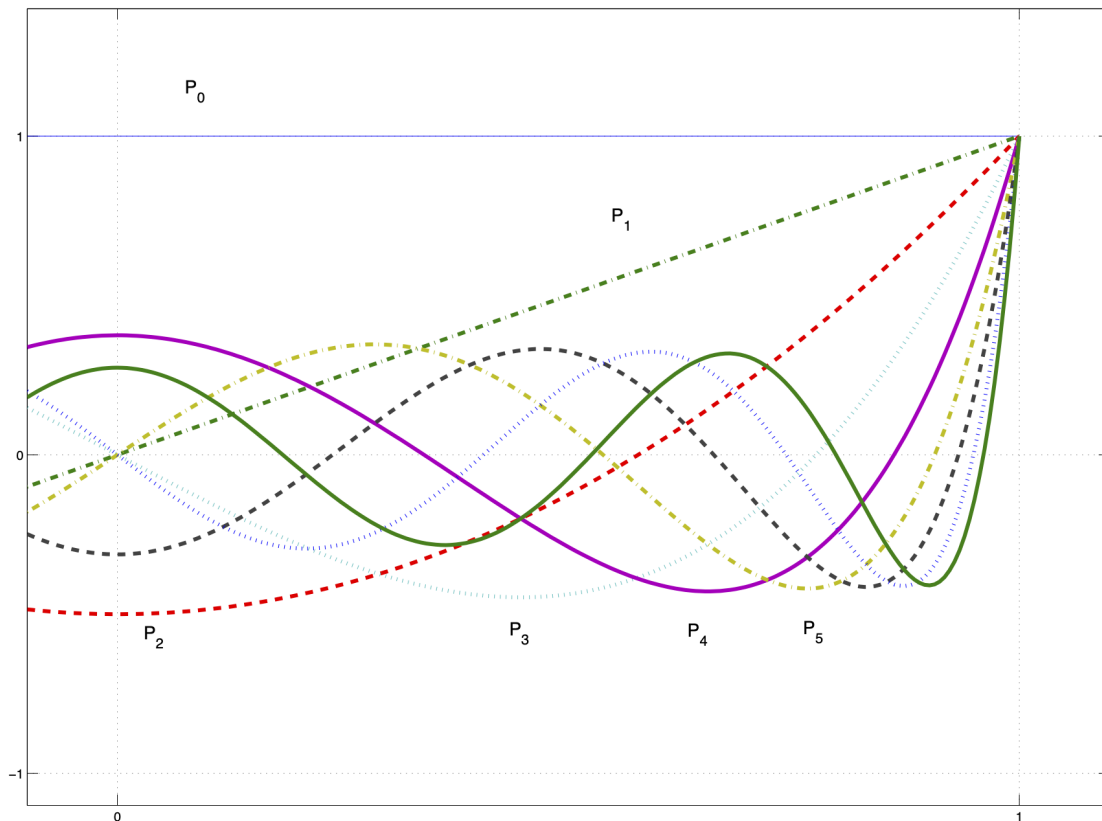


Abbildung 7.2: Legendre-Polynome $P_n, n = 0, \dots, 8$.

7.2.3 Jacobi-Polynome

Die Jacobi-Polynome $P_n^{(\alpha, \beta)}$ sind orthogonal auf $(-1, 1)$ bezüglich der Gewichtsfunktion $(1-x)^\alpha(1+x)^\beta$ mit $\alpha, \beta > -1$. Für $\alpha = \beta = 0$ erhalten wir die Legendre-Polynome und für $\alpha = \beta = -1/2$ die Tschebyscheff-Polynome.

Satz 7.2.11 — [Sze67]. Seien $|\alpha|, |\beta| \leq 1/2$ und $x_k = \cos \theta_k$, $k = 1, \dots, n$, die Nullstellen von $P_n^{(\alpha, \beta)}(x)$ in absteigender Folge, d.h.

$$1 > x_1 > x_2 > \dots > x_n > -1; \text{ also } 0 < \theta_1 < \theta_2 < \dots < \theta_n < \pi.$$

Dann gilt

$$\frac{k + (\alpha + \beta - 1)/2}{n + (\alpha + \beta + 1)/2} \pi < \theta_k < \frac{k}{n + (\alpha + \beta + 1)/2} \pi, \quad k = 1, 2, \dots$$

und im Falle $\alpha = \beta$ gilt

$$\theta_k \geq \frac{k + \alpha/2 - 1/4}{n + \alpha + 1/2} \pi, \quad k = 1, 2, \dots, \left[\frac{n}{2}\right],$$

wobei die Gleichheit für $\alpha = \beta = -1/2$ oder $\alpha = \beta = 1/2$ gilt.

7.3 Nullstellen von Orthogonalpolynomen

Wie bereits erwähnt, spielen Nullstellen von Orthogonalpolynomen bei der numerischen Integration eine wichtige Rolle.

Satz 7.3.1 Es sei $\omega \in C(a, b)$ eine positive Gewichtsfunktion, $(\cdot, \cdot)_\omega$ das zugehörige Skalarprodukt. Dann hat ein Orthogonalpolynom p_k von echtem Grad k genau k einfache Nullstellen in (a, b) .

Beweis. Es seien x_1, \dots, x_m die $m \leq k$ verschiedenen Nullstellen $a < x_i < b$, an denen p_k sein Vorzeichen wechselt. Das Polynom $q(x) := (x - x_1) \cdots (x - x_m) \in \mathbb{P}_m$ wechselt dann an den gleichen Stellen sein Vorzeichen, sodass die Funktion $\omega p_k q$ aufgrund der Positivität von ω ihr Vorzeichen in (a, b) nicht ändert. Daher gilt

$$(q, p_k)_\omega = \int_a^b q(x) p_k(x) \omega(x) dx \neq 0.$$

Da jedoch $p_k \perp_\omega \mathbb{P}_{k-1}$, folgt unmittelbar, dass $m = \text{Grad } q \geq k$ und damit $m = k$. ■

Der obige Satz ist ein reines Existenzresultat, er liefert keine Möglichkeit zur Berechnung der Nullstellen. Der Schlüssel hierzu ist die Drei-Term-Rekursion (7.7) aus Satz 7.2.3. Schematisch dargestellt bedeutet dies:

$$\begin{array}{lll} p_1 = (a_1 + b_1 x) p_0 & \Leftrightarrow & -\frac{a_1}{b_1} \cdot p_0 + \frac{1}{b_1} \cdot p_1 = x p_0 \\ p_2 = (a_2 + b_2 x) p_1 + c_2 p_0 & \Leftrightarrow & -\frac{c_2}{b_2} \cdot p_0 - \frac{a_2}{b_2} \cdot p_1 + \frac{1}{b_2} \cdot p_2 = x p_1 \\ \vdots & & \vdots \\ p_{n+1} = (a_{n+1} + b_{n+1} x) p_n + c_{n+1} p_{n-1} & \Leftrightarrow & -\frac{c_{n+1}}{b_{n+1}} \cdot p_{n-1} - \frac{a_{n+1}}{b_{n+1}} \cdot p_n = x p_n - \frac{1}{b_{n+1}} \cdot p_{n+1} \end{array}$$

Aus dem Schema ist aber ersichtlich, dass wir die Drei-Term-Rekursion auch als lineares

Gleichungssystem interpretieren können, d.h.

$$\underbrace{\begin{pmatrix} -\frac{a_1}{b_1} & \frac{1}{b_1} & 0 & \cdots & \cdots & 0 \\ -\frac{c_2}{b_2} & -\frac{a_2}{b_2} & \frac{1}{b_2} & & & \vdots \\ 0 & -\frac{c_3}{b_3} & -\frac{a_3}{b_3} & \frac{1}{b_3} & & \vdots \\ \vdots & & \ddots & \ddots & & 0 \\ \vdots & & & \ddots & \ddots & \frac{1}{b_n} \\ 0 & \cdots & 0 & -\frac{c_{n+1}}{b_{n+1}} & -\frac{a_{n+1}}{b_{n+1}} & \end{pmatrix}}_{=: \mathbf{A}_n} \cdot \underbrace{\begin{pmatrix} p_0(x) \\ \vdots \\ p_n(x) \end{pmatrix}}_{=: \mathbf{p}_n} = x \cdot \underbrace{\begin{pmatrix} p_0(x) \\ \vdots \\ p_n(x) \end{pmatrix}}_{=: \mathbf{p}_n} + \underbrace{\begin{pmatrix} 0 \\ \vdots \\ 0 \\ -\frac{p_{n+1}(x)}{b_{n+1}} \end{pmatrix}}_{=: \mathbf{r}_n}$$

also in Matrixschreibweise also

$$\mathbf{A}_n \mathbf{p}_n = x \mathbf{p}_n + \mathbf{r}_n. \quad (7.19)$$

Korollar 7.3.2 — Nullstellen von Orthogonalpolynomen. Die Nullstellen von p_{n+1} sind die Eigenwerte der Matrix \mathbf{A}_n , d.h., die Lösungen der Gleichung $\mathbf{A}_n \mathbf{p}_n = x^{(n)} \mathbf{p}_n$ für $\mathbf{p}_n \neq 0$. Die Eigenvektoren zu Eigenwerten $x_k^{(n)}$, $k = 1, \dots, n+1$, lauten bis auf ein Vielfaches $\mathbf{p}_n^{(k)} := (p_0(x_k^{(n)}), \dots, p_n(x_k^{(n)}))^T$, $k = 0, \dots, n$.

Das obige Eigenwertproblem lässt sich mit den Methoden aus der Vorlesung *Numerische Lineare Algebra* effizient lösen. Man beachte, dass die Normierung der Polynome keinen Einfluss auf die Nullstellen hat.

Darstellung der Orthogonalpolynome

Für die spätere Berechnung von Quadraturformeln bei der Gauß-Quadratur benötigen wir eine wichtige Identität für orthogonale Polynome. Diese Identität ist als Christoffel-Darboux-Formel⁵ bekannt.

Satz 7.3.3 — Christoffel-Darboux. Sei $(p_n)_{n \in \mathbb{N}_0}$ eine Folge orthogonaler Polynome der Form (7.6). Dann gilt mit h_n gemäß für alle $x, y \in \mathbb{R}$ (7.8)

$$\sum_{i=0}^n \frac{p_i(x)p_i(y)}{h_i} = \frac{k_n}{k_{n+1} \cdot h_n} \cdot \frac{p_{n+1}(x)p_n(y) - p_n(x)p_{n+1}(y)}{x - y}. \quad (7.20)$$

Beweis. Für $n = 0$ erhält man die Identität (7.20) durch Einsetzen von $p_0(x) = k_0$ und $p_1(x) = k_1x + c_1$ mit einer Konstanten $c_1 \in \mathbb{R}$. Dann lautet die linke Seite $\frac{p_0(x)p_0(y)}{h_0} = \frac{k_0^2}{h_0}$ und die rechte Seite

$$\frac{k_0}{k_1 \cdot h_0} \cdot \frac{p_1(x)p_0(y) - p_0(x)p_1(y)}{x - y} = \frac{k_0}{k_1 \cdot h_0} \cdot \frac{(k_1x + c_1)k_0 - k_0(k_1y + c_1)}{x - y} = \frac{k_0^2}{h_0}.$$

Sei nun $n \geq 1$. Dann folgt mit (7.7) durch jeweiliges Einsetzen für p_{n+1}

$$\begin{aligned} p_{n+1}(x)p_n(y) - p_n(x)p_{n+1}(y) &= ((a_{n+1} + b_{n+1}x)p_n(x) + c_{n+1}p_{n-1}(x))p_n(y) \\ &\quad - p_n(x)((a_{n+1} + b_{n+1}y)p_n(y) + c_{n+1}p_{n-1}(y)) \end{aligned}$$

⁵Elwin Bruno Christoffel, 1829-1900; Jean-Gaston Darboux, 1842-1917.

$$= b_{n+1}(x-y)p_n(x)p_n(y) + c_{n+1}(p_{n-1}(x)p_n(y) - p_n(x)p_{n-1}(y)).$$

Mit (7.8) erhalten wir für $x \neq y$ ($b_{n+1} = k_{n+1}/k_n \neq 0$ nach Voraussetzung)

$$\begin{aligned} & \frac{1}{b_{n+1}} \cdot \frac{p_{n+1}(x)p_n(y) - p_n(x)p_{n+1}(y)}{x-y} \\ &= p_n(x)p_n(y) + \frac{c_{n+1}}{b_{n+1}} \cdot \frac{p_{n-1}(x)p_n(y) - p_n(x)p_{n-1}(y)}{x-y}, \text{ d.h.} \\ & \frac{k_n}{k_{n+1}} \cdot \frac{p_{n+1}(x)p_n(y) - p_n(x)p_{n+1}(y)}{x-y} \\ &= p_n(x)p_n(y) + \frac{k_{n+1}k_{n-1}h_nk_n}{k_n^2 h_{n-1} k_{n+1}} \cdot \frac{p_n(x)p_{n-1}(y) - p_{n-1}(x)p_n(y)}{x-y} \\ &= p_n(x)p_n(y) + \frac{k_{n-1}h_n}{k_n h_{n-1}} \cdot \frac{p_n(x)p_{n-1}(y) - p_{n-1}(x)p_n(y)}{x-y}. \end{aligned} \tag{7.21}$$

Gelte nun (7.20) für ein $n-1 \geq 0$ dann schließen wir mit (7.21) auf n wie folgt

$$\begin{aligned} & \frac{k_n}{k_{n+1}} \cdot \frac{p_{n+1}(x)p_n(y) - p_n(x)p_{n+1}(y)}{x-y} \\ &= p_n(x)p_n(y) + \frac{k_{n-1}h_n}{k_n h_{n-1}} \cdot \frac{p_n(x)p_{n-1}(y) - p_{n-1}(x)p_n(y)}{x-y} \\ &= p_n(x)p_n(y) + h_n \sum_{i=0}^{n-1} \frac{p_i(x)p_i(y)}{h_i}. \end{aligned}$$

Division durch h_n liefert dann die Behauptung. ■

Kapitel 8

Numerische Integration

Nach dem Hauptsatz der Integral- und Integralrechnung existiert zu jeder stetigen Funktion eine Stammfunktion. Doch der Satz von Liouville¹ besagt weiter, dass nicht alle Stammfunktionen mit Hilfe der elementaren Funktionen (wie den ganz-rationalen Funktionen, den Wurzelfunktionen, den trigonometrischen Funktionen und ihren Umkehrfunktionen oder auch der Exponential- und der Logarithmusfunktion) dargestellt werden können. Dies trifft bereits auch schon so einfache Funktionen zu wie:

$$e^{-x^2}, \quad \frac{\sin x}{x}, \quad \frac{\log x}{x-1}, \quad x^x, \quad \sqrt{\cos x}, \quad \sqrt{\log x}, \quad \sqrt{1+x^{-4}}, \quad \sqrt[6]{1-x^{-7}}, \quad \frac{1}{\log x}.$$

Daher stellen wir numerische Methoden zur näherungsweisen Berechnung von Integralen vor, die nicht mit Formeln für deren Stammfunktion ausgedrückt werden können. Wir beschränken uns auf die Berechnung des Riemann-Integrals² einer Funktion $f : [a, b] \rightarrow \mathbb{R}$, d.h.,

$$I(f) := \int_a^b f(x) \, dx.$$

Wir wollen stets annehmen, dass die Funktion f zumindest an ausgewählten Knoten auswertbar ist. Da für viele solcher Funktionen f das Riemann- und das Lebesgue-Integral³ identisch sind, spielt die Wahl des Riemann-Integrals hier zunächst keine Rolle.

8.1 Mittelpunkts- und Trapezregel

Wir beginnen mit sehr einfachen Näherungen für Integrale. Die entstehenden Formeln heißen auch *Quadraturformeln*, da sie zur Berechnung des Flächeninhalts eines von Kurven begrenzten Bereiches verwendet wurden und werden.

¹Joseph Liouville, 1809-1882.

²Georg Friedrich Bernhard Riemann, 1826-1866.

³Henri Léon Lebesgue, 1875-1941.

Die Mittelpunktsregel

Nehmen wir an, dass wir Integral einer Funktion $f \in C([a, b])$ berechnen wollen. Wir betrachten dazu die **Mittelpunktsregel**

$$Q^{\text{Mi}}[f] := (b - a) \cdot f\left(\frac{a+b}{2}\right). \quad (8.1)$$

Den Fehler können wir leicht mit dem Taylor-Restglied abschätzen:

$$\begin{aligned} \left| \int_a^b f(x) dx - Q^{\text{Mi}}[f] \right| &= \left| \int_a^b \left(f(x) - f\left(\frac{a+b}{2}\right) \right) dx \right| \\ &\leq \int_a^b \left| f(x) - f\left(\frac{a+b}{2}\right) \right| dx \leq \left(\sup_{x \in [a, b]} |f'(x)| \right) \int_a^b \left| x - \frac{a+b}{2} \right| dx \\ &= \frac{1}{4} (b - a)^2 \|f'\|_{\infty}, \end{aligned}$$

also quadratisch in der Intervalllänge $b - a$, falls $f \in C^1([a, b])$.

Wir erkennen in diesem Beispiel auch einen allgemeinen Zugang, den wir weiter verfolgen werden. Mit $x_0 := \frac{a+b}{2}$ können wir die Mittelpunktsregel auch dadurch motivieren, dass wir

1. die Funktion f zunächst durch das Interpolationspolynom $p_0 = P(f|x_0) \in \mathbb{P}_0$ ersetzen und dann
2. die Quadraturformal als exaktes Integral über das Interpolationspolynom wählen, $Q[f] := \int_a^b p_0 dx$.

Dann können wir den Quadraturfehler auch direkt mittels der Darstellung des Interpolationsfehlers in Satz 6.3.7 abschätzen. Mit einem $\xi \in (a, b)$ gilt

$$\left| \int_a^b f(x) dx - Q^{\text{Mi}}[f] \right| \leq \int_a^b |f(x) - (f|x_0)| dx = \int_a^b \left| \frac{x-x_0}{1!} f'(\xi) \right| dx$$

und dies führt auf die gleiche Abschätzung.

Eine weitere Beobachtung ist, dass die Mittelpunktsregel **exakt** für lineare Funktionen ist, also

$$I(p) = Q^{\text{Mi}}[p] \text{ für alle } p \in \mathbb{P}_1. \quad (8.2)$$

Damit können wir die Fehlabschätzung für Funktionen $f \in C^2([a, b])$ verbessern: Sei $p_1 \in \mathbb{P}_1$ das eindeutige Hermite-Interpolationspolynom an f mit $p_1(x_0) = f(x_0)$, $p_1'(x_0) = f'(x_0)$ und $x_0 = \frac{a+b}{2}$. Mit (8.2) folgt $I(p_1) = Q^{\text{Mi}}[p_1]$ und dann wiederum mit Satz 6.3.7

$$\begin{aligned} \left| \int_a^b f(x) dx - Q^{\text{Mi}}[f] \right| &= \left| \int_a^b (f(x) - p_1(x)) dx \right| = \left| \int_a^b (f(x) - P(f|x_0, x_0)(x)) dx \right| \\ &\leq \int_a^b |f(x) - P(f|x_0, x_0)(x)| dx \\ &= \int_a^b \left| \frac{(x-x_0)^2}{2!} f''(\xi) \right| dx \leq \frac{1}{24} (b - a)^3 \|f''\|_{\infty}, \end{aligned}$$

also kubisch in der Intervalllänge!

Bemerkung 8.1.1 Man kann sich leicht überlegen, dass die Mittelpunktsregel optimal bzgl. aller Quadraturformeln mit nur einem Knoten x_0 ist. Insbesondere ist der Fehler der linken ($x_0 = a$) und rechten ($x_0 = b$) Rechteckregel nur linear in der Intervalllänge.

Die Trapezregel

Bei der Mittelpunktsregel haben wir nur einen Knoten x_0 , den Mittelpunkt, verwendet, um daraus eine Quadraturformel zu konstruieren. Als nächste Idee liegt es nahe, die beiden Endpunkte des Intervalls, also $x_0 = a$ und $x_1 = b$ als Interpolationsknoten zu wählen und so die **Trapezregel**

$$Q^{\text{Tr}}[f] := I(P(f|a, b)) = \frac{b-a}{2}(f(a) + f(b)) \quad (8.3)$$

zu erhalten. Die Analyse des Fehlers ist vollkommen analog zur Mittelpunktsregel:

$$\begin{aligned} \left| \int_a^b f(x) dx - Q^{\text{Tr}}[f] \right| &= \left| \int_a^b (f(x) - P(f|a, b)(x)) dx \right| \\ &\leq \int_a^b |f(x) - P(f|a, b)(x)| dx \\ &= \int_a^b \left| \frac{(x-a)(x-b)}{2!} f''(\xi) \right| dx \leq \frac{1}{12} (b-a)^3 \|f''\|_{\infty}, \end{aligned}$$

für $f \in C^2([a, b])$. Auch die Trapezregel ist exakt für lineare Funktionen im Sinne von (8.2).

Summierte Mittelpunkts- und Trapezregel

Wir haben gesehen, dass sich der Quadraturfehler quadratisch bzw. kubisch in der Intervalllänge verhält. Was tun wir, wenn das Intervall groß ist? Die Antwort ist naheliegend und einfach: wir unterteilen das Intervall und wenden auf jedem Teilintervall die Mittelpunkts- oder Trapezregel an.

Sei dazu $n \in \mathbb{N}$ die Anzahl der Teilintervalle, $h := \frac{b-a}{n}$ die Schrittweite und $x_k := k h$, $k = 0, \dots, n$, die Knoten. Dann verwenden wir den Ansatz

$$\int_a^b f(x) dx = \sum_{k=0}^{n-1} \int_{x_k}^{x_{k+1}} f(x) dx \approx \sum_{k=0}^{n-1} Q_n^k[f],$$

wobei Q_n^k eine Quadraturformel auf dem Teilintervall $[x_k, x_{k+1}]$, $k = 0, \dots, n-1$, ist. Wählen wir Q_n^k als die Mittelpunktsregel, dann ergibt sich die **summierte Mittelpunktsregel**

$$Q_n^{\text{Mi}}[f] := h \sum_{k=1}^n f(x_{k-1/2}), \quad \text{mit } x_{k-1/2} := a + (k - \frac{1}{2})h, \quad k = 1, \dots, n, \quad (8.4)$$

bzw. bei Wahl von Q_n^k als die Trapezregel analog die **summierte Trapezregel**

$$Q_n^{\text{Tr}}[f] := h \left(\frac{f(a)}{2} + \sum_{k=1}^{n-2} f(x_k) + \frac{f(b)}{2} \right). \quad (8.5)$$

Die Fehleranalyse ergibt sich durch Summation unter der Annahme, dass der Quadraturfehler in jedem Teilintervall durch $C h^p$ mit einer Konstante C , der Teilintervalllänge h

und einer Ordnung $p \geq 1$ beschränkt werden kann. Dann gilt

$$\left| \int_a^b f(x) dx - Q_n[f] \right| \leq \sum_{k=0}^{n-1} \left| \int_{x_k}^{x_{k+1}} f(x) dx - Q_n^k[f] \right| \leq C \sum_{k=0}^{n-1} h^p = C(b-a) h^{p-1},$$

da $n = \frac{b-a}{h}$ ist. Wir verlieren also eine Ordnung und erhalten für die summierte Mittelpunktsregel

$$\left| \int_a^b f(x) - Q_n^{\text{Mi}}[f] dx \right| \leq \frac{1}{24} (b-a) h^2 \|f''\|_{\infty} \quad (8.6)$$

und für die summierte Trapezregel

$$\left| \int_a^b f(x) - Q_n^{\text{Tr}}[f] dx \right| \leq \frac{1}{12} (b-a) h^2 \|f''\|_{\infty}, \quad (8.7)$$

jeweils für $f \in C^2([a, b])$. Die beiden summierten Regeln sind in Abbildung 8.1 dargestellt.

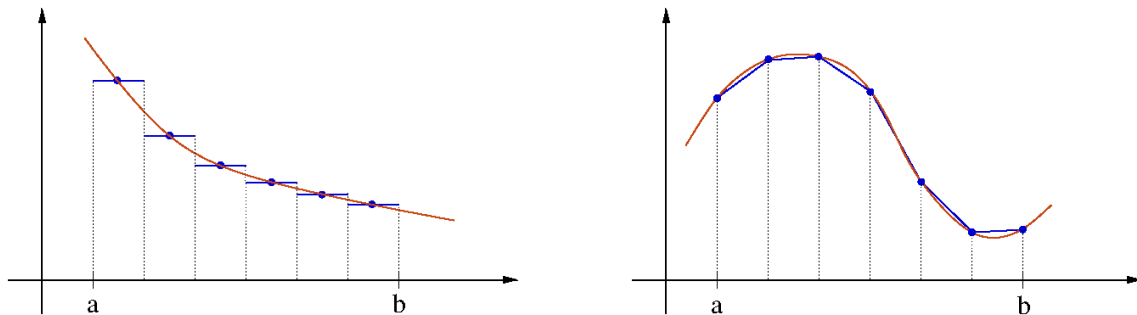


Abbildung 8.1: Summierte Mittelpunkts- (links) und summierte Trapezformel (rechts).

MAPLE-Beispiel:

Möchten wir das Integral

$$\int_0^{e^{-2}} \frac{dx}{\log|x|}$$

```
> restart:
> n:=10^500:
> 2*n*int(-1/log(x), x=0..exp(-2)/(2*n)):
> evalf(%);
```

mit einem Fehler $< \frac{1}{2}10^{-4}$ und einer summierten Trapezformel berechnen, so reichen dazu $n = 10^{500}$ Knoten nicht aus!

0.0001171749067

8.2 Newton-Cotes-Formeln

Wir haben sowohl die Mittelpunkts- als auch die Trapezregel als exaktes Integral eines Interpolationspolynoms an den Integranden schreiben können. Nutzt man dies als allgemeines Konstruktionsprinzip, so erhält man die *Newton-Cotes-Formeln*, eine ganze Klasse von Quadraturverfahren, wobei Mittelpunkts- und Trapezregel Spezialfälle sind.

Definition 8.2.1 (i) Sei $f \in C([a, b])$, $a \leq x_0 \leq \dots \leq x_n \leq b$. Dann heißt

$$Q_n[f] := I(P(f|x_0, \dots, x_n))$$

interpolatorische Quadraturformel bzgl. der Knoten x_0, \dots, x_n .

(ii) Eine Quadraturformel heißt **exakt vom Grad** $k \in \mathbb{N}$ (oder auch von der **Ordnung** $k+1$), falls $Q[p] = I(p)$ für alle $p \in \mathbb{P}_k$ und k dabei maximal ist.

Interpolatorische Quadraturformeln Q_n sind exakt vom Grad n .

Lemma 8.2.2 Zu $(n+1)$ paarweise verschiedenen Knoten $x_0, \dots, x_n \in [a, b]$ gibt es genau eine Quadraturformel

$$Q_n[f] = (b-a) \sum_{i=0}^n \lambda_i^{(n)} f(x_i), \quad (8.8)$$

die exakt vom Grad n ist. Die Zahlen $\lambda_i^{(n)} \in \mathbb{R}$ heißen **Gewichte**.

Beweis. Sei $p \in \mathbb{P}_n$, dann gilt $p = \sum_{i=0}^n p(x_i) L_{n,i}$ mit den Lagrange-Basis-Polynomen $L_{n,i}$ aus (6.2). Dann gilt einerseits, dass

$$I(p) = \int_a^b p(x) dx = \sum_{i=0}^n p(x_i) \int_a^b L_{n,i}(x) dx$$

und andererseits $Q_n[p] = (b-a) \sum_{i=0}^n \lambda_i^{(n)} p(x_i)$. Für

$$\lambda_i^{(n)} := \frac{1}{(b-a)} \int_a^b L_{n,i}(x) dx \quad (8.9)$$

erhalten wir $I(p) = Q_n[p]$ für alle $p \in \mathbb{P}_n$. Die Eindeutigkeit sieht man, indem man $p = L_{n,j}$, $j = 0, \dots, n$, einsetzt und so erkennt, dass die Wahl der Gewichte eindeutig ist. ■

Bemerkung 8.2.3 Die Gewichte $\lambda_j^{(n)}$ können auch durch Lösen eines linearen Gleichungssystems berechnet werden. Sei p_0, \dots, p_n eine beliebige Basis von \mathbb{P}_n , dann gilt

$$\int_a^b p_j dx = \sum_{k=0}^n \lambda_k^{(n)} p_j(x_k) \quad \text{für alle } j = 0, \dots, n.$$

Diese Gleichung lässt sich als lineares Gleichungssystem formulieren:

$$\begin{pmatrix} p_0(x_0) & \cdots & p_0(x_n) \\ \vdots & & \vdots \\ p_n(x_0) & \cdots & p_n(x_n) \end{pmatrix} \begin{pmatrix} \lambda_0^{(n)} \\ \vdots \\ \lambda_n^{(n)} \end{pmatrix} = \begin{pmatrix} \int_a^b p_0 dx \\ \vdots \\ \int_a^b p_n dx \end{pmatrix}. \quad (8.10)$$

Die Matrix auf der linken Seite von (8.10) ist eine transponierte Vandermonde-Matrix und deshalb regulär für paarweise verschiedene Knoten.

MATLAB-Beispiel:

Wir betrachten das Intervall $[0, 1]$ mit der Monombasis. Das Gleichungssystem (8.10) lautet nun

$$\begin{pmatrix} 1 & \cdots & 1 \\ x_0 & \cdots & x_n \\ \vdots & & \vdots \\ x_0^n & \cdots & x_n^n \end{pmatrix} \begin{pmatrix} \omega_0 \\ \vdots \\ \omega_n \end{pmatrix} = \begin{pmatrix} 1 \\ \frac{1}{2} \\ \vdots \\ \frac{1}{n+1} \end{pmatrix} \quad (8.11)$$

und für $x_j = \frac{j}{n}$ erhalten wir die Gewichte der zugehörigen (abgeschlossenen, s.u.) Newton-Cotes-Gewichte. Die folgende Matlab-Funktion berechnet die Stützstellen x_j und die zugehörigen Gewichte λ_j .

```
1 function [x,lambda] = ClosedNewtonCotes(n)
2 x = [0:n]/n;
3 A = (ones(n+1,1)*x).^([0:n]'+ones(1,n+1));
4 b = 1./[1:n+1]';
5 lambda = A\b;
6 % kompakte Version: lambda = (1./[1:n+1])/fliplr(vander([0:n]/n))
```

Definition 8.2.4 — Newton-Cotes-Formeln. Seien $a \leq x_0 < x_1 < \cdots < x_n \leq b$ äquidistant, d.h. $h = x_{k+1} - x_k$ für alle $k = 0, \dots, n-1$. Die resultierende interpolatorische Quadraturformel Q_n heißt eine **Newton-Cotes-Formel**.

i) Sie heißt **abgeschlossen**, wenn

$$x_0 = a \text{ und } x_n = b, \text{ also } h = \frac{b-a}{n} \text{ und } x_k := a + k h, k = 0, \dots, n;$$

ii) und **offen**, wenn $h = \frac{b-a}{n+2}$, $x_k := a + (k+1)h$, $k = 0, \dots, n$.

Interpolatorische Quadraturformeln zu $h = \frac{b-a}{n+1}$ und $x_k = a + (i + \frac{1}{2})h$, $k = 0, \dots, n$, werden als **Euler-MacLaurin-Formeln** bezeichnet.⁴

Die Formel für die Gewichte $\lambda_i^{(n)}$ der abgeschlossenen Newton-Cotes-Formeln verein-

⁴Leonhard Euler, 1707-1783; Colin Maclaurin, 1698-1746.

facht sich durch die Substitution $s := \frac{(x-a)}{h}$ zu

$$\lambda_i^{(n)} = \frac{1}{b-a} \int_a^b \prod_{\substack{j=0 \\ j \neq i}}^n \frac{(x-x_j)}{(x_i-x_j)} dx = \frac{1}{n} \int_0^n \prod_{\substack{j=0 \\ j \neq i}}^n \frac{(s-j)}{(i-j)} ds, \quad (8.12)$$

und für die offenen Newton-Cotes-Formeln erhält man

$$\lambda_i^{(n)} = \frac{1}{b-a} \int_a^b \prod_{\substack{j=0 \\ j \neq i}}^n \frac{(x-x_j)}{(x_i-x_j)} dx = \frac{1}{n+2} \int_{-1}^{n+1} \prod_{\substack{j=0 \\ j \neq i}}^n \frac{(s-j)}{(i-j)} ds. \quad (8.13)$$

Sind die ursprünglichen Definitionen der Gewichte $\lambda_i^{(n)}$ vom Intervall $[a, b]$ abhängig, so sind es die letztgenannten Formeln nicht. Sie hängen nur noch von n ab und können unabhängig vom Integrationsintervall berechnet und gespeichert werden.

Einige der abgeschlossenen Newton-Cotes-Formeln sind unter speziellen Namen bekannt:

$n = 1$: **Trapezregel**,

$n = 2$: **Simpson-Regel** oder **Keplersche-Fassregel**,⁵

$n = 3$: **Newtonsche 3/8-Regel**,

$n = 4$: **Milne-Regel**.⁶

Mittels der folgenden MAPLE®-Anweisungen lassen sich die Gewichte einfach bestimmen.

MAPLE-Beispiel: Berechnung der Quadraturgewichte

```

1 > # geschlossene Newton-Cotes-Formel
2 > n:=2:
3 > zaehler:= (x,i) -> quo(product((x-k),k=0..n),(x-i),x):
4 > seq(int(zaehler(x,m)/subs(x=m, zaehler(x,m)),x=0..n)/n,m=0..n);
5
6          1      2      1
7          - , - , -
8          6      3      6
9
10 # offene Newton-Cotes-Formel
11 > n:=2:
12 > zaehler:= (x,i) -> quo(product((x-k),k=0..n),(x-i),x):
13 > seq(int(zaehler(x,m)/subs(x=m, zaehler(x,m)),
14         x=-1..n+1)/(n+2),m=0..n);
15
16          2      -1      2
17          - , - , -
18          3      3      3
19
20 # MacLaurin-Formel
21 > n:=2:
22 > zaehler:= (x,i) -> quo(product((x-k),k=0..n),(x-i),x):
23 > seq(int(zaehler(x,m)/subs(x=m, zaehler(x,m)),
24         x=-1/2..n+1/2)/(n+1),m=0..n);
25

```

⁵Thomas Simpson, 1710-1761; Johannes Kepler, 1571-1630.

⁶Edward Arthur Milne, 1896-1950.

26	3	1	3
27	–	,	–
28	8	4	8

8.3 Konvergenz von Quadraturformeln

Bei den Newton-Cotes-Formeln treten für $n \geq 8$ auch negative Gewichte auf, was zu Stabilitätsproblemen führen kann. Für offene Newton-Cotes-Formeln geschieht dies bereits bei $n = 6$. Der Satz von Kuzmin⁷ ([Pla04, Satz 6.11]) besagt sogar, dass

$$\lim_{n \rightarrow \infty} \sum_{j=0}^n |\lambda_j^{(n)}| = \infty$$

gilt. Es stellt sich die Frage nach der Konvergenz der Quadraturformel $Q_n[f]$ gegen das Integral $I(f)$ mit $n \rightarrow \infty$.

Definition 8.3.1 — Konvergenz einer Quadraturformel. Eine Folge $(Q_n)_{n \in \mathbb{N}}$ von Quadraturformeln heißt **konvergent**, falls

$$\lim_{n \rightarrow \infty} Q_n[f] = \int_a^b f(x) dx \quad \text{für alle } f \in C([a, b]). \quad (8.14)$$

Der folgende Satz gibt eine vollständige Charakterisierung.⁸

Satz 8.3.2 — Szegő. Für Knoten $a \leq x_0^{(n)} < \dots < x_n^{(n)} \leq b$ und Gewichte $\lambda_k^{(n)} \in \mathbb{R}$, $k = 0, \dots, n$, seien Quadraturformeln definiert durch

$$Q_n[f] := \sum_{k=0}^n \lambda_k^{(n)} f(x_k^{(n)}). \quad (8.15)$$

Dann ist $(Q_n)_{n \in \mathbb{N}}$ genau dann konvergent, wenn

- 1.) $\sup_{n \in \mathbb{N}} \sum_{k=0}^n |\lambda_k^{(n)}| < \infty$;
- 2.) $(Q_n)_{n \in \mathbb{N}}$ konvergiert für Polynome, d.h. $I(p) = \lim_{n \rightarrow \infty} Q_n[p]$ für alle $p \in \mathbb{P}$.⁹

Beweis. Für den Beweis benötigt man einige Aussagen aus der Funktionalanalysis (u.a. die Sätze von Weierstraß und Banach-Steinhaus). Dies würde den Rahmen hier sprengen. Wir verweisen auf [Wer18, Satz IV.2.8]. ■

Korollar 8.3.3 Die Folge $(Q_n^{\text{Mi}})_{n \in \mathbb{N}_0}$ der summierten Mittelpunktsregel ist konvergent.

Beweis. Nach (8.4) gilt für die Gewichte $\lambda_k^{(n)} = \frac{b-a}{n}$, $k = 0, \dots, n-1$, also ist die Summe der Gewichte gleich $b-a$ unabhängig von n , womit die erste Bedingung des Satzes von Szegő

⁷Rodion Osievich Kuzmin, 1891-1949.

⁸Gábor Szegő, 1895-1985.

⁹Mit \mathbb{P} bezeichnen wir die Menge der Polynome beliebigen Grades.

erfüllt ist. Nun sei p ein beliebiges Polynom. Dann gilt mit (8.6)

$$\left| \int_a^b p(x) - Q_n^{\text{Mi}}[p] \, dx \right| \leq \frac{1}{24}(b-a)h^2 \|p''\|_\infty = \frac{1}{24} \frac{(b-a)^3}{n^2} \|p''\|_\infty \longrightarrow 0 \quad \text{mit } n \rightarrow \infty,$$

womit auch die zweite Bedingung gezeigt ist und Satz 8.3.2 die Behauptung ergibt. ■

Der Satz von Szegő liefert eine sehr handhabbare hinreichende Konvergenzbedingung, die als Satz von Steklov¹⁰ bekannt ist.

Satz 8.3.4 — Steklov. Für die Quadraturformeln in (8.15) gelte

$$\lambda_k^{(n)} \geq 0 \quad \text{für alle } n \in \mathbb{N} \text{ und alle } 0 \leq k \leq n, \quad (8.16)$$

dann konvergiert die Folge $(Q_n)_{n \in \mathbb{N}_0}$, falls sie für alle $p \in \mathbb{P}$ konvergiert.

Beweis. Da die zweite Bedingung aus dem Satz von Szegő ist nach Voraussetzung erfüllt. Damit gilt für $p \equiv 1 \in \mathbb{P}$, dass

$$b-a = I(p) = \lim_{n \rightarrow \infty} Q_n[p] = \lim_{n \rightarrow \infty} \sum_{k=0}^n \lambda_k^{(n)}.$$

Mit (8.16) folgt dann die erste Voraussetzung aus Satz 8.3.2 und damit die Behauptung. ■

Bemerkung 8.3.5 Wie bereits zu Beginn dieses Abschnittes erwähnt, treten bei den Newton-Cotes-Formeln auch negative Gewichte auf. Also kann der Satz 8.3.4 von Steklov nicht angewendet werden. Leider hilft auch der Satz 8.3.2 von Szegő nicht weiter, da es nach einem Resultat von Pólya¹¹ mindestens eine Funktion gibt, für welche die Newton-Cotes-Formeln *nicht* konvergieren.

Die folgenden Zeilen MAPLE®-Code berechnen die Gewichte der abgeschlossenen Newton-Cotes-Formeln für $n = 8$ und die der Euler-MacLaurin-Formeln für $n = 6$, beide Quadratur-Formeln haben negative Gewichte. Für die offenen Newton-Cotes-Formeln erhalten wir schon für $n = 2$ negative Gewichte, wie wir im Maple-Beispiel auf Seite 127 gesehen haben.

MAPLE-Beispiel:

```

1 > n:=8:
2 > zaehler:= (x,i) -> quo(product((x-k),k=0..n),(x-i),x):
3 > seq(int(zaehler(x,m)/subs(x=m, zaehler(x,m)),x=0..n)/n,m=0..n);
4
5 989 2944 -464 5248 -454 5248 -464 2944 989
6 ----, ----, ----, ----, ----, ----, ----, ----, ----
7 28350 14175 14175 14175 2835 14175 14175 14175 28350
8
9 > n:=6:
10 > zaehler:= (x,i) -> quo(product((x-k),k=0..n),(x-i),x):
11 > seq(int(zaehler(x,m)/subs(x=m, zaehler(x,m)),

```

¹⁰Vladimir Andreevich Steklov, 1864-1926.

¹¹George Pólya, 1887-1985.

```

12      x=-1/2..n+1/2)/(n+1),m=0..n);
13
14      4949      49      6223      -6257      6223      49      4949
15      -----, -----, -----, -----, -----, -----, -----
16      27648      7680      15360      34560      15360      7680      27648

```

Satz 8.3.6 Sei Q_n eine Quadraturformel bzgl. Knoten $a \leq x_0^{(n)} < \dots < x_n^{(n)} \leq b$, dann gilt für die Ordnung

$$k \leq 2n + 2.$$

Beweis. Wende die Quadraturformel auf das Polynom

$$p \in \mathbb{P}_{2n+2} \text{ mit } p(x) := \prod_{k=0}^n (x - x_k^{(n)})^2 > 0, \quad x \in [a, b]$$

an. Dann gilt

$$I(p) - Q_n[p] = \int_a^b \underbrace{p(x)}_{>0} dx - \sum_{k=0}^n \lambda_k^{(n)} \underbrace{p(x_k^{(n)})}_{=0} > 0,$$

also kann Q_n nicht exakt vom Grad $2n + 2$ sein, die Ordnung ist also maximal $2n + 2$. ■

8.4 Gauß-Quadratur

Das negative Resultat aus Satz 8.3.6 lässt die Frage offen, ob es Quadraturformeln Q_n gibt, welche die maximale Ordnung erreichen, die also exakt für \mathbb{P}_{2n+1} sind. Auf den ersten Blick erscheint dies nicht hoffnungslos: Q_n ist durch je $n+1$ Knoten und Gewichte bestimmt, wir haben also $2n + 2$ Freiheitsgrade. Auf der anderen Seite gilt $\dim(\mathbb{P}_{2n+1}) = 2n + 2$, also haben wir genau so viele Freiheitsgrade wie Bedingungen. Wenn man zur Bestimmung dieser Unbekannten den folgenden monomialen Ansatz wählt

$$(b-a) \sum_{i=0}^n \lambda_i^{(n)} (x_i^{(n)})^j = Q_n[(\cdot)^j] \stackrel{!}{=} I((\cdot)^j) = \int_a^b x^j dx = \frac{b^{j+1} - a^{j+1}}{j+1}, \quad j = 0, \dots, 2n+1,$$

so ist dies ein *nichtlineares Gleichungssystem* mit $2n + 2$ Gleichungen und $2n + 2$ Unbekannten. Hat dieses Gleichungssystem eine Lösung, so integriert die resultierende Quadraturformel Polynome bis zum Grad $2n + 1$ exakt. Mit zunehmendem n wird dieses Gleichungssystem allerdings immer unübersichtlicher. Und insbesondere gehen die Knoten nichtlinear ein. Wir suchen daher einen allgemeinen Ansatz, um Quadraturformeln maximaler Exaktheit zu konstruieren.

8.4.1 Integrale mit Gewichtsfunktion

Dazu betrachten wir nun gewichtete Integrale, also

$$I_\omega(f) := \int_a^b f(x) \omega(x) dx,$$

mit einer positiven Gewichtsfunktion, vgl. Beispiel 7.1.2. Solche Gewichte können z.B. Wahrscheinlichkeitsdichten sein, so dass obiges Integral ein Erwartungswert sein kann. Für $\omega \equiv 1$ erhalten wir den „Standardfall“ und es wird sich herausstellen, dass die Erweiterung auf gewichtete Integrale nicht nur bzgl. möglicher Anwendungen adäquat ist, sondern auch für die mathematische Analyse geradezu passend ist.

Lemma 8.4.1 — Charakterisierung via Orthogonalität. Sei Q_n exakt vom Grad $2n + 1$ und

$$p_{k+1}(x) := \prod_{i=0}^k (x - x_i^{(k)}) \in \mathbb{P}_{k+1}, \quad k = 0, \dots, n, \quad (8.17)$$

dann gilt $(p_j, p_{n+1})_\omega = 0$ für alle $j = 0, \dots, n$.

Beweis. Wegen $p_k \in \mathbb{P}_k$ gilt $p_j p_{n+1} \in \mathbb{P}_{2n+1}$ für alle $0 \leq j \leq n$, also

$$(p_j, p_{n+1})_\omega = I_\omega(p_j p_{n+1}) = Q_n[p_j p_{n+1}] = (b-a) \sum_{i=0}^n \lambda_i^{(n)} p_j(x_i^{(n)}) \underbrace{p_{n+1}(x_i^{(n)})}_{=0} = 0$$

und damit die Behauptung. ■

Bemerkung 8.4.2 Der Satz sagt also, dass die Knoten einer Quadraturformel maximaler Exaktheit Nullstellen von orthogonalen Polynomen sein müssen. Die Nullstellen von orthogonalen Polynomen scheinen also eine gute Wahl für die gesuchten optimalen Knoten zu sein!

Wie bereits bei den Newton-Cotes-Formeln gesehen, sind mit der Wahl der Knoten auch die Gewichte bestimmt: falls Q_n exakt vom Grad n ist, muss dies insbesondere für die *Lagrange*-Polynome $L_{n,i}$ bzgl. $x_k^{(n)}$ aus (6.2) gelten, d.h.

$$\begin{aligned} 0 &= I_\omega(L_{n,i}) - Q_n[L_{n,i}] = \int_a^b L_{n,i}(x) \omega(x) dx - (b-a) \sum_{j=0}^n \lambda_j^{(n)} \underbrace{L_{n,i}(x_j^{(n)})}_{=\delta_{ij}} \\ &= \int_a^b L_{n,i}(x) \omega(x) dx - (b-a) \lambda_i^{(n)}, \end{aligned}$$

also –vollkommen analog zu (8.9)–

$$\lambda_i^{(n)} = \frac{1}{(b-a)} \int_a^b L_{n,i}(x) \omega(x) dx.$$

Damit gilt (durchaus erstaunlicherweise):

Satz 8.4.3 Sei $(p_n)_{n \in \mathbb{N}}$ eine Folge orthogonaler Polynome bzgl. der Gewichtsfunktion ω und $x_i^{(n)}$, $i = 0, \dots, n$, seien die Nullstellen von p_{n+1} . Dann ist die Quadraturformel Q_n mit den Knoten $x_i^{(n)}$ und den Gewichten $\lambda_i^{(n)}$ genau dann exakt für \mathbb{P}_n , wenn Q_n exakt für \mathbb{P}_{2n+1} ist.

Beweis. „ \Leftarrow “ ist trivial. Sei also umgekehrt Q_n exakt auf \mathbb{P}_n und $p \in \mathbb{P}_{2n+1}$. Mit dem Euklidischen Algorithmus¹² folgt, dass Polynome $q, r \in \mathbb{P}_n$ existieren mit

$$p = q p_{n+1} + r. \quad (8.18)$$

Die orthogonalen Polynome $(p_k)_{k=0, \dots, n}$ bilden eine Basis für \mathbb{P}_n , also wegen $p_{n+1} \perp_{\omega} \mathbb{P}_n$ damit $(q, p_{n+1})_{\omega} = 0$ für alle $q \in \mathbb{P}_n$ und somit (wegen $r \in \mathbb{P}_n$)

$$I_{\omega}(p) = I_{\omega}(q p_{n+1}) + I_{\omega}(r) = \underbrace{(q, p_{n+1})_{\omega}}_{=0} + I_{\omega}(r) = Q_n[r]$$

und andererseits

$$Q_n[r] = (b-a) \sum_{i=0}^n \lambda_i^{(n)} r(x_i^{(n)}) = (b-a) \sum_{i=0}^n \lambda_i^{(n)} \left\{ q(x_i^{(n)}) \underbrace{p_{n+1}(x_i^{(n)})}_{=0} + r(x_i^{(n)}) \right\} = Q_n[p],$$

also $Q_n[p] = I_{\omega}(p)$ für alle $p \in \mathbb{P}_{2n+1}$. ■

8.4.2 Gauß-Christoffel-Quadratur

Wir fassen die Überlegungen in dem folgenden Satz zusammen. Die entsprechende Methode wird Gauß-Christoffel-Quadratur¹³ oder auch Gauß-Legendre-Quadratur¹⁴ genannt.

Satz 8.4.4 — Gauß-Christoffel-Quadratur. Es sei $\omega \in C(a, b)$ eine positive Gewichtsfunktion auf (a, b) und $(p_k)_{k \in \mathbb{N}_0}$ eine Folge von Orthogonalpolynomen bzgl. des Skalarprodukts $(\cdot, \cdot)_{\omega}$. Des Weiteren sei $n \in \mathbb{N}$ und $x_0^{(n)} < \dots < x_n^{(n)}$ seien die Nullstellen von $p_{n+1}(x)$. Dann existieren (von $[a, b]$ unabhängige) eindeutig bestimmte Gewichte $\lambda_0^{(n)}, \dots, \lambda_n^{(n)} \in \mathbb{R}_+$, so dass Q_n exakt vom Grad $2n+1$ ist.

Darüberhinaus sind alle Gewichte positiv und es gilt

$$\lambda_k^{(n)} = \frac{\|p_n\|_{\omega}^2}{p'_{n+1}(x_k^{(n)}) p_n(x_k^{(n)})} \quad (8.19)$$

Beweis. Die Existenz folgt aus Satz 8.4.3 und der einfachen Überlegung, dass Q_n exakt vom Grad n ist. Die Gewichte sind wegen (8.9) eindeutig bestimmt und aufgrund von (8.12) bzw. (8.13) unabhängig vom Intervall $[a, b]$.

Sei $q_{ik} \in \mathbb{P}_{2n+1}$ mit $q_{ik}(x_i^{(n)}) = 0$ für $i \neq k$ und $q_{ik}(x_k^{(n)}) \neq 0$. Aufgrund der Exaktheit der

¹²Polynomdivision mit Rest.

¹³Elwin Bruno Christoffel, 1829-1900.

¹⁴Adrien-Marie Legendre, 1752-1833.

Quadraturformel gilt dann

$$\int_a^b q_{ik}(x) \omega(x) dx = I_\omega(q_{ik}) = Q_n[q_{ik}] = \lambda_k^{(n)} q_{ik}(x_k^{(n)}), \quad \text{also}$$

$$\lambda_k^{(n)} = \frac{1}{q_{ik}(x_k^{(n)})} \int_a^b q_{ik}(x) \omega(x) dx.$$

Wir wählen speziell $q_{ik}(x) := \left(\frac{p_{n+1}(x)}{x - x_k^{(n)}} \right)^2$, so hat dieses Polynom die geforderten Eigenschaften und es gilt

$$\lambda_k^{(n)} = \int_a^b \left(\frac{p_{n+1}(x)}{p'_{n+1}(x_k^{(n)})(x - x_k^{(n)})} \right)^2 \omega(x) dx > 0.$$

Nun wählen wir $q_{ik}(x) := \frac{p_{n+1}(x)}{x - x_k^{(n)}} p_n(x)$, was wiederum die gewünschten Eigenschaften besitzt. Weiter gilt

$$\lambda_k^{(n)} = \frac{1}{p'_{n+1}(x_k^{(n)}) p_n(x_k^{(n)})} \int_a^b \underbrace{\frac{p_{n+1}(x)}{x - x_k^{(n)}} p_n(x)}_{=p_n(x)+q_{n-1}(x)} \omega(x) dx \quad (8.20)$$

mit einem $q_{n-1} \in \mathbb{P}_{n-1}$, da das Polynom oberhalb der Klammer den führenden Koeffizienten Eins hat. Da aber $p_n \perp_\omega \mathbb{P}_{n-1}$ folgt (8.19). ■

Bemerkung 8.4.5 (a) Der offensichtliche Vorteil der Gauß-Quadratur ist die optimale Exaktheit.

(b) Nachteile sind einerseits, dass die Knoten festgelegt sind und andererseits eine nachträgliche Erhöhung der Genauigkeit durch Aufdatierung der Knoten nicht möglich ist – beim Übergang von n auf $n + 1$ muss alles neu berechnet werden.

(c) Die Berechnung der Gewichte und Knoten für beliebige ω wird mit der Drei-Term-Rekursion gemacht, s.u.

(d) Integrale auf mehrdimensionalen Gebieten („Kubatur“) auf $[a_1, b_1] \times \cdots \times [a_d, b_d]$ mit $d \gg 1$ kann man mit *Tensorprodukt-Quadratur-Formeln* $Q_{n_1} \otimes \cdots \otimes Q_{n_d}$ berechnen. Die Komplexität steigt jedoch stark mit der Raumdimension d („Fluch der Dimensionen“). Nimmt man beispielsweise die Trapezregel, so hat man in einer Raumdimension $n_1 = 2$ Punkte. In d Raumdimensionen sind es $n_d = 2^d$, was schnell jede Rechnerkapazität sprengen würde. Mögliche Auswege sind Monte-Carlo, Quasi-Monte-Carlo, Dünngitter (siehe Vorlesung *Numerical Finance*).

Nun zur Darstellung des Fehlers. Analog zu den interpolatorischen Quadraturformeln kann man vom Exaktheitsgrad auf den Approximationsfehler schließen.

Satz 8.4.6 — Fehler der Gauß-Quadratur. Für $f \in C^{2n+2}([a, b])$ gilt für ein $\xi \in (a, b)$

$$E_n(f) = I_\omega(f) - Q_n[f] = \frac{f^{2n+2}(\xi)}{(2n+2)!} \|P_{n+1}\|_\omega^2 \quad (8.21)$$

Beweis. Betrachte den Hermite-Interpolanten $p \in \mathbb{P}_{2n+1} = P(f|x_0^{(n)}, x_0^{(n)}, \dots, x_n^{(n)}, x_n^{(n)})$ an f zu den jeweils doppelten Knoten $x_k^{(n)}$, $k = 0, \dots, n$. Dafür gilt mit einem $\xi \in (a, b)$

$$\begin{aligned} f(x) - p(x) &= \underbrace{(x - x_0^{(n)})^2 \dots (x - x_n^{(n)})^2}_{=P_{n+1}(x)^2} \frac{f^{2n+2}(\xi)}{(2n+2)!}, \text{ also} \\ I_\omega(f) &= \underbrace{I_\omega(p)}_{=Q_n[p]} + \frac{f^{2n+2}(\xi)}{(2n+2)!} I_\omega(P_{n+1}^2) = (b-a) \underbrace{\sum_{i=0}^n \lambda_i^{(n)} \underbrace{p(x_i^{(n)})}_{=f(x_i^{(n)})}}_{=Q_n[f]} + \frac{f^{2n+2}(\xi)}{(2n+2)!} \underbrace{(P_{n+1}, P_{n+1})_\omega}_{=\|P_{n+1}\|_\omega^2} \end{aligned}$$

und damit die Behauptung. ■

8.4.3 Berechnung der Knoten und Gewichte

Für die Umsetzung der Gauß-Quadratur benötigen wir Knoten und Gewichte. Diese sind mittlerweile in Tabellenform (auch in vielen Software-Paketen) dokumentiert:



Gib mir bitte Knoten und Gewichte für die Gauß-Quadratur



Die Gauss-Quadratur ist eine Methode zur numerischen Integration, bei der bestimmte Gewichtungen und Knoten verwendet werden, um die Integration von Funktionen zu approximieren. Es gibt verschiedene Varianten der Gauss-Quadratur, abhängig von der Anzahl der Knoten und Gewichte.

Hier sind die Knoten und Gewichte für die Gauss-Quadratur mit einem bestimmten Grad n :

1. Gauss-Quadratur Grad $n = 1$:

Knoten: $x_1 = 0$

Gewicht: $w_1 = 2$

2. Gauss-Quadratur Grad $n = 2$:

Knoten: $x_1 = -1/\sqrt{3}, x_2 = 1/\sqrt{3}$

Gewicht: $w_1 = w_2 = 1$

3. Gauss-Quadratur Grad $n = 3$:

Knoten: $x_1 = -\sqrt{3/5}, x_2 = 0, x_3 = \sqrt{3/5}$

Gewicht: $w_1 = w_3 = 5/9, w_2 = 8/9$

4. Gauss-Quadratur Grad $n = 4$:

Knoten: $x_1 = -\sqrt{(3 + 2\sqrt{6/5})/7}, x_2 = -\sqrt{(3 - 2\sqrt{6/5})/7},$

$x_3 = \sqrt{(3 - 2\sqrt{6/5})/7}, x_4 = \sqrt{(3 + 2\sqrt{6/5})/7}$

Gewicht: $w_1 = w_2 = (18 - \sqrt{30})/36, w_3 = w_4 = (18 + \sqrt{30})/36$

Bitte beachten Sie, dass diese Werte speziell für das Intervall $[-1, 1]$ gelten. Wenn Sie die Gauss-Quadratur auf ein anderes Intervall anwenden möchten, müssen Sie die Knoten und Gewichte entsprechend skalieren. Die Formeln zur Berechnung der Knoten und Gewichte sind komplex und werden oft durch numerische Methoden bestimmt. Die obigen Werte sind einige der gebräuchlichsten Beispiele für die Gauss-Quadratur.

Wir wollen uns der „komplexen“ Berechnung und den entsprechenden numerischen Methoden widmen. Die Knoten sind die Nullstellen der orthogonalen Polynome. Damit hatten wir uns in §7.3 (Seite 118 ff.) bereits beschäftigt. In Korollar 7.3.2 hatten wir gezeigt, dass wir die Nullstellen als Eigenvektoren eines Eigenwertproblems berechnen können. Die Gewichte kann man dann beispielsweise aus der Christoffel-Darboux-Formel in Satz 7.3.3 ermitteln.

Satz 8.4.7 Sei $(p_n)_{n \in \mathbb{N}_0}$ eine Folge orthogonaler Polynome der Form (7.6) mit Nullstellen $x_k^{(n)}$, dann gilt für die Gewichte $\lambda_k^{(n)}$ der Gauß-Christoffel-Quadratur

$$\lambda_k^{(n)} = \frac{k_{n+1} h_n^2}{p'_{n+1}(x_k^{(n)}) p_n(x_k^{(n)}) k_n} = h_n \left(\sum_{i=0}^n \frac{p_i^2(x_k^{(n)})}{h_i} \right)^{-1}, \quad k = 0, \dots, n.$$

Beweis. Mit den Nullstellen $y := x_k^{(n)}$ von p_{n+1} folgt aus der Christoffel-Darboux-Formel (7.20) (Satz 7.3.3)

$$\frac{k_n}{k_{n+1} \cdot h_n} \cdot \frac{p_{n+1}(x)}{x - x_k^{(n)}} p_n(x_k^{(n)}) = \sum_{i=0}^n \frac{p_i(x) p_i(x_k^{(n)})}{h_i} \text{ für alle } y \in \mathbb{R} \quad (8.22)$$

und mit dem Grenzübergang $x \rightarrow x_k^{(n)}$ ergibt sich¹⁵

$$\frac{k_n}{k_{n+1} \cdot h_n} p'_{n+1}(x_k^{(n)}) p_n(x_k^{(n)}) = \sum_{i=0}^n \frac{p_i(x_k^{(n)})^2}{h_i}. \quad (8.23)$$

Nun setzen wir (8.22) in (8.20) ein und erhalten

$$\begin{aligned} \lambda_k^{(n)} &= \frac{1}{p'_{n+1}(x_k^{(n)}) p_n(x_k^{(n)})} \int_a^b \frac{p_{n+1}(x)}{x - x_k^{(n)}} p_n(x) \omega(x) dx \\ &= \frac{1}{p'_{n+1}(x_k^{(n)}) p_n(x_k^{(n)})} \frac{k_{n+1} \cdot h_n}{k_n} \frac{1}{p_n(x_k^{(n)})} \sum_{i=0}^n \frac{p_i(x_k^{(n)})}{h_i} \underbrace{\int_a^b p_i(x) p_n(x) \omega(x) dx}_{=\delta_{i,n} h_n^2} \\ &= \frac{k_{n+1} h_n^2}{p'_{n+1}(x_k^{(n)}) p_n(x_k^{(n)}) k_n}, \end{aligned}$$

also die erste behauptete Identität. Nun wenden wir noch (8.23) an und erhalten

$$\lambda_k^{(n)} = h_n \left(\frac{k_n}{k_{n+1} h_n} p'_{n+1}(x_k^{(n)}) p_n(x_k^{(n)}) \right)^{-1} = h_n \left(\sum_{i=0}^n \frac{p_i(x_k^{(n)})^2}{h_i} \right)^{-1}$$

und damit ist alles bewiesen. ■

Bemerkung 8.4.8 — Gauß-Quadratur in höheren Dimensionen. In höheren Raumdimensionen hat man die Theorie der Orthogonalpolynome nicht, aus der man optimale Quadraturformeln gewinnen könnte. Hier bleibt einem häufig nichts anderes übrig, als diese näherungsweise als Lösung nichtlinearer Gleichungen zu bestimmen.

8.4.4 Radau- und Lobatto-Formeln

Insbesondere bei Anfangs- oder Randwertproblemen von Differenzialgleichungen (vgl. z.B. die Vorlesung *Numerik gewöhnlicher Differenzialgleichungen*) ist es sinnvoll, Knoten an einem Rand oder beiden Endpunkten mit in die Quadraturformel einzubringen. Sei dies z.B. der linke Randnoten a . Wir beschränken uns hier auf das Intervall $I = [-1, 1]$.

¹⁵Beachte: $\lim_{x \rightarrow x_k^{(n)}} \frac{p_{n+1}(x)}{x - x_k^{(n)}} = \lim_{x \rightarrow x_k^{(n)}} \frac{p_{n+1}(x) - p_{n+1}(x_k^{(n)})}{x - x_k^{(n)}} = p'_{n+1}(x_k^{(n)})$.

Gauß-Lobatto-Quadratur

Schreiben wir die beiden Knoten $\bar{x}_0 = -1$ und $\bar{x}_n = 1$ vor, so basiert die Gauß-Quadratur maximaler Exaktheit auf den $n + 1$ Nullstellen des Polynoms

$$\bar{\omega}_{n+1}(x) := p_{n+1}(x) + a p_n(x) + b p_{n-1}(x),$$

wobei wiederum $\{p_n\}_{n \in \mathbb{N}_0}$ orthogonale Polynome zur Gewichtsfunktion ω sind. Die Konstanten a und b werden so gewählt, dass $\bar{\omega}_{n+1}(-1) = \bar{\omega}_{n+1}(1) = 0$ gilt. Diese Nullstellen seien bezeichnet mit $-1 = \bar{x}_0, \bar{x}_1, \dots, \bar{x}_{n-1}, \bar{x}_n = 1$, dann erhalten wir die Gewichte analog zu (8.9)

$$\bar{\lambda}_k^{(n)} := \frac{1}{2} \int_a^b \bar{L}_{n,k}(x) \omega(x) dx$$

mit den Lagrange-Basispolynomen $\bar{L}_{n,k}$ auf $(-1, 1)$ bzgl. der Knoten \bar{x}_k , $k = 0, \dots, n$. Die entsprechende Quadratur

$$Q_n^{\text{GL}}[f] := \sum_{k=0}^n \bar{\lambda}_k^{(n)} f(\bar{x}_k)$$

heißt *Gauß-Lobatto-Formel*.¹⁶ Sie hat den Exaktheitsgrad $2n - 1$, was man analog zum Beweis von Satz 8.4.3 zeigt, [QSS07, §10.2]. Für das Tschebyscheff-Gewicht $\omega(x) := \frac{1}{\sqrt{1-x^2}}$ sind die inneren Gauß-Lobatto-Knoten die Tschebyscheff-Abszissen aus Lemma 7.2.8 (ix), also die Nullstellen der Ableitung T'_n der Tschebyscheff-Polynome.

Die besondere Bedeutung der Gauß-Lobatto-Quadratur liegt bei der numerischen Lösung von partiellen Differenzialgleichungen mithilfe sogenannter *spektraler Kollokationsmethoden*, vgl. Vorlesung *Numerik partieller Differentialgleichungen*.¹⁷ Sie werden auch zur numerischen Differentiation eingesetzt, vgl. §8.6.

Gauß-Radau-Quadratur

Hier benutzt man die Gewichtsfunktion $\omega \equiv 1$ und schreibt vor, dass der linke Endpunkt des Intervalls ein Quadraturknoten ist, also $\bar{x}_0 = -1$. Wählt an die übrigen n Knoten so, dass der Exaktheitsgrad maximal wird, so führt dies auf die Nullstellen des Polynoms

$$\frac{P_{n-1}(x) + P_n(x)}{1+x}$$

mit den Legendre-Polynomen P_n aus §7.2.2. Wiederum sind die Gewichte durch die Knoten bestimmt. Der Exaktheitsgrad der entstehenden Gauß-Radau-Quadratur¹⁸ ist $2n$.

8.5 Extrapolation und Romberg-Integration

¹⁶Rehuel Lobatto, 1797-1866.

¹⁷Von besonderem Interesse ist dabei, dass es Fehlerabschätzung für die Gauß-Lobatto-Interpolation gibt für Funktionen in Lebesgue- und Sobolev-Räumen, die für partielle Differentialgleichungen besondere Bedeutung haben.

¹⁸Jean Charles Rodolphe Radau, 1835-1911.

Bemerkung 8.5.1 — Ineffizienz bei hoher gewünschter Genauigkeit. Angenommen, man möchte mit der summierten Trapezregel (8.5) eine Genauigkeit des Integrals von 10^{-6} erreichen. Die Fehlerschätzung (8.7) besagt dann

$$|I(f) - Q_n^{\text{Tr}}[f]| = \mathcal{O}(h^2) \stackrel{!}{\leq} 10^{-6},$$

also muss die Größe der Teilintervalle in der Größenordnung $h \sim 10^{-3}$ sein. Das bedeutet aber, dass wir $n \sim 1000$ Teilintervalle und damit Knoten haben. Das wäre viel zu ineffizient, wir brauchen Alternativen!

Die Gauß-Quadratur wäre ggf. (bei hinreichender Glattheit des Integranden) ein Ausweg, es gibt aber (wie bereits erwähnt) zwei wesentliche Nachteile:

1. die Wahl der Stützstellen ist fest; insbesondere müssen wir die Ordnung der Quadraturformel *a priori* festlegen; und diese ist durch die globale Glattheit des Integranden festgelegt;
2. eine nachträgliche (*a posteriori*) Erhöhung der Ordnung (ggf. nur lokal) ist nicht in effizienter Form möglich.

Wir wollen daher nun eine Strategie zur nachträglichen Steigerung der Genauigkeit kennenlernen, die auch eine lokal angepasste („adaptive“) Wahl der Stützstellen erlaubt. Bisher haben wir Quadraturen bzgl. eines festen Knotengitters betrachtet. Nun verwenden wir eine Folge von Knotengittern: Zu einer gegebenen Funktion f und einer Folge von absteigenden Schrittweiten $(h_\ell)_{\ell=1,2,\dots}$, $h_\ell = \frac{b-a}{n_\ell}$, $\mathbb{N} \ni n_\ell \nearrow$, betrachten wir Quadraturformeln $Q_{h_\ell}[f]$ – man beachte, dass der Index hier nicht wie bisher die Anzahl der Knoten ist, sondern die Schrittweite. Der Schlüssel zur Effizienz der Quadratur mittels Extrapolation liegt in folgendem Konstrukt.

Definition 8.5.2 Eine Quadraturformel Q_h mit Gitterweite $h > 0$ besitzt eine **asymptotische Fehlerentwicklung** (der Ordnung 2), falls

$$E_h(f) := |Q_h[f] - I(f)| = \tau_2 h^2 + \tau_4 h^4 + \dots + \tau_{2\nu} h^{2\nu} + \mathcal{O}(h^{2\nu+2}) \quad (8.24)$$

für alle $f \in C^{2\nu}([a, b])$, $\nu \in \mathbb{N}$, mit Koeffizienten $\tau_{k\ell} \in \mathbb{R}$, die **unabhängig von h sind**.

Ein bekanntes Beispiel ist die Trapezsumme aus (8.5), bei der die asymptotische Entwicklung als Euler-MacLaurin-Summenformel bekannt ist.¹⁹ Prinzipiell lässt sich das Vorgehen in diesem Abschnitt auf alle Quadraturen anwenden, die eine asymptotische Fehlerentwicklung erlauben. Wir beschränken uns jedoch der Einfachheit halber auf die summierte Trapezregel.

Lemma 8.5.3 — Euler-MacLaurin-Summenformel. Die Trapezsumme besitzt eine asymptotische Fehlerentwicklung (8.24) mit von h unabhängigen Koeffizienten

$$\tau_{2\ell} = \frac{B_{2\ell}}{(2\ell)!} (f^{(2\ell-1)}(b) - f^{(2\ell-1)}(a)), \quad (8.25)$$

¹⁹Leonhard Euler, 1707-1783, Colin Maclaurin, 1698-1746.

wobei $B_{2\ell}$ die *Bernoulli-Zahlen* sind, also

$$B_0 = 1, \quad \sum_{\nu=0}^n \binom{n+1}{\nu} B_\nu = 0. \quad (8.26)$$

Beweis. Taylor-Entwicklung plus Symmetrie, vgl. [Sto94, §3.3]. ■

Die Idee der **Extrapolation** ist nun wie folgt: zu gegebener Funktion f nehmen wir zunächst an, dass

$$\lim_{h \rightarrow 0} |I(f) - Q_h[f]| = 0.$$

Wir betrachten die Quadraturformel als Funktion der Schrittweite, d.h.

$$h \mapsto T(h) := Q_h[f] \text{ für eine feste Funktion } f.$$

Weiter nehmen wir zunächst an, dass eine asymptotische Fehlerentwicklung der Ordnung 2 gelte.²⁰ Dann sieht das Vorgehen wie folgt aus:

- (i) seien h_1 und h_2 zwei Schrittweiten (später h_1, \dots, h_k , also hier $k = 2$)
- (ii) sei $P = P(T|h_1^2, h_2^2) \in \mathbb{P}_{k-1} = \mathbb{P}_1$ das Interpolationspolynom, d.h. $P(h^2) = T(h)$, also

$$P(h^2) = T_{h_1}(f) + \frac{T_{h_2}(f) - T_{h_1}(f)}{h_2^2 - h_1^2} (h^2 - h_1^2). \quad (8.27)$$

vgl. Abbildung 8.2.

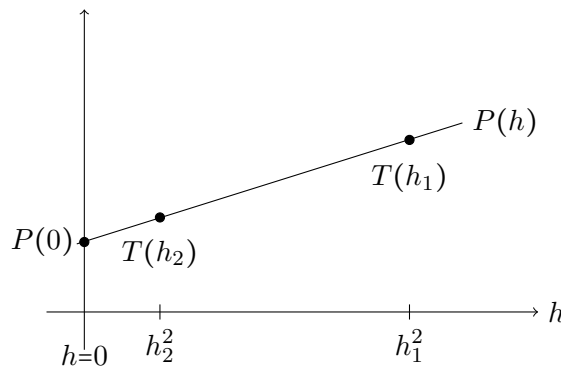


Abbildung 8.2: Extrapolation einer Geraden bei $h = 0$.

- (iii) *extrapoliere* $P(h)$ auf $h = 0$, also

$$\begin{aligned} P(0) &= \frac{1}{h_2^2 - h_1^2} (T_{h_1}(f)(h_2^2 - h_1^2) - h_1^2(T_{h_2}(f) - T_{h_1}(f))) \\ &= (h_2^2 - h_1^2)^{-1} (h_2^2 T_{h_1}(f) - h_1^2 T_{h_2}(f)), \end{aligned}$$

und damit $|I(f) - P(0)| = \mathcal{O}(h_1^2 h_2^2)$, falls T_h die asymptotische Entwicklung (8.24) besitzt.

²⁰Die Modifikationen bei anderen Ordnungen p wird später klar.

Da $0 \notin [h_k, h_1] \subset \mathbb{R}^+$ liegt, spricht man von **Extrapolation**, im Gegensatz zur **Interpolation**, bei der ein Interpolation an einer Stelle innerhalb des Knotenintervalls ausgewertet wird.

■ **Beispiel 8.5.4** Sei $h_1 = 2h_2$ mit h_2 fest. Dann erhalten wir mit (8.27) eine neue Quadraturformel

$$P(T|h_1^2, h_2^2)(0) = \frac{4}{3}T(h_2) - \frac{1}{3}T(2h_2) =: \tilde{T}(h_2) \quad (8.28)$$

durch Extrapolation. ■

Wir können dieses Vorgehen mittels des Interpolationspolynoms auch anhand der asymptotischen Entwicklung des Fehlers motivieren:

$$\begin{aligned} E_h(f) &= \tau_2 h^2 + \tau_4 h^4 + \mathcal{O}(h^6) \\ E_{2h}(f) &= 4\tau_2 h^2 + 16\tau_4 h^4 + \mathcal{O}(h^6), \end{aligned}$$

also können wir den führenden Term (vor h^2) durch $\tilde{T}(h)$ aus (8.28) eliminieren:

$$\begin{aligned} \tilde{T}(h) - I(f) &= \frac{4}{3}(T(h) - I(f)) - \frac{1}{3}(T(2h) - I(f)) \\ &= \frac{4}{3}\tau_2 h^2 + \frac{4}{3}\tau_4 h^4 + \dots + \mathcal{O}(h^{2\nu+2}) \\ &\quad - \frac{4}{3}\tau_2 h^2 - \frac{16}{3}\tau_4 h^4 + \dots + \mathcal{O}(h^{2\nu+2}) = \mathcal{O}(h^4), \end{aligned}$$

also ist $\tilde{T}(h)$ durch geschickte Kombination von $T(h)$ und $T(2h)$ eine Quadratur mit Fehlerordnung $\mathcal{O}(h^4)$! Diese Idee kann man analog für allgemeine Schrittweiten h_1, h_2 anstelle von h und $2h$, oder auch für andere Quadraturformeln realisieren.

Die Frage ist nun, wie macht man das praktisch, d.h., wie kommt man an geeignete Kombinationen wie in (8.28)? Eine Möglichkeit wäre, die entsprechenden Gleichungssysteme aufzustellen und zu lösen, dies ist jedoch unbequem, aufwendig und instabil für $h_1 \approx h_2$! Wir können den Fehler allgemein abschätzen.

Satz 8.5.5 — Fehlerabschätzung. Seien $f \in C^{2k}([a, b])$, $h_1, \dots, h_k > 0$, $k \in \mathbb{N}$ und $P \in \mathbb{P}_{k-1}$ mit $P(h_\ell^2) = T(h_\ell) = Q_{h_\ell}[f]$ für $\ell = 1, \dots, k$, dann gilt

$$\left| \int_a^b f(x) dx - P(0) \right| = \mathcal{O}(h_1^2 \dots h_k^2). \quad (8.29)$$

Beweis. Verwende die Lagrange-Darstellung (6.4) mit den Basis-Polynomen $L_{n,i}$ aus (6.3) als Funktionen in h^2 , also

$$P(h) = \sum_{i=1}^k T(h_i) L_{k-1,i-1}(h^2),$$

also mit der asymptotischen Entwicklung und Lemma 6.1.5 (Seite 93)

$$\begin{aligned} P(0) &= \sum_{i=1}^k T_{h_i}(f) L_{k-1,i-1}(0) \\ &= \sum_{i=1}^k L_{k-1,i-1}(0) \left\{ I(f) + \sum_{\ell=1}^k \tau_{2\ell} h_i^{2\ell} + \mathcal{O}(h_i^{2k+2}) \right\} \end{aligned}$$

$$\begin{aligned}
&= I(f) \underbrace{\sum_{i=1}^k L_{k-1,i-1}(0)}_{=1} + \sum_{\ell=1}^k \tau_{2\ell} \underbrace{\sum_{i=1}^k L_{k-1,i-1}(0) h_i^{2\ell}}_{= \begin{cases} 0, & \ell \leq k-1, \\ (-1)^{k-1} h_1^2 \cdots h_k^2, & \ell = k, \end{cases}} + \underbrace{\sum_{i=1}^k L_{k-1,i-1}(0) \mathcal{O}(h_i^{2k+2})}_{\text{Terme höherer Ordnung}} \\
&= I(f) + (-1)^{k-1} h_1^2 \cdots h_k^2 \tau_{2k} + \sum_{i=1}^k \mathcal{O}(h_i^{2k+2}),
\end{aligned}$$

und damit die Behauptung. ■

Dies kann man für k verschiedene Stützstellen und einem Verfahren der Ordnung p zu einer **Rekursion** machen:

1. **Erzeuge Datenpaare:** Berechne $T(h)$ für k verschiedene Stützstellen h_{i-k+1}, \dots, h_i , $i = 1, 2, \dots$
2. **Interpolation:** Bestimme dann das Interpolationspolynom in h^p zu den Wertepaaren $(h_{i-j}^p, T(h_{i-j}))$, $j = 0, \dots, k-1$

$$P_{i,k} := P(T|h_{i-k+1}^p, \dots, h_i^p) \in \mathbb{P}_{k-1}.$$

3. **Extrapolation:** $T_{i,k} := P_{i,k}(0)$ für $1 \leq k \leq i$.

Zur Erinnerung, wir können das *Neville-Aitken-Schema* in Algorithmus 6.2.2 zur Auswertung eines Interpolationspolynoms bzgl. der Knoten x_0, \dots, x_n an eine Funktion f (mit $f_i := f(x_i)$) verwenden:

$$P_{i,0} = f_i, \quad P_{i,k} = P_{i,k-1} + \frac{x - x_i}{x_i - x_{i-k}} (P_{i,k-1} - P_{i-1,k-1}), \quad i \geq k,$$

und dies überträgt sich dann für $T_{i,k}$ zu der folgenden **Rekursion**

$$\begin{aligned}
T_{i,1} &:= T(h_i), & \text{für } i = 1, 2, \dots, \\
T_{i,k} &= T_{i,k-1} + \frac{T_{i,k-1} - T_{i-1,k-1}}{\left(\frac{h_i}{h_{i-k+1}}\right)^p - 1}, & \text{für } 2 \leq k \leq i.
\end{aligned}$$

Bemerkung 8.5.6 — Unterschied zum Neville-Aitken-Schema. Man beachte, dass die vorstehende Rekursion im Gegensatz zum Neville-Aitken-Schema um einen Index verschoben ist. Das Extrapolationstableau, welches man analog zum Schema in §6.2 erhält, fängt erst beim Spaltenindex 1 an.

Wir werden nun einen Algorithmus auf Basis obiger Rekursion formulieren: Geht man davon aus, dass eine Grundschriftweite h als Startwert gegeben ist, dann bildet man die Schrittweiten

$$h_i := \frac{h}{m_i} \quad \text{mit einer wachsenden Folge } \mathbb{N} \ni m_i \nearrow, i = 1, 2, \dots$$

Algorithmus 8.5.7 — Romberg²¹-Quadratur. Gegeben sei eine Grundschriftweite h .

Wähle $(m_i)_{i \in \mathbb{N}}$ und setze $h_i := \frac{h}{m_i}$.

Für $i = 1, 2, \dots$

1. Bestimme $T_{i,1} := T(h_i)$.
2. Für $k = 2, \dots, i$
 - Berechne $T_{i,k} = T_{i,k-1} + \frac{T_{i,k-1} - T_{i-1,k-1}}{\left(\frac{m_i}{m_{i-k+1}}\right)^p - 1}$.
3. Falls $T_{i,i}$ ein geeignetes Abbruchkriterium erfüllt: STOP.

Bemerkung 8.5.8 (a) Obiges Verfahren funktioniert für alle Quadratur-Verfahren mit asymptotischer Entwicklung p -ter Ordnung. Analog zu Satz 8.5.5 ist der Fehler dann $\mathcal{O}(h_1^p \dots h_i^p)$. Die Trapezregel hat $p = 2$; dieser Spezialfall heißt **Romberg-Quadratur**.
 (b) Die Schrittweiten-Folge bestimmt durch $(m_i)_{i \in \mathbb{N}}$ ist noch zu wählen.

Beispiele für Knotenfolgen

Die Schrittweiten-Folge bestimmt die Wahl der Quadraturknoten. Mit einer geschickten Wahl der Knoten kann man

- eine möglichst hohe Genauigkeit erreichen und/oder
- den Berechnungsaufwand möglichst gering halten.

Die Genauigkeit in Bezug auf die Grundschriftweite h ist gegeben durch

$$\mathcal{O}(h_1^p \dots h_i^p) = \mathcal{O}\left(h^{ip} \left(\prod_{k=1}^i m_k\right)^{-p}\right).$$

Der Aufwand A_i zur Berechnung von $T_{i,i}$ hängt i.W. von der Anzahl der Auswertungen von f ab, also

$A_i :=$ Anzahl der zur Berechnung von $T_{i,i}$ notwendigen f -Auswertungen.

Nun ordnen wir der Folge $(m_i)_{i \in \mathbb{N}}$ eine Aufwandsfolge zu

$$\mathcal{A} := \{A_1, A_2, \dots\}$$

und geben die entsprechenden Werte für einige bekannte Beispiele an.

Romberg-Folge

Definition 8.5.9 — Romberg-Folge. Die Folge $(m_i^R)_{i \in \mathbb{N}}$ definiert durch

$$m_i^R := 2^{i-1}, \quad \text{d.h. } \{1, 2, 4, 8, 16, \dots\}. \quad (8.30)$$

wird **Romberg-Folge** genannt.

Der Vorteil der Romberg-Folge ist die Möglichkeit der effizienten rekursiven Berechnung der $T_{i,i}$, also $A_i^R = m_i^R + 1$, d.h. $\{2, 3, 5, 9, 17, \dots\}$, wie Tabelle 8.1 zeigt:

²¹Werner Romberg, 1909-2003.

m_i^R	h_i	Knoten	A_i^R
1	1	● ————— ●	2
2	$\frac{1}{2}$	○ ————— ● ————— ○	1 zusätzliche
4	$\frac{1}{4}$	○ —●— ○ —●— ○	2 zusätzliche
8	$\frac{1}{8}$	○ —●— ○ —●— ○ —●— ○	4 zusätzliche

Tabelle 8.1: Funktionsauswertungen bei der Romberg-Folge.

Bemerkung 8.5.10 — Stabilität bei Verwendung der Romberg-Folge. Die Romberg-Folge führt auf eine Quadraturformel mit positiven Gewichten (*Übung*), was aus Stabilitätsgründen wichtig ist.

Bulirsch-Folge

Vom Aufwand her günstiger ist die **Bulirsch-Folge**²²:

Definition 8.5.11 — Bulirsch-Folge. Die Folge $(m_i^B)_{i \in \mathbb{N}}$ definiert durch

$$1, 2, 3, 4, 6, 8, 12, 16, 24, \dots, \quad m_i^B = \begin{cases} 2^j, & i = 2j, \\ 3 \cdot 2^{j-1}, & i = 2j + 1, \\ 1, & i = 1, \end{cases} \quad (8.31)$$

mit $A_i^B = 2, 3, 5, 7, 9, 13, 17, \dots$, d.h. $A_i^B = m_i^B + 1$, wird **Bulirsch-Folge** genannt.

Da $m_i^B < m_i^R$ für $i > 2$ gilt, folgt $A_i^B < A_i^R$ für $i > 2$, was zeigt, dass der Aufwand für die Bulirsch-Folge geringer als für die Romberg-Folge ist.

Bemerkung 8.5.12 (a) Die Verwendung einer asymptotischen Entwicklung setzt stillschweigend die entsprechende Glattheit von f voraus!

(b) Wenn man f nicht kennt, oder f lokal stark variiert, ist es sinnvoll, die Schrittweite *adaptiv* zu steuern, d.h.,

- große Schrittweiten lokal in Bereichen, in denen f glatt ist,
- kleine Schrittweiten lokal in Bereichen, in denen f variiert.

Dazu benötigt man berechenbare so genannte **Fehlerschätzer**. Man kann zeigen, dass $|T_{k,k-1} - T_{k,k}|$ ein solcher ist, d.h.

$$E_k(f) := |I(f) - T_{k,k}| \approx |T_{k,k-1} - T_{k,k}| =: \Delta E_k(f).$$

Das Prinzip lautet dann

- $\Delta E_k(f)$ „groß“: Unterteile das Intervall und verwende niedrige Ordnung,
- $\Delta E_k(f)$ „klein“: Verwende hohe Ordnung und vergrößere eventuell das Intervall.

Details dazu findet man z.B. in [DH19, §9.5-9.7]. Dies ist ein erstes Beispiel einer sogenannten *hp*-Methode, bei der man Schrittweite h und Ordnung p adaptiv variiert.

Wiederum probieren wir, mit Hilfe von ChatGBT ein MATLAB[®]-Programm zu erzeugen:

²²Roland Zdeněk Bulirsch, 1932-2022.



Gib mir bitte ein Matlab-Perogramm zur adaptiven Quadratur



Selbstverständlich! Hier ist ein einfaches Beispiel für ein adaptives Quadraturprogramm in MATLAB, das die adaptive Simpson-Regel verwendet, um das Integral einer gegebenen Funktion zu berechnen:

MATLAB-Funktion per ChatGBT: adaptive_quadrature.m

```

1 function result = adaptive_quadrature(func, a, b, tol)
2     fa = func(a);
3     fb = func(b);
4     fc = func((a + b) / 2);
5
6     h = b - a;
7     integral_approx = (h / 6) * (fa + 4*fc + fb);
8
9     left_approx = (h / 12) * (fa + 4*func((a + (a+b)/2) / 2) +
10         2*fc);
11     right_approx = (h / 12) * (fc + 4*func(((a+b)/2 + b) / 2) +
12         fb);
13
14     error_estimate = abs(integral_approx - (left_approx +
15         right_approx))
16
17     if error_estimate < tol
18         result = integral_approx;
19     else
20         result = adaptive_quadrature(func, a, (a + b) / 2, tol
21             /2) + ...
22             adaptive_quadrature(func, (a + b) / 2, b, tol
23                 /2);
24     end
25 end

```



Du kannst dieses Programm verwenden, indem du deine eigene Funktion func definierst und die Intervallgrenzen a und b sowie die Toleranz tol angibst. Zum Beispiel:

MATLAB-Funktion per ChatGBT: run_adaptive_quadrature.m

```

1 % Die zu integrierende Funktion
2 func = @(x) x.^2;
3
4 % Intervallgrenzen
5 a = 0;
6 b = 1;
7
8 % Toleranz fuer die adaptive Quadratur
9 tolerance = 1e-3;
10

```

```

11 % Aufruf der adaptiven Quadratur-Funktion
12 result = adaptive_quadrature_my(func, a, b, tolerance);
13
14 disp(['Approximiertes Integral: ', num2str(result)]);

```



Bitte beachte, dass dies ein einfaches Beispiel ist und für komplexere Funktionen und Anforderungen möglicherweise weiterentwickelt werden muss.

Allerdings tut auch dieses Programm nicht, was es tun soll. Versuchen Sie, es zu korrigieren. Der exakte Wert des Integrals ist natürlich $I(f) = \frac{1}{3}$.

8.6 Numerische Differentiation

Ähnlich zur numerischen Approximation von Integralen kann man Methoden zur näherungsweisen Bestimmung von Ableitungen konstruieren. Die sicherlich einfachste Möglichkeit dazu sind *Finite Differenzen*, also z.B. die Vorwärtsdifferenz $u_i^{\text{VD}} := \frac{1}{h}(f(x_{i+1}) - f(x_i))$ oder die zentrale Differenz $u_i^{\text{ZD}} := \frac{1}{2h}(f(x_{i+1}) - f(x_{i-1}))$ zur Approximation von $f'(x_i)$, wenn $x_i = ih$ äquidistante Punkte sind. Den Fehler dieser Approximationen kann man mithilfe der Taylorentwicklung leicht abschätzen

$$u_i^{\text{VD}} - f'(x_i) = \frac{h}{2} f''(\xi_i), \quad u_i^{\text{ZD}} - f'(x_i) = \frac{h^3}{6} f'''(\eta_i),$$

mit geeigneten Zwischenpunkten ξ_i und η_i , jeweils unter geeigneten Glattheitsannahmen. Allerdings haben Differenzenmethoden entscheidende Nachteile. Einerseits muss die Schrittweite h klein gewählt werden, damit obige Fehlerabschätzungen jeweils gute Approximationen liefern. Auf der anderen Seite führt eine kleine Schrittweite zu massiven Stabilitätsproblemen.

Ein möglicher Ausweg ist die Verwendung der Gauß-Lobatto-Interpolation, die wir in §8.4.4 kennengelernt haben. Wir beschränken uns wieder auf den Fall $I = [-1, 1]$.

Definition 8.6.1 Die **pseudo-spektrale Ableitung** einer Funktion $f \in C^0(I)$ ist definiert als

$$\mathcal{D}_n f := (P_{n,\omega}^{\text{GL}} f)' \in \mathbb{P}_{n-1}(I),$$

wobei $P_{n,\omega}^{\text{GL}}$ der Gauß-Lobatto-Interpolant von f zur Gewichtsfunktion ω ist.

Die Idee ist also zunächst f durch den Gauß-Lobatto-Interpolanten zu approximieren und diesen dann abzuleiten. Letzteres ist nicht schwierig, da es sich um ein Polynom handelt. Wir können die Fehlerabschätzungen aus der Interpolation benutzen, um eine Fehleranalyse der pseudo-spektrale Ableitung anzugeben. Dazu seien $-1 = \bar{x}_0, \bar{x}_1, \dots, \bar{x}_{n-1}, \bar{x}_n = 1$ die Gauß-Lobatto-Knoten.

Satz 8.6.2 (a) Es gilt $\|f' - \mathcal{D}_n f\|_\omega \leq C n^{1-k} \|f^{(k)}\|_\omega$ für alle $f \in C^k(I)$.²³

(b) Für das Gauß-Gewicht ω hat der Operator \mathcal{D}_m folgende Form

$$(\mathcal{D}_n f)(\bar{x}_k) = (D_n f_n)_k, \quad f_n := (f(\bar{x}_k))_{k=0, \dots, n}$$

mit $d_0 := d_n := 2$ sowie $d_k := 1$ für $k = 1, \dots, n-1$ und der Ableitungsmatrix $D^{(n)}$

$$[D_n]_{k,j} := \begin{cases} \frac{d_k}{d_j} \frac{(-1)^{k+j}}{\bar{x}_k - \bar{x}_j}, & k \neq j, \\ \frac{-\bar{x}_j}{2(1-\bar{x}_j^2)}, & 1 \leq k = j \leq n-1, \\ -\frac{2m^2+1}{6}, & k = j = 0, \\ -\frac{2m^2+1}{2}, & k = j = n. \end{cases} \quad (8.32)$$

Beweis. Den Beweis findet man in [Can+06, S. 89ff]. ■

Der Konvergenzfaktor n^{1-k} ist der Grund für die Bezeichnung „pseudo-spektrale“ Approximation – „spektral“ wäre n^{-k} . Die pseudo-spektrale Approximation f_n'' der zweiten Ableitung erhält man durch $f_n'' := D_n^2 f$.

²³Diese Aussage kann man auch für Sobolev-Funktionen $f \in H_\omega^s(I)$ zeigen.

Anhang A

Einige 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 $\|\cdot\| : X \rightarrow \mathbb{R}$ heißt *Norm* auf X , falls

$$\|x\| \geq 0 \text{ für alle } x \in X \quad \text{und} \quad \|x\| = 0 \Leftrightarrow x = 0, \quad (\text{Definitheit}) \quad (\text{A.1a})$$

$$\|x + y\| \leq \|x\| + \|y\| \text{ für alle } x, y \in X, \quad (\text{Dreiecksungleichung}) \quad (\text{A.1b})$$

$$\|\alpha x\| = |\alpha| \|x\| \text{ für alle } \alpha \in \mathbb{R} \text{ und } x \in X. \quad (\text{Homogenität}) \quad (\text{A.1c})$$

Wir nennen $x \in X$ mit $\|x\| = 1$ einen *Einheitsvektor* bzgl. der Norm $\|\cdot\|$.

Für $X = \mathbb{R}^n$ nennen wir eine Funktion $\|\cdot\| : \mathbb{R}^n \rightarrow \mathbb{R}$, die die Normaxiome (A.1) erfüllt, auch *Vektornorm*. Besonders häufig verwendete Vektornormen sind die p -Normen definiert

für $x = (x_1, \dots, x_n)^T$ durch

$$\|x\|_p := \begin{cases} \left(\sum_{k=0}^n |x_k|^p \right)^{\frac{1}{p}}, & 1 \leq p < \infty, \\ \max_{1 \leq j \leq n} |x_j| & p = \infty. \end{cases} \quad (\text{A.2})$$

Für die Spezialfälle $p = 1$ und $p = 2$ ergibt sich:

$$\begin{aligned} \|x\|_1 &= |x_1| + \dots + |x_n| \\ \|x\|_2 &= (|x_1|^2 + \dots + |x_n|^2)^{\frac{1}{2}} = (x^T x)^{\frac{1}{2}} \end{aligned} \quad (\text{A.3})$$

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

$$|x^T y| \leq \|x\|_p \|y\|_q \quad \text{gilt mit } \frac{1}{p} + \frac{1}{q} = 1, \quad 1 \leq p, q \leq \infty. \quad (\text{A.4})$$

Ein Spezialfall der Hölder-Ungleichung für $p = q = 2$ ist die **Cauchy-Schwarz-Ungleichung**, die wie folgt lautet

$$|x^T y| \leq \|x\|_2 \|y\|_2. \quad (\text{A.5})$$

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 $0 < c_1 \leq c_2 < \infty$, so dass

$$c_1 \|x\|_p \leq \|x\|_{p'} \leq c_2 \|x\|_p, \quad x \in \mathbb{R}^n. \quad (\text{A.6})$$

Für obige Fälle und $x \in \mathbb{R}^n$ gilt

$$\begin{aligned} \|x\|_2 &\leq \|x\|_1 \leq \sqrt{n} \|x\|_2, \\ \|x\|_\infty &\leq \|x\|_2 \leq \sqrt{n} \|x\|_\infty, \\ \|x\|_\infty &\leq \|x\|_1 \leq n \|x\|_\infty. \end{aligned} \quad (\text{A.7})$$

Achtung: Wir erkennen, dass diese Konstanten von der Raumdimension n abhängen, insbesondere gilt im Allgemeinen, dass $c_1(n) \rightarrow 0$ und $c_2(n) \rightarrow \infty$ mit $n \rightarrow \infty$. 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 isomorph¹ zum \mathbb{R}^{mn} könnten wir einfach eine Vektornorm für solche „langen“ Vektoren verwenden, um eine Matrixnorm als eine Abbildung $\|\cdot\| : \mathbb{R}^{m \times n} \rightarrow \mathbb{R}$ mit den Eigenschaften (A.1) zu definieren.

¹ Dies bedeutet, dass eine bijektive Abbildung $\varphi : \mathbb{R}^{m \times n} \rightarrow \mathbb{R}^{mn}$ 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

$$\|A\|_F := \sqrt{\sum_{i=1}^m \sum_{j=1}^n |a_{ij}|^2} \quad (\text{A.8})$$

und die p -Normen ($1 \leq p \leq \infty$)

$$\|A\|_p := \sup_{x \neq 0} \frac{\|Ax\|_p}{\|x\|_p}. \quad (\text{A.9})$$

Man kann leicht zeigen, dass

$$\sup_{x \neq 0} \frac{\|Ax\|_p}{\|x\|_p} = \sup_{x \neq 0} \left\| A \frac{x}{\|x\|_p} \right\|_p = \sup_{\|x\|_p=1} \|Ax\|_p \quad (\text{A.10})$$

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 \mathbb{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.

$$\|A \cdot B\| \leq \|A\| \|B\|. \quad (\text{A.11})$$

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)

$$\begin{aligned} \|A \cdot B\|_p &= \max_{x \neq 0} \frac{\|ABx\|_p}{\|x\|_p} = \max_{x \neq 0} \frac{\|ABx\|_p \|Bx\|_p}{\|Bx\|_p \|x\|_p} \\ &\leq \max_{x \neq 0} \frac{\|ABx\|_p}{\|Bx\|_p} \cdot \max_{x \neq 0} \frac{\|Bx\|_p}{\|x\|_p} = \max_{y \neq 0} \frac{\|Ay\|_p}{\|y\|_p} \cdot \max_{x \neq 0} \frac{\|Bx\|_p}{\|x\|_p} = \|A\|_p \|B\|_p, \end{aligned}$$

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 Matrixberechnungen verwendet werden.

Lemma A.2.4 Es sei $A \in \mathbb{R}^{m \times n}$, dann gelten folgende Ungleichungen

$$\|A\|_2 \leq \|A\|_F \leq \sqrt{n} \|A\|_2, \quad (\text{A.12})$$

$$\|A\|_{\max} \leq \|A\|_2 \leq \sqrt{m n} \|A\|_{\max}, \quad (\text{A.13})$$

$$\frac{1}{\sqrt{n}} \|A\|_{\infty} \leq \|A\|_2 \leq \sqrt{m} \|A\|_{\infty}, \quad (\text{A.14})$$

$$\frac{1}{\sqrt{m}} \|A\|_1 \leq \|A\|_2 \leq \sqrt{n} \|A\|_1. \quad (\text{A.15})$$

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

$$\begin{aligned} \|A\|_{\infty} &= \max_{\|x\|_{\infty}=1} \|Ax\|_{\infty} = \max_{\|x\|_{\infty}=1} \left\{ \max_{1 \leq j \leq m} \left| \sum_{k=1}^n a_{jk} x_k \right| \right\} = \max_{1 \leq j \leq m} \left\{ \max_{\|x\|_{\infty}=1} \left| \sum_{k=1}^n a_{jk} x_k \right| \right\} \\ &= \max_{1 \leq j \leq m} \sum_{k=1}^n |a_{jk}|. \end{aligned} \quad (\text{A.16})$$

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:

$$\begin{aligned} \|A\|_1 &= \max_{\|x\|_1=1} \|Ax\|_1 = \max_{\|x\|_1=1} \sum_{j=1}^m \left| \sum_{k=1}^n a_{jk} x_k \right| = \max_{\|x\|_1=1} \sum_{j=1}^m \sum_{k=1}^n |a_{jk}| |x_k| \text{sign}(a_{jk} x_k) \\ &= \max_{\|x\|_1=1} \sum_{k=1}^n |x_k| \sum_{j=1}^m |a_{jk}| \text{sign}(a_{jk} x_k) = \max_{\|x\|_1=1} \sum_{k=1}^n |x_k| \sum_{j=1}^m |a_{jk}| \\ &= \max_{1 \leq k \leq n} \sum_{j=1}^m |a_{jk}|. \end{aligned} \quad (\text{A.17})$$

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 $A^T A$ zum Eigenwert $\mu^2 = \|A\|_2^2$ ist. Also ist $\|A\|_2^2$ eine Nullstelle des charakteristischen Polynoms von $A^T A$

$$\varrho(z) = \det(A^T A - \lambda I).$$

Man kann sogar zeigen, dass die 2-Norm von A die *Wurzel des größten Eigenwerts von $A^T A$* 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

$$g : \mathbb{R}^n \rightarrow \mathbb{R} \quad g(x) := \frac{1}{2} \frac{\|Ax\|_2^2}{\|x\|_2^2} = \frac{1}{2} \frac{x^T A^T A x}{x^T x} \quad (\text{A.18})$$

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 $i = 1, \dots, n$

$$\frac{\partial}{\partial x_i} g(z) = \left[x^T x \sum_{j=1}^n (A^T A)_{ij} x_j - (x^T A^T A x) x_i \right] / (x^T x)^2. \quad (\text{A.19})$$

Wegen $\nabla g(z) = 0$ folgt also

$$0 = z^T z \sum_{j=1}^n (A^T A)_{ij} z_j - (z^T A^T A z) z_i = \left(z^T z (A^T A z) - (z^T A^T A z) z \right)_i$$

für alle $i = 1, \dots, n$. Da $1 = \|z\|_2^2 = z^T z$, erhalten wir daraus die Gleichung

$$A^T A z = (z^T A^T A z) z = \|Az\|_2^2 z = \mu^2 z,$$

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 $A : X \rightarrow X$ eine lineare Abbildung. Die Operatornorm ist definiert durch

$$\|A\|_X := \sup_{X \ni x \neq 0} \frac{\|Ax\|_X}{\|x\|_X}. \quad (\text{A.20})$$

² ∇g ist der Gradient von g , also $\left(\nabla := \left(\frac{\partial}{\partial x_1}, \frac{\partial}{\partial x_2}, \dots, \frac{\partial}{\partial x_n} \right) \right)^T$ (als Spaltenvektor).

Analog zu (A.10) kann man leicht zeigen, dass

$$\sup_{x \neq 0} \frac{\|Ax\|_X}{\|x\|_X} = \sup_{x \neq 0} \left\| A \frac{x}{\|x\|_X} \right\|_X = \sup_{\|x\|_X=1} \|Ax\|_X. \quad (\text{A.21})$$

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 Operatornorm, dann gilt für alle $x \in X$

$$\|Ax\|_X \leq \|A\|_X \|x\|_X. \quad (\text{A.22})$$

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 $x \in \mathbb{R}^n$

$$\|Ax\| \leq \|A\| \|x\| \quad (\text{A.23})$$

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

$$\|A\|_2 \leq \sqrt{\|A\|_1 \|A\|_\infty}. \quad (\text{A.24})$$

Beweis. Nach Satz A.2.7 existiert ein Vektor $0 \neq z \in \mathbb{R}^n$ mit $A^T A z = \mu^2 z$ und $\mu = \|A\|_2$. Dann gilt wegen Bemerkung A.2.13 und Korollar A.2.11

$$\mu^2 \|z\|_1 = \|A^T A z\|_1 \leq \|A^T\|_1 \|A z\|_1 \leq \|A\|_\infty \|A\|_1 \|z\|_1, \quad (\text{A.25})$$

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:

$$Av = \lambda v. \quad (\text{A.26})$$

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

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

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 :

$$\det(A) = \prod_{i=1}^n \lambda_i, \quad \text{Spur}(A) := \sum_{i=1}^n a_{ii} = \sum_{i=1}^n \lambda_i.$$

Beweis. Die zweite Identität erhält man z.B. dadurch, daß man den Koeffizienten vor λ^{n-1}

im charakteristischen Polynom untersucht.

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

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 $\text{Bild}(X)$ 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 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

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

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 $A \in \mathbb{C}^{n \times n}$ hermitesch und $X \in \mathbb{C}^{n \times n}$ regulär. Dann haben A und $X^H A X$ 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

$$E_k(\lambda) = \begin{pmatrix} \lambda & 1 & & 0 \\ & \ddots & \ddots & \\ & & \ddots & 1 \\ 0 & & & \lambda \end{pmatrix}. \quad (\text{A.27})$$

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

$$A = T^{-1} J T,$$

wobei J , die durch die Paare $(\lambda_1, n_1), \dots, (\lambda_k, n_k)$ 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 $A \in \mathbb{C}^{n \times n}$. Dann gibt es eine unitäre Matrix $Q \in \mathbb{C}^{n \times n}$ ($Q Q^H = Q^H Q = I$), so daß

$$Q^H A Q = R$$

ist, wobei $R \in \mathbb{C}^{n \times n}$ eine obere Dreiecksmatrix ist und die Diagonaleinträge r_{11}, \dots, 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 $Q^T A Q$ in oberer Quasi-Dreiecksform

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

ist, wobei jedes R_{ii} entweder ein 1×1 oder 2×2 Block ist. Dabei sind die 1×1 Blöcke unter R_{11}, \dots, 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 $A^H A = A A^H$.

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 Matrizen. 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 $Q^H A Q = D = \text{diag}(\lambda_1, \dots, \lambda_n)$.

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

$$R^H R = Q^H A^H Q Q^H A Q = Q^H A^H A Q = Q^H A A^H Q = Q^H A Q Q^H A^H Q = R R^H.$$

Die Behauptung folgt nun aus der Tatsache, daß eine normale, obere Dreiecksmatrix eine Diagonalmatrix ist. ■

Korollar A.3.13 — Schur-Zerlegung von symmetrischen Matrizen. 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

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

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

Anhang B

Einige Grundlagen aus der Analysis

Wir stellen hier einige wenige Tatsachen aus der Analysis zusammen, die im vorliegenden Skript benötigt werden und die vielleicht nicht unbedingt zum Standard-Stoff jeder Analysis-Vorlesung gehören.

B.1 Der Satz von Rolle

Satz B.1.1 — Verallgemeinerter Satz von Rolle¹. Sei $f \in C^n([a, b])$ zwei oder mehr verschiedene Nullstellen der Gesamtordnung $n + 1$, so existiert ein $\xi \in (a, b)$ mit $f^{(n)}(\xi) = 0$.

Beweis. Vgl. [Wal17, S. 433-434]. ■

B.2 Der Satz von Taylor

Zu einer Funktion $f \in C^n([a, b])$ heißt

$$p_n(f|y; x) := \sum_{k=0}^n \frac{1}{k!} f^{(k)}(y) (x - y)^k$$

n -tes **Taylor-Polynom²** von f in $y \in [a, b]$. Der Ausdruck $R_n(f|x - y) := f(x) - p_n(f|y; x)$ heißt **Taylor-Restglied** und $f(x) = p_n(f|y; x) + R_n(f|x - y)$ ist als **Taylor-Formel** bekannt.

Satz B.2.1 — Satz von Taylor. Seien $f \in C^{n+1}([a, b])$, $y \in [a, b]$ und $h > 0$ so, dass $x = y + h \in [a, b]$. Dann gilt

$$R_n(f|h) = \frac{h^{n+1}}{n!} \int_0^1 (1-t)^n f^{(n+1)}(y + th) dt.$$

Beweis. Z.B. [Hil05, Satz III.13.1]. ■

¹Michel Rolle, 1652-1719.

²Brook Taylor, 1685-1731.

Es gibt weitere Darstellungen für das Restglied, u.a.³

$$R_n(f|x-y) = \frac{h^{n+1}}{n!} (1-\vartheta)^n f^{(n+1)}(y+\vartheta h) \quad (\text{Cauchy-Restgliedformel}),$$

$$R_n(f|x-y) = \frac{h^{n+1}}{n!} f^{(n+1)}(y+\tilde{\vartheta}h) \quad (\text{Lagrange-Restgliedformel}),$$

mit geeigneten $\vartheta, \tilde{\vartheta} \in (0, 1)$ und $x = y + h$.

Wenn f keine skalarwertige Funktion $f : \mathbb{R} \rightarrow \mathbb{R}$ ist, sondern ein **Skalarfeld** $f : \Omega \rightarrow \mathbb{R}$, $\Omega \subseteq \mathbb{R}^n$, dann übernimmt der Gradient $\nabla f = f'$ die Rolle der ersten Ableitung und die Hesse-Matrix⁴ $H_f = f''$ die Rolle der zweiten Ableitung. Gegeben sei also eine stetig partiell differenzierbare Funktion $f : \Omega \rightarrow \mathbb{R}$ mit $x = (x_1, \dots, x_n)^T$ und zusätzlich ein Entwicklungspunkt $y = (y_1, \dots, y_n)^T$. Dann heißen

$$\begin{aligned} p_0(f|y; x) &:= f(y) && \text{das nullte,} \\ p_1(f|y; x) &:= f(y) + \nabla f(y) \cdot (x - y) && \text{das erste,} \\ p_2(f|y; x) &:= f(y) + \nabla f(y) \cdot (x - y) + \frac{1}{2}(x - y)^T H_f(y)(x - y) && \text{das zweite,} \end{aligned}$$

Taylor-Polynom von f im **Entwicklungspunkt** y . Falls f hinreichend glatt ist, d.h. $f \in C^{n+1}(\Omega)$, gilt wie im eindimensionalen $|f(x) - p_n(f|y; x)| = \mathcal{O}(\|x - y\|^{n+1})$.

Allgemeiner lautet die **Taylor-Reihe** von $f \in C^\infty(\Omega, \mathbb{R}^m)$ in einem Entwicklungspunkt $y \in \Omega$

$$\sum_{s=0}^{\infty} \sum_{|\alpha|=s} \frac{1}{\alpha!} D^\alpha f(y) (x - y)^\alpha.$$

Mehr zur Taylor-Entwicklung im Mehrdimensionalen inklusive der Konvergenzanalyse findet man z.B. in [Hil03, Kap. I.6].

³Augustin-Louis Cauchy, 1789-1857; Joseph-Louis Lagrange, 1736-1813.

⁴Ludwig Otto Hesse, 1811-1874.

Anhang C

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].

C.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

>>

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.

C.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 \	$17/4 = 4 \setminus 17$
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 ausgewertet werden.

C.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
<code>ans</code>	standard Variablenname benutzt für Ergebnisse
<code>pi</code>	3.1415...
<code>eps</code>	Maschinengenauigkeit
<code>flops</code>	Zähler für die Anzahl der Fließkommaoperationen
<code>inf</code>	steht für Unendlich (eng. infinity). z.B. 1/0
<code>NaN</code>	englisch <i>Not a Number</i> , z.B. 0/0
<code>i</code> (und) <code>j</code>	$i = j = \sqrt{-1}$

Der Speicherplatz, der durch Variablen belegt ist, durch den Befehl `clear` wieder freigegeben werden, z.B.

```
>> clear a b c
```

C.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.

C.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
<code>abs(x)</code>	Absolutbetrag
<code>cos(x)</code>	Kosinus
<code>exp(x)</code>	Exponentialfunktion: e^x
<code>fix(x)</code>	rundet auf die nächste, vom Betrag her kleinere ganze Zahl
<code>floor(x)</code>	rundet auf die nächste, kleinere ganze Zahl
<code>gcd(x,y)</code>	größter gemeinsamer Teiler von x und y
<code>lcm(x,y)</code>	kleinstes gemeinsames Vielfaches von x und y
<code>log(x)</code>	natürlicher Logarithmus
<code>rem(x,y)</code>	Modulo (engl. remainder of division), z.B. <code>rem(5,2)=1</code>
<code>sign(x)</code>	Signum Funktion, z.B. <code>sign(2.3) = 1</code> , <code>sign(0) = 0</code> , <code>sign(-.3) = -1</code>
<code>sin(x)</code>	Sinus
<code>sqrt(x)</code>	Quadratwurzel
<code>tan(x)</code>	Tangens

Ein kleines Beispiel:

```
>> y = cos(pi)
y =
    -1
```

C.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	
<code>disp(ans)</code>	zeigt den Wert der Variablen <code>ans</code> , ohne ihren Namen auszugeben
<code>input</code>	erwartet vom Benutzer eine Eingabe
<code>keyboard</code>	übergibt zeitweise die Kontrolle an die Tastatur
<code>pause</code>	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
```

C.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-Management-Funktionen	
<code>cd path</code>	wechselt in das Verzeichnis <code>path</code>
<code>delete beispiel</code>	löscht die Datei <code>beispiel.m</code>
<code>ls</code>	zeigt alle Dateien im aktuellen Verzeichnis an
<code>pwd</code>	zeigt den aktuellen Verzeichnispfad an
<code>type beispiel</code>	zeigt den Inhalt der Datei <code>beispiel.m</code> im Befehl-Fenster
<code>what</code>	zeigt alle M- und MAT-Dateien im aktuellen Verzeichnis an

C.1.7 Hilfe

MATLAB® bietet verschiedene Hilfoptionen 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 chollesky

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.

C.2 Mathematik mit Matrizen

MATLAB® ist eine Abkürzung für **Matrix Laboratory**. 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.

C.2.1 Matrixkonstruktion und Adressierung

Wir zeigen zunächst einige Möglichkeiten, Matrizen in MATLAB® zu definieren.

einfache Matrix Konstruktionen	
<code>x=[1 4 2*pi 4]</code>	erstelle einen Zeilenvektor x mit Einträgen
<code>x=anfang:ende</code>	erstelle einen Zeilenvektor x beginnend mit <code>anfang</code> , Inkrement 1 und endend mit <code>ende</code>
<code>x=anfang:inkrement:ende</code>	Ähnliches wie oben mit dem Inkrement <code>inkrement</code>
<code>x=linspace(anfang,ende,n)</code>	erzeugt einen Zeilenvektor der Dimension n mit
	$x(i) = \frac{(n-i) \cdot \text{anfang} + (i-1) \cdot \text{ende}}{n-1}$

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
```

C.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
```

C.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 \backslash und $.\backslash$ zu unterscheiden. In nachfolgender Tabelle sind die Matrixoperationen beschrieben:

komponentenweise Matrixoperationen	
Beispieldaten	$a = [a_1, a_2, \dots, a_n], b = [b_1, b_2, \dots, b_n], c$ ein Skalar
komp. Addition	$a+c = [a_1 + c, a_2 + c, \dots, a_n + c]$
komp. Multiplikation	$a*c = [a_1 \cdot c, a_2 \cdot c, \dots, a_n \cdot c]$
Matrix-Addition	$a+b = [a_1 + b_1, a_2 + b_2, \dots, a_n + b_n]$
komp. Matrix-Multiplikationen	$a.*b = [a_1 \cdot b_1, a_2 \cdot b_2, \dots, a_n \cdot b_n]$
komp. Matrix-Div. von rechts	$a./b = [a_1/b_1, a_2/b_2, \dots, a_n/b_n]$
komp. Matrix-Div. von links	$a.\backslash b = [b_1/a_1, b_2/a_2, \dots, b_n/a_n]$
komp. Matrix-Potenz	$a.^c = [a_1^c, a_2^c, \dots, a_n^c]$ $c.^a = [c^{a_1}, c^{a_2}, \dots, c^{a_n}]$ $a.^b = [a_1^{b_1}, a_2^{b_2}, \dots, a_n^{b_n}]$

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
```

C.2.4 Matrix-Operationen und -Funktionen

Nun einige Operationen die sich auf Matrizen anwenden lassen:

Matrixfunktionen	
<code>reshape(A,m,n)</code>	erzeugt aus den Einträgen der Matrix A eine $m \times n$ -Matrix, wobei die Einträge spaltenweise aus A gelesen werden.
<code>diag(A)</code>	ergibt die Diagonale von A als Spaltenvektor
<code>diag(v)</code>	erzeugt Diagonalmatrix mit dem Vektor v in der Diagonalen
<code>tril(A)</code>	extrahiert den unteren Dreiecksanteil der Matrix A
<code>triu(A)</code>	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 C.2.1 Der \ Operator ist auch für Matrizen definiert und liefert in Kombination mit Vektoren für reguläre Matrizen ihre Inverse, d.h. $A \backslash x = A^{-1}x$.

C.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

C.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
<code>find(A)</code>	findet Indizes von Nichtnulleinträgen
<code>nnz(A)</code>	Anzahl an Nichtnulleinträgen
<code>spdiags(v)</code>	erzeugt eine Sparse-Diagonalmatrix mit Vektor <i>v</i> als Diagonale
<code>speye(n)</code>	erzeugt eine Sparse-Einheitsmatrix
<code>spy(A)</code>	visualisiert die Struktur der Matrix <i>A</i>

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 Besetzungsstruktur einer 3D-FEM¹ Steifigkeitsmatrix in Abb. C.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 $256 \times 256 = 65.536$ Einträgen).

C.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.

C.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.

¹FEM: 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.

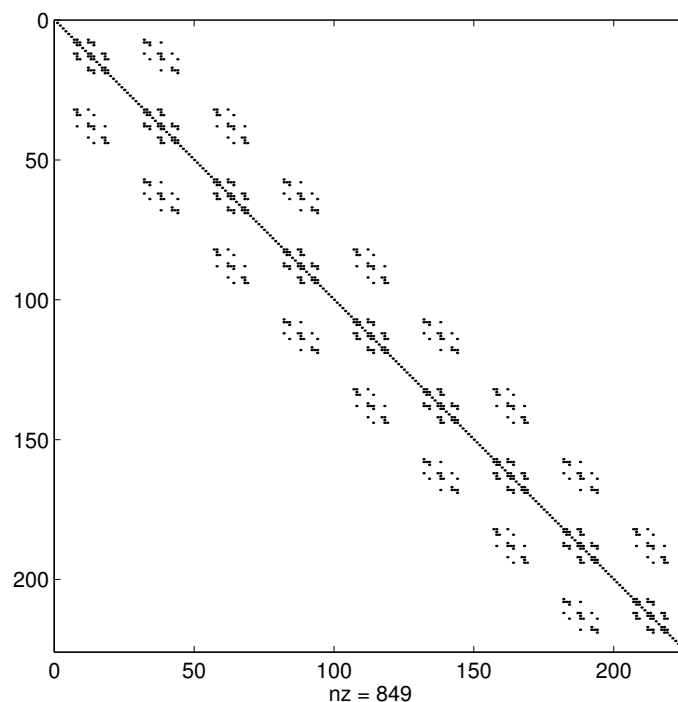


Abbildung C.1: Besetzungsstruktur einer 3D-FEM Steifigkeitsmatrix.

```
>> 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.

C.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

C.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
<code>fprintf('%.0e\n',1.234567)</code>	1e00+
<code>fprintf('%.2e\n',1.234567)</code>	1.23e00+
<code>fprintf('%.5e\n',1.234567)</code>	1.23456e00+
<code>fprintf('%10.0e\n',1.234567)</code>	1e00+
<code>fprintf('%10.2e\n',1.234567)</code>	1.23e00+
<code>fprintf('%10.5e\n',1.234567)</code>	1.23456e00+
<code>fprintf('%10.2f\n',1.234567)</code>	1.23
<code>fprintf('%10.5f\n',1.234567)</code>	1.23457

Beachte: MATLAB® rundet numerische Werte bei der Ausgabe, wenn nötig!



C.5 Kontrollbefehle

Man kann in MATLAB® analog zu einer höheren Programmiersprache programmieren. Wir zeigen einige Beispiele dazu.

C.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
```

C.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
```

C.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

```

C.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

C.6 Graphische Darstellung

Ergebnisse von numerischen Simulationen müssen in vielen Fällen graphisch dargestellt werden. Hierzu sind in MATLAB® einige sehr nützliche Befehle realisiert.

C.6.1 Zweidimensionale Graphiken

Wir beginnen mit einigen Befehlen zur Darstellung von Funktionen $f : \mathbb{R} \rightarrow \mathbb{R}$ in einem zweidimensionalen Graphen.

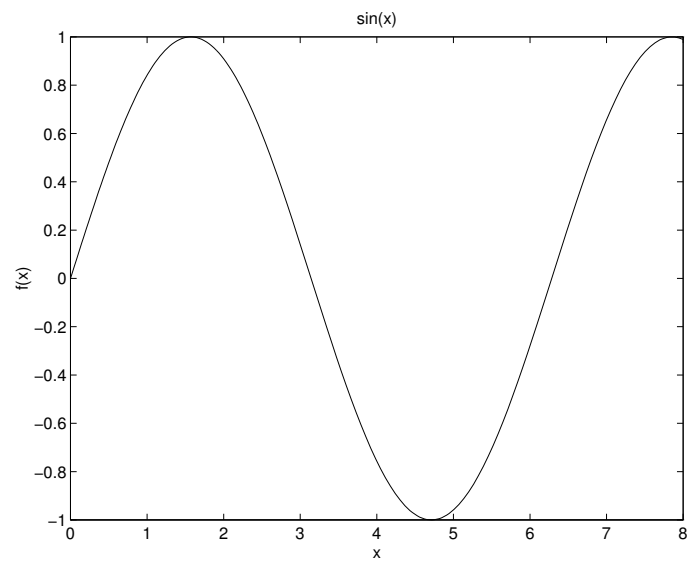
```

>> f='sin(x)';
>> fplot(f,[0 8]);
>> title(f),xlabel('x'),ylabel('f(x)');

```

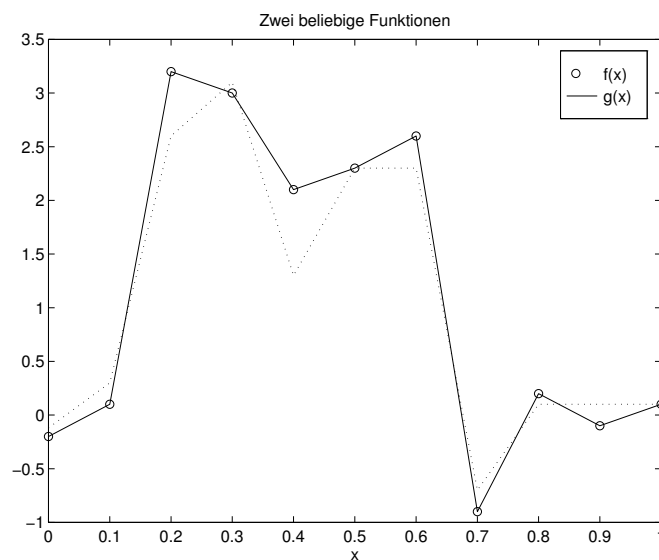
Das Ergebnis sieht dann wie folgt aus:

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	.	Punkt
m	magenta	o	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

C.6.2 Dreidimensionale Graphiken

Funktionen $f : \mathbb{R}^3 \rightarrow \mathbb{R}$ 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

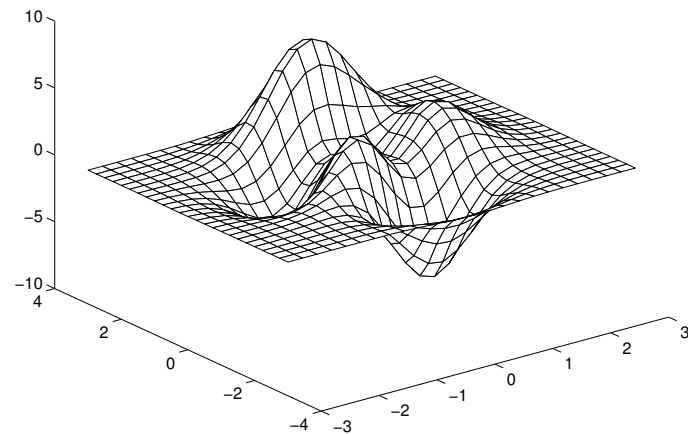
$$f(x, y) := 3(1 - x^2)e^{-x^2 - (y+1)^2} - 10\left(\frac{x}{5} - x^3 - y^5\right)e^{-x^2 - y^2} - \frac{1}{3}e^{-(x+1)^2 - y^2}$$

visualisieren. Diese Funktion ist durch den Befehl `peaks` in MATLAB® realisiert.

```
>> [X,Y,Z] = peaks(25)
>> mesh(X,Y,Z)
```

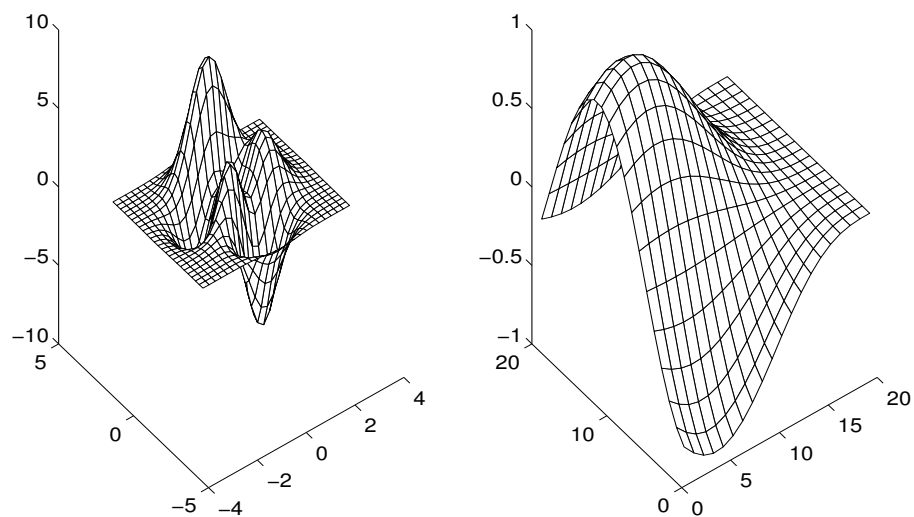
Als Ergebnis erhalten wir:

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:



C.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 Postscript
	-depssc	Encapsulated Color Postscript
Optionen	-P<Drucker>	Spezifiziert den zu benutzenden Drucker

Die Eingabe

```
>> print fig4 -deps
```

erzeugt die Datei fig4.eps.

C.7 Fortgeschrittenes

C.7.1 MATLAB®-Skripte

Wie schon in Abschnitt C.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 Editor² 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 Pfad³ 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).

²Dies kann, muss aber nicht, der MATLAB®-interne Editor sein.

³Deswegen muss man den entsprechenden Pfad vorher setzen!

C.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).
- [Bor16] F. Bornemann. *Numerische lineare Algebra. Eine konzise Einführung mit MATLAB und Julia*. Springer, 2016 (siehe S. iii).
- [Can+06] C. Canuto, M. Y. Hussaini, A. Quarteroni und T. Zang. *Spectral methods. Fundamentals in Single Domains*. Springer-Verlag, Berlin, 2006 (siehe S. 146).
- [DR08] W. Dahmen und A. Reusken. *Numerik für Ingenieure und Naturwissenschaftler*. 2. Aufl. Berlin: Springer, 2008 (siehe S. 3).
- [DH19] P. Deufhard und A. Hohmann. *Numerische Mathematik 1*. 5. Aufl. De Gruyter Studium. Eine algorithmisch orientierte Einführung. De Gruyter, Berlin, 2019 (siehe S. iii, 143).
- [Fab14] G. Faber. „Über die interpolatorische Darstellung stetiger Funktionen“. In: *Jahresbericht DMV* 23 (1914), S. 192–210 (siehe S. 105).
- [Fis13] G. Fischer. *Lineare Algebra*. 18. Aufl. Grundkurs Mathematik [Foundational Course in Mathematics]. Vieweg-Verlag, Braunschweig, 2013 (siehe S. A-9).
- [FH07] R. Freund und R. Hoppe. *Stoer/Bulirsch: Numerische Mathematik 1*. 10. Auflage. Springer, 2007 (siehe S. iii).
- [GV13] G. H. Golub und C. F. Van Loan. *Matrix computations*. 4. Aufl. Johns Hopkins University Press, Baltimore, MD, 2013 (siehe S. iii).
- [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, 71).
- [Har22] H. Harbrecht. *Algorithmische Mathematik. Graphen, Numerik und Probabilistik*. Springer Spektrum, 2022 (siehe S. iv).
- [Hil03] S. Hildebrand. *Analysis 2*. Springer-Verlag, Berlin, 2003 (siehe S. B-0).
- [Hil05] S. Hildebrand. *Analysis 1*. 2. Aufl. Springer-Verlag, Berlin, 2005 (siehe S. B-1).

- [HW64] R. A. Hunt und G. Weiss. „The Marcinkiewicz interpolation theorem“. In: *Proc. Amer. Math. Soc.* 15 (1964), S. 996–998 (siehe S. 105).
- [Kan05] C. Kanzow. *Numerik linearer Gleichungssysteme. Direkte und iterative Verfahren*. 1. Aufl. Springer, 2005 (siehe S. iii).
- [Mar39] J. Marcinkiewicz. „Sur l’interpolation d’operations“. In: *C.R. Acad. Sci. Paris* 208 (1939), S. 1272–1273 (siehe S. 105).
- [Mei15] A. Meister. *Numerik linearer Gleichungssysteme: Eine Einführung in moderne Verfahren*. Mit MATLAB®-Implementierungen von C. Vömel. Wiesbaden: Springer Spektrum, 2015 (siehe S. iii).
- [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, 137).
- [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, C-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).
- [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, 139).
- [SB90] J. Stoer und R. Bulirsch. *Numerische Mathematik*. 2. 3. Aufl. Springer-Lehrbuch. Springer-Verlag, Berlin, 1990 (siehe S. iii).
- [Sze67] G. Szegő. *Orthogonal polynomials*. 3. Aufl. American Mathematical Society, Providence, R.I., 1967 (siehe S. 118).
- [Wal17] G. Walz, Hrsg. *Lexikon der Mathematik. Band 4, Moo bis Sch*. 2. Aufl. Springer Spektrum, 2017 (siehe S. B-1).
- [Wer18] D. Werner. *Funktionalanalysis*. 8. Aufl. Springer Berlin, 2018 (siehe S. 128).
- [WS78] H. Werner und R. Schaback. *Praktische Mathematik II*. 2. Aufl. Springer-Verlag, Berlin, 1978 (siehe S. 105).
- [Zyg59] A. Zygmund. *Trigonometric Series*. Cambridge University Press, 1959 (siehe S. 105).