# Numerik 1

**Skriptum zur Vorlesung WS 2000/01**

Hermann Schichl

# Inhaltsverzeichnis


Kapitel 1. Einleitung I 3
1. Vorbemerkungen 3
2. Computer 5
3. Zahlendarstellung im Computer 12
4. Auswertung von Ausdrücken 15
5. Fehler 19

Kapitel 2. Modellierung I 39
1. Modelle 39
2. Beispiele 46
3. Andere Anwendungen 54

Kapitel 3. Numerische Lineare Algebra I: Lineare Gleichungssysteme 57
1. Grundlagen 57
2. Direkte Verfahren 64
3. Fehleranalyse 75
4. Systeme mit speziellen Eigenschaften 87
5. Software 93

Kapitel 4. Numerische Lineare Algebra II: Über– und unterbestimmte lineare Gleichungssysteme 97
1. Grundlagen 97
2. Lineare überbestimmte Systeme 98
3. Lineare unterbestimmte Systeme 126
4. Software 127

Kapitel 5. Interpolation I: Eindimensionaler Fall 129
1. Grundlagen 129
2. Rationale Interpolation 137
3. Trigonometrische Interpolation 148
4. Splines 156
5. Extrapolation 171
6. Software 173

Kapitel 6. Differentialrechnung und Integration I: Eindimensionaler Fall 175
1. Grundlagen 175
2. Integration - einfache Verfahren 177
3. Extrapolationsverfahren 180
4. Interpolatorische Quadraturformeln 185
5. Gauß–Quadratur 187
6. Adaptive Verfahren 192

7. Transformationen 193
8. Numerische Differentiation 194
9. Software 199

Kapitel 7. Nichtlineare Gleichungssysteme I: Eindimensionaler Fall, Nullstellen 201
1. Grundlagen 201
2. Bisektionsverfahren 202
3. Sekantenverfahren 204
4. Global superlinear konvergente Verfahren 209
5. Genauere Konvergenzuntersuchung 212
6. Fehleranalyse 215
7. Newton-Verfahren 216

Kapitel 8. Datenanalyse und Funktionsapproximation 217
1. Grundlagen 217
2. Auswertung von Reihen, Konvergenzbeschleunigung 217
3. Algorithmen zur Klassifikation 217

Kapitel 9. Numerische Lineare Algebra III: Eigenwertprobleme 219
1. Grundlagen 219
2. Transformationsverfahren 222
3. QR–Verfahren 224
4. Andere Verfahren 233
5. Berechnung der Singulärwertzerlegung 236
6. Software 240

Kapitel 10. Numerische Lineare Algebra IV: Iterationsverfahren 241
1. Grundlagen 241
2. Arnoldi Iteration 242
3. Das GMRES Verfahren 246
4. Lanczos Iteration 248
5. Konjugierte Gradienten 250
6. Überblick über andere Iterationsverfahren 252
7. Vorkonditionierung 253

Literaturverzeichnis 255

KAPITEL 1

# Einleitung I

## 1. Vorbemerkungen

Die Numerische Mathematik gehört im Gegensatz zu anderen Gebieten wie die Zahlentheorie oder die Topologie hauptsächlich zur angewandten Mathematik, also zu dem Teil der Mathematik, der sich vorwiegend darum kümmert, anderen Wissenschaften *Methoden* zur Verfügung zu stellen, mit deren Hilfe sich Probleme bearbeiten lassen. Das bedeutet nicht, daß die Topologie oder die Zahlentheorie keine Verwendung außerhalb der Mathematik finden, doch liegt in diesen Gebieten das Hauptaugenmerk im allgemeinen nicht auf den Anwendungen.

Was sind die Auswirkungen dieser Ausrichtung auf die Anwendungen, wenn man die Arbeit eines Numerischen Mathematikers (eines Angewandten Mathematikers) und den Aufbau der Vorlesung betrachtet?

- Die Arbeit eines Angewandten Mathematikers beginnt meistens zwei Schritte vor der rein mathematischen Arbeit mit dem *Gespräch mit einem Anwender* über dessen zu behandelnde Probleme.
  - Er muß dabei die „Sprache“ des Anwenders verstehen lernen:
    Was bedeuten die Begriffe?
    Welche *unausgesprochenen* Beziehungen haben die einzelnen Begriffe miteinander?
  - Der Kern des Problems muß herausgearbeitet werden:
    Was soll berechnet/simuliert werden?
    Was ist für den Anwender ein *brauchbares* Ergebnis?
    Wie schnell und wie oft benötigt der Anwender eine Berechnung?
- Der nächste Schritt ist die *Modellierung des Problems*, das heißt die Übersetzung aus der Sprache des Anwenders in den mathematischen Formalismus (siehe 2). Erst dieser Schritt, in dem unwesentliche Objekte, Zusammenhänge und Komplikationen unterdrückt (wegabstrahiert) werden, macht das Problem angreifbar durch mathematische Methoden. Dabei muß umsichtig vorgegangen werden, denn zu starke *Abstraktion* macht die Ergebnisse, die aus dem Modell gewonnen werden, unbrauchbar, weil sie mit dem tatsächlichen Problem zu wenig Ähnlichkeit aufweisen oder wesentliche Vorgänge nicht berücksichtigt werden. Zu wenig Abstraktion auf der anderen Seite macht das Modell unübersichtlich und birgt die Gefahr, daß es für die mathematischen Methoden unbehandelbar (zu komplex) wird.
- Danach wird die gängige mathematische Theorie verwendet, um einen *Lösungsalgorithmus* zu entwickeln und zu beweisen, daß er auch in der Lage ist, das *mathematische Problem* zu lösen (Konvergenz, Robustheit,...). Dieser Algorithmus wird dann auf einem *Computer* (siehe 2) entweder neu implementiert oder mit bereits vorhandener *Software* realisiert.
- Dann wird das mathematische Problem gelöst, wobei dem Übersetzen und der Analyse der Eingabedaten eine nicht zu unterschätzende Bedeutung zukommt. Außerdem sollte der maximale Gesamtfehler abgeschätzt werden, um die Genauigkeit der Approximation der Lösung des mathematischen Modells zu bestimmen.

- Auch nach Lösung des mathematischen Problems sind noch zwei weitere Schritte anzuschließen, die bei rein mathematischen Fragestellungen im allgemeinen nicht oder nicht in diesem Ausmaß nötig sind. Der erste ist die *Interpretation der Ergebnisse*. Dabei werden die vom Algorithmus produzierten Zahlen wieder in Begriffe und Größen der „Wirklichkeit des Anwenders“ rückübersetzt, zusammen mit den Fehlerschranken, damit die Brauchbarkeit der Lösung überprüft werden kann.
- Die letzte Aufgabe ist dann die interpretierten Ergebnisse bzw. die Vorhersagen des Modells mit der Wirklichkeit zu vergleichen, um die *Gültigkeit des Modells überprüfen* zu können. Sollte sich das Modell dabei als unbrauchbar erweisen, so muß erneut mit dem Modellierungsschritt begonnen werden. Eine Analyse der Abweichungen kann dabei einen Hinweis auf falsche oder zu starke Vereinfachungen, auf falsche mathematische Übersetzungen oder einen zu kleinen Gültigkeitsbereich des Modells liefern, die zur Unbrauchbarkeit des Modells geführt haben könnten.

**1.1. Struktur der Vorlesung und des Proseminars.** Die Vorlesung beschäftigt sich nach einigen einführenden Abschnitten über grundlegende Probleme bei der mathematischen Verwendung von Computern und einem Kapitel über Modelle, Modellbildung und Modellprüfung sowie einigen Beispielen im weiteren Verlauf mit den mathematischen Grundlagen der Numerischen Mathematik. Dabei liegt im ersten Teil der zweiteiligen Vorlesung das Hauptaugenmerk auf der Linearen Algebra, die der Hauptbestandteil der meisten numerischen Algorithmen ist. Als Ergänzung dienen noch Grundlagen der Datenanalyse und eindimensionale Interpolation, Integration und die Lösung nichtlinearer Gleichungssysteme. Während der gesamten Vorlesung wird versucht, die Relevanz der Methoden und deren Anwendbarkeit in den Vordergrund zu stellen, moderne Verfahren zu erläutern auch wenn mathematische Beweise über deren Funktionstüchtigkeit nur aus der Literatur zitiert werden können. Manche Stoffgebiete, die üblicherweise in Numerik–Vorlesungen vorgetragen werden, die aus heutiger Sicht jedoch bereits weniger relevant sind, oder die keine hohe Komplexität besitzen, sollen im Proseminar anhand von kleinen Programmen und Kurzvorträgen von den Studenten unter Anleitung selbst erarbeitet werden.

**1.2. Relevanz der Vorlesung: Berufsleben, Zeitungsausschnitte.** Die Relevanz der angewandten Mathematik und damit der numerischen Mathematik als einer Grundlage im Berufsleben liegt auf der Hand. Möchte man als Mathematik außerhalb der Universität in der Privatwirtschaft eine Anstellung finden, so steht typischerweise eines der folgenden Jobprofile zur Auswahl: Software-Entwicklung, Versicherungen, Banken, mathematische Berechnungen, Management, Unternehmensberater.

Für die ersten vier Jobs ist angewandte Mathematik und die damit verbundene Beschäftigung mit Computern eine wichtige Grundlage. Besonders wenn es um Berechnungen geht, sind Firmen an Mathematikern vor allem interessiert, die aus Fragestellungen relevante Ergebnisse berechnen können. Die am häufigsten in der Privatwirtschaft untersuchten Probleme kommen aus den Gebieten Optimierung, Differentialgleichungen und Statistik (werden alle zumindest zum Teil in Numerik 2 abgehandelt).

Wählen Firmen Mathematiker für Management–Positionen oder zur Unternehmensberatung, so sind sie primär an der analytischen Denkfähigkeit interessiert. Für solche Positionen sind angewandte Mathematiker nicht grundsätzlich bevorzugt, doch hilft die „Grundausbildung im Gespräch mit dem Anwender“ auch in diesem Fall.

Daß mathematische Methoden, obwohl bekannt, nicht immer richtig angewendet werden, zeigen zwei Katastrophen in der Raketentechnologie in der jüngeren Vergangenheit:

- Der Tod von 28 amerikanischen Soldaten im Golfkrieg am 25. Februar 1991 nach einem Angriff mit einer irakischen Scud–Rakete hätte verhindert werden können, wenn das Patriot Raketenabwehrsystem nicht versagt hätte.
- Der Jungfernflug der Ariane 5 Rakete endete am 4. Juni 1996 nur vierzig Sekunden nach dem Abheben in einer gigantischen Explosion. Zwei einige Milliarden Dollar teure Satelliten wurden bei diesem Unfall zerstört.

Beide Katastrophen hatten Ursachen, die bei konsequenter Anwendung der Grundprinzipien der numerischen Mathematik vermeidbar gewesen wären.

## 2. Computer

Das wichtigste Hilfsmittel neben der mathematischen Theorie ist in der Angewandten Mathematik der *Computer*. Bei der Verwendung eines Computersystems spielen drei Grundelemente die wesentlichen Rollen:

**Hardware:** „Der Teil des Computers, den man sieht.“

**Betriebssystem:** Ist spezielle Software (Programme), mit deren Hilfe Benutzerprogramme gestartet, die verschiedenen Benutzer verwaltet werden, und die sich um die Kommunikation mit den Peripheriegeräten kümmert.

**Anwendersoftware:** Alle Programme, die auf einem Computer laufen — hier werden unter diesem Punkt nur den Mathematiker unterstützende Programmpakete angesprochen.

**2.1. Hardware.** Im Gegensatz zu früheren Jahren sind im Moment fast ausschließlich *Digitalrechner* im Einsatz. Die Bedeutung von *Analogrechnern* (Rechner, die mit kontinuierlichen Größen rechnen. In ihnen werden mathematische Probleme durch analoge physikalische Prozesse simuliert) hat nicht zuletzt durch die große Steigerung der Geschwindigkeit der Digitalrechner in den letzten zwei Jahrzehnten praktisch auf Null abgenommen. Der Rechenschieber wurde durch den Taschenrechner völlig verdrängt, und auch alle Aufgaben eines Planimeters können durch Digitalrechner mit geeigneter Software leicht ersetzt werden. Aus diesem Grund wird auf Analogrechner im weiteren Verlauf der Vorlesung auch nicht mehr eingegangen.

Der Grundaufbau der Computerhardware besteht heutzutage aus folgenden Teilen:

**CPU:** Die CPU (central processing unit) ist das Herzstück eines modernen Computers. In ihr werden alle Rechenvorgänge und die meisten Steuervorgänge ausgeführt. Sie besteht meist aus einem *Chip* (integrierter Schaltkreis, ein Halbleiterbaustein) von 0.4–5 $\text{cm}^2$ Größe, auf dem mehrere Millionen bis Milliarden Transistoren untergebracht sind.

Die Leistungsfähigkeit einer CPU hängt von mehreren Faktoren ab: Der Güte der Implementation der elementaren Vorgänge wie *Speicherzugriff*, *ganzzahlige Rechnung (Integerarithmetik)*, *Gleitkommaarithmetik (Floating–Point–Arithmetik)* und *Sprünge und Unterprogrammaufrufe*; der *Taktfrequenz* (im wesentlichen der Anzahl der abgearbeiteten Befehle pro Sekunde); der Anzahl der Befehle, die gleichzeitig (parallel) abgearbeitet werden können, bzw. der Größe der Datenmenge, die in einem einzigen Arbeitsschritt manipuliert werden kann (Vektorisierung). Im Normalfall hat ein Computer nur eine CPU, es kommen jedoch in letzter Zeit schon PCs auf den Markt, die mit zwei bis vier CPUs ausgestattet werden können, die Benutzerprogramme parallel verarbeiten. Die Anbieter größerer Rechnersysteme verkaufen schon seit längerer Zeit Computeserver mit vier bis 128 CPUs. Hersteller

von Spezialrechensystemen offerieren jedoch für Spezialaufgaben, bei denen massive Parallelverarbeitung von Vorteil ist, auch Systeme mit mehr als 100000 CPUs (Transputer).

Die Gesamtleistung solcher Systeme und der einzelnen CPUs wird mit verschiedenen *Benchmark*–Tests bestimmt, bei denen für bestimmte häufig anfallende Probleme Ausführungszeiten bestimmt werden. Für mathematische Anwendungen sind die SPEC–Flops (floating point operations per second) und die diversen LINPACK– und LAPACK–Tests wesentlich.

**Hauptspeicher:** Der Hauptspeicher (RAM — random access memory) ist das zentrale Gedächtnis des Computers. In diesem Teil müssen alle Daten, die für momentan laufende Berechnungen benötigt werden, sowie die gerade ausgeführten Programme abgelegt werden. Jedes Datum, das im Hauptspeicher gelagert wird, hat eine eindeutige *Adresse*, über die Programme auf dessen Wert zugreifen können. Das RAM ist mit der CPU über mehrere Leitungen verbunden, von denen die wichtigsten die Adress– und die Datenleitungen sind. Die CPU teilt dem Hauptspeicher über die Adressleitungen die Adresse des benötigten Datums mit und der Hauptspeicher „antwortet“ auf diese Anfrage auf den Datenleitungen. Der Hauptspeicher hält nur dann Daten, wenn der Computer eingeschaltet ist. Bei Unterbrechung der Stromversorgung gehen alle Daten verloren, die im Hauptspeicher gelagert sind.

Nachdem es zwei grundlegend verschiedene Techniken gibt, Speicherbausteine herzustellen, die *dynamischen Speicher* (bieten eine hohe Speicherdichte, haben aber eine längere Zugriffszeit — die Zeit zwischen Bekanntgabe der Adresse und der Antwort des Speichers) und die *statischen Speicher* (bieten eine hohe Zugriffsgeschwindigkeit, aber eine geringe Speicherdichte), werden in allen modernen Computersystemen zwischen RAM und CPU ein oder mehrere Zwischenspeicher eingerichtet (der *Cache*), die mit eigenen Logikbausteinen dafür sorgen, daß Daten, die bald benötigt bzw. häufig wiederverwendet werden, vom Hauptspeicher in die Zwischenspeicher kopiert werden, um der CPU raschestmöglichen Zugriff auf die benötigten Daten zu ermöglichen.

Die Größe des Hauptspeichers wird in *Byte* gemessen. Ein Byte besteht aus acht *Bit*, der kleinsten Informationseinheit — eine Binärstelle. Durchschnittliche PCs haben etwa 32–64 MB (Megabyte — 1 Megabyte = $2^{20}$ Byte) Hauptspeicher, während bei größeren Workstations Speicherkapazitäten von 256 MB–2 GB (Gigabyte — 1 GB = 1024 MB = $2^{30}$ Byte) nicht unüblich sind. In großen Datenbankservern oder in Rechnersystemen, die für riesige Simulationen verwendet werden, sind allerdings auch Hauptspeicher von 128 GB–2 TB (Terabyte — 1 TB = 1024 GB = $2^{40}$ Byte) eingebaut.

Bei modernen Betriebssystemen ist es darüber hinaus möglich, das RAM dadurch scheinbar zu vergrößern, daß Teile davon auf die Festplatte ausgelagert werden (geswappt), wenn sie gerade nicht benötigt werden. Solch eine *virtuelle Speicherverwaltung (virtual memory management)* kann auf Kosten von Festplattenkapazität den Platz des Hauptspeichers auf ein Vielfaches erhöhen. Allerdings kann es bei einem zu großen Mißverhältnis zwischen tatsächlich vorhandenem RAM und benötigtem Hauptspeicher durch ständiges Austauschen von Daten zwischen Festplatte und Hauptspeicher zu einem Einbruch der Performance führen.

**BIOS:** Das BIOS (basic input/output system) dient der grundlegenden Kommunikation mit der Steuerlogik für die Peripheriegeräte. Es führt auch den Selbsttest von CPU und Speicher durch und lädt das Bootprogramm von Diskette, CDROM oder Harddisk. Es ist in einem ROM (read only memory), genauer einem EEPROM

(electrically eraseable programmable ROM), von geringer Kapazität (ca. 64 kB (1 kilo Byte = $2^{10}$ Byte)) untergebracht.

**Festplatte:** Die Festplatte (Harddisk) dient einerseits zur Sicherung der Daten, die nicht für gerade laufende Programme benötigt werden, und andererseits zur Sicherung aller Daten und Programme auch während der Zeit, in denen der Computer abgeschaltet ist.

Harddisks können viel größere Datenmengen speichern als im Hauptspeicher abgelegt werden können, und durch Einbau mehrerer Harddisks kann die Speicherkapazität eines Computersystems beinahe unbegrenzt ausgebaut werden. Ein durchschnittlicher PC hat in der heutigen Zeit eine Harddiskkapazität von mehr als 10 GB. Ein Fileserver eines durchschnittlichen mittleren computerisierten Betriebes stellt den Benutzern eine Speicherkapazität von 50–500 GB zur Verfügung. Die großen Datenbankserver von Krankenhäusern oder Bibliotheken haben Speicherkapazitäten jenseits von 128 TB.

Die Zugriffszeit auf die Daten auf der Festplatte liegen in der Höhe einiger Millisekunden (bei perfekter Ausnutzung aller technischer Möglichkeiten in der Höhe vieler Mikrosekunden), während die Zugriffszeit von dynamischen RAMs bei 45 Nanosekunden liegt. Daher wird im Interesse einer schnellen Verarbeitung auch in diesem Bereich mit Cachespeichern gearbeitet, die die Zugriffsgeschwindigkeit erheblich erhöhen können.

**Konsole:** Mit Konsole (console) werden die Ein– und Ausgabegeräte zur Benutzerkommunikation zusammengefaßt: Die Tastatur, der Monitor, die Maus sind heute Standardausrüstung jeder Computerkonsole. In den letzten Jahren wurden aber immer mehr PCs und Workstations zusätzlich mit Multimediazubehör ausgestattet wie Mikrophonen, Tonerzeugern (bis zu HiFi Qualität), Videoschnittstellen, ... Im CAD– und Bildbearbeitungsbereich sind Grafiktabletts und Digitalisierlupen weit verbreitet und bei Computern, die im öffentlichen Bereich eingesetzt werden, werden Tastatur, Maus und Monitor oft zu einem Touchscreen (berührungsempfindlicher Bildschirm) zusammengefaßt. Die neueste Entwicklung sind Spracheingabesysteme, die versuchen, Anweisungen in natürlicher Sprache in Computerbefehle zu verwandeln und die Mensch-Maschine–Schnittstelle auch für ungeübte Benutzer einfach handhabbar zu machen.

**Mainboard:** Das Mainboard (Motherboard, Hauptplatine) eines Computers ist die zentrale Verbindungsstelle zwischen CPU, Hauptspeicher und den Schnittstellen zu den Peripheriegeräten. In den modernen kompakten Computersystemen wie Workstations und PCs sind CPU, Steckplätze für den Hauptspeicher, der Cache und die wichtigsten Schnittstellen direkt auf dem Mainboard untergebracht. Graphikprozessoren, Videoschnittstellen und ähnliche Erweiterungen können meist mit Hilfe einfacher Steckverbindungen am Motherboard untergebracht werden.

**Schnittstellen:** Die Verbindung zwischen Computern und Peripheriegeräten erfolgt über genormte Schnittstellen (interface). Solche Schnittstellen unterscheiden sich in der Art und Geschwindigkeit der Datenübertragung und in der maximal zulässigen Entfernung zwischen den Endgeräten. Die bekanntesten Schnittstellen sind „die serielle“ Schnittstelle (RS-232, sie dient vor allem der Verbindung von Computern mit langsamen Peripheriegeräten und Modems zur Datenfernübertragung), „die parallele“ Schnittstelle (über sie werden meist Drucker an das System angeschlossen), der USB (universal serial bus) (ein standardisiertes Interface für alle Arten von Erweiterungsgeräten), Firewire (eine Schnittstelle, die hauptsächlich zur Bild- und

Tonübertragung verwendet wird), die Ethernet, FDDI, ATM und andere Netzwerkschnittstellen (dienen der Anbindung von Computern an ein größeres Computernetzwerk wie das Internet), SCSI und IDE Schnittstellen (dienen zur Verbindung zwischen der Hauptplatine und schnellen Peripheriegeräten, wie Harddisks und CD-ROM Laufwerken, wo eine hohe Datenübertragunsrate benötigt wird).

**andere Massenspeicher:** Zusätzlich zu den Harddisks kommen im Computerbereich noch andere Massenspeicher zum Einsatz. Sie dienen dem bequemen Transport kleiner bis mittelgroßer Datenmengen (Disketten, ZIP-Drive), als Backupmedium zur Datensicherung (DAT- und andere Bandlaufwerke) und zur Auslieferung und Archivierung großer Datenmengen (CD-ROM und DVD).

**Ausgabegeräte:** Abgesehen vom Monitor sind Drucker die wichtigsten Geräte zur Datenausgabe. Sie sind in einer großer Variantenvielfalt für alle möglichen Anwendungen erhältlich. Die im Augenblick verbreitetsten Druckertypen sind Farbtintenstrahldrucker zur Farbausgabe und Laserdrucker zur qualitativ hochwertigen Schwarz-Weiß-Ausgabe. Die meisten Drucker können nur wenige hochspezialisierte Druckersprachen interpretieren, mit deren Hilfe die auszugebenden Seiten definiert werden können. Auf beinahe allen Computersystemen besteht jedoch zumindest die Möglichkeit Dateien im PostScript–Format (einer portablen Druckerprogrammiersprache) auszudrucken. Manche Druckertypen können diese Sprache auch direkt interpretieren.

Neben den Druckern sind im CAD Bereich auch noch Plotter verbreitet. Diese arbeiten nicht seiten- oder zeichenorientiert sondern linienorientiert. Sie werden vor allem zur Erstellung von Plänen auf sehr großen Blättern (A0) meist in Tusche auf Transparentpapier verwendet.

**andere Peripheriegeräte:** Zusätzlich zu den oben genannten sind auch noch Scanner (zur Eingabe von Bildern), FAX Schnittstellen (zum Versenden und Empfangen von FAXen), Meß– und Steuergeräte,... im Umfeld von Computern anzutreffen.

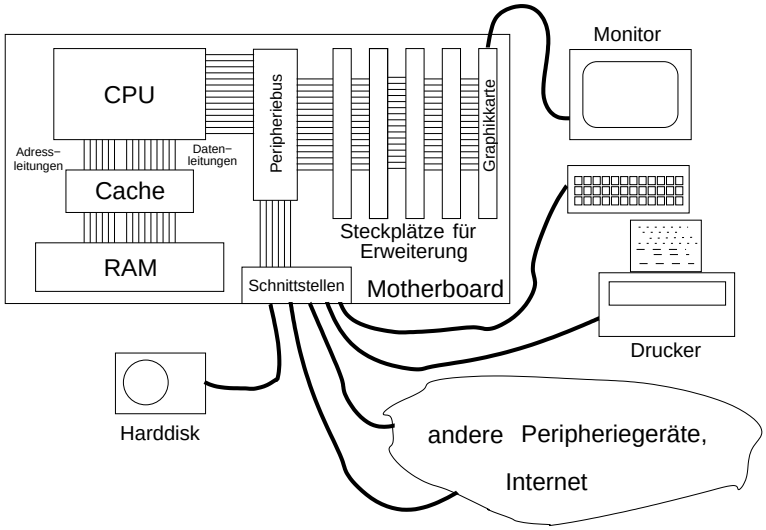



Abbildung 1.1. Aufbau eines Computers (skizzenhaft)

**2.2. Betriebssysteme.** Die erste Schicht Software, mit der jeder Benutzer eines Computersystems konfrontiert wird, ist das Betriebssystem. Es dient dazu, dem Benutzer den Zugriff auf Daten und Peripheriegeräte so einfach wie möglich zu gestalten. Es kümmert sich um Anbindung des Computers an Netzwerke, um Druckerformate, um den Zugriff auf

Dateien, die auf Massenspeichern untergebracht sind. Außerdem dient es dazu, die Daten verschiedener Benutzer voneinander zu trennen und im günstigsten Fall vor fremdem Zugriff zu schützen.

Nach ihren Eigenschaften unterscheidet man verschiedene Betriebssysteme danach, ob auf ihnen gleichzeitig ein oder mehrere Programme ablaufen können (single tasking/ multi tasking) und ob gleichzeitig nur ein oder mehrere Benutzer arbeiten können (single user/multi user). Zusätzlich arbeiten manche Betriebssysteme entweder kommandozeilenorientiert (die einzelnen Programme werden durch Eintippen von Befehlen gestartet) oder graphisch orientiert (die einzelnen Programme werden durch Anklicken von Icons (kleinen graphischen Bildern) ausgeführt).

Die verbreitetsten Betriebssysteme sind

**MSDOS:** Ein kommandozeilenorientiertes single-user single-tasking Betriebssystem für PCs. Diese Minimalvariante eines Betriebssystems ist zwar nicht mehr ganz zeitgemäß aber immer noch verbreitet. Außerdem bietet es keine virtuelle Speicherverwaltung.

**Windows (95/98/ME):** Dies ist zwar im engeren Sinn kein eigenständiges Betriebssystem sondern ein graphischer Aufsatz auf MSDOS, aber nachdem einige wesentliche Erweiterungen eingebaut sind, ist es zweckmäßig, über Windows einen eigenen Abschnitt zu reservieren. Windows ist graphisch orientiert, ein single-user single-tasking (mit einigen Erweiterungen für sehr stark eingeschränktes multi-tasking) Betriebssystem, das auch über eine eingeschränkte virtuelle Speicherverwaltung verfügt.

**Windows NT/2000:** Ein graphisch orientiertes multi-user multi-tasking Betriebssystem, das mit Windows nur das Aussehen gemeinsam hat. Windows NT/2000 ist das erste (wirkliche) Betriebssystem in der Aufzählung, das in der Lage ist, verschiedene Benutzer zu verwalten, deren Dateien voneinander zu trennen und Zugriffe Unberechtigter zu verhindern (oder zumindest zu erschweren). In letzter Zeit nimmt die Verbreitung dieses Betriebssystems stark zu. Es ist auch das erste Betriebssystem in dieser Aufzählung, das auf mehreren Rechnertypen läuft und nicht nur auf PCs beschränkt ist.

**Mac OS:** Das graphisch orientierte single-user multi-tasking Betriebssystem von Apple Macintosh Computern. Ähnlich zu Windows (95/98/ME) ist auch Mac OS auf eine einzelne Plattform beschränkt aber mit dieser Plattform weit verbreitet (vor allem an amerikanischen Bildungsinstituten). Die neueste Version Mac OS X hat im Gegensatz zu den alten Versionen auch multi-user Fähigkeiten. Es ist mit den früheren Versionen nur mehr beschränkt vergleichbar und ähnelt UNIX mit fest integrierter fensterorientierter Oberfläche.

**UNIX:** Ein kommandozeilenorientiertes multi-user multi-tasking Betriebssystem, das vor allem im Workstationbereich und auf Universitäten weite Verbreitung gefunden hat. Im PC Bereich wird es durch die Einführung von LINUX (ein gratis erhältliches UNIX-Derivat) auch immer beliebter. Dieses sehr mächtige Betriebssystem ist zu seinem Nachteil in vielen nicht ganz kompatiblen Abwandlungen (AIX, HP-UX, BSD, LINUX, Solaris, Digital UNIX, ULTRIX,...) im Umlauf. Im Prinzip hat jeder Hardwareanbieter sein eigenes UNIX-Derivat erfunden. Um die Nachteile des kommandozeilenorientierten Zugangs, der durch das Alter des Betriebssystems bedingt ist, zu verringern, gibt es eine auf allen Plattformen lauffähige graphische Erweiterung, das X-Windows. Im Gegensatz zu Windows (95, 98, ME, NT, 2000) und Mac OS kann man damit auch graphische Anwendungen von anderen Rechnersystemen aus starten, um zum Beispiel Performancevorteile großer Computeserver

nutzen zu können. Durch die Entwicklung integrierter Desktop–Oberflächen, die ungeübten Benutzern ein ähnliches „Look–And–Feel“ wie Windows vermitteln (KDE, GNOME, CDE) fallen auch die früher großen Nachteile der komplizierten Benutzeroberfläche nicht mehr ins Gewicht.

**andere:** VMS, R 2000, CMV und andere spezielle Betriebssysteme für Großrechenanlagen (Mainframes) (allesamt multi-user und multi-tasking, kommandozeilenorientiert) verlieren durch ihre Komplexität und ihre geringe Verbreitung trotz vieler Vorteile zunehmend an Bedeutung. Als Mathematiker trifft man sie höchstens noch an, wenn man größere Simulationen oder Berechnungen im Batchbetrieb (man startet das Programm, z.B. über Nacht, und wird später über dessen Ergebnis informiert) auf Großrechenanlagen ausführt.

**2.3. Anwendersoftware.** Mit Hilfe des Betriebssystems kann der Benutzer andere Programmpakete auf einem Computer starten. Wir wollen uns hier nur um die mathematische Anwendersoftware kümmern. Andere Benutzerprogramme wie Textverarbeitung, Tabellenkalkulation und Datenbankanwendungen werden daher im weiteren ausgeklammert, auch wenn manche Tabellenkalkulationsprogramme einiges an grundlegender numerisch mathematischer Funktionalität eingebaut haben.

Die mathematischen Anwendungsprogramme kann man in mehrere große Klassen einteilen:

**2.3.1. Programmpakete zum vorwiegend symbolischen Rechnen.** Software wie Mathematica, Maple, MuPad, Derive, Axiom, MACSYMA,... dient der interaktiven Manipulation von mathematischen Ausdrücken. Sie erlaubt es, analytische Mathematik direkt am Computer zu betreiben, symbolisch zu differenzieren, zu integrieren, und Ausdrücke zu vereinfachen.

Die umfassendsten Programmpakete in dieser Kategorie Mathematica, Maple und MuPAD (ist frei erhältlich und kann in der neuesten Version zum Teil mit den beiden erstgenannten mithalten) sind zwar zuvorderst Computer-Algebrasysteme, haben aber auch Graphik- und einiges an Numerik-Funktionalität.

Im Gegensatz zu anderen Softwarepaketen können die symbolischen Pakete auch rundungsfehlerfrei rechnen. Viele mathematische Aufgaben können mit Hilfe der umfangreichen Manipulationsmöglichkeiten dieser Programme gut gelöst werden, doch ihre mangelnde Arbeitsgeschwindigkeit beschränkt ihr Einsatzgebiet auf kleine Probleme oder rein mathematisch, symbolische Umformungen.

**2.3.2. Programmpakete zum vorwiegend numerischen Rechnen.** Vorwiegend zur Lösung mittelgroßer numerischer Probleme sind Programmpakete, die vorwiegend matrixorientiert arbeiten wie MATLAB, OCTAVE, SCILAB, MLAB, GAUSS, XMATH,... konzipiert.

Die beiden prominentesten Vertreter sind MATLAB und SCILAB, das zweitere ist eine frei erhältliche Implementation, die sich stark an MATLAB orientiert, in der Funktionalität aber ein klein wenig zurückbleibt.

**MATLAB** ist ein interaktives Programmpaket, das vorwiegend auf numerische lineare Algebra aufgebaut ist. Man kann sehr einfach Gleichungen lösen, Daten manipulieren und selbstdefinierte Funktionen hinzufügen. Auch einiges an Graphikfunktionalität ist enthalten, allerdings kann diese nicht mit derjenigen von Programmen wie Mathematica und Maple mithalten. Um aufwendigere Graphiken zu erstellen, lohnt es sich im allgemeinen, Datenfiles zu erzeugen und spezielle Plotpakete wie **gnuplot** oder **XMgr** zu verwenden.

Zusätzlich zur linearen Algebra enthält MATLAB auch noch numerische Verfahren zur Nullstellenbestimmung von Polynomen, zur schnellen Fouriertransformation (FFT), zur numerischen Lösung von Anfangswertproblemen gewöhnlicher Differentialgleichungen,...

MATLAB läßt sich über eine Interpretersprache programmieren, die leider im Vergleich zu anderen Systemen relativ langsam arbeitet. Als Abhilfe kann man aber aus MATLAB Programmen automatisch C Programme erstellen, um die Ausführungsgeschwindigkeit zu erhöhen. Eine Reihe von Toolboxen, die für spezielle Anwendungen wie Simulationen, Signalverarbeitung, Splines, Optimierung, neuronale Netze, Regelungssysteme, Statistik,... entwickelt wurden, sind zusätzlich erhältlich, und eine Schnittstelle zu Maple macht aus MATLAB heraus auch den Bereich der symbolischen Manipulation mathematischer Ausdrücke zugänglich.

**2.3.3. Programme, die speziell zur Lösung bestimmter Probleme angefertigt wurden.** Wenn einmal Probleme gelöst werden müssen, die zu groß für die oben aufgezählten Programmpakete sind, oder für die keine ausreichende Funktionalität vorgesehen ist, dann muß Spezialsoftware entwickelt werden. Nur selten werden alle Teile des neu zu entwickelnden Programmes von den Entwicklern selbst verfaßt.

Im Sinne der Wiederverwertung bereits existierender Algorithmen und Programme werden in den meisten Fällen auch bei neuen Programmen wesentliche Teile mit Hilfe von Standardalgorithmen erledigt. Besonders im Bereich der linearen Algebra (einem zentralen Teil der meisten numerischen Algorithmen) existieren Softwarebibliotheken, in denen die jeweils robustesten und schnellsten Algorithmen zusammengefaßt sind, sodaß sie innerhalb selbstgeschriebener Programme als Unterprogramme aufgerufen werden können.

Im folgenden werden die wichtigsten großen Algorithmenpakete aufgezählt. Interessante Spezialalgorithmen sowie Hinweise auf einzelne Teile der unten aufgeführten Softwarebibliotheken werden später im Rahmen der einzelnen Kapitel gegeben.

**ISML:** (International Mathematical and Statistical Libraries) wird von der Firma Visual Numerics Inc. (Houston, USA) vertrieben. Diese Bibliothek ist eine große Sammlung numerischer und statistischer Algorithmen für die Programmiersprachen C und FORTRAN. Im Gegensatz zu früheren Zeiten, in denen die ISML nur für IBM Großrechner erhältlich war, werden heute Versionen für alle bekannten Rechnersysteme verkauft.

**NAG:** Die zweite umfassende Softwarebibliothek von internationaler Bedeutung wird von der Numerical Algorithms Group (Oxford, GB) vertrieben. Auch diese Bibliothek gibt es für die Programmiersprachen C und FORTRAN, doch der Umfang der FORTRAN Bibliothek ist noch größer. Es ist jedoch auf allen modernen Rechnersystemen ohne Probleme möglich, Programmiersprachen bei der Erstellung neuer Programme zu mischen.

**andere:** Es gibt noch eine Reihe anderer Softwarebibliotheken, die jedoch nicht so große Verbreitung haben. PORT (Lucent Technologies (früher AT&T Bell Labs), USA), CMLIB (National Institute of Standards, USA), Harwell Subroutine Library (Harwell Laboratory, GB), SLATEC Common Mathematical Library (Computing Division, Los Alamos Scientific Laboratory, USA), BOEING Math. Software Library (Boeing Computer Services Company, USA).

Zusätzlich zu den Softwarebibliotheken gibt es auch noch eine Reihe wichtiger spezialisierter numerischer Softwarepakete. Diese Pakete werden im Gegensatz zu den Bibliotheken nicht regelmäßig gewartet, sind aber oftmals von sehr hoher Qualität. Die wichtigsten sind

**BLAS:** Basic Linear Algebra Subroutines. Dieses Paket enthält in drei Teilen Vektor-Vektor Routinen (inneres Produkt, Summe,...) (BLAS 1), Matrix-Vektor Routinen (BLAS 2) und Matrix-Matrix Routinen (Summe, Produkt,...) (BLAS 3). Das Paket ist auf allen wichtigen Rechnerplattformen in speziell auf diese Plattform hin optimierten Versionen erhältlich.

**LAPACK:** Linear Algebra Pack ist ein Paket von Unterprogrammen zur Lösung vielfältiger Probleme der numerischen linearen Algebra (Gleichungssysteme, LR Zerlegung, Choleskyfaktorisierung, Eigenwertberechnung, Ausgleichsprobleme,...). LAPACK baut auf die BLAS Routinen auf und ist auch für alle wichtigen Computersysteme erhältlich.

**QUADPACK:** Zur schnellen Integration und Integraltransformation eindimensionaler Funktionen.

**andere:** ITPACK (iterative Lösung riesiger linearer Gleichungssysteme mit dünn besetzten Matrizen), MINPACK (nichtlineare Gleichungssysteme und Optimierungsaufgaben), ODEPACK (Anfangswertprobleme gewöhnlicher Differentialgleichungen), FFTPACK (schnelle Fouriertransformation), ODRPACK (nichtlineare Ausgleichsprobleme), ELLPACK (elliptische partielle Differentialgleichungen in zwei bis drei Dimensionen),...

**2.3.4. andere Programmsysteme.** Es gibt noch eine Reihe von Mischformen: Programmsysteme, die für eine große Klasse ähnlicher Probleme entwickelt worden sind, aber nicht so universell einsetzbar sind wie Mathematica, Maple oder MATLAB, die aber unter Verwendung von Softwarepaketen erstellt worden sind und vielleicht aus anderen Programmen heraus aufgerufen werden können oder eine eigene interaktive Benutzerschnittstelle besitzen. GLOPT, das von der CMA der Uni Wien für globale Optimierungsaufgaben entwickelt wird, ist ein Beispiel für ein solches Mischsystem. Softwarehinweise auf solche Systeme werden einzeln in den passenden Kapiteln gegeben.

## 3. Zahlendarstellung im Computer

Will man mathematische Software schreiben oder verwenden, so ist es notwendig, sich mit den Eigenheiten der Zahlendarstellung am Computer vertraut zu machen. Entgegen der üblichen Dezimaldarstellung arbeiten Computer im allgemeinen in der Binärdarstellung. So werden z.B. ganze Zahlen (Integerzahlen) folgendermaßen dargestellt:

$$z = \pm(a_0 a_1 \dots a_n)_2 := \pm a_0 + a_1 2 + a_2 2^2 + \dots + a_n 2^n, \quad a_i \in \{0, 1\}.$$

Nachdem für menschliche Benutzer die Binärarithmetik nicht nur gewöhnungsbedürftig sondern auch ausgesprochen unübersichtlich ist ($(795)_{10} = (1100011011)_2$), wird in der Eingabe–Ausgabeschnittstelle (I/O) zwischen Binär- und Dezimalsystem umgewandelt. Alternativ werden auch das Oktal- und das Hexadezimalsystem verwendet, die sich sehr einfach in das Binärsystem umwandeln lassen und eine ähnlich übersichtliche Darstellung wie das Dezimalsystem zulassen. Beim Oktalsystem werden Dreiergruppen von Binärzahlen zusammengefaßt ($(795)_{10} = (1433)_8 = (001\,100\,011\,011)_2$), während beim Hexadezimalsystem Vierergruppen gebildet werden. Dabei werden die Zahlen 10–15 durch die Buchstaben A–F repräsentiert ($(795)_{10} = (31\mathrm{B})_{16} = (0011\,0001\,1011)_2$).

Diese Beobachtung gewinnt erst im Zusammenhang mit den folgenden Fakten Bedeutung:

- Ganze Zahlen und reelle Zahlen werden unterschiedlich repräsentiert.
- Das Hin– und Herwandeln zwischen den Repräsentationen ist nicht in allen Fällen eindeutig möglich.
- Im Computer ist nur eine endliche Teilmenge der natürlichen bzw. der reellen Zahlen darstellbar.

Reelle Zahlen werden im Computer ebenfalls im Binärsystem als sogenannte Gleitkommazahlen *(floating point numbers)* repräsentiert. Die Darstellung einer solchen Gleitkommazahl im Computer sieht folgendermaßen aus:

$$z = \pm(d_1 2^{-1} + d_2 2^{-2} + \dots + d_n 2^{-n}) 2^e.$$

Die Zahl $(.d_1d_2\dots d_n)_2$ heißt die *Mantisse (mantissa, fraction)* von $z$ und $e$ heißt der *Exponent (exponent)*. Die Zahlen $d_1\dots d_n$ heißen auch die *Stellen* von $z$. Solch eine Gleitkommazahl heißt *normalisiert* (normalized), wenn $z = 0$ oder $d_1 \neq 0$. In modernen Computersystemen werden Gleitkommazahlen der IEEE-Norm folgend repräsentiert. Die IEEE-Norm kennt zwei verschiedene Arten der Darstellung, die sich im Wesentlichen in der Länge der Mantisse ($n$) und der maximalen Größe des Exponenten unterscheiden, die Zahlen *einfacher Genauigkeit (single precision)* und die Zahlen *doppelter Genauigkeit (double precision)*:

| Zahlentyp | Mantissenlänge | Exponentenbereich |
|---|---|---|
| single precision | 24 | [-126,127] |
| double precision | 53 | [-1022,1023] |

Zusätzlich können nach der IEEE-Norm noch 0 und $\pm\infty$ dargestellt werden. Manche besonders kleine Zahlen (Exponent $< -126$) können als nicht-normalisierte (denormalized) Zahlen dargestellt werden, indem auf die Forderung $d_1 \neq 0$ verzichtet wird. Nach dem IEEE–Standard sind solche Zahlen zugelassen, doch implementieren nicht alle Compiler diesen Zusatz korrekt. Weiters beachte man, daß nicht-normalisierte Zahlen eine niedrigere Genauigkeit aufweisen als normalisierte Zahlen. Alle Zahlen, die nicht diesem Schema entsprechen sind ungültige Gleitkommazahlen (NaN, not a number) und führen zu Laufzeitfehlern, wenn sie bei Rechenoperationen verwendet werden (floating point exception).

| Zahlentyp | kleinste positive Zahl | größte Zahl |
|---|---|---|
| single precision | $2^{-149}$ | $2^{128} - 2^{104}$ |
| double precision | $2^{-1074}$ | $2^{1024} - 2^{971}$ |

Diese Erklärungen sind nicht ganz exakt aber ein gemeinsamer Nenner der Implementationen der IEEE-Arithmetik auf den Geräten der führenden Hardwarehersteller. Eine genauere Darstellung kann z.B. in [**Überhuber 1995a**, 4.4.3] gefunden werden; den Standard kann man nachlesen in [**IEEE 754–1985**], oder unter

`http://www.psc.edu/general/software/packages/ieee/ieee.html`.

**3.1. Runden.** Alle reellen Zahlen $z$, die nicht exakt darstellbar sind, können auf verschiedene Art in eine Gleitkommazahl $\mathrm{rd}(z)$übersetzt werden. Es gibt dabei vier in der IEEE-Norm definierte Möglichkeiten das zu tun. Dabei müssen wir aber voraussetzen, daß $z$ im Größenbereich der Gleitkommazahlen bleibt (im Fall der einfachen Genauigkeit ist das die Menge $[-2^{128} + 2^{104}, -2^{-149}] \cup \{0\} \cup [2^{-149}, 2^{128} - 2^{104}]$).

**Optimales Runden:** Die üblichste Methode, nicht darstellbare reelle Zahlen $z$ in Gleitkommazahlen $\mathrm{rd}(z)$ zu verwandeln ist das *Runden (rounding)*. In diesem Fall wird als $\mathrm{rd}(z)$ diejenige Gleitkommazahl gewählt, die $z$ am nächsten liegt. Wenn zwei Gleitkommazahlen gleich weit von $z$ entfernt liegen, wird diejenige gewählt, bei der $d_n = 0$ gilt.

**Abschneiden:** Eine weitere Methode, die auch schon in Computersystemen vor Einführung der IEEE-Norm verwendet worden ist, ist das *Abschneiden (chopping)*. Um $z$ zu repräsentieren, wird $\mathrm{rd}(z)$ als diejenige Gleitkommazahl gewählt, die zwischen 0 und $z$ am nächsten bei $z$ liegt, d.h. in der Binärdarstellung werden einfach alle Stellen weggelassen, die nicht gespeichert werden können.

**Aufrunden:** Beim *Aufrunden (rounding up, rounding towards infinity)* wird die kleinste Gleitkommazahl, die größer als $z$ ist als $\mathrm{rd}(z)$ gewählt.

**Abrunden:** $\mathrm{rd}(z)$ ist beim *Abrunden (rounding down, rounding towards minus infinity)* die größte Gleitkommazahl, die kleiner als $z$ ist.

Man beachte, daß eine Zahl, die im Dezimalsystem als abbrechende Dezimalzahl darstellbar ist, im allgemeinen keine abbrechende Binärdarstellung besitzt, und daher im Computer

nicht exakt repräsentierbar ist. Z.B.

$$(0.1)_{10} = (0.0\,0011\,0011\ldots)_2 = (.1100\,1100\,1100\,1100\ldots)_2 2^{-3}.$$

Auf jedem Computersystem, das dem IEEE-Standard bei der Gleitkommaarithmetik entspricht, kann man sich bei jeder Repräsentation einer reellen Zahl entscheiden, welche der vier Rundungsmethoden gewählt werden soll.

In jedem der vier Fälle hängt der Unterschied zwischen der reellen Zahl $z$ und der Gleitkommazahl $\mathrm{rd}(z)$, der *Rundungsfehler (roundoff error)*, von der Größe von $z$ ab und wird daher am besten relativ zu $z$ gemessen:

$$\mathrm{rd}(z) = z(1+\delta), \tag{1}$$

wobei $\delta = \delta(z)$ eine Zahl ist, die von $z$ abhängt. Im Fall des Rundens ist $|\delta| < 2^{-n}$. In den anderen Fällen ist $|\delta| < 2^{-n+1}$, doch ist das Vorzeichen von $\delta$ a priori bekannt. Der höchstmögliche Wert von $\delta$ wird auch als *Maschinengenauigkeit (unit roundoff)* (im folgenden mit eps) bezeichnet.

Liegt eine umzuwandelnde Zahl $z$ nicht im Größenbereich der Gleitkommazahlen, dann gibt es zwei Möglichkeiten, die beide zu einem Laufzeitfehler des Programmes (floating point exception) führen:

- Ist das Ergebnis größer (kleiner) als die größte (kleinste) im Computer repräsentierbare Zahl, so passiert ein *Überlauf (overflow)*.
- Wenn das Ergebnis eine Zahl ist, deren Betrag kleiner ist als die betragsmäßig kleinste darstellbare Zahl ungleich Null, so passiert ein *Unterlauf (underflow)*.

Obwohl Unterläufe eine Exception produzieren sollten, wird abhängig vom Rechnersystem das Ergebnis manchmal auch als Null repräsentiert, was nicht dem Standard entspricht und mitunter zu seltsamen Effekten und zu starker Fehlerverstärkung (siehe 5) führen kann.

Grundsätzlich kann durch vorsichtige Skalierung der Daten das Überlauf-/Unterlaufproblem vermieden werden, und wir werden im folgenden nicht mehr darauf eingehen und annehmen, daß alle Daten und Ergebnisse zu Gleitkommazahlen gerundet werden können.

**3.2. Grundrechenarten, Körperaxiome.** Wenn eine arithmetische Operation $\circ$ auf zwei Gleitkommazahlen angewendet wird, ist das Resultat im allgemeinen keine im Computer repräsentierbare Gleitkommazahl mehr. In diesem Fall wird das Ergebnis wieder entsprechend einer der vier Rundungsmethoden in eine darstellbare Zahl verwandelt. Die so entstehenden Ersatzoperationen $\circ_*$, die Gleitkommaarithmetik, haben andere Eigenschaften als die arithmetischen Grundrechenarten auf reellen Zahlen.

**Beispiel 3.2.1.** *In diesen Beispielen wird aus Gründen der Übersichtlichkeit auf die Umwandlung in die Binärdarstellung verzichtet. Stattdessen wird die analoge Darstellung im Dezimalsystem ($z = (d_1 \ldots d_n)_{10} \cdot 10^e$ mit $n = 5$) mit Rundung auf 5 signifikante Stellen verwendet, um die prinzipiellen Ideen aufzuzeigen.*

$$\begin{aligned} 12.457 +_* 1.1465 &= 13.604 \\ 1.1297 *_* 1.6574 &= 1.8724 \\ 1378.6 -_* 0.046732 &= 1378.6 \\ 3453.3/_* 0.018394 &= 187740 \end{aligned}$$

Eine wichtige Auswirkung der Rundung in der Definition der Gleitkommaoperationen ist die Verletzung der Körperaxiome. Beinahe keine der Standardrechenregeln (ausgenommen die Kommutativgesetze und die *Existenz* neutraler Elemente) behält Gültigkeit. Besonderes

Augenmerk sollte dabei auf die Tatsache gelenkt werden, daß das Assoziativgesetz verletzt ist.

**Beispiel 3.2.2.** *Wie im obigen Beispiel wird wieder auf 5 signifikante Stellen gerundet.*

$$(0.31564 +_* 0.38436) +_* 17845 = 17846$$
$$0.31564 +_* (0.38436 +_* 17845) = 17845$$

$$0.31564 + 0.38436 + 17845 = 17845.7$$

Wenn man das obige Beispiel genauer betrachtet, erkennt man immerhin, daß man den Rundungsfehler minimieren kann, wenn man die einzelnen Terme vom kleinsten beginnend summiert. Eine genauere Analyse dieses Effektes wird im Abschnitt 5 gegeben. Dort wird auch ein Hinweis auf Summationsverfahren gegeben, die so konstruiert sind, daß sie den relativen Rundungsfehler minimal halten.

## 4. Auswertung von Ausdrücken

Ein interessantes Problem der Numerik, das direkt mit Rundungsfehlern und deren Fortpflanzung zu tun hat, und dessen Auswirkungen gerne unterschätzt oder übersehen werden, ist die Frage, welchen von mehreren mathematisch äquivalenten Ausdrücken man wählen soll, um das Ergebnis zu berechnen. Wir werden an mehreren Beispielen sehen, daß das große Unterschiede machen kann. Das führt auch dazu, daß die Wichtigkeit analytischer Rechnungen im numerischen Bereich gerne überschätzt wird. Manchmal ist das Ergebnis genauer, wenn man z.B. schwierige bestimmte Integrale numerisch löst statt sie zuerst symbolisch zu integrieren und dann die entstehenden komplizierten Funktionen numerisch an den Grenzen auszuwerten.

**4.1. Analytisch vs. Numerisch.** Es lohnt sich, einmal einen Blick auf die Vorgänge bei der numerischen Auswertung eines komplizierten analytischen Ausdrucks zu werfen. Betrachten wir einmal folgendes Beispiel:

$$\begin{aligned} f(x) = &- 0.271668\cos(x) + 0.0804941\cos(3x) - 0.0381288\cos(5x) + \\ &+ 0.0190644\cos(7x) - 0.00917915\cos(9x) + 0.00409648\cos(11x) - \\ &- 0.00165777\cos(13x) + 0.00059864\cos(15x) - 0.000190156\cos(17x) + \\ &+ 5.23507\cdot 10^{-5}\cos(19x) - 1.22798\cdot 10^{-5}\cos(21x) + 2.40257\cdot 10^{-6}\cos(23x) - \\ &- 3.81097\cdot 10^{-7}\cos(25x) + 4.7049\cdot 10^{-8}\cos(27x) - 4.23912\cdot 10^{-9}\cos(29x) + \\ &+ 2.47852\cdot 10^{-10}\cos(31x) - 7.05547\cdot 10^{-12}\cos(33x) \end{aligned}$$

Dieser Ausdruck ist entstanden aus symbolischer Integration von $\sin^{33}(x)$ und Berechnung der Koeffizienten auf 6 signifikante Dezimalstellen. Was ist bei der Auswertung dieses Ausdrucks zu tun? Zuerst ist die Funktion $\cos(x)$ an einigen verschiedenen Stellen auszuwerten. Dies kann man zum Beispiel mit Hilfe der bekannten Reihendarstellung von $\cos(x)$ durchführen. (Man beachte, daß die häufig verwendeten transzendenten Funktionen selbst schon numerische Schwierigkeiten aufwerfen!) Dabei hat man das Problem bis zum wievielten Reihenglied man summieren soll. Mit Hilfe von Fehleranalyse und Ausnützung der Periodizität des Cosinus kann man dieses Problem so lösen, daß der Fehler in der Größenordnung der Rechengenauigkeit bleibt. Danach hat man jedoch 32 Multiplikationen und 31 Additionen auszuführen, bei denen die Summanden verschiedene Vorzeichen haben. Bei solchen Rechenschritten ist der Fehler nur sehr schwer unter Kontrolle zu halten. Eine direkte numerische Integration von $\sin^{33}(x)$ ist jedoch mit den geeigneten Methoden mit vorgegebener Genauigkeit ohne Probleme durchführbar und benötigt etwa vergleichbar viele Rechenschritte.

Zusätzlich deuten auch rein logische Überlegungen darauf hin, daß analytische Rechnungen nicht immer Vorteile bringen, wenn man dabei auf transzendente Funktionen stößt und man vielleicht alternativ das Problem direkt z.B. durch eine Reihenentwicklung approximieren kann. Denn nur für den Menschen sind analytische Ausdrücke von Vorteil, wenn er sich ein Bild von Funktionen oder Werten machen möchte. Für den Computer sind transzendente Funktionen auch nur Reihendarstellungen, Kettenbruchentwicklungen oder Tabellenwerte, die interpoliert werden müssen, um zu Gleitkommazahlen zu führen.

Abgesehen davon gibt es viele Ausdrücke, die sich gar nicht analytisch berechnen lassen, wie z.B.

$$\int_1^5 \frac{\sin x}{x}.$$

**Beispiel 4.1.1.** *Zum Abschluß noch ein Beispiel, das eine weitverbreitete Meinung widerlegt: „Wenn man alle Rechnungen auf dem Computer mit allen möglichen vorhandenen verschiedenen Rechengenauigkeiten durchführt (single precision, double precision,...), und wenn dann die Ergebnisse in den ersten n Dezimalstellen übereinstimmen, dann sind diese übereinstimmenden Dezimalstellen auch korrekt.“ Das Beispiel stammt von Rump (*[**Rump 1988**]*) und kann auch in* [**Hansen 1992**, p. 3] *nachgelesen werden.*

*Die Rechnungen wurden auf einem S/370 Computer durchgeführt. Alle Eingangszahlen sind auf dem Computer exakt repräsentierbar; daher entstehen alle Rechenfehler aus Rundungen. Das Problem ist, den Wert der Funktion*

$$f(x,y) = \tfrac{1335}{4}y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + \tfrac{11}{2}y^8 + \frac{x}{2y}$$

*an* $(x,y) = (77617, 33096)$ *zu bestimmen. Die Rechenergebnisse mit single, double und extended precision (ungefähr äquivalent zu 6, 17 und 34-stelliger Dezimalarithmetik) waren*

$$\begin{array}{ll} \textit{single precision:} & f = 1.172603\ldots \\ \textit{double precision:} & f = 1.1726039400531\ldots \\ \textit{extended precision:} & f = 1.172603940053178\ldots \end{array}$$

*Alle Ergebnisse stimmen in den ersten sieben Dezimalstellen überein. Die beiden genauer berechneten Ergebnisse unterscheiden sich sogar in den ersten 14 Dezimalstellen nicht. Trotzdem haben sie mit dem wirklichen Ergebnis (kann zum Beispiel in Mathematica mit rundungsfehlerfreier Arithmetik bestimmt werden)* $f = -0.8273960599468213\ldots$ *nicht einmal das Vorzeichen gemein.*

**4.2. Konvergenz, Konvergenzbeschleunigung.** Die moderne Analysis basiert auf dem Begriff der Konvergenz. Die meisten grundlegenden Konzepte wie Ableitung, Integral, Stetigkeit,... werden über konvergierende Folgen definiert. Wie oben erwähnt, werden auch die meisten elementaren Funktionen entweder über Reihen definiert (manchmal werden sie zwar nicht so definiert, aber die Reihenentwicklungen werden in der Regel zusätzlich bewiesen).

Nachdem alle Resultate bei numerischen Berechnungen nur bis zu einer bestimmten Genauigkeit benötigt werden, ist es nur natürlich, diese Folgendefinitionen auszunutzen und anstelle des Grenzwertes ein Folgenglied zu berechnen, das den Limes gut genug approximiert. Wie findet man nun so ein Folgenglied? Die etwas saloppe Definition aus der Analysis, die das Problem aber am besten wiederspiegelt, ist folgendermaßen:

> Die Folge $\{a_n\}$ konvergiert gegen den Wert $a$, wenn für jede positive reelle Zahl $\varepsilon$ gilt, daß für alle $n$, die „groß genug“ sind (d.h. es gibt eine Zahl $N(\varepsilon)$, sodaß für alle $n > N(\varepsilon)$), $|a_n - a| < \varepsilon$ ist.

Vom numerischen Standpunkt ist diese Definition mehr als unbefriedigend. Zum ersten ist es oft unmöglich, zu sagen, wann $n$ groß genug ist, d.h. die Funktion $N(\varepsilon)$ ist meist unkontrollierbar. Manchmal ist „groß genug“ auch einfach viel zu groß.

**Beispiel 4.2.1.** *Der genaue Wert der Zahl $\pi/4$ läßt sich durch die Reihe*

$$\sum_{i=0}^{\infty} \frac{(-1)^i}{2i+1} = 1 - \sum_{j=1}^{\infty} \frac{2}{16j^2-1}$$

*berechnen. Definiert man*

$$a_n := 1 - \sum_{j=0}^{n} \frac{2}{16j^2-1},$$

*so erhält man eine Folge, die monoton fallend gegen $\pi/4$ konvergiert. Aus der ersten Darstellung durch die alternierende Reihe erhält man für alle $n$ die Abschätzung*

$$0 \le a_n - \tfrac{\pi}{4} \le \tfrac{1}{4n+3}.$$

*Um also $\pi/4$ bis auf einen Fehler von $10^{-6}$ zu berechnen, muß man $n \approx 250000$ wählen, was für eine vernünftige Berechnung viel zu hoch ist.*

Um mit solchen Problemen zurechtzukommen, ist etwas Notation aus der Analysis nötig, die hier kurz wiederholt werden soll. Sie dient dazu, die Konvergenzgeschwindigkeit zweier Folgen zu vergleichen:
Seien $\{a_n\}$ und $\{b_n\}$ zwei konvergente Folgen. Man sagt die Folge $\{a_n\}$ konvergiert mit Ordnung $\{b_n\}$ ($\{a_n\}$ ist groß O von $\{b_n\}$), also

$$a_n = O(b_n), \quad \text{wenn } |a_n| \le K|b_n|$$

für eine Konstante $K$ und alle genügend großen $n$, und man sagt, daß $\{a_n\}$ schneller als $\{b_n\}$ konvergiert ($\{a_n\}$ ist klein o von $\{b_n\}$), also

$$a_n = o(b_n), \quad \text{wenn } \lim_{n\to\infty} \tfrac{a_n}{b_n} = 0.$$

**Beispiel 4.2.2.**

$$\left.\begin{array}{c} 500/n \\ 3/n - 40/n^2 + e^{-2n} \\ 1/n^2 \end{array}\right\} = O(1/n)$$

$$\left.\begin{array}{c} 1/(n\log n) \\ e^{-2n} \\ 1/n^2 \end{array}\right\} = o(1/n)$$

Diese Notation wird überwiegend auf der rechten Seite von Gleichungen verwendet, um Fehlerterme darzustellen ohne sich um die genaue Größe oder multiplizierende Konstanten kümmern zu müssen.

Die Abschätzung für die Reihe, die $\pi/4$ darstellt, kann man mit Hilfe dieser Notation auch so schreiben:

$$1 - \sum_{j=1}^{\infty} \frac{2}{16j^2-1} = \tfrac{\pi}{4} + O(\tfrac{1}{n}).$$

Die Abschätzung mit $o$ statt $O$ gilt jedoch bereits nicht mehr. Man erkennt also, daß für numerische Berechnungen eine Konvergenzrate von $1/n$ viel zu gering ist.

Um Abschätzungen dieser Art allerdings für das Problem zu verwenden, bei welchem Folgenglied man abbrechen soll, muß man versuchen, sie in die folgende Form zu bringen:

$$a_n = a + b_n + o(b_n) = a + b_n(1+\varepsilon_n),$$

mit einer Nullfolge $\{\varepsilon_n\}$. Obwohl man normalerweise nicht *beweisen* kann, daß $n$ groß genug ist, kann man die *Hypothese*, daß $n$ groß genug ist, *testen*, indem man $|a_{k+1}-a_k|$ mit $|b_{k+1}-b_k|$ vergleicht. Wenn $\frac{|a_{k+1}-a_k|}{|b_{k+1}-b_k|} \approx 1$ für einige $k$ in der Umgebung von $n$ gilt, dann akzeptiert man die Hypothese, daß $n$ groß genug ist, daß $a_n \approx a+b_n$ gilt und daher $b_n$ eine gute Abschätzung für den Fehler $|a_n - a|$ ist.

Die Konvergenzgeschwindigkeit der geometrischen Reihe $\sum 1/q^k$ ist ausreichend für numerische Berechnungen. Man erhält die Abschätzung

$$\sum_{i=0}^{n} \tfrac{1}{q^i} = \tfrac{1}{1-q} - \tfrac{q^{n+1}}{1-q} = \tfrac{1}{1-q} + O(q^n) = \tfrac{1}{1-q} + o(r^n),$$

für jedes $q < r < 1$. Wenn eine Folge so rasch konvergiert wie die geometrische Reihe, so sagt man sie konvergiert exponentiell (geometrisch). Für viele numerische Anwendungen ist dies die Mindestanforderung an die Konvergenzordnung.

Die Folgen $\{a_n\}$ werden zwar oft durch Reihen erzeugt, häufiger jedoch werden sie durch Iterationsverfahren generiert. Wenn $\Phi$ eine reellwertige Abbildung ist, dann kann man durch

$$a_{n+1} = \Phi(a_n),$$

rekursiv eine Folge mit Startwert $a_0$ definieren. Wenn $\{a_n\}$ konvergiert, so muß der Grenzwert $a$ ein Fixpunkt der Funktion $\Phi$ sein. Man sagt, daß das durch $\Phi$ definierte Iterationsverfahren Konvergenzordnung $p$ hat, falls für alle Startwerte $a_0$ in einer genügend kleinen Umgebung $U(a)$ gilt

$$|a_{n+1} - a| \le C|a_n - a|^p, \quad \text{für alle } n \ge o,$$

wobei $C < 1$ für $p = 1$ vorausgesetzt sei. Eine leichte Abschätzung zeigt, daß ein Iterationsverfahren erster Ordnung eine Folge erzeugt, die exponentiell konvergiert. Meist versucht man jedoch, Iterationsverfahren mit höherer als *linearer Konvergenz* ($p = 1$) zu erzeugen. *Superlineare Konvergenz* ($1 < p < 2$), *quadratische Konvergenz* ($p = 2$) und *kubische Konvergenz* ($p = 3$) sind übliche Bezeichnungen. Z.B. ist das Newton–Verfahren zur Nullstellenbestimmung für nicht-lineare Funktionen ein Iterationsverfahren mit quadratischer Konvergenz (siehe 7).

Die Notationen $O$ und $o$ lassen sich auch auf reellwertige Funktionen verallgemeinern: Sei $\lim_{x\to x_0} f(x) = f_0$. Dann sagen wir die Konvergenz ist $O(g(h))$, wenn

$$\frac{f(x) - f_0}{|g(x)|} \le K$$

für eine Konstante $K$ und für alle $x$ mit $|x - x_0|$ klein genug. Gilt das für alle $K > 0$, ist also

$$\lim_{x\to x_0} \frac{f(x) - f_0}{g(x)} = 0,$$

so schreiben wir $f(x) = o(g(x))$ bei $x_0$.

**Beispiel 4.2.3.** *Bei* 0 *gilt*

$$\frac{\sin x}{x} = 1 + \sum_{i=1}^{\infty} \tfrac{1}{(2i+1)!} x^{2i} = 1 + O(x^2) =$$
$$= 1 - \tfrac{x^2}{6} + o(x^3).$$

Es gibt auch einige analytische Tricks, die Konvergenz von durch Reihen definierte Folgen zu beschleunigen. Diese *Konvergenzbeschleunigungsverfahren* sind meist nicht von einem Beispiel auf ein anderes übertragbar und werden daher auch in dieser Vorlesung nicht behandelt. Einzig für geometrisch konvergierende Folgen und Folgen, die asymptotisch geometrische Konvergenz erreichen, gibt es ein Standardverfahren — das $\Delta^2$–Verfahren von Aitken.

Iterationsverfahren, die lineare Konvergenz aufweisen, können mit dem $\Delta^2$–Verfahren in Iterationsverfahren mit quadratischer Konvergenz verwandelt werden. Dieses Verfahren und das damit verwandte Verfahren von Steffensen findet man z.B. in [**Stoer 1994a**, 5.10].

**Beispiel 4.2.4.** *Zum Abschluß noch ein Kuriosum, das aber auch mit Konvergenzraten zu tun hat: Die „berühmteste Reihe der Analysis“, die harmonische Reihe*

$$\sum_{i=0}^{\infty} \frac{1}{i},$$

*divergiert gegen $+\infty$. Das ist jedem bekannt. Die Divergenzrate ist jedoch so langsam, daß man dieses Verhalten auf keinem Computer auch nur annähernd bestätigen kann. Nach einiger Zeit sind die Glieder der Reihe so klein, daß sie kleiner sind als der relative Rundungsfehler, daß also gilt $s_{n+1} = s_n +_* a_{n+1} = s_n$. Dann hat die Summation im Computer ihren Grenzwert erreicht (ohne einen Überlauf zu verursachen). Dieser Grenzwert ist meist „weit von Unendlich entfernt“ — selbst bei sehr hoher Rechengenauigkeit wird 50 kaum überschritten.*

## 5. Fehler

Bei jeder Berechnung für eine Anwendung entstehen *Fehler*. Eine der wichtigsten Aufgaben der numerischen Mathematik ist es, die Größenordnung dieser Fehler abzuschätzen und Algorithmen zu finden, die die Fehler möglichst gering halten.

Von der Beobachtung der Wirklichkeit, über die Bildung des Modells bis zur Berechnung der numerischen Resultate entstehen verschieden Arten von Fehlern, die die Genauigkeit des Ergebnisses einschränken:

**Modellierungsfehler:** Diese Fehler (sie werden nicht in der gesamten numerischen Literatur erwähnt) entstehen im Schritt vor der tatsächlichen Berechnung während der Modellbildung. Schon während des Abstraktionsschrittes entstehen Fehler, deren Einfluß auf das Modell nur schwer abgeschätzt werden kann. Nach erfolgter Abstraktion und Übersetzung in ein mathematisches Modell kann es zusätzlich sein, daß das Modell immer noch zu komplex für die numerischen Methoden ist. Dann wird es oftmals noch vereinfacht (z.B. durch Linearisierung). Dabei „verfälscht“ man das Modell und handelt sich Fehler ein, die durch eine Änderung in der Modellierung entstehen, die auch den Gültigkeitsbereich des Modells verkleinern.

**Datenfehler:** Fehler in den Eingabedaten lassen sich nicht vermeiden. Oft entstehen die Eingabedaten zu Berechnungen aus Messungen, die nur von beschränkter Genauigkeit sind. Manchmal müssen sogar Schätzwerte als Eingangsdaten herhalten. Wichtig ist es hier, vom Anwender zu erfahren, welche Größenordnung der Fehler zu erwarten ist.

**Rundungsfehler:** Dieser Fehlertyp entsteht, wenn man (was in beinahe allen numerischen Berechnungen geschieht) nur mit Zahlen einer endlichen aber festen Zahl signifikanter Stellen rechnet. Die Fortpflanzung solcher Fehler während einer Rechnung bestimmt zu einem guten Teil die Güte eines numerischen Algorithmus. Theorie zu deren Abschätzung wird in Abschnitt 5.1 entwickelt.

**Approximationsfehler:** Viele Methoden zur Berechnung von Größen liefern selbst bei rundungsfehlerfreier Rechnung nicht die eigentlich gesuchte Lösung sondern nur

eine Approximation der Lösung. Fehler, die aufgrund solcher Rechenmethoden entstehen, heißen Approximationsfehler. Soll zum Beispiel $e^x$ mit Hilfe der Reihenentwicklung

$$e^x = \sum_{k=0}^{\infty} \frac{1}{k!} x^k$$

berechnet werden, so muß die Summation nach endlich vielen Gliedern abgebrochen werden (truncation error, Abbruchfehler). Auch bei der heute weit verbreiteten Darstellung durch Kettenbrüche ([**Hart 1968**]) treten Abbruchfehler auf. Unabhängig davon, wie spät man abbricht, ist ein Fehler auch bei rundungsfehlerfreier Rechnung unvermeidlich.

Häufig werden Probleme auch durch Diskretisierung approximiert: Integrale werden durch endliche Summen angenähert, Differentialquotienten durch Differenzenquotienten, mehrdimensionale Gebiete werden durch eine Vereinigung von endlich vielen Rechtecken approximiert,... Fehler, die in diesem Zusammenhang auftreten, werden auch als Diskretisierungsfehler (error of discretization) bezeichnet.

In vielen modernen Gleitkommaprozessoren werden analytische Funktionen tabelliert, und Zwischenwerte werden durch Interpolationsmethoden berechnet. Fehler, die bei diesen Verfahren auftreten, heißen auch Interpolationsfehler.

Beispiele für die verschiedenen Fehlerarten kann man zu Beginn des Kapitels 2 in der Fallstudie über das Pendel finden.

### 5.1. Fehlerfortpflanzung: Chaos.

**Definition 5.1.1.** *Sei $z$ eine Zahl und $\tilde{z}$ eine für $z$ berechnete Näherung. Wenn man den Unterschied von $z$ und $\tilde{z}$ betrachtet, hat man grundsätzlich zwei verschiedene Möglichkeiten: Man berechnet den* absoluten Fehler $\Delta z = \tilde{z} - z$*, oder man bestimmt (für $z \neq 0$) den* relativen Fehler $\varepsilon_z = (\tilde{z} - z)/z$.

In numerischen Berechnungen wird meist der relative Fehler abgeschätzt, da dieser in der Praxis eine weitaus größere Bedeutung hat und auch bei den Implementationen der Gleitkommaarithmetik der einzig abschätzbare Fehler für die arithmetischen Grundoperationen ist.

Wie bereits der Ausführung oben zu entnehmen war, ist die Abschätzung der Rundungsfehler und Eingangsdatenfehler und deren Fortpflanzung für die Konstruktion eines numerischen Algorithmus von wesentlicher Bedeutung. Im Abschnitt 3.2 haben wir bereits gesehen, daß verschiedene mathematisch äquivalente Ausdrücke nicht unbedingt zu gleichen Ergebnissen führen, wenn Rundungsfehler mit im Spiel sind. Es muß also ein Verfahren gefunden werden zu entscheiden, welche von mehreren mathematisch äquivalenten Methoden zur Berechnung eines bestimmten numerischen Ausdruckes herangezogen werden soll.

**Beispiel 5.1.2.** *Wie unterschiedlich die Resultate verschiedener Verfahren zur Auswertung ein und derselben Funktion ausfallen können, sei anhand der Funktion*

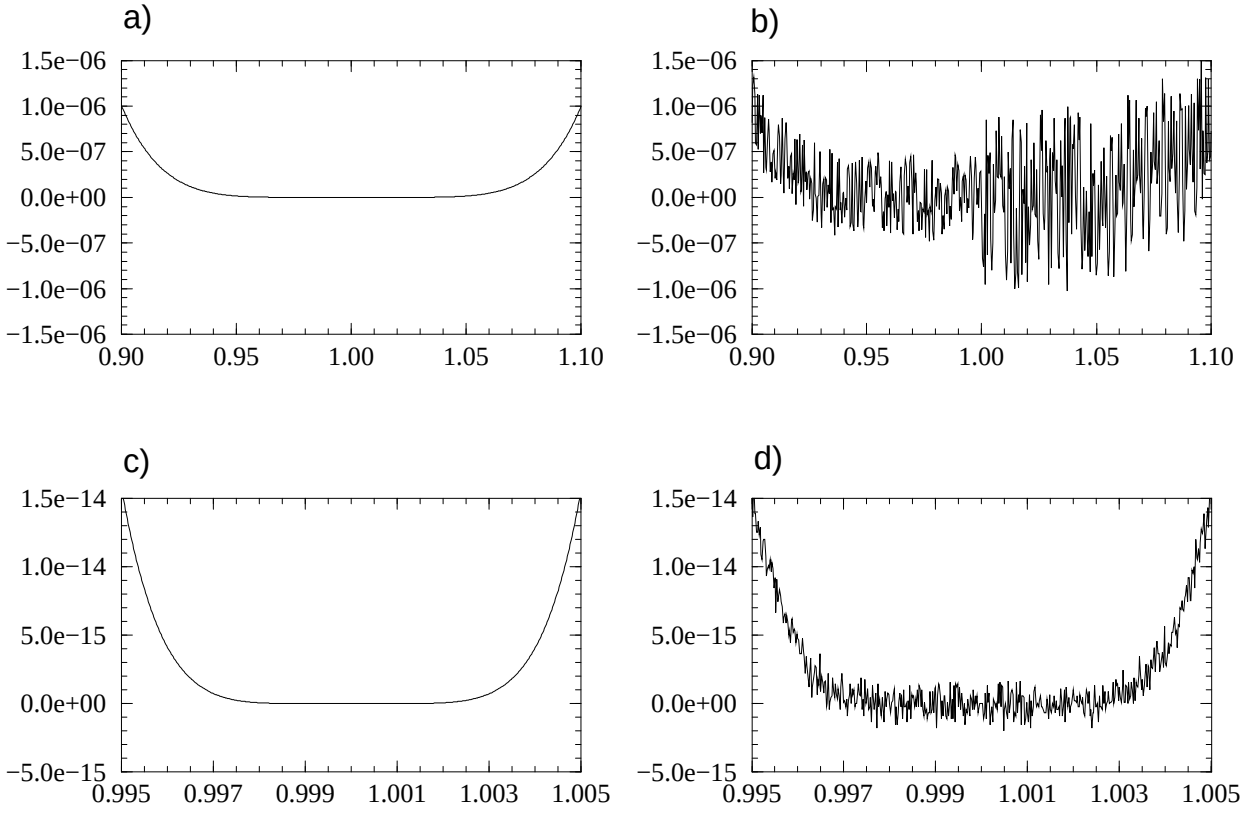
$$f(x) = (1-x)^6 = 1 - 6x + 15x^2 - 20x^3 + 15x^4 - 6x^5 + x^6$$

*gezeigt. Die folgenden vier Diagramme zeigen die Auswertung des ersten Ausdruckes, die naive Auswertung des zweiten Ausdruckes und die Auswertung des zweiten Ausdruckes mit Hilfe des Horner-Schemas:*

$$f(x) = 1 + x(-6 + x(15 + x(-20 + x(15 + x(-6 + x))))).$$

*Diagramm* a) *zeigt die Auswertung mit Hilfe des ersten Ausdruckes und des Hornerschemas, während Diagramm* b) *die naive Auswertung des expandierten Ausdrucks zeigt an 500 Stützstellen im Intervall* $[0.9, 1.1]$*. Daß auch das Hornerschema nur begrenzt verwendbar ist zeigt*

*der Vergleich zwischen Hornerschema und Auswertung von* $(1-x)^6$ *in den Diagrammen* c) *und* d) *im Intervall* $[0.995, 1.005]$ *wieder mit 500 Stützstellen. Alle Auswertungen geschehen mit einfacher Genauigkeit (ungefähr Rundung auf acht Stellen) und linearer Interpolation zwischen den Stützstellen.*




*Daran sieht man auch, daß bei numerischen Berechnungen Eigenschaften wie Monotonie und Stetigkeit von Funktionen scheinbar verloren gehen können.*

Auf **qualitative** Weise bezeichnen wir numerische Berechnungsmethoden, die den linken Diagrammen (a), c)) entsprechen als *numerisch stabiler* als Algorithmen, die ein Verhalten wie in den rechten Diagrammen (b), d)) zeigen.

Die folgenden und ähnliche Beispiele für Stabilität und Instabilität können auch in [**Neumaier 2000**, 1.3] gefunden werden. In allen Beispielen wird gerundete Rechnung auf 12 signifikante Dezimalstellen verwendet (ein Taschenrechner TI 59).

**Beispiel 5.1.3.** *Wir werten die Funktion*

$$f(x) = (x + \tfrac{1}{3}) - (x - \tfrac{1}{3})$$

*an mehreren Punkten aus. Das Ergebnis ist immer* $\frac{2}{3}$*, da* $f$ *nur ein komplizierter Ausdruck für eine konstante Funktion ist. Trotzdem weichen die Ergebnisse mitunter erheblich vom*

*Erwarteten ab.*

| $x$ | $\widetilde{f(x)}$ |
|---|---|
| 1 | 0.666666666666 |
| $10^3$ | 0.666666666000 |
| $10^6$ | 0.666666000000 |
| $10^9$ | 0.667000000000 |
| $10^{10}$ | 0.660000000000 |
| $10^{11}$ | 0.600000000000 |
| $10^{12}$ | 0 |

**Beispiel 5.1.4.** *Auch bei der Funktion*

$$f(x) = \frac{(3 + \frac{x^2}{3}) - (3 - \frac{x^2}{3})}{x^2}$$

*iat das Ergebnis ist immer $\frac{2}{3}$, und auch hier treten seltsame Phänomene auf.*

| $x$ | $\widetilde{f(x)}$ |
|---|---|
| 1 | 0.666666666666 |
| $10^{-1}$ | 0.666666666000 |
| $10^{-2}$ | 0.666666660000 |
| $10^{-3}$ | 0.666666000000 |
| $10^{-4}$ | 0.666600000000 |
| $10^{-5}$ | 0.660000000000 |
| $2 \cdot 10^{-6}$ | 0.500000000000 |
| $10^{-6}$ | 0 |

Zwei Gründe existieren für den Genauigkeitsverlust in den Beispielen 5.1.3 und 5.1.4.

- Die zu addierenden bzw. subtrahierenden Zahlen unterscheiden sich um mehrere Größenordnungen (z.B. $10^{12} + 3$). Sind $x_i = m_i \cdot 10^{e_i}$ für $i = 1, 2$ zwei Zahlen mit $d := e_1 - e_2 - 1 > 0$, dann hat $x_2$ keinen Einfluß auf die ersten $d$ Dezimalstellen der Summe oder der Differenz $x_1 \pm x_2$. Die Informationen von $x_2$ sind nur in den weniger signifikanten Stellen des Ergebnisses, und die letzten $d$ Stellen von $x_2$ gehen im Rundungsprozeß völlig verloren.
- Die Subtraktion zweier Zahlen fast gleicher Größe führt zur Auslöschung signifikanter Stellen und verschiebt bereits mit Rundungsfehlern behaftete Dezimalstellen von hinteren nicht-signifikanten in vordere signifikantere Positionen. Dabei steigt der **relative** Fehler um mehrere Größenordnungen.

Dividiert man durch sehr kleine Zahlen oder multipliziert man in den Rechnungen mit sehr großen Zahlen, so wird zwar der relative Fehler nicht verändert, der **absolute** Fehler kann aber gewaltig anwachsen. Glücklicherweise ist dies meist kein großes Problem.

Im Fall der Auslöschung beachte man noch, daß die Subtraktion selbst im allgemeinen **ohne Rundungsfehler** ausgeführt wird. Die Instabilität stammt in diesem Fall aus der **Verstärkung früherer Fehler**, wie mit dem folgenden Beispiel belegt.

**Beispiel 5.1.5.** *Man berechnet die Differenz zweier Zahlen $x$, $y$ auf 7 bzw. 6 signifikante Stellen.*

**7 Stellen:**

| | | |
|---|---|---|
| $x$: | 0.3344689 | |
| $y$: | 0.3344342 | |
| $x - y$: | 0.0000347 | *kein Rundungsfehler* |
| $x - y$: | $0.347 \cdot 10^{-4}$ | |

**6 Stellen:**

| | *gerundet* | *relativer Fehler* |
|---|---|---|
| $x$: | 0.334469 | $\approx 3 \cdot 10^{-7}$ |
| $y$: | 0.334434 | $\approx 6 \cdot 10^{-7}$ |
| $x - y$: | 0.000035 | *kein Rundungsfehler* |
| $x - y$: | $0.35 \cdot 10^{-4}$ | $\approx 8.6 \cdot 10^{-3}$ |

*Obwohl bei der Berechnung der Differenz kein Rundungsfehler aufgetreten ist, wurden also die alten Rundungsfehler in den Eingabedaten um etwa einen Faktor* 10000 *verstärkt.*

**5.2. Fehlerfortpflanzung - mathematische Betrachtung.** Im folgenden werden wir manchmal die folgende etwas saloppe Schreibweise verwenden: Steht für einen arithmetischen Ausdruck $A$ fest, auf welche Art und Weise er berechnet werden soll (ev. durch geeignete Klammerung vorgeschrieben), so bezeichnen wir mit $\mathrm{gl}(A)$ das Ergebnis, welches durch Gleitkommarechnung zustande kommt. Auch die Auswertung von elementaren und transzendenten Funktionen durch Ersatzfunktionen (möglicherweise mit Approximationsfehlern behaftet) wird solcherart bezeichnet.

**Beispiel 5.2.1.**

$$\begin{aligned} \mathrm{gl}(x * y) &= x *_* y \\ \mathrm{gl}(a + (b + c)) &= a +_* (b +_* c) \\ \mathrm{gl}((a + b) + c) &= (a +_* b) +_* c \\ \mathrm{gl}(\sqrt{x}) &= \sqrt{x}_* \end{aligned}$$

Die arithmetischen Operationen sowie alle Funktionen, für die Gleitkomma-Ersatzfunktionen implementiert sind, werden in der Folge als *einfache Operationen* bezeichnet. In der Folge werden wir o.B.d.A. annehmen, daß alle elementaren Operationen $E$ so implementiert sind, daß

$$\mathrm{gl}(E) = E(1 + \delta), \quad |\delta| \leq \mathsf{eps} \tag{2}$$

gilt.

Um den Begriff des Algorithmus etwas zu formalisieren, nehmen wir an, daß jedes Problem darin besteht, aus einem Satz von endlich vielen Eingabedaten $x_1, \ldots, x_n$ endlich viele Ausgabedaten $y_1, \ldots, y_m$ zu berechnen. Solch ein Problem zu lösen heißt also, den Wert einer Funktion $\varphi : D \to \mathbb{R}^m$ zu bestimmen mit $D \subseteq \mathbb{R}^n$. Ein Algorithmus ist eine eindeutige Rechenvorschrift zur Berechnung von $\varphi(x)$. In jedem Zwischenschritt des Algorithmus sind die Zwischenergebnisse in Form eines reellen Vektors $x^{(i)} \in \mathbb{R}^{n_i}$ gegeben. Der Übergang zum nächsten Zwischenergebnis wird durch eine *elementare Abbildung* $\varphi^{(i)} : D_i \to D_{i+1}$, $D_k \subseteq \mathbb{R}^{n_k}$, $x^{(i+1)} = \varphi^{(i)}(x^{(i)})$ vermittelt. Diese elementaren Abbildungen sind durch den Algorithmus eindeutig bestimmt (unter Vernachlässigung der trivialen Permutationsinvarianz in den Zwischenergebnissen).

Die Abfolge der elementaren Operationen bestimmt also eine Dekomposition der Funktion $\varphi$ in eine Folge elementarer Abbildungen $\varphi^{(i)}$ mit

$$\varphi = \varphi^{(r)} \circ \varphi^{(r-1)} \circ \ldots \circ \varphi^{(0)}, \quad D_0 = D, \quad D_{r+1} \subseteq \mathbb{R}^{n_{r+1}} = \mathbb{R}^m.$$

**Beispiel 5.2.2.** *Wegen $a^2 - b^2 = (a+b)(a-b)$ kann man zur Berechnung von $\varphi(a, b) = a^2 - b^2$ zwei unterschiedliche Algorithmen heranziehen:*
*Algorithmus 1:*

$$\varphi^{(0)}(a, b) = \begin{pmatrix} a^2 \\ b \end{pmatrix}, \quad \varphi^{(1)}(u, v) = \begin{pmatrix} u \\ v^2 \end{pmatrix}, \quad \varphi^{(2)}(u, v) = u - v.$$

*Algorithmus 2:*

$$\varphi^{(0)}(a,b) = \begin{pmatrix} a+b \\ a \\ b \end{pmatrix}, \quad \varphi^{(1)}(u,v,w) = \begin{pmatrix} u \\ v-w \end{pmatrix}, \quad \varphi^{(2)}(u,v) = u\,v.$$

Mit Hilfe dieser Form der Darstellung wollen wir jetzt untersuchen, welche Gründe dafür verantwortlich sind, daß verschiedene Algorithmen unterschiedliche Resultate liefern, um Kriterien für die Beurteilung der Güte von Algorithmen zu gewinnen. Die schon angesprochene Fortpflanzung der Rundungsfehler spielt dabei eine wichtige Rolle.

Kommen wir für den Anfang der Untersuchungen zur Verletzung des Assoziativgesetzes der Addition zurück. Sei also $\varphi(a,b,c) = a+b+c$. Bei Gleitpunktrechnung erhält man statt $y = \varphi(a,b,c)$ einen Näherungswert $\tilde{y} = \mathrm{gl}((a+b)+c)$, für den wegen 2 gilt

$$\begin{aligned} \eta &:= \mathrm{gl}(a+b) = (a+b)(1+\delta_1) \\ \tilde{y} &= \mathrm{gl}(\eta + c) = (\eta + c)(1+\delta_2) = ((a+b)(1+\delta_1)+c)(1+\delta_2) = \\ &= (a+b+c)(1+\frac{a+b}{a+b+c}\delta_1(1+\delta_2)+\delta_2). \end{aligned}$$

Für den relativen Fehler $\varepsilon_y$ erhält man daher

$$\varepsilon_y = \frac{a+b}{a+b+c}\delta_1(1+\delta_2)+\delta_2.$$

Da eps meist eine sehr kleine Zahl ist und nach Voraussetzung immer $|\delta_i| \leq \mathsf{eps}$ gilt, genügt es in erster Näherung Terme höherer als erster Ordnung in den $\delta_i$ zu ignorieren, und wir erhalten

$$\varepsilon_y \doteq \frac{a+b}{a+b+c}\delta_1 + 1 \cdot \delta_2$$

($\doteq$ bezeichne auch im folgenden „gleich bis auf Terme zweiter oder höherer Ordnung“). Die Verstärkungsfaktoren $\frac{a+b}{a+b+c}$ und 1 bestimmen, wie stark sich die Rundungsfehler $\delta_i$ in den elementaren Funktionen auf den relativen Gesamtfehler $\varepsilon_y$ auswirken. In diesem Beispiel ist der kritische Faktor der Bruch vor $\delta_1$. Wertet man die Summe in der anderen Reihenfolge aus ($\tilde{y} = \mathrm{gl}(a + (b+c))$), so ist der entsprechende Vorfaktor $\frac{b+c}{a+b+c}$. Im Beispiel aus dem Abschnitt 3.2 sind die beiden Faktoren

$$\frac{a+b}{a+b+c} \approx 3.910^{-5}, \quad \frac{b+c}{a+b+c} \approx 1,$$

was die größere Genauigkeit der Berechnung nach der ersten Methode erklärt. Im allgemeinen ist es also numerisch stabiler, die Summe $a+b+c$ nach der Formel $(a+b)+c$ zu berechnen, wenn $|a+b| < |b+c|$ gilt.

**5.2.1. Kondition, numerische Stabilität.**

**Beispiel 5.2.3.** *Sei*

$$f(x) = \frac{1}{1-x}.$$

*Bei $x = 0.999$ gilt $f(x) = 1000$. Untersuchen wir den Fehler analytisch, so ergibt sich für $\tilde{x} = 0.999 + \varepsilon$ mit kleinem Fehlerterm $\varepsilon$*

$$\begin{aligned} f(x) = \frac{1000}{1-1000\varepsilon} &= \\ &= 1000(1+10^3\varepsilon+10^6\varepsilon^2+\dots). \end{aligned}$$

*Verwendet man dieses Resultat, um den relativen Fehler zu bestimmen, so erhält man die Ausdrücke*

$$\frac{|x-\tilde{x}|}{|x|} = \frac{\varepsilon}{0.999} \approx 1.001\varepsilon,$$
$$\frac{|f(x)-f(\tilde{x})|}{f(x)} = 10^3\varepsilon + 10^6\varepsilon^2 + \ldots.$$

*Der relative Fehler steigt also etwa um einen Faktor* $10^3$*, obwohl wir für die Berechnung von* $f$ *Rundungsfehler freie Rechnung angenommen haben.*

Problemstellungen wie in Beispiel 5.2.3 nennt man *schlecht konditioniert (ill-conditioned).* Auch dieser Begriff hat eine **qualitative** Bedeutung ohne exakte Definition.

Die qualitativen Begriffe Stabilität und Konditioniertheit wollen wir im folgenden mathematisch zu fassen versuchen. Wir bedienen uns dabei einer Methode, Fehlerfortpflanzung durch Vernachlässigung Terme höherer Ordnung zu untersuchen, der *differentiellen Fehleranalyse* des Algorithmus

$$\varphi = \varphi^{(r)} \circ \varphi^{(r-1)} \circ \ldots \circ \varphi^{(0)}.$$

Zuerst genügt es zu untersuchen, wie sich die Eingangsfehler $\Delta x$ von $x$ auf das Endresultat $y = \varphi(x)$ auswirken. Wir setzen voraus, daß die Funktion $\varphi : D \subseteq \mathbb{R}^n \to \mathbb{R}^m$ stetig differenzierbar ist. Ersetzt man die Eingabedaten $x \in \mathbb{R}^n$ durch abgeänderte Daten $\tilde{x} \in \mathbb{R}^n$, so erhält man als Resultat $\tilde{y} = \varphi(\tilde{x})$ anstatt $y = \varphi(x)$. Durch Taylorentwicklung bis zum ersten Glied ergibt sich unter Vernachlässigung Termen höherer als erster Ordnung

$$\Delta y = \varphi(\tilde{x}) - \varphi(x) \doteq D\varphi(x)\Delta x, \tag{3}$$

wobei

$$D\varphi(x) = \begin{pmatrix} \frac{\partial\varphi_1(x)}{\partial x_1} & \cdots & \frac{\partial\varphi_1(x)}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial\varphi_m(x)}{\partial x_1} & \cdots & \frac{\partial\varphi_m(x)}{\partial x_n} \end{pmatrix}$$

die Jacobimatrix (Funktionalmatrix) von $\varphi$ bei $x$ bezeichnet. Der einzelne Faktor $\frac{\partial\varphi_i(x)}{\partial x_j}$ mißt die Empfindlichkeit, mit der $y_i$ auf Änderungen in $x_j$ reagiert. Ist $y_i \neq 0$ und ist $x_j \neq 0$ für alle $j$, so folgt aus (3) für den relativen Fehler

$$\varepsilon_{y_i} \doteq \sum_{j=1}^{n} \frac{x_j}{\varphi_i(x)} \cdot \frac{\partial\varphi_i(x)}{\partial x_j} \cdot \varepsilon_{x_j}. \tag{4}$$

Die Verstärkungsfaktoren $\kappa_{ij} := \frac{x_j}{\varphi_i(x)} \cdot \frac{\partial\varphi_i(x)}{\partial x_j}$ für die relativen Fehler, die unabhängig sind von der Skalierung der $y_i$ und der $x_j$, heißen die *Konditionszahlen* der Funktion $\varphi$ bei $x$. $\kappa_{ij}$ gibt dabei den Einfluß vom Eingabeparameter $x_i$ auf die Ergebniskomponente $f_j(x)$ an. Sind die Konditionszahlen klein, liegt ein *gut konditioniertes* andernfalls ein *schlecht konditioniertes* Problem vor. Bei schlecht konditionierten Problemen bewirken, wie wir oben gesehen haben, relativ kleine Fehler in den Eingangsdaten $x$ große Fehler in den Ergebnissen, unabhängig von der verwendeten Berechnungsmethode, selbst bei Rundungsfehler freier Rechnung.

Ist eine Aufgabe sehr schlecht konditioniert unabhängig davon, welchen Algorithmus man zu ihrer Lösung verwendet, dann tritt chaotisches Verhalten (*Chaos*) auf, d.h. die Ergebnisse werden scheinbar nicht mehr durch die Funktion $\varphi$ sondern nur noch durch die Fehler in den Eingabedaten bestimmt. Solches Verhalten tritt bei der Behandlung mancher dynamischer Systeme, die durch nichtlineare gewöhnliche oder partielle Differentialgleichungen oder durch nichtlineare Differenzengleichungen beschrieben werden in natürlicher Weise auf (auch bei

exakter Rechnung), z.B. bei der Beschreibung von viskosen Strömungen mit Hilfe der Navier–Stokes Gleichungen (siehe Numerik 2) oder der logistischen Differenzengleichung, die z.B. Räuber–Beute–Modelle in der Biologie beschreibt.

Die Definition der Konditionszahlen von oben hat den Nachteil, daß für eine Funktion $\varphi : \mathbb{R}^n \to \mathbb{R}^m$ immerhin $m \cdot n$ Zahlen berechnet werden müssen, um die Kondition eines Problems zu bestimmen. Daher werden diese Zahlen oft zu einer einzigen Konditionszahl $\kappa$ zusammengefaßt. Eine Zahl $\kappa$ heißt Konditionszahl, wenn bezüglich einer geeigneten Norm

$$\frac{\|\Delta y\|}{\|y\|} \leq \kappa \frac{\|\Delta x\|}{\|x\|}$$

gilt. Eine Konditionszahl kann man daher nach 3 berechnen als

$$\frac{\|\Delta y\|}{\|y\|} \leq \frac{\|D\varphi(x)\|_M \|\Delta x\|}{\|\varphi(x)\|}, \quad \text{daher}$$
$$\kappa = \frac{\|x\|}{\|\varphi(x)\|} \cdot \|D\varphi(x)\|_M,$$

für eine zu $\| \quad \|$ passende Matrixnorm $\| \quad \|_M$. Besonders für lineares $\varphi$ ist die obige Definition üblich, und im Kapitel 3 werden wir wieder darauf treffen.

Für eindimensionale Funktionen $f : \mathbb{R} \to \mathbb{R}$ ist die Untersuchung der Kondition am Punkt $x^*$ sehr einfach. Im wesentlichen untersuchen wir drei verschiedene Fälle:

1. **Einfache Nullstelle:** Ist $f(x^*) = 0$ und $f'(x^*) \neq 0$, hat also $f$ bei $x^*$ eine *einfache Nullstelle*, dann strebt $\kappa \to \infty$ für $x \to x^*$. $f$ ist also in der Umgebung einer einfachen Nullstelle $x^* \neq 0$ schlecht konditioniert.
2. **Mehrfache Nullstellen, Pole:** Existiert eine ganze Zahl $k$ mit

   $$f(x) = (x - x^*)^k g(x), \qquad g(x^*) \neq 0,$$

   dann hat $f$ eine Nullstelle der Ordnung $k$ bei $x^*$, falls $k > 0$ gilt. Ist $k < 0$, so sprechen wir von einem Pol der Ordnung $k$. Immer gilt.

   $$f'(x) = k(x - x^*)^{k-1} g(x) + (x - x^*)^k g'(x)$$
   und
   $$\kappa = \left| \frac{x f'(x)}{f(x)} \right| =$$
   $$= |x| \cdot \left| \frac{m}{x - x^*} + \frac{g'(x)}{g(x)} \right| \doteq$$
   $$\doteq |m| \cdot \left| \frac{x - x^*}{x} \right|^{-1}.$$

   Für $x \to x^*$ finden wir also

   $$\kappa \to \begin{cases} \infty & x^* \neq 0, \\ |m| & x^* = 0. \end{cases}$$

   Für einen Pol bzw. eine Nullstelle bei $x^*$ sehen wir also wieder, daß $f$ schlecht konditioniert ist, wenn $x^* \neq 0$ gilt. Befindet sich der kritische Punkt aber bei Null, so ist das Problem gut konditioniert.
3. **Spitzen:** Ist $f$ bei $x^*$ nicht differenzierbar, und gilt $\lim_{x \to x^*} f'(x) = \infty$ ($f$ hat eine „Spitze“ bei $x^*$), so ist $f$ schlecht konditioniert bei $x^*$.

**Beispiel 5.2.4.** *Für die arithmetischen Operationen und die Quadratwurzel berechnet man*

$$\begin{aligned}
\varphi(x,y) := x * y: &\quad \varepsilon_{xy} \doteq \varepsilon_x + \varepsilon_y,\\
\varphi(x,y) := x/y: &\quad \varepsilon_{x/y} \doteq \varepsilon_x - \varepsilon_y,\\
\varphi(x,y) := x \pm y: &\quad \varepsilon_{x\pm y} \doteq \frac{x}{x\pm y}\varepsilon_x \pm \frac{y}{x\pm y}\varepsilon_y, \quad \textit{falls } x \pm y \neq 0,\\
\varphi(x) := \sqrt{x}: &\quad \varepsilon_{\sqrt{x}} \doteq \tfrac{1}{2}\varepsilon_x.
\end{aligned}$$

Man sieht also, daß die einzigen numerisch gefährlichen arithmetischen Operationen die Addition und die Subtraktion sind. Beschränken wir die weiteren Untersuchungen auf die Addition. Haben $x$ und $y$ gleiches Vorzeichen, so liegt jede der beiden Konditionszahlen zwischen 0 und 1, ihre Summe ist gleich 1, und daher haben wir

$$|\varepsilon_{x+y}| \leq \max\{|\varepsilon_x|, |\varepsilon_y|\}.$$

Wenn ein Summand klein ist gegenüber dem anderen, so kann er mit einem großen relativen Fehler behaftet sein, und das Ergebnis kann trotzdem nur einen kleinen relativen Fehler aufweisen, wenn nur der andere Summand einen kleinen relativen Fehler hat. In diesem Fall spricht man von *Fehlerdämpfung.*

Andererseits, wenn die beiden Summanden verschiedenes Vorzeichen haben und vom Betrag ungefähr gleich sind, dann ist mindestens einer der beiden Faktoren $|x/(x+y)|$, $|y/(x+y)|$ (möglicherweise viel) größer als 1, und es wird mindestens einer der Fehler $\varepsilon_x$ oder $\varepsilon_y$ verstärkt. Dies beschreibt das Phänomen der *Auslöschung (Verlust signifikanter Stellen)* mathematisch genauer. Aufgrund der Wichtigkeit noch ein Beispiel:

**Beispiel 5.2.5.** *Nehmen wir an, daß wir mit Rundung auf acht Stellen rechnen. Seien* $x = 0.348335866229$ *und* $y = -0.348313234991$*, und* $x_* = 0.\underline{34833587}$ *und* $y_* = -0.\underline{34831323}$ *Gleitkommaapproximationen zu* $x$ *bzw.* $y$*. Es berechnet sich* $z_* := x_* + y_* = 0.\underline{226}40000 \cdot 10^{-4}$*, was sich exakt darstellen läßt; deshalb tritt an dieser Stelle auch kein zusätzlicher Rundungsfehler auf. Die Approximationen* $x_*$ *und* $y_*$ *stimmen mit* $x$ *und* $y$ *in mindestens sieben Stellen überein. Die achte Stelle ist schon fehlerbehaftet. Betrachtet man nun* $z_*$*, so sieht man, daß die* 4 *aus eben diesen fehlerbehafteten Stellen berechnet worden ist, deshalb selbst bereits fehlerhaft ist. Die Nullen dahinter geben überhaupt keine Auskunft mehr über* $z = x - y = 0.22631238 \cdot 10^{-4}$*. Wir sehen also, daß die Approximation* $z_*$ *für die wahre Differenz* $z$ *nur mehr auf drei signifikante Stellen genau ist. Wir haben durch* Auslöschung *vier signifikante Stellen verloren — also ist der relative Fehler um einen Faktor* 10000 *angewachsen.*

Daß Stabilität und Kondition zwei grundverschiedene Dinge sind, soll schließlich das folgende Beispiel aus [**Neumaier 2000**, 1.4] unterstreichen:

**Beispiel 5.2.6.** *Sei*

$$f(x) = \sqrt{x^{-1} - 1} - \sqrt{x^{-1} + 1}, \quad \textit{für } (0 < x < 1).$$

*Bei* $x \approx 0$ *gilt* $x^{-1} - 1 \approx x^{-1} + 1$*, und es tritt Auslöschung auf, also ist der Ausdruck instabil. Bei* $x \approx 1$ *sind* $\sqrt{x^{-1} - 1} \approx 0$ *und* $\sqrt{x^{-1} + 1} \approx \sqrt{2}$*, der Ausdruck ist also stabil.*

*Andererseits berechnet sich die Konditionszahl*

$$\kappa = \frac{1}{2\sqrt{1 - x^2}},$$

*und für* $x \approx 0$ *ist* $\kappa \approx \frac{1}{2}$ *während* $\lim_{x\to 1} \kappa = \infty$ *gilt.*

*Die Funktion* $f$ *ist also bei* 0 *gut konditioniert aber obiger Ausdruck für* $f$ *ist instabil. Bei* 1 *ist die Kondition schlecht, doch der Ausdruck ist stabil. Die schlechte Kondition bei* 1

*kann man durch mathematische Transformationen nicht verändern, doch die Instabilität bei* 0 *kann sehr leicht durch äquivalente Umformung des Ausdrucks für* $f$ *beseitigt werden:*

$$f(x) = \frac{\sqrt{x^{-1}-1}+\sqrt{x^{-1}+1}}{\sqrt{x^{-1}-1}+\sqrt{x^{-1}+1}}\left(\sqrt{x^{-1}-1}-\sqrt{x^{-1}+1}\right) = \\ = \frac{\sqrt{x^{-2}-1}}{\sqrt{x^{-1}-1}+\sqrt{x^{-1}+1}}.$$

*In dieser Form für* $f$ *tritt nahe* 0 *keine Auslöschung auf.*

Um die Fortpflanzung von Fehlern in mehrstufigen Algorithmen zu analysieren, verwenden wir wieder Gleichung (3). Betrachten wir wieder den Algorithmus als Zerlegung

$$\varphi = \varphi^{(r)} \circ \varphi^{(r-1)} \circ \ldots \circ \varphi^{(0)},$$

mit stetig differenzierbaren $\varphi^{(i)} : D_i \subseteq \mathbb{R}^{n_i} \to D_{i+1} \subseteq \mathbb{R}^{n_{i+1}}$. Setzen wir wieder $x^{(i+1)} = \varphi^{(i)}(x^{(i)})$ mit $x^{(0)} = x$, und seien die Restabbildungen $\psi^{(i)}$ definiert durch

$$\psi^{(i)} = \varphi^{(r)} \circ \ldots \circ \varphi^{(i)}.$$

Dann gelten für alle $i$ nach der Kettenregel

$$D\varphi(x) = D\varphi^{(r)}(x^{(r)})D\varphi^{(r-1)}(x^{(r-1)}) \ldots D\varphi^{(0)}(x) \quad (5)$$

$$D\psi^{(i)}(x^{(i)}) = D\varphi^{(r)}(x^{(r)})D\varphi^{(r-1)}(x^{(r-1)}) \ldots D\varphi^{(i)}(x^{(i)}). \quad (6)$$

In jedem Schritt erhält man durch den Einfluß der Eingangs- und Rundungsfehler statt der exakten Zwischenresultate $x^{(i)}$ Näherungswerte $\tilde{x}^{(i)}$. Der Zusammenhang ist $\tilde{x}^{(i+1)} = \mathrm{gl}(\varphi^{(i)}(\tilde{x}^{(i)}))$. Für den absoluten Fehler $\Delta x^{(i+1)}$ haben wir

$$\Delta x^{(i+1)} = \big(\mathrm{gl}(\varphi^{(i)}(\tilde{x}^{(i)})) - \varphi^{(i)}(\tilde{x}^{(i)})\big) + \big(\varphi^{(i)}(\tilde{x}^{(i)}) - \varphi^{(i)}(x^{(i)})\big). \quad (7)$$

Es gilt in erster Näherung nach Gleichung 3

$$\varphi^{(i)}(\tilde{x}^{(i)}) - \varphi^{(i)}(x^{(i)}) \doteq D\varphi^{(i)}(x^{(i)}) \cdot \Delta x^{(i)}. \quad (8)$$

Für den zweiten Term verwenden wir die Voraussetzung 2 und erhalten die Abschätzung

$$\mathrm{gl}(\varphi^{(i)}(z)) = (\mathbb{I} + E_{i+1}) \cdot \varphi^{(i)}(z),$$

wobei $\mathbb{I}$ die Einheitsmatrix bezeichnet, und $E_{i+1}$ die diagonale Fehlermatrix

$$E_{i+1} = \begin{pmatrix} \varepsilon_1 & & & 0 \\ & \varepsilon_2 & & \\ & & \ddots & \\ 0 & & & \varepsilon_{n_{i+1}} \end{pmatrix}.$$

Dies führt zu der Abschätzung

$$\mathrm{gl}(\varphi^{(i)}(\tilde{x}^{(i)})) - \varphi^{(i)}(\tilde{x}^{(i)}) = E_{i+1} \cdot \varphi^{(i)}(\tilde{x}^{(i)}).$$

Nachdem $\tilde{x}^{(i)}$ in erster Näherung gleich $x^{(i)}$ ist, gilt auch in erster Näherung durch Taylorentwicklung

$$\mathrm{gl}(\varphi^{(i)}(\tilde{x}^{(i)})) - \varphi^{(i)}(\tilde{x}^{(i)}) \doteq E_{i+1}\varphi^{(i)}(x^{(i)}) = \\ = E_{i+1}x^{(i+1)} =: \alpha_{i+1}. \quad (9)$$

$\alpha_{i+1}$ ist der bei der Auswertung von $\varphi^{(i)}$ *neu entstehende* absolute Rundungsfehler, die Diagonalelemente von $E_{i+1}$ sind die dazugehörigen relativen Rundungsfehler. Aus den Gleichungen (7), (8) und (9) folgt eine rekursive Formel für die absoluten Fehler

$$\Delta x^{(i+1)} \doteq \alpha_{i+1} + D\varphi^{(i)}(x^{(i)})\Delta x^{(i)} = E_{i+1}\, x^{(i+1)} + D\varphi^{(i)}(x^{(i)})\Delta x^{(i)}, \quad (10)$$

wo $\Delta x^{(0)} = \Delta x$. Löst man die Rekursion auf, so erhält man für den Gesamtfehler unter Verwendung von (5) und (6)

$$\begin{aligned}\Delta y = \Delta x^{(r+1)} &\doteq D\varphi^{(r)} \dots D\varphi^{(0)}\Delta x + D\varphi^{(r)} \dots D\varphi^{(1)}\alpha_1 + \dots + \alpha_{r+1} = \\ &= D\varphi(x)\Delta x + D\psi^{(1)}(x^{(1)})\alpha_1 + \dots + D\psi^{(r)}(x^{(r)})\alpha_r + \alpha_{r+1} = \\ &= D\varphi(x)\Delta x + D\psi^{(1)}(x^{(1)})E_1x^{(1)} + \dots + D\psi^{(r)}(x^{(r)})E_rx^{(r)} + E_{r+1}y.\end{aligned} \tag{11}$$

Entscheidend für den Einfluß des bei der Berechnung von $x^{(i)}$ begangenen Rundungsfehlers $\alpha_i$ (bzw. $E_i$) ist also die Größe der Funktionalmatrix $D\psi^{(i)}$ der Restabbildung $\psi^{(i)}$.

Wenn man nun zur Berechnung von $y = \varphi(x)$ einen anderen Algorithmus (eine andere Zerlegung) verwendet, so ändert sich zwar $D\varphi(x)$ nicht, wohl aber die Elementarfunktionen $\varphi^{(i)}$ und damit die Restabbildungen $\psi^{(i)}$, also auch die Matrizen $D\psi^{(i)}$, die die Fortpflanzung der Rundungsfehler messen. Damit ändert sich aber auch der Gesamteinfluß aller Rundungsfehler

$$D\psi^{(1)}(x^{(1)})\alpha_1 + \dots + D\psi^{(r)}(x^{(r)})\alpha_r + \alpha_{r+1}.$$

Ein Algorithmus ist dann *numerisch stabiler* als ein anderer, wenn dieser Gesamteinfluß der Rundungsfehler bei dem ersten kleiner ist als bei dem zweiten. Am Beispiel der beiden Algorithmen aus Beispiel 5.2.2 wollen wir diese Analyse einmal durchführen:

**Beispiel 5.2.7.** *Algorithmus 1:*

$$x = x^{(0)} = \begin{pmatrix} a \\ b \end{pmatrix}, \quad x^{(1)} = \begin{pmatrix} a^2 \\ b \end{pmatrix}, \quad x^{(2)} = \begin{pmatrix} a^2 \\ b^2 \end{pmatrix}, \quad x^{(3)} = y = a^2 - b^2,$$

$$\psi^{(1)}(u,v) = u - v^2, \quad \psi^{(2)}(u,v) = u - v,$$

$$D\varphi(a,b) = (2a, -2b),$$

$$D\psi^{(1)}(x^{(1)}) = (1, -2b), \quad \psi^{(2)}(x^{(2)}) = (1, -1),$$

$$\alpha_1 = \begin{pmatrix} \varepsilon_1 a^2 \\ 0 \end{pmatrix}, \quad E_1 = \begin{pmatrix} \varepsilon_1 & 0 \\ 0 & 0 \end{pmatrix} \quad \alpha_2 = \begin{pmatrix} 0 \\ \varepsilon_2 b^2 \end{pmatrix}, \quad E_2 = \begin{pmatrix} 0 & 0 \\ \varepsilon_2 & 0 \end{pmatrix}$$

$$\alpha_3 = \varepsilon_3(a^2 - b^2)$$

$|\varepsilon_i| < \mathsf{eps}$ *für alle* $i$. *Für den Gesamtfehler erhält man daher mit* $\Delta x = \begin{pmatrix} \Delta a \\ \Delta b \end{pmatrix}$

$$\Delta y \doteq 2a\Delta a - 2b\Delta b + a^2\varepsilon_1 - b^2\varepsilon_2 + (a^2 - b^2)\varepsilon_3.$$

*Der Gesamteinfluß der Rundungsfehler kann demnach abgeschätzt werden als*

$$|a^2\varepsilon_1 - b^2\varepsilon_2 + (a^2 - b^2)\varepsilon_3| \leq \mathsf{eps}(a^2 + b^2 + |a^2 - b^2|).$$

*Analog berechnet man für Algorithmus 2:*

$$x = x^{(0)} = \begin{pmatrix} a \\ b \end{pmatrix}, \quad x^{(1)} = \begin{pmatrix} a \\ b \\ a+b \end{pmatrix}, \quad x^{(2)} = \begin{pmatrix} a+b \\ a-b \end{pmatrix}, \quad x^{(3)} = y = a^2 - b^2,$$

$$\psi^{(1)}(a,b,u) = u(a-b), \quad \psi^{(2)}(u,v) = u\,v,$$

$$D\varphi(x) = (2a, -2b),$$

$$D\psi^{(1)}(x^{(1)}) = (a+b, -a-b, a-b), \quad D\psi^{(2)}(x^{(2)}) = (a-b, a+b),$$

$$\alpha_1 = \begin{pmatrix} 0 \\ 0 \\ \varepsilon_1(a+b) \end{pmatrix}, \quad E_1 = \begin{pmatrix} 0 & & \\ & 0 & \\ & & \varepsilon_1 \end{pmatrix},$$

$$\alpha_2 = \begin{pmatrix} 0 \\ \varepsilon_1(a-b) \end{pmatrix}, \quad E_2 = \begin{pmatrix} 0 & \\ & \varepsilon_2 \end{pmatrix},$$

$$\alpha_3 = \varepsilon_3(a^2 - b^2), \quad E_3 = \varepsilon_3,$$

*wieder mit* $|\varepsilon_i| < \mathsf{eps}$ *für alle* $i$*. In diesem Fall berechnet sich der Gesamtfehler als*

$$\Delta y \doteq 2a\Delta a - 2b\Delta b + (a^2 - b^2)(\varepsilon_1 + \varepsilon_2 + \varepsilon_3).$$

*In diesem Fall gilt für den Gesamteinfluß der Rundungsfehler*

$$|(a^2 - b^2)(\varepsilon_1 + \varepsilon_2 + \varepsilon_3)| \leq 3\,\mathsf{eps}\,|a^2 - b^2|.$$

*Jetzt können wir also endlich untersuchen, welcher Algorithmus der numerisch stabilere ist. Wir müssen also untersuchen, für welche* $(a, b)$

$$\mathsf{eps}(a^2 + b^2 + |a^2 - b^2|) < 3\,\mathsf{eps}\,|a^2 - b^2|$$

*gilt. Eine einfache Rechnung zeigt, daß dies für* $1/3 < |a/b|^2 < 3$ *der Fall ist. Algorithmus 2 ist also für diese* $(a, b)$ *numerisch stabiler. In den anderen Fällen (ausgenommen den Grenzfällen, für die beide Algorithmen gleich stabil sind) ist Algorithmus 1 numerisch stabiler.*

*Man erhält zum Beispiel für* $a = 0.3237$*,* $b = 0.3134$ *bei Rechnung auf vier Stellen:*

*Algorithmus 1:* $a *_* a = 0.1048$*,* $b *_* b = 0.9882\,10^{-1}$*,*
$(a *_* a) -_* (b *_* b) = 0.6580\,10^{-2}$*,*
*Algorithmus 2:* $a +_* b = 0.6371$*,* $a -_* b = 0.1030\,10^{-1}$*,*
$(a +_* b) *_* (a -_* b) = 0.6562\,10^{-2}$*,*
*Das exakte Resultat ist* $a^2 - b^2 = 0.656213\,10^{-2}$*;* $|a/b|^2 = 1.0668\ldots$*.*

Diese Resultate sind also dazu geeignet, zu unterscheiden, welchen von zwei Algorithmen man in welchem Fall zur Berechnung des Resultates heranziehen soll. Es ist aber nicht zweckmäßig (eigentlich ist es sogar unmöglich), alle möglichen Algorithmen zur Auswertung einer bestimmten Funktion $\varphi$ zu analysieren. Man benötigt daher eine Methode, einen einzelnen gegebenen Algorithmus auf seine Verwendbarkeit zu überprüfen. Dazu betrachten wir noch einmal Gleichung (11): Der erste und der letzte Term sind unabhängig von der Zerlegung $\varphi = \varphi^{(r)} \circ \ldots \circ \varphi^{(0)}$. Einfache Abschätzungen liefern

$$|E_{r+1}y| \leq \mathsf{eps}\,|y|$$
$$|D\varphi(x)\Delta x| \leq \mathsf{eps}\,|D\varphi(x)||x|,$$

da bereits die Eingangsdaten $x$ durch Runden wenigstens einen Fehler der Größenordnung $|\Delta x| \leq \mathsf{eps}\,|x|$ aufweisen, wobei Absolutbeträge von Matrizen und Vektoren (auch im weiteren) komponentenweise gebildet werden. Die Summe dieser beiden Terme ist ein Fehler,

der bei jeder Berechnung von $\varphi$ auftritt, egal welchen Berechnungsalgorithmus wir wählen. Daher heißt

$$\Delta^{(0)}y := \mathsf{eps}\big(|\Delta\varphi(x)||x| + |y|\big) \tag{12}$$

der *unvermeidbare Fehler* von $y$. Nachdem man aber ohnehin mit einem Fehler der Größenordnung $\Delta^{(0)}y$ zu rechnen hat, ist es nicht vernünftig, von den Rundungsfehlern $\alpha_i$ (bzw. $E_i$) eines Algorithmus zu verlangen, daß ihre Beiträge zum Gesamtfehler wesentlich kleiner sind. Ein Rundungsfehler $\alpha_i$ ($E_i$) heißt also *harmlos*, falls sein Beitrag zum Gesamtfehler $\Delta y$ höchstens dieselbe Größenordnung wie der unvermeidbare Fehler $\Delta^{(0)}y$ ist:

$$|D\psi^{(i)}(x^{(i)})\alpha_i| = |D\psi^{(i)}(x^{(i)})E_i x^{(i)}| \approx \Delta^{(0)}y.$$

Sind alle Rundungsfehler in einem gegebenen Algorithmus harmlos, so nennt man den Algorithmus *gutartig*.

Der Begriff der Gutartigkeit eines Algorithmus ist zentral für die Numerische Mathematik. Solche Algorithmen sind grundsätzlich *numerisch stabil*. Hat man einen gutartigen Algorithmus für die Lösung eines Problemes gefunden, so reicht dies für die meisten Anwendungen aus. Lediglich, wenn $r$ (die Anzahl der elementaren Funktionen) sehr groß ist, kann es trotzdem notwendig sein, zu versuchen Algorithmen zu suchen, die noch stabiler sind, also Algorithmen, in denen viele Rundungsfehler Beiträge liefern, für die die Größenordnung der Fehlerbeiträge kleiner als $\Delta^{(0)}y$ ist, falls dies möglich ist.

Zusammenfassend kann man (als Faustregel) sagen, daß ein Algorithmus ziemlich sicher gutartig ist, wenn die Kondition aller Abbildungen $\varphi^{(i)}$ ungefähr gleich gut wie oder besser als die Kondition der Abbildung $\varphi$ ist. Es sollte jedoch in jedem Fall vermieden werden, einen Algorithmus so aufzubauen, daß Zwischenergebnisse $x^{(i)}$ gebildet werden, von denen das Endergebnis $y$ wesentlich empfindlicher abhängt als von den Eingangsdaten $x$.

**Beispiel 5.2.8.** *Die beiden Algorithmen aus den Beispielen 5.2.2 und 5.2.7 sind beide gutartig. Der unvermeidbaree Fehler ist nämlich*

$$\Delta^{(0)}y = \mathsf{eps}\left(\begin{pmatrix}2|a| & 2|b|\end{pmatrix}\begin{pmatrix}|a|\\|b|\end{pmatrix} + |a^2-b^2|\right) = \mathsf{eps}(2(a^2+b^2)+|a^2-b^2|).$$

*Ein Vergleich mit den Beiträgen, die in Beispiel 5.2.7 ausgerechnet worden sind, zeigt, daß der* gesamte *Rundungsfehlereinfluß beider Algorithmen dem Betrag nach höchstens gleich $\Delta^{(0)}y$ ist.*

**Beispiel 5.2.9** (Summation)**.** *Will man die Summe vieler ($N$) Zahlen $z_i$ berechnen, so muß man die Rundungsfehler kontrollieren. Ähnlich wie für drei Zahlen kann man ableiten, daß der Fehler minimal ist, wenn man die einzelnen Zahlen in der Reihenfolge ansteigender Absolutbeträge summiert.*

*Der Aufwand für das Sortieren der $z_i$ ist jedoch in der Größenordnung $N \log N$, was für große $N$ deutlich größer als der Aufwand von $N$ Additionen für die Standardmethode, die* rekursive Summation *(von $z_1$ beginnend), ist.*

*Die Frage ist also, ob man eine Methode finden kann, die Aufwand $O(N)$ hat, aber trotzdem die Rundungsfehler unter Kontrolle hält. Im folgenden werden zwei Summationsmethoden der Ordnung $O(N)$ vorgestellt, die die Rundungsfehler besser unter Kontrolle halten als die rekursive Summation. Für eine tiefgehende Untersuchung von Summationsmethoden und weitere effiziente Algorithmen sein auf* [**Neumaier 1974**] *verwiesen.*

*Die einfachste Verbesserung ist die Methode der* paarweisen Summation*. Hierbei werden jeweils zwei benachbarte Elemente der Folge $(z_1, \ldots, z_N)$ addiert, was zu einer neuen Folge*

$$\left(z_1^{(1)}, z_2^{(1)}, \ldots, z_{\lceil N/2\rceil}^{(1)}\right), \quad \textit{mit } z_i^{(1)} := z_{2i-1} + z_{2i}$$

*führt. Falls $N$ ungerade ist, setzen wir $z^{(1)}_{\lceil N/2\rceil} := z_N$. Auf die so entstandene Summe wird wieder die paarweise Summation angewandt, was eine weitere Folge $z^{(2)}_i$ definiert, und dieser Vorgang wird so lange wiederholt bis das Endergebnis*

$$s = z_1^{(\lceil \log_2 N\rceil)} = z_1 + \cdots + z_N$$

*vorliegt. Diese Form der Summation eignet sich hervorragend für Parallelrechner, da die einzelnen Summationen in jeder der $\log_2 N$ Stufen unabhängig von einander berechnet werden können. Zusätzlich erhält man aus der Fehlerabschätzung die Fehlerschranke*

$$\Delta s \le R_k \sum_{i=1}^{N} |x_i| \quad \textit{mit } R_k := \frac{k\,\mathsf{eps}}{1 - k\,\mathsf{eps}}$$

*und $k = \lceil \log_2 N\rceil$. Diese Fehlerschranke ist im allgemeinen kleiner als die Schranke für die rekursive Summation, da sie proportional zu $\log_2 N$ wächst und nicht mit $N$ wie im Standardfall.*

*W. Kahan hat eine Methode der Summation vorgeschlagen, die Aufwand $4N$ hat und deren Gesamtrundungsfehler in erster Ordnung von $N$ unabhängig ist. Sie basiert darauf, den Rundungsfehler in jedem einzelnen Schritt zu schätzen und die darauffolgenden Schritte demgemäß zu kompensieren. Für Binärarithmetik läßt sich der Rundungsfehler dem Schema aus Abb. 1.2 nach exakt bestimmen, in dem die Kästchen die Mantissen der Summanden darstellen:*

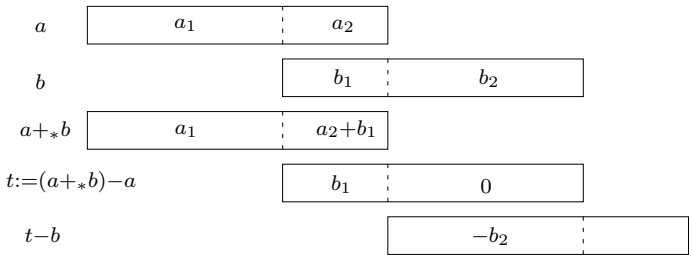



ABBILDUNG 1.2. Ermittlung des Rundungsfehlers $(a +_* b) - (a + b) = -b_2$

*Es gilt demnach*

$$e = ((a +_* b) -_* a) -_* b$$

*für $a \ge b$. Dieses Faktum nützt man aus, um den Rechenfehler in jedem Schritt zu schätzen und bei den folgenden Summationschritten wieder einzubringen. Dieses Prinzip führt zur* fehlerkompensierenden Summation *(Kahan–Babuška–Summation):*

**Algorithmus 5.2.10.** *Kahan–Babuška–Summation*

$s = x_1$
$e = 0$
**for** $i = 2$ **to** $n$ **do**
  $y = x_i - e$
  $z = s + y$
  $e = (z - s) - y$
  $s = z$
**done**

*Eine genauere Analyse der Rundungsfehler führt noch zur folgenden Verbesserung, die allerdings $N$ Vergleichsoperationen zusätzlich benötigt, die auf modernen Computer etwa denselben Aufwand benötigen wie Additionen. Der Gesamtaufwand beträgt also etwa $5N$.*

**Algorithmus 5.2.11.** *verbesserte Kahan–Babuška–Summation*

$s = x_1$
$e = 0$
**for** $i = 2$ **to** $n$ **do**
  $z = s + x_i$
  **if** $|x_m| \leq |s|$ **then**
    $e = e + (x_m + (s - z))$
  **else**
    $e = e + (s + (x_m - z))$
  **endif**
**done**
$s = s + e$

*Für diese Algorithmen wurde in* [**Neumaier 1974**] *folgender Satz bewiesen:*

**Theorem 5.2.12** (Kahan–Babuška–Summation)**.** *Der Rechenfehler bei der fehlerkompensierenden Summation kann für Algorithmus 5.2.10 durch*

$$|\Delta s| \leq \mathsf{eps}\,|s| + \left(\mathsf{eps} + \mathsf{eps}^2\left(\frac{3}{4}n^2 + \frac{7}{2}n\right)\right)\sum_{i=1}^{n}|x_i|$$

*bzw.*

$$|\Delta s| \leq \mathsf{eps}\,|s| + \left(\mathsf{eps}\,n + \mathsf{eps}^2\left(\frac{1}{4}n^3 + 3n^2 + 4n\right)\right)\max_{1\leq i\leq n}|x_i|$$

*abgeschätzt werden, sofern* $n\,\mathsf{eps} \leq \frac{1}{3}$ *gilt, was wohl immer der Fall sein dürfte.*

*Unter derselben Voraussetzung gelten für Algorithmus 5.2.11 die Fehlerabschätzungen*

$$|\Delta s| \leq \mathsf{eps}\,|s| + \mathsf{eps}^2\left(\frac{3}{4}n^2 + n\right)\sum_{i=1}^{n}|x_i|,$$

$$|\Delta s| \leq \mathsf{eps}\,|s| + \mathsf{eps}^2\left(\frac{1}{4}n^3 + \frac{5}{2}n^2 + n\right)\max_{1\leq i\leq n}|x_i|.$$

Man kann also mit fehlerkompensierender Summation ähnlich gute Ergebnisse erzielen wie mit doppelt genauer Rechnung und anschließendem Runden, ein Verfahren, das etwa den doppelten Aufwand von Summation mit einfacher Genauigkeit aufweist. Der höhere (insgesamt vierfache bzw. fünffache) Aufwand zahlt sich also nur dann aus, wenn keine höhere Rechengenauigkeit auf dem Computersystem vorhanden ist, bzw. die maximale vorhandene Rechengenauigkeit zu hohe Fehler produziert

**5.3. Fehlerkontrolle, Fehlerminimierung, Fehlervermeidung.** Die Abhandlungen des Abschnitts 5.1 geben eine gute Methode, die Fortpflanzung der Fehler in einem Algorithmus genau zu untersuchen. Dieses Analyseverfahren, *Vorwärtsanalyse (forward analysis)* genannt, beschreibt ja direkt, im Rahmen gewisser Vereinfachungen, wo die Fehler auftreten und wie sie sich weiter ausbreiten. Wenn jedoch die Anzahl $r$ der Zwischenschritte zu groß wird, steigt der Aufwand für diese Herleitung in einem solchen Ausmaß an, daß diese Methode nicht mehr praktikabel ist.

**5.3.1. Rückwärtsanalyse.** Für eine gewisse Größe und manche Klassen von Algorithmen kann man statt dessen versuchen, die Gutartigkeit des Algorithmus statt wie oben durch *Rückwärtsanalyse (backward analysis)* zu beweisen.

Um diese Methode zu erklären, nehmen wir an, daß wir einen Algorithmus $\Phi$ zur Berechnung der Funktion $\varphi$ untersuchen wollen. Sei $y = \varphi(x)$ das korrekte Ergebnis zu den Eingabedaten $x$, und sei $\tilde{y} = y + \Delta y = \Phi(x)$ das mit Fehlern behaftete Resultat, das der Algorithmus berechnet. Dabei wollen wir o.B.d.A. annehmen, daß die Eingabedaten ohne

weitere Rundungsfehler im Computer repräsentierbar seien. Bei der Rückwärtsanalyse wird nun versucht zu zeigen, daß sich $\tilde{y}$ immer in der Form

$$\tilde{y} = \varphi(x + \Delta x) = \varphi(\tilde{x}) \tag{13}$$

schreiben läßt und damit als das Ergebnis einer Rechnung mit exakter Arithmetik aber mit verfälschten Eingabedaten interpretiert werden kann.

Wenn wir die Gleichung 13 weiter untersuchen, finden wir

$$\tilde{y} = y + D\varphi(x)\Delta x + O(\Delta x)^2$$

und damit

$$\Delta y \doteq D\varphi(x)\Delta x. \tag{14}$$

Ist die Funktion $\varphi$ gut konditioniert, so hat $\Delta y$ die gleiche Größenordnung wie $\Delta x$. Um also zu gewährleisten, daß $\Delta y$ nicht wesentlich größer als $\Delta^{(0)}y$ ist, muß man schließlich noch beweisen, daß

$$\Delta x \dot{\leq} K\,\mathsf{eps}\,|y|$$

mit kleiner Konstante $K$ ist.

Die Methode der Rückwärtsanalyse wurde, weil sie quasi dafür geschaffen ist, zur Untersuchung der Algorithmen der linearen Algebra mit Erfolg angewendet. Eine typische Abschätzung, die etwa in Kapitel 3 Abschnitt 4.1.5 zu finden ist:

Ein Algorithmus zur Lösung des linearen Gleichungssystemes $Ax = b$ mit symmetrischer positiv definiter Matrix $A$ ist die Cholesky–Faktorisierung, die $A = LL^*$ als „Quadrat“ einer unteren Dreiecksmatrix darstellt und dann die einfachen Gleichungssysteme $Lu = b$ und $L^*x = u$ löst. Ist $\tilde{x}$ die mit Rundungsfehlern behaftete Lösung, mit dem Algorithmus errechnet. Dann gilt

$$(A + E)\tilde{x} = b$$

mit einer Fehlermatrix $E$, die der Abschätzung

$$||E|| = K\,\mathsf{eps}\,||A||$$

genügt. $K$ hängt von der Dimension von $A$ ab, ist aber nicht sehr hoch, und damit ist die Stabilität des Algorithmus „Cholesky–Faktorisierung“ bewiesen.

Für die meisten großen nicht-linearen Probleme sind jedoch weder Vorwärts- noch Rückwärtsanalyse geeignet. Es gibt noch zwei weitere Möglichkeiten, die Rundungsfehler zu kontrollieren, die *statistische Analyse* und die *Intervallrechnung*.

**5.3.2. Statistische Fehleranalyse.** Diese Art der Fehlerabschätzung beruht auf einem häufig verwendeten Trick in der Numerischen Mathematik: Wenn ein Problem zu groß wird, um es rigoros zu behandeln, dann verwende statistische Methoden, um einiges an Komplexität aus dem Problem zu nehmen. Man erhält bei diesen Methoden allerdings nur Wahrscheinlichkeitsaussagen über die Größe der Rundungsfehler; auf exakte Ergebnisse und Abschätzungen muß man dabei verzichten. Die relativen Rundungsfehler bei den elementaren Operationen werden als *Zufallsvariable* angesehen. Dadurch werden auch die Resultate des Algorithmus und die Zwischenschritte zu Zufallsvariablen. Man trifft dabei meist die folgenden vereinfachenden Annahmen:

- Die relativen Rundungsfehler bei den elementaren Operationen sind auf dem Intervall $[-\,\mathsf{eps}, \mathsf{eps}]$ gleichverteilt und voneinander *unabhängig*.
- Bei der Berechnung der Varianzen werden nur die Terme niedrigster Ordnung in $\mathsf{eps}$ berücksichtigt.
- Alle Varianzen bleiben stets so klein, daß in erster Näherung bei allen Berechnungen für arithmetische Operationen $\circ$ gilt:

$$E(x \circ y) \doteq E(x) \circ E(y).$$

Dann verwendet man die Rechenregeln für Erwartungswerte und Varianzen, um alle diese Rundungsfehler miteinander zu kombinieren und zu einem Erwartungswert für den gesamten Rundungsfehler zu kommen und zu einer Varianz. Diese Rechnungen können im Verlauf des Algorithmus neben der wirklichen Berechnung mitgemacht werden und liefern am Ende eine Wahrscheinlichkeitsaussage über den Gesamtfehler. Eine genauere Abhandlung dieses Themas kann man in [**Stoer 1994a**, pp. 32–35] finden. Anhand eines Beispieles kann man die Grundideen jedoch erläutern:

**Beispiel 5.3.1.** *Wieder sei das Problem, $a^2 - b^2$ zu berechnen, unser Beispielfall. Wir wählen dazu Algorithmus 1. Es gilt $E(a) = a$, $E(b) = b$, $\sigma_a^2 = \sigma_b^2 = 0$. Weiters haben wir unter den obigen Annahmen*

$$\begin{gathered} E(\varepsilon_i) = 0, \quad \sigma_{\varepsilon_i}^2 = \tfrac{1}{3}\,\mathsf{eps} =: \overline{\mathsf{eps}} \\ \eta_1 = a^2(1+\varepsilon_1), \quad E(\eta_1) = a^2, \quad \sigma_{\eta_1}^2 = a^4\overline{\mathsf{eps}}^2 \\ \eta_2 = b^2(1+\varepsilon_2), \quad E(\eta_2) = b^2, \quad \sigma_{\eta_2}^2 = b^4\overline{\mathsf{eps}}^2 \\ y = (\eta_1 - \eta_2)(1+\varepsilon_3), \quad E(y) = E(\eta_1 - \eta_2)E(1+\varepsilon_3) = a^2 - b^2. \end{gathered}$$

*Für $\sigma_y^2$ berechnen wir*

$$\begin{aligned} \sigma_y^2 &= \sigma_{\eta_1-\eta_2}^2\sigma_{1+\varepsilon_3}^2 + E(\eta_1-\eta_2)^2\sigma_{\mathbb{I}+\varepsilon_3}^2 + E(1+\varepsilon_3)^2\sigma_{\eta_1-\eta_2}^2 = \\ &= (\sigma_{\eta_1}^2 + \sigma_{\eta_2}^2)\overline{\mathsf{eps}}^2 + (a^2-b^2)^2\overline{\mathsf{eps}}^2 + 1(\sigma_{\eta_1}^2 + \sigma_{\eta_2}^2) = \\ &= (a^4+b^4)\overline{\mathsf{eps}}^4 + ((a^2-b^2)^2 + a^4 + b^4)\overline{\mathsf{eps}}^2 \doteq \\ &\doteq ((a^2-b^2)^2 + a^4 + b^4)\overline{\mathsf{eps}}^2, \end{aligned}$$

*wobei wir hier bei $\doteq$ Terme der Ordnung 4 in* eps *gegenüber Termen der Ordnung zwei vernachlässigt haben.*

*Für das Zahlenbeispiel $a = 0.3237$ und $b = 0.3134$ und $\mathsf{eps} = 5 \cdot 10^{-4}$ berechnen wir die Streuung $\sigma_y \doteq 0.144\,\overline{\mathsf{eps}} = 0.415 \cdot 10^{-4}$. Das ist in der Größenordnung des wahren Fehlers $\Delta y = 0.1787 \cdot 10^{-4}$. Die Fehlerschranke, die von der Vorwärtsanalyse in Beispiel 5.2.7 berechnet wird, beläuft sich dagegen auf $0.10478 \cdot 10^{-3}$.*

**5.3.3. Intervallrechnung.** Eine weitere sehr wichtige Methode zur Verfolgung von Rundungsfehlern ist die *Intervallrechnung*. Sie ist eine Art automatisches rigoroses Fehleranalyseverfahren, das auch als essentielles Werkzeug für computerunterstütztes Beweisen dient.

Bei diesem Verfahren werden alle Zahlen und Ergebnisse durch Intervalle repräsentiert. Z.B. kann man jede reelle Zahl $z$ aus den Eingangsdaten anstatt sie zu runden durch das Intervall $[z] := [\underline{z}, \overline{z}]$ darstellen, wobei $\underline{z}$ die abgerundete und $\overline{z}$ die aufgerundete Gleitkommazahl sind. Allgemeiner kann man jede Zahl $x$, die mit einem Fehler der maximalen Größe $r$ behaftet ist, durch das Intervall $[x] = [x - r, x + r]$ darstellen. Im gesamten Algorithmus verwendet man dann anstelle von Gleitkommaarithmetik Intervallarithmetik, die wir unten kurz beleuchten werden.

Im folgenden seien $\mathrm{mid}([x]) = \frac{1}{2}(\underline{x} + \overline{x})$ und $\mathrm{rad}([x]) = \frac{1}{2}(\overline{x} - \underline{x})$ der Mittelpunkt bzw. der Radius von $[x]$.

Die Grundzüge der Intervallrechnung basieren auf folgenden Rechengesetzen:

- Seien $[a]$ und $[b]$ zwei Intervalle, und sei $\circ$ eine arithmetische Operation. Dann ist $[a] \circ [b]$ definiert als das kleinste Intervall $[c]$, für das gilt

$$[c] \supseteq \{a \circ b | a \in [a], b \in [b]\}.$$

  Z.B. gilt $[1,2] + [2,4] = [3,6]$ oder $[-1,2] * [-7,4] = [-14,8]$.

- Sei $[a]$ ein Intervall und sei $f$ eine elementare Funktion. Dann ist $f([a])$ definiert als das kleinste Intervall $[c]$, für das gilt
$$[c] \supseteq \{f(a)|a \in [a]\}.$$
Z.B. ist $[-3,4]^2 = [0,16]$.

Man beachte, daß bekannte Rechenregeln für reelle Zahlen sich nicht einfach auf Intervalle übertragen lassen. So finden wir etwa in
$$[x] - [x] \neq 0,$$
$$[a]([b] + [c]) \neq [a][b] + [a][c]$$
die Verletzung einfacher elementarer Rechengesetze.

Wird Intervallarithmetik auf einem Computer implementiert, so muß eine geeignete Rundungsmethode zusätzlich definiert werden, die *optimale Intervallrundung* oder *Außenrundung*. Die elementaren Computeroperationen für Intervalle sind dabei so definiert, daß sie mit den elementaren Intervalloperationen verträglich sind:

- Seien $[a]$ und $[b]$ zwei Intervalle und sei $\circ$ eine arithmetische Operation. Dann ist $[a] \circ [b]$ definiert als das kleinste durch Maschinenzahlen begrenzte Intervall $[c]$, für das gilt
$$[c] \supseteq \{a \circ b|a \in [a], b \in [b]\}.$$
Z.B. ist $[-0.34561, 2.5674] + [-1.1225, 0.45683] = [-1.4682, 3.0243]$ bei Rechnung auf fünf signifikante Stellen.
- Sei $[a]$ ein Intervall und sei $f$ eine elementare Funktion. Dann ist $f([a])$ definiert als das kleinste durch Maschinenzahlen begrenzte Intervall $[c]$, für das gilt
$$[c] \supseteq \{f(a)|a \in [a]\}.$$
Z.B. ist $\sin([0.24657, 1.8683]) = [0.24407, 1]$, wiederum bei Rechnung auf fünf Stellen.

Um Funktionen auszuwerten, die komplexere Ausdrücke haben, muß man daher noch ein wenig mehr an Beobachtung leisten. Hat man etwa zwei verschiedene arithmetische Ausdrücke $\Phi_1$, $\Phi_2$, die dieselbe Funktion $\varphi$ in elementaren Operationen ausdrücken, so wird im allgemeinen die Intervallauswertung von $\Phi_1$ ein anderes Ergebnis liefern als die Auswertung von $\Phi_2$. In jedem Fall stimmen jedoch die folgenden Aussagen:

- Gilt $[x] \subseteq [z]$, so gilt die Inklusionsisotonizität
$$\Phi_i([x]) \subseteq \Phi_i([z]).$$
- Weiters gilt die Wertebereichseinschließung
$$\{\varphi(x)|x \in [x]\} \subseteq \Phi_i([x]).$$
- Die naive Intervallauswertung mit Hilfe der Zerlegung in elementare Funktionen liefert Einschließungen, deren Güte linear von der Größe der Ausgangsintervalle abhängt. Es gilt
$$\operatorname{rad} \Phi_i([x]) = O(\operatorname{rad}([x])),$$
wenn alle in $\Phi_i$ auftretenden elementaren Operationen nur über Intervallen ausgewertet werden, auf denen sie differenzierbar sind.

Es ist jedoch in höchstem Maße schwierig, vorherzusagen welcher von vielen äquivalenten Ausdrücken für eine gegebene Funktion $\varphi$ die genaueste Einschließung für $\varphi([x])$ produziert.

Unter Verwendung von Intervallrechnung kann man in jedem Fall *eine* exakte Einschließung des Resultats erhalten. Andererseits erhält man bei unkritischer Anwendung meist viel zu schlechte Fehlerschranken. Man kann bei der Anpassung der Algorithmen nicht einfach alle Zahlen durch Intervalle und alle arithmetischen Operationen und elementaren Funktionen

durch ihre Intervallanaloga ersetzen. Um vernünftig funktionierenden Intervallalgorithmen zu entwickeln ist einige zusätzliche Arbeit zu investieren. Dies geht jedoch über den Rahmen dieser Vorlesung hinaus. Weiterführende Informationen kann man z.B. in [**Neumaier 1990**] und [**Hansen 1992**] finden. Einige wesentliche zusätzliche Informationen sind auch im Kapitel 18 über globale Optimierung in Teil 2 zu finden.

Zusammenfassend kann man als *Faustregel* jedoch angeben, daß man die Aufschaukelung der Rundungsfehler am besten durch Beachtung einiger einfacher Regeln reduzieren kann: Man vermeide möglichst

- Subtraktion von fast gleich großen Zahlen, da durch Auslöschung die relativen Fehler verstärkt werden.
- Division durch sehr kleine Zahlen, da absolute Fehler verstärkt werden.
- Multiplikation mit sehr großen Zahlen, da auch hier absolute Fehler verstärkt werden.

**5.4. Validierung numerischer Rechnungen.** Grundsätzlich sind numerische Berechnungen aller Art mit Unsicherheiten allen möglichen Ursprungs behaftet, und es besteht im allgemeinen keine Möglichkeit, sich Gewißheit zu verschaffen, ob die Ergebnisse der Rechnung den gestellten Genauigkeitsanforderungen entsprechen. Es ist jedoch die Pflicht des Numerischen Mathematikers, die Unsicherheiten zu minimieren.

Eine der Methoden, dieses Ziel zu erreichen, ist gewissenhafte Rechenfehleranalyse durchzuführen. Es gibt jedoch noch Fehlerquellen, an die man meist zu denken vergißt und einige Methoden, zusätzliche Information zu gewinnen, um die Unsicherheit zu reduzieren.

**Modellvalidierung:** Modelle (siehe Kapitel 2) versucht man dadurch zu validieren, daß man die Rechenergebnisse den zu modellierenden Phänomenen gegenüberstellt. Dazu sollte man, wenn möglich, die gesuchten Größen in geplanten Experimenten empirisch ermitteln. Dabei sollte der Gültigkeitsbereich des Modells überprüft (manchmal auch bestimmt) werden und festgestellt werden, für welche Datenkonstellationen sich das Modell stabil und für welche es sich instabil verhält.

Aus diesen Untersuchungen kann man Informationen erhalten, die von der Form sind: „Ist das Modell zu grob?“ „Wenn ja, in welchen Bereichen muß das Modell verfeinert werden?“ „Werden die tatsächlichen Phänomene mit der benötigten Genauigkeit wiedergegeben?“

**Sensitivitätsanalyse:** Darunter versteht man Untersuchungen, die dazu dienen herauszufinden, wie stark das Ergebnis der Rechnung von *kleinen Veränderungen* des Modells, der Eingabedaten oder der Steuerparameter des verwendeten Algorithmus abhängt. Sie wird durchgeführt, indem man z.B. mehrere Durchläufe der Berechnung ausführt, jedoch die Eingabedaten zufällig leichten Störungen unterwirft und dann die Abweichungen im berechneten Resultat bestimmt. Bei einem Modell, das nach einem Vorgang modelliert wurde, der experimentell untersucht werden kann, sollte man die Störung der Eingabeparameter entsprechend den tatsächlichen Verhältnissen wählen. Dann kann man die Streuung in den Rechenergebnissen mit der der experimentell bestimmten Ergebnisse vergleichen.

Ähnlich verhält es sich mit den Veränderung der Algorithmenparameter. Die Abhängigkeit der Ergebnisse von diesen Parametern sollte untersucht werden. Ist die Veränderung sehr groß, obwohl der Parameter nur schwach verändert worden ist, so deutet das darauf hin, daß der Algorithmus mit der Berechnung Probleme hat und für die Problemlösung möglicherweise ungeeignet ist.

**Softwarefehler:** Ein Softwarefehler liegt immer dann vor, wenn das Programm etwas anderes produziert als der Benutzer erwartet. Ein großes Problem jeder Arbeit

mit Computern, also auch bei der Durchführung numerischer Berechnungen, ist das Auftreten solcher Softwarefehler (bugs). Es gibt kaum eine Möglichkeit, ein Programm herzustellen, das nicht an irgendeiner Stelle fehlerhaft arbeitet. Die Anzahl der Fehler auf ein erträgliches Mindestmaß zu reduzieren, ist die Aufgabe des Testens.

Dabei wird die Software anhand möglichst vieler Eingaben, deren Ergebnis bekannt ist, überprüft. Alle auftretenden Fehler werden mit speziellen Fehlersuchprogrammen (debugger) genauer untersucht und entfernt. Man sollte zum Testen das Programm in kleine einfach zu überblickende Teile (Module) zerlegen und diese Teile einzeln testen. Dadurch kann man die Anzahl der übersehenen Fehler meist minimieren.

Eine eingehende Untersuchung von Programmen und deren Bugs hat ergeben, daß die Anzahl der Fehler überlinear von der Länge der Programme und sehr stark von der Übersichtlichkeit und Komplexität der Programmausdrücke abhängt. Um die Anzahl der Fehler zu minimieren, sollte also „Wurstcode“ (sehr lange Funktionen) vermieden werden. Die Programme sollten in kleine unabhängige, möglichst wiederverwendbare Teile zerlegt werden, diese sollten gut kommentiert und übersichtlich gestaltet werden. Schließlich sollte übergroße Komplexität von Ausdrücken vermieden und statt dessen mit Zwischenergebnissen gearbeitet werden.

KAPITEL 2

# Modellierung I

In diesem Kapitel behandeln wir Theorie und Praxis des Modellierens beginnend mit ein wenig Theorie gefolgt von vielen Beispielen, die auch aufzeigen sollen, in welchen Gebieten mathematische Modellierung eingesetzt wird und welche mathematischen Probleme dabei auftreten.

## 1. Modelle

Der Begriff Modell hat im Sprachgebrauch viele Bedeutungen. Daher wollen wir hier eine (etwas saloppe) Definition geben ([**Ören 1979**]). Über diese kann man zwar streiten, aber wenigstens erläutert sie den Begriff so weit, daß wenig Raum für Misverständnisse bleibt.

**Definition 1.0.1.** *Ein* Modell *ist ein künstlich geschaffenes Objekt, das wesentliche Merkmale, Beziehungen (Struktur) und Funktionen eines zu untersuchenden Objekts (Original) in vereinfachter Form wiedergibt, nachbildet und damit den Prozeß der Informationsgewinnung über dieses Objekt erleichtert.*
*Die Person, die das Modell entwickelt oder verwendet, wird mit* Modellierer *bezeichnet.*

In den meisten Fällen ist das Modell ein *formalisiertes* Abbild des Originals. Ein *mathematisches Modell* verwendet mathematische Ausdrücke, um die relevanten Eigenschaften eines Modells zu beschreiben. Andere Modellformen sind *Simulationsmodelle* (sie verwenden Computerprogramme, um das Original abzubilden), *graphische Modelle* (Balkendiagramm),...

**1.1. Original und Modell.** Die Beziehungen zwischen Original und Modell bestehen als *Analogien*. Die Vorteile des Modelles resultieren im allgemeinen aus den beim Modellierungsvorgang getroffenen Vereinfachungen. Es ist eine *Reduktion* (*Abstraktion*) des Originals. Dabei richtet sich die Auswahl der abzubildenden Merkmale nach dem Zweck, dem das Modell dienen soll. Doch ein Modell hat nicht unbedingt nur weniger Eigenschaften als das Original. Meist besitzt das Modell auch Eigenschaften, die keine Entsprechung beim Original haben. (Wenn man z.B. bei der Modellierung der Strömungen im Inneren eines Sternes den Stern in erster Näherung als kugelförmig annimmt, dann hat das Modell des Sternes einige Symmetrien, die der modellierte Stern nicht besitzt.)

Aus diesen Bemerkungen folgt, daß es für *ein* Original nicht nur ein einziges Modell gibt. Jedes dieser Modelle erfaßt immer nur Teilaspekte des Originals, und damit ist es für die Untersuchung und Beschreibung der einzelnen zu beschreibenden Teile des Originals mehr oder weniger gut geeignet. Der Modellierer muß sich aus diesem Grund immer überlegen, welche Teilaspekte gut und welche schlecht beschrieben werden, d.h. er muß den *Gültigkeitsbereich des Modelles* angeben.

**Beispiel 1.1.1.** *Es gibt mehrere verschiedene Möglichkeiten, die Bewegung eines Körpers im physikalischen Umfeld zu beschreiben. Mit zunehmender Komplexität des Modells wird der Gültigkeitsbereich immer größer. Wir werden im folgenden nur wenige Teilaspekte der physikalischen Situation beleuchten.*

**Newtonsche Mechanik:** *Sie beschreibt die Bewegung von Massenkörpern, wenn die Geschwindigkeiten klein gegenüber der Lichtgeschwindigkeit und die Massen klein gegenüber der Sonnenmasse, aber groß gegenüber der Protonenmasse bleiben.*

**Spezielle Relativitätstheorie:** *Dieses Modell beschreibt die Bewegung von Körpern auch für hohe Geschwindigkeiten. Die anderen Einschränkungen bleiben bestehen.*

**Allgemeine Relativitätstheorie:** *Wenn zusätzlich noch große Massen beschrieben werden sollen, dann muß man diese Theorie verwenden, auch wenn die Komplexität bereits sehr groß ist.*

**Quantenmechanik:** *Wenn die beteiligten Massen sehr klein sind, bzw. die Anzahl der beteiligten Teilchen sehr gering ist, muß man Quanteneffekte beachten. Dazu dient das Quantenmechanische Modell. Die Quantenmechanik kommt in zwei Komplexitätsstufen. Die* Schrödingergleichung *beschreibt kleine Massen mit niedrigen Geschwindigkeiten. Sollen relativistische Effekte berücksichtigt werden (mit Ausnahme der Gravitation), muß man kompliziertere Strukturen untersuchen, wie etwa das* Standardmodell*, das auch die spezielle Relativitätstheorie enthält.*

**was nicht modelliert ist:** *Sollen jedoch Situationen beschrieben werden, in denen sich Elementarteilchen mit hohen Geschwindigkeiten in der Nähe großer Massen mit kleiner Ausdehnung bewegen, dann müßte man eine Theorie verwenden, die sowohl allgemein relativistische als auch quantenmechanische Effekte beschreibt. Es gibt jedoch trotz großen Bemühens der theoretischen Physiker bis dato kein konsistentes Modell (Stichwort:* GUT *— grand unified theory), das diese Situation abdeckt.*

Wozu benötigt man nun Modelle? Modelle können mehreren Zielen dienen. Ein Zweck ist die *Erkenntnisgewinnung*. Das Original wird durch das Modell ersetzt mit dem Ziel, über Teilaspekte des Modells neue Erkenntnisse zu gewinnen. Ein anderer möglicher Nutzen ist die *Erkenntnisvermittlung*. Dabei dient das Modell dazu, eine kommunikative oder didaktische Funktion zu übernehmen, um Uninformierten bekannte Beziehungen näherzubringen. Ein graphisches Modell (Diagramm) dient zum Beispiel diesem Zweck. Manchmal werden Modelle sogar dazu verwendet, zeitweilig die Funktion originaler Systeme zu übernehmen. Der „Autopilot“, ein Computer–Echtzeitsystem mit speziellen Programmen, übernimmt — auf der Basis von Modellen für Aerodynamik, Triebwerksverhalten,. . . — die Steuerung eines Flugzeugs nach vorgegebenen Parametern.

Nach der Untersuchung des Modells überträgt der Modellierer schließlich die gewonnenen Erkenntnisse im Analogieschluß auf das Original. D.h. der Modellierer entnimmt dem Modell Informationen, um sich dem Original gegenüber angemessen verhalten zu können. Empirische Experimente am Original können dadurch allerdings nicht ersetzt werden. Alle gewonnenen Informationen müssen immer auf ihre Relevanz im Originalbereich getestet werden.

**Beispiel 1.1.2.** *Die verschiedenen graphisch–mathematischen Verfahren der medizinischen Datenverarbeitung (Computer–Tomographie, Sonographie,. . . ) liefern Modelle von Teilen des menschlichen Körpers. Diese Modelle sind als Diagnosehilfsmittel in der modernen Medizin zwar nicht mehr wegzudenken, doch diese Modelle enthalten immer nur bruchstückhafte Information über den Körper und daher müssen sie durch andere diagnostische Maßnahmen unterstützt werden. (Es gibt zum Beispiel Fälle, wo man in einem Computer–Tomogramm einen Gehirnschlag nicht von einem Tumor unterscheiden kann.)*

**1.2. Modellierung in der Praxis.** Um ein praxistaugliches Modell zu erstellen, muß man üblicherweise mehrere Durchläufe des in Abbildung 2.1 dargestellten Kreisprozesses absolvieren. Dieser Prozeß beginnt und endet beim oben dargestellten Gespräch mit dem Anwender. Die einzelnen Einträge in diesem Kreisprozeß sind nicht ganz konsistent in der Literatur, doch das Prinzip ist immer dasselbe: Meist ist es mit einem Modell und der Berechnung eines Ergebnisses nicht getan. Im Normalfall ist Modellierung ein sehr aufwendiger

Vorgang, der viele Gespräche, Tests und Berechnungen umfaßt. Der kleine Punkt „Modellierung“ im Kreisprozeß zerfällt selbst wieder in eine ganze Reihe wichtiger Unterpunkte, wie im Abschnitt 1.3 beschrieben.

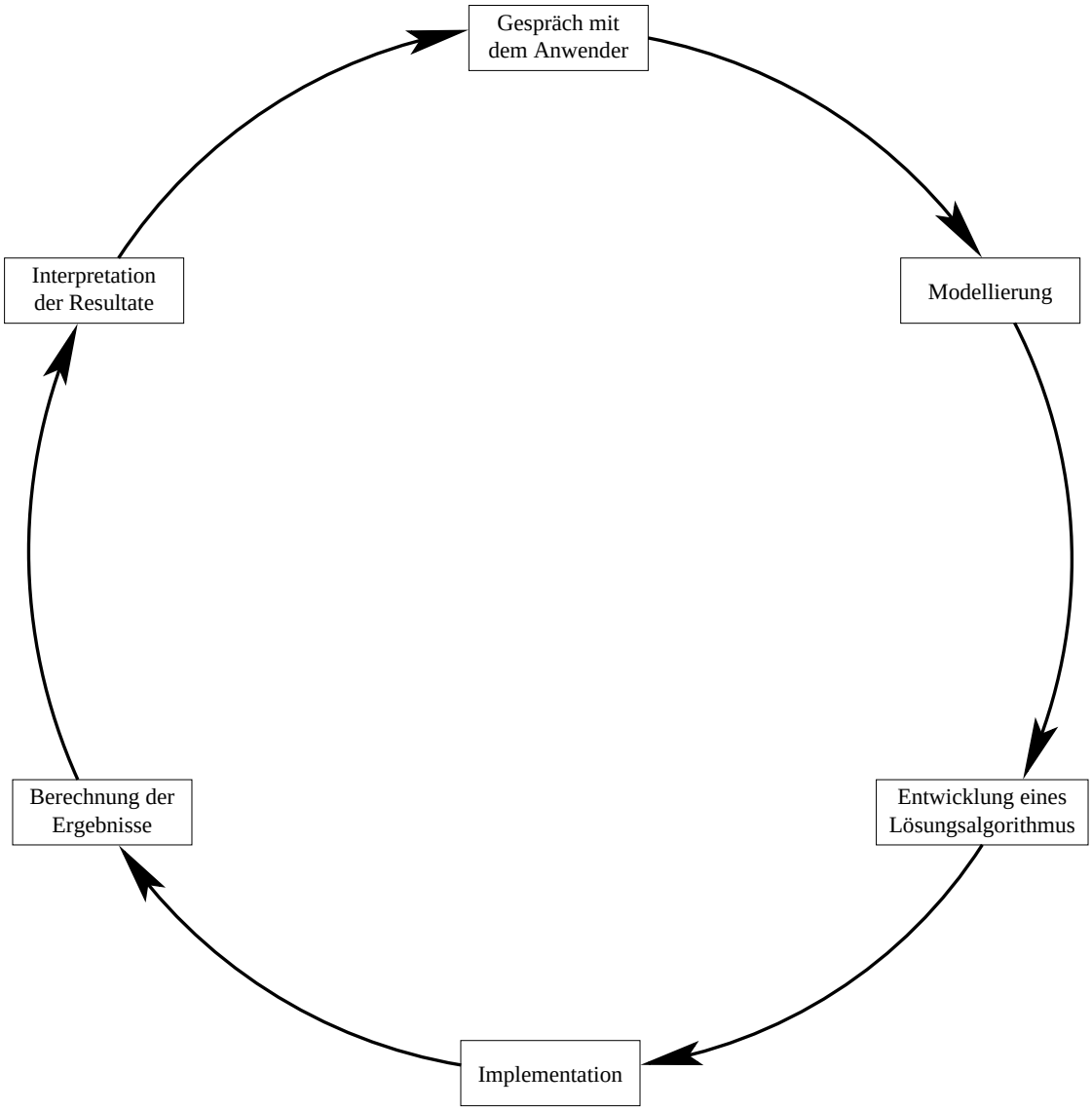



Abbildung 2.1. Modellierungskreislauf

**1.3. Modellbildung.** Um ein Modellabbild eines Originals zu erschaffen, durchläuft der Modellierer in mehr oder weniger ausgeprägter Form eine Reihe von Schritten:

**Problemspezifikation:** Der erste Schritt bei der Modellbildung dient zur Bestimmung von Ziel und Zweck der Untersuchung, die Eingrenzung des Problemes und die Festlegung der erforderlichen Genauigkeit, gegebenenfalls auch der maximal aufwendbaren Rechenzeit.

**Konzept:** Beim Erstellen des Konzeptes wird die zuvor ungegliederte Struktur des Originals begrifflich zerlegt und gegliedert. Wesentliche und unwesentliche Eigenschaften werden herausgearbeitet.

**Entwurf:** In diesem Abschnitt des Modellierungsprozesses wird der Modelltyp ausgewählt. Dabei sind sowohl methodologische Prinzipien als auch Kosten–Nutzen–Überlegungen maßgebend. Die beiden wichtigsten Kriterien sind jedoch Adäquatheit

und Einfachheit. Das bedeutet, daß das Modell die richtige *qualitative* Beschreibung des Originals, wie sie zur Lösung des gestellten Problems erforderlich ist, ermöglichen muß. Darüberhinaus muß es die richtige *quantitative* Beschreibung hinsichtlich des in der Spezifikation festgelegten Genauigkeitsgrades erlauben (Adäquatheit). Unter Einfachheit ist zu verstehen, daß „bei zwei sonst gleichwertigen Modellen jenes vorzuziehen ist, das mit weniger Annahmen und geringeren Mitteln auskommt" (Minimalitätsprinzip — bereits vor vielen Jahrhunderten von William Occam formuliert, in der englischen Literatur auch *Occam's razor* genannt).

**Festlegung der Parameterwerte:** Bei der Auswahl des Modelltyps müssen auch Anzahl und *geeignete* Werte der Modellparameter festgelegt werden. Wesentlich ist dabei, die Anzahl dieser Parameter möglichst gering zu halten, besonders wenn die Parameter aus vorhandenen Daten bestimmt werden müssen (siehe Kapitel 8), da sonst die Gefahr besteht, daß Datenfehler mitinterpoliert werden.

**Beispiel 1.3.1.** *Am relativ einfachen Problem der Beschreibung der Schwingungsdauer $T$ eines* Pendels *kann man die verschiedenen Schritte der Modellbildung und die verschiedenen Fehlerarten gut erklären. (siehe auch* [**Überhuber 1995a**, 2,p.13]*).*

*Die experimentelle Bestimmung der Schwingungsdauer eines um eine feste Achse drehbaren Pendels läßt sich leicht bewerkstelligen (in den meisten Physikpraktika ist solch ein Experiment vorhanden). Die Umkehrung dieser Fragestellung auf der anderen Seite, die Bestimmung der Pendellänge $l$ zu einer vorgegebenen Schwingungsdauer ist experimentell äußerst aufwendig. Aus diesem Grund ist es günstiger, zusätzlich zu den Untersuchungen an realen Pendeln ein mathematisches Modell zu erstellen, mit dem man die Abhängigkeit der Schwingungsdauer von den relevanten Einflußgrößen bestimmen kann.*

**Problemspezifikation:** *Zuerst muß der Begriff Schwingungsdauer spezifiziert werden: Unter Schwingungsdauer wird jene Zeit verstanden, die vom Loslassen des Pendels bis zum erstmaligen Erreichen der Maximalauslenkung auf der Seite der Ausgangslage verstreicht. Als Genauigkeitsanforderung wird z.B.*

$$|T_{ermittelt} - T_{tatsächlich}| < \varepsilon$$

*verlangt, wobei etwa $\varepsilon$ so gewählt werden könnte, daß es in der Gegend der Meßgenauigkeit liegt.*

**Konzept:** *Die Strukturanalyse eines Pendels liefert die folgenden Teile mit den relevanten physikalischen Größen: Pendelkörper (Masse, Ausdehnung, Luftwiderstand), Pendelaufhängung ≡ Stange oder Seil (Masse, Länge), Lager (Reibung), Gravitation (Fallbeschleunigung). Andere Größen können bei (fast) allen Problemen vernachlässigt werden (die Masse des Lagers, die Ausdehnung der Pendelaufhängung, die Luftbewegungen im Raum,. . . ).*

**Entwurf:** *Bei der Auswahl des Modelltyps kann man noch zusätzliche Vereinfachungen treffen:*

**Das mathematische Pendel:** *Im einfachsten Fall, der aber schon ausreicht, um die wesentlichen Eigenschaften der Pendelschwingung zu beschreiben, werden folgende zusätzliche vereinfachende Annahmen (*Idealisierungen*) getroffen:*

- *Die Schwingung verläuft* ungedämpft*: Es wird angenommen, daß keine Reibungskräfte (Luftwiderstand, Lagerreibung) auftreten.*
- *Die Aufhängung ist masselos, und der Pendelkörper hat keine Ausdehnung (d.h. die Masse ist in einem Punkt konzentriert).*

*Die Bewegungsgleichung des mathematischen Pendels ist eine gewöhnliche Differentialgleichung zweiter Ordnung*

$$\varphi''(t) = -\omega^2 \sin(\varphi(t)), \quad \textit{mit } \varphi(0) = \varphi_0,\ \varphi'(0) = 0, \tag{15}$$

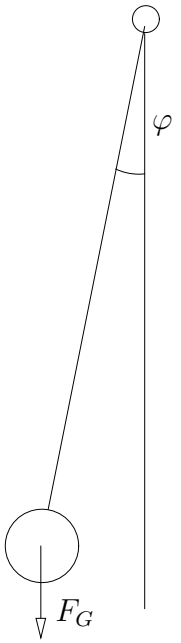



ABBILDUNG 2.2. Pendel

*wobei* $\omega^2 := g/l$ *und* $g := 9.80665ms^{-2}$ *die Fallbeschleunigung bezeichnet.* $l$ *ist die Pendellänge und* $\varphi_0$ *die Auslenkung des Pendels zum Zeitpunkt* $t = 0$ *(beim Auslassen).*

**Linearisierung:** *Nachdem nicht-lineare Differentialgleichungen schwierig zu behandeln sind, trifft man oft noch die* zusätzliche Annahme*, daß nur* kleine Auslenkungen $\varphi$ *von Bedeutung sind, d.h.* $\varphi$ *bleibt so nahe bei* 0*, daß* $\varphi \approx \sin(\varphi)$ *gilt. Man kann dann die Differentialgleichung durch ihre Linearisierung ersetzen:*

$$\varphi''(t) = -\omega^2\varphi(t). \tag{16}$$

*Der Vorteil dieser Beschreibung ist, daß die Gleichung explizit gelöst werden kann:*

$$\varphi(t) = \varphi_0 \cos \omega t.$$

*Die Lösung des nicht-linearen Problems ist dagegen nicht elementar darstellbar (ausgenommen mit den Jacobischen Elliptischen Funktionen, die aber im allgemeinen nicht zu den elementaren Funktionen gezählt werden).*

**Physikalisches Pendel:** *Im Gegensatz zum mathematischen Pendel wird beim physikalischen Pendel nicht vorausgesetzt, daß die Aufhängung masselos ist. Auch auf die Unterdrückung der Ausdehnung des Pendelkörpers wird verzichtet. Um ein mathematisches Modell für diesen Fall zu bilden, muß man das Trägheitsmoment der Stange und des Pendelkörpers in Betracht ziehen. Man kann zeigen, daß sich ein physikalisches Pendel, wenn man die Pendelaufhängung und den Pendelkörper idealisierend als starr annimmt, durch dieselbe Gleichung wie das mathematische Pendel beschreiben läßt. Man kann alle auftretenden Größe in die Pendellänge einrechnen. Diese* reduzierte Pendellänge $l^*$ *errechnet sich zu*

$$l^* = \frac{J}{ma},$$

*wo* $a$ *den Abstand des Schwerpunktes des Pendels zum Lager,* $J$ *das Massenträgheitsmoment bezogen auf den Lagerpunkt und* $m$ *die Masse des Pendels bezeichnen.*

**Komplexere Modelle:** *Wenn man auch die Reibungskräfte nicht vernachlässigen will, so führt die Modellierung dieser Kräfte zu einem weiteren Term in der Bewegungsgleichung des Pendels:*

$$\varphi''(t) = -\omega^2 \sin(\varphi(t)) + R(\varphi'(t)),$$

*mit einer reellwertigen Funktion $R$, die bei realistischer Modellierung meist nichtlinear ist. Will man auch noch auf die Starrheit von Aufhängung und Pendelkörper verzichten, so verläßt man bei der Modellierung das Gebiet der gewöhnlichen Differentialgleichungen und muß auf eine Formulierung mit nicht-linearen partiellen Differentialgleichungen ausweichen. Diese Modelle sind jedoch bereits so aufwendig, daß nur eine sehr spezialisierte Anwendung den Aufwand rechtfertigt (sonst Verstoß gegen das Einfachheitsprinzip).*

**Berechnung der Schwingungsdauer:** *Am einfachsten läßt sich $T$ für das* **Lineare Modell** *berechnen. Die Schwingungsdauer berechnet sich aus der Kreisfrequenz $\omega$ als*

$$T_L = 2\pi\sqrt{\frac{l}{g}},$$

*wie man aus der expliziten Lösung der Differentialgleichung leicht erkennen kann. Das nächst einfachere Modell, das* **mathematische Pendel***, wirft schon viel größere Probleme bei der Berechnung der Schwingungsdauer auf. Das folgende (nicht elementar lösbare) Integral — ein* vollständiges elliptisches Integral erster Gattung*) — drückt in diesem Fall die Schwingungsdauer aus:*

$$T_M = \frac{4}{\omega} E(\sin(\tfrac{\varphi_0}{2}))$$

$$E(y) = \int_0^{\pi/2} \frac{1}{\sqrt{1 - y^2 \sin^2 t}}\, dt.$$

*Dieses Integral kann man auf numerischem Weg z.B. durch die Methode von Gauß oder durch eine Doppelfolge (eine Intervallschachtelung) bestimmen:*

$$a_0 := \sqrt{1 - y^2} \quad b_0 := 1$$

$$a_{n+1} := \sqrt{a_n b_n} \quad b_{n+1} := \frac{a_n + b_n}{2}.$$

*Die Folgen $a_n$ und $b_n$ sind monoton steigend bzw. fallend und konvergieren gegen einen gemeinsamen Grenzwert $c_y$. Der Wert des elliptischen Integrals ist dann $E(y) = \frac{\pi}{2c_y}$.*

*Die Schwingungsdauer für die komplizierteren Modelle zu bestimmen ist meist schwieriger und abhängig von der genauen Modellierung, und daher hier nicht durchgeführt.*

*Ein Zahlenbeispiel für $\varphi = 0.07679448$ ($\equiv 4.4°$) und $l = 0.85m$ zeigt zum Abschluß noch die Unterschiede zwischen den beiden einfachsten Modellen:*

$$T_M \approx 1.850500016s, \quad T_L \approx 1.849817967s.$$

*Die Differenz ist $|T_M - T_L| = -6.82 \cdot 10^{-4}s$. Falls als Genauigkeit also eine Millisekunde ausreicht, genügt es das lineare Modell zu verwenden, wenn der Einfluß der anderen nicht modellierten Einflüsse ebenfalls unter dem geforderten Genauigkeitsniveau von $\varepsilon = 10^{-3}s$ bleibt.*

*Für $\varphi = \pi/4$ bleibt bei gleicher Pendellänge die Schwingungsdauer nach dem linearen Modell gleich; für das nicht-lineare Modell ergibt sich jedoch $T_M \approx 2.18415248s$. In diesem Fall ist das lineare Modell unakzeptabel selbst bei geringsten Genauigkeitsanforderungen. Das ist aber auch klar, da die Modellierungsanforderung „geringe Auslenkung“ nicht erfüllt ist — der Gültigkeitsbereich des linearen Modells wurde also verlassen.*

**1.4. Testen und Validierung.** Jedes Modell muß, bevor es für die ihm zugedachten Aufgaben eingesetzt wird, auf Validität (Gültigkeit) getestet werden. Nachdem die Gültigkeit von Modellen im allgemeinen nicht bewiesen werden kann, gibt es nur die Möglichkeit, sie zu widerlegen (zu falsifizieren). Die *Verifikation* von Modellen kann nur *indirekt* erfolgen. Man zeigt also entweder, daß eine Falsifikation nicht möglich ist (sehr selten) oder daß sie trotz intensiver Bemühung nicht gelingt.

Die folgenden Fragen müssen dabei untersucht werden: Stimmt das Modellverhalten mit dem Verhalten des Originals hinreichend genau überein? Ist die Modellstruktur zu hoch aggregiert (zu grob) und muß sie daher weiter detailliert werden? Ist die Modellstruktur zu komplex (und daher zu fein) und kann sie vergröbert werden ohne die Genauigkeitsanforderungen zu verfehlen? Wo liegen die Grenzen des Modells und wie verhält es sich in den Grenzbereichen?

Folgende Methoden werden zur Modellverifikation eingesetzt:

**Indirekte Verifikation:** Wie schon oben erwähnt, wird versucht mittels intensiven *Modelltestens* das Modell zu falsifizieren. Ist das nicht erfolgreich, so sagt man das Modell sei indirekt verifiziert.

**Statistische Verifikation:** In diesem Fall wird mit statistischen Methoden untersucht, wie zufällig die nicht vom Modell abgedeckten Datenanteile sind.

**Sensitivitätsanalyse:** Mit dieser Methode wird untersucht, wie empfindlich das Modell auf Parameter- oder Strukturveränderungen reagiert.

Sind diese Methoden gewissenhaft durchgeführt und positiv abgeschlossen, so spricht man von einem *validierten Modell* wohlwissend, daß es lediglich nicht gelungen ist, es zu falsifizieren.

**1.5. Algorithmus, Implementation, Berechnung.** Nach Beendigung der Modellierung müssen Informationen aus dem Modell gewonnen werden. Im Fall der mathematischen Modellierung müssen die Endergebnisse aus den Eingabedaten berechnet werden. Hier kommt die numerische Mathematik ins Spiel.

Zuerst muß ein *Berechnungsalgorithmus* ausgewählt, öfter neu erfunden, werden. Dieser sollte numerisch stabil sein oder zumindest die Möglichkeit bieten, brauchbare Fehlerkontrolle und -analyse zu betreiben. Nach einer Test- und Beweisphase, die vor allem die theoretischen Aspekte des Algorithmus beleuchtet, wird meist eine abgespeckte Version (*Prototyp*) erstellt, die nur eine eingeschränkte Menge von Ein- und Ausgabedaten behandelt, aber die prinzipielle Funktionstüchtigkeit zeigt.

Der Prototyp und danach der vollständige Algorithmus werden am Computer *implementiert*, und nach einer Testphase, bei der anhand vorhandener Testdaten (oft vom Prototypen erzeugt) Softwarefehler gesucht und ausgebessert werden, werden *Ergebnisse berechnet*.

**1.6. Interpretation.** Ein Schritt dessen Wichtigkeit oft unterschätzt wird ist die *Interpretation der Ergebnisse*. Wird dieser Schritt vernachlässigt, erhält der Anwender einen „Zahlenfriedhof“ mit dem er meist nichts anfangen kann. Es ist im allgemeinen so, daß Informationen, die dem Modellierer ausreichen, die Ergebnisse zu überblicken, dem Anwender nichts sagen. Um zu vermeiden, daß der Anwender die Ergebnisse ignoriert und zu dem Schluß kommt, daß Mathematik nur unnütze Antworten produziert, muß der Modellierer selbst noch die Ergebnisse analysieren und für den Anwender aufbereiten.

Zu diesem Zweck ist es notwendig, die folgenden Schritte durchzuführen:

(1) Die Ergebnisse des mathematischen Modells sind aufzutrennen in wesentliche Informationen und Zusatz- bzw. Fehlerinformation.

(2) Danach sollte man die wesentlichen Resultate wieder in Begriffe des Anwenders rückübersetzen und zum Original in Beziehung bringen.

(3) Gegebenenfalls kann man mit Hilfe graphischer Aufbereitung die Information zusätzlich strukturieren.
(4) Die Genauigkeitsinformationen bzw. die Fehlerinformation sollte dann zusammengefaßt und mit den Anforderungen verglichen werden. Eventuell könnten dann diese Ergebnisse dem Anwender als Zusatzinformationen in einem Anhang zur Verfügung gestellt werden.

Erst nach diesem Interpretationsschritt kann das Problem wirklich als gelöst betrachtet werden, wenn der Anwender sich mit dem Ergebnis zufrieden zeigt. Ist das nicht der Fall, muß mit der Modellierungsarbeit wieder von vorne begonnen werden, und der Kreisablauf beginnt von vorne.

## 2. Beispiele

Im Anschluß werden einige Anwendungsbeispiele vorgestellt, die aus den verschiedensten Bereichen stammen, um zu unterstreichen wie wichtig mathematische Modellierung im wissenschaftlichen und sozialen Umfeld ist. Diese Beispiele führen ferner auf mathematische Probleme, die mit den Methoden des ersten Teils bearbeitet werden können und für die Entwicklung eben dieser Methoden Motivierung sein sollen.

**2.1. Architektur: Brückenbau, Baustatik, Festigkeitslehre.** Werden Brücken, Häuser, U–Bahn–Stationen oder ähnliche Bauwerke errichtet, so müssen die Architekten und Bauingeneure sicher gehen, daß die errichteten Bauten die ihnen zugedachten Belastungen (plus einen ausreichenden Sicherheitsspielraum) aushalten. Zu diesem Zweck müssen die Kräfte in den einzelnen Bauteilen unter den vorgeschriebenen Belastungen bestimmt werden.

Die Baustatik beschäftigt sich mit der Modellierung von Tragkonstruktionen. Die einfachsten Konstruktionen sind die statisch bestimmten Fachwerke. Ein Beispiel für ein solches Fachwerk bietet Abb. 2.3.

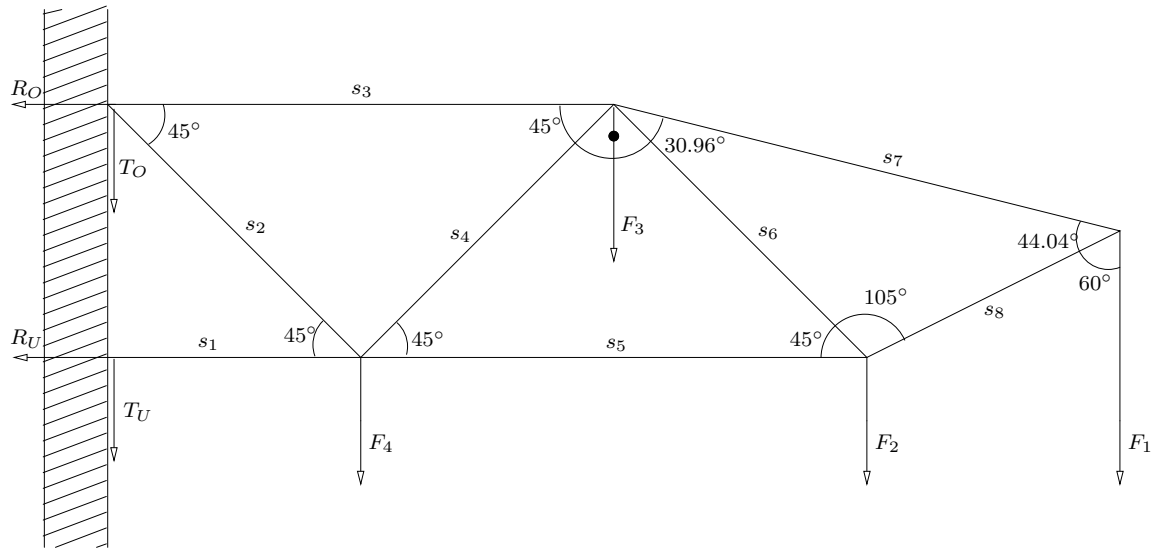



Abbildung 2.3. Statisch bestimmtes Fachwerk

In diesem Fall sind bei vorgegebenen Belastungen $F_i$, $i = 1, \ldots, 4$ sowohl die Belastungen $s_j$, $j = 1, \ldots, 8$ der einzelnen Stäbe als auch die Kräfte, die auf die Aufhängungen wirken ($R_U$, $R_O$, $T_U$, $T_O$) gesucht.

Um dieses Problem zu lösen, bespricht man sich mit dem Architekten und klärt mit ihm ab, daß

- alle Belastungen in einem Bereich bleiben sollen, in dem sich die Stäbe nicht verbiegen.
- alle Stäbe sehr lang sind im Vergleich zu ihrem Durchmesser.
- die Nahtstellen so beschaffen sind, daß die „Stäbe brechen bevor die Nähte zerstört werden“.
- die Wand viel stabiler und fester ist als die Stäbe.

Aus diesen Tatsachen ergeben sich die folgenden vereinfachenden Annahmen:

(1) Alle Verbindungsstäbe werden als ideal gerade und als unendlich dünn angenommen.
(2) Die Wand sei unendlich fest und ideal senkrecht.
(3) Die auftretenden Verfomungskräfte in den Stäben, den Aufhängungen und den Nahtstellen werden vernachlässigt.

All dies führt zu folgendem Modell, das im wesentlichen auf dem Kräfteparallelogramm (Abb. 2.4) fußt, bei dem gilt

$$|F_i| = |F_{\text{ges}}| \left| \frac{\sin \alpha_i}{\sin \beta} \right|.$$

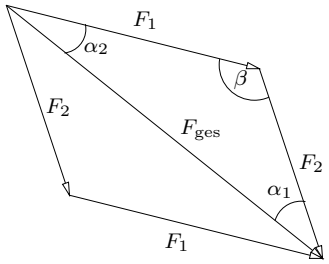



Abbildung 2.4. Kräfteparallelogramm

Eine weitere wichtige Beobachtung ist, daß sowohl die Stabkräfte als auch die Aufhängungskräfte linear von den einzelnen Belastungen $F_i$ abhängen. Es genügt also, die Einflüsse der einzelnen Belastungen getrennt zu analysieren (dabei können sie sogar auf 1 gesetzt werden) und am Ende geeignet gewichtet aufzusummieren. Die Ergebnisse können dann in Matrixnotation zusammengefaßt werden:

$$\begin{pmatrix} s_1 \\ s_2 \\ \vdots \\ s_8 \\ T_O \\ T_U \\ R_O \\ R_U \end{pmatrix} = \begin{pmatrix} E_{11} & E_{12} & E_{13} & E_{14} \\ E_{21} & E_{22} & E_{23} & E_{24} \\ \vdots & \vdots & \vdots & \vdots \\ E_{81} & E_{82} & E_{83} & E_{84} \\ E_{T_U 1} & E_{T_U 2} & E_{T_U 3} & E_{T_U 4} \\ E_{T_O 1} & E_{T_O 2} & E_{T_O 3} & E_{T_O 4} \\ E_{R_U 1} & E_{R_U 2} & E_{R_U 3} & E_{R_U 4} \\ E_{R_O 1} & E_{R_O 2} & E_{R_O 3} & E_{R_O 4} \end{pmatrix} \cdot \begin{pmatrix} F_1 \\ F_2 \\ F_3 \\ F_4 \end{pmatrix}.$$

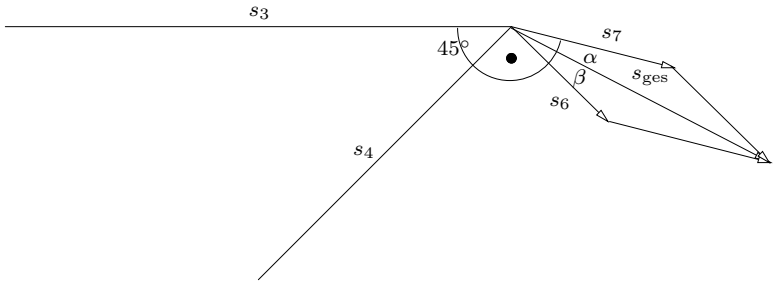
Was bleibt, ist die Elemente $E_{ij}$ der Einflußmatrix zu berechnen. Das sei an den $E_{k1}$ vorgeführt:
$F_1$ zerrt an den beiden Stäben $s_7$ und $s_8$. Dies führt nach dem Kräfteparallelogramm zu den Stabkräften

$$|s_7| = \frac{\sin 60^\circ}{\sin 44.04^\circ} \approx 1.3955$$
$$|s_8| = \frac{\sin 75.96^\circ}{\sin 44.04^\circ} \approx 1.2458,$$

wobei beide Kräfte $s_7$ und $s_8$ nach unten weisen. Der Stab $s_8$ überträgt seine Kräfte auf die Stäbe $s_5$ und $s_6$, welche beide von der Wand weg weisen und die Beträge

$$|s_5| = |s_8|\frac{\sin 105^\circ}{\sin 45^\circ} \approx 1.9063$$
$$|s_6| = |s_8|\frac{\sin 30^\circ}{\sin 45^\circ} \approx 0.9868$$

haben. Die Kraft $s_6$ wirkt zusammen mit der Kraft $s_7$ auf die vordere obere Naht und verteilt sich dort auf die Stäbe $s_3$ und $s_4$. Die Stabkraft $s_3$ wirkt nach außen, während $s_4$ nach innen und unten weist. Zuerst ist jedoch $s_{\text{ges}}$ nach dem Schema unten zu berechnen.




$$|s_{\text{ges}}| = \sqrt{|s_7|^2 + |s_6|^2 - 2|s_7||s_6|\cos(180^\circ - 30.96^\circ)} \approx 2.7948$$
$$\alpha = \text{Arcsin}(\frac{|s_6|}{|s_{\text{ges}}|}\sin 149.04^\circ) \approx 16.47^\circ$$

dann

$$|s_4| = |s_{\text{ges}}|\frac{\sin 30.50^\circ}{\sin 45^\circ} \approx 0.9823$$
$$|s_3| = |s_{\text{ges}}|\frac{\sin 104.50^\circ}{\sin 45^\circ} \approx 1.8737$$

Ähnliche Rechnungen führen auf

$$|s_1| \approx 3.5173, \quad |s_2| \approx 1.4569,$$
$$|R_O| \approx 1.0302, \quad |R_U| \approx 3.5173,$$
$$|T_O| \approx 1.0302, \quad |T_U| \approx 0.$$

Für die Einflußfaktoren in der Matrix ergibt sich dann z.B.

$$E_{1k} = \pm|s_k|.$$

Aus analogen Rechnungen, die sich übrigens leicht algorithmisieren lassen, kann man alle Einträge der Matrix bestimmen. Bei gegebenen $F_i$ wird dann die Größe der Belastungen einfach durch eine Matrix–Vektor–Multiplikation bestimmt. Einfach ist auch, bei vorgegebener Festigkeit der Stäbe und Wandanschlüsse die maximal zulässigen Belastungen $F_{i,\max}$ durch die Lösung eines linearen Gleichungssystemes zu berechnen.

Kompliziertere Konstruktionen sind meist statisch unbestimmt. Um für diese statische Untersuchungen anzustellen, benötigt man ein Modell über den inneren Aufbau der tragenden Elemente. Bei diesen Modellen werden dann Differentialgleichungen verwendet. Die Behandlung dieser komplexen Modelle geht über den Stoff des ersten Vorlesungsteiles hinaus.

**2.2. Geographie: Ozeanographie.** Eine interessante Frage in der Ozeanographie ist, inwieweit Meeresbewegungen und Strömungen (z.B. El Niño) angeregte Eigenschwingungen des Wassers sind und inwieweit ihre Bewegungen andere Ursachen hat. Zu diesem Zweck modelliert man die Wasserbewegungen mit Hilfe einer partiellen Differentialgleichung auf einem

komplexen dreidimensionalen Raum (dem Meeresbecken), die man zur Berechnung diskretisiert. Die Untersuchungen der Eigenschwingungen führt dann auf ein Eigenwertproblem für eine *sehr große* Matrix. Da allerdings nur die betragsgrößten Eigenwerte interessant sind, kann man zur Lösung ein Iterationsverfahren (etwa den Lanczos–Algorithmus) verwenden, wie in Kapitel 10 beschrieben.

**2.3. Informatik: Bildkompression.** In der modernen Multimedia-dominierten Informatik wird die Kompression von Bildinformaten eine immer wichtiger werdende Aufgabe. Unabhängig vom Ursprung ist ein Bild immer darstellbar als $m \times n$ Matrix von Vektoren positiver reeller Zahlen. Der Vektor $v_{ij}$ beschreibt z.B. die Farbe des Punktes in der $i$–ten Reihe und $j$–ten Spalte. Üblicherweise ist $v_{ij} \in \mathbb{R}^3$ und stellt den Rot–, Blau– und Grünanteil der Farbe dar — es gibt aber auch Darstellungsarten mit vier und mehr Komponenten. Im Fall von Schwarzweißbildern genügt auch ein eindimensionaler Vektor.

Zwei mögliche Aufgaben können in diesem Zusammenhang gestellt werden:

- Speichere ein Bild unter Verwendung von möglichst wenig Speicher ab ohne es zu verfälschen.
- Speichere das Bild unter Verwendung von möglichst wenig Speicher ab und verändere es dabei nur so wenig, daß ein Mensch ohne genau zu schauen keinen Unterschied erkennen kann.

Die erste Aufgabe ist ein rein informatisches Problem, das durch verschiedene Kodierungsverfahren (z.B. GIF) gelöst werden kann. Die zweite Aufgabe hingegen bietet genug Raum für mathematische Methoden (JPEG, FIF).

Auch interessant ist die Kompression von Musik oder Videos. Auch in diesen Bereichen werden mathematische Methoden (vor allem der Signalverarbeitung) eingesetzt (auch Fouriertransformation — FFT, wie in Kapitel 5, Abschnitt 3 beschrieben). Die momentan meist verwendeten Kompressionsverfahren heißen MP3 für Musik und MPEG bzw. AVI für Video.

**2.4. Informatik: Schrift–, Bild– und Spracherkennung.** Ein wichtiger Forschungsbereich in der Informatik ist die Verbesserung der Mensch–Maschine–Schnittstelle. Besonders die für viele Menschen unituitive Eingabe mit Tastatur und Maus soll durch natürliche Kommunikation ersetzt werden. (Hand)schrifterkennung und die Verarbeitung natürlicher Sprache (Spracherkennung) und Mimik (Bilderkennung) stehen dabei im Vordergrund. Es werden vor allem Methoden der Datenanalyse und Klassifikation (siehe Kapitel 8) verwendet.

**2.5. Medizin: Computertomographie.** Für die Diagnose von Krankheiten im Bereich der Weichteile des menschlichen Körpers (Gehirn, Rückenmark, Gelenkskapseln,...) ist es für den Arzt wesentlich, ein Modell des betroffenen Teils zu haben, an dem er sich Orientierung verschaffen kann. Eine Möglichkeit, sich Überblick zu verschaffen, ist, ebene Schnittbilder zu erstellen. Natürlich wäre es besser, ein dreidimensionales Modell zu gewinnen, doch das ist mit den heutigen Techniken noch kaum realisierbar.

Im Gegensatz zu Röntgenaufnahmen sind CT–Bilder keine direkten Aufnahmen (Photographien), die direkt durch Projektion von Strahlungen auf eine Photoplatte entstehen. Solche Schnittbilder entstehen als *errechnete Bilder* erst im Computer. Das Meßverfahren selbst wäre nicht geeignet, direkt Bilder zu produzieren.

Die ersten in der Praxis verwendbaren Computertomographen wurden von dem Engländer Godfrey Newbold Hounsfield und von dem US-Amerikaner südafrikanischer Herkunft Allen McLead Cormack um das Jahr 1973 gebaut. Im Jahr 1979 erhielten die beiden dafür den Nobelpreis für Medizin.

Heute gibt es viele verschiedene Varianten von Tomographen (Scannern), die anstelle von Röntgenstrahlen andere physikalische Meßprozesse verwenden, jedoch bei der Berechnung der Bilder dieselben mathematischen Methoden verwenden (Sonographen, Kernspin-Magnetresonanz-Tomographen,...).

Alle diese Verfahren beruhen auf einer mathematischen Theorie, die der Österreicher Johann Radón (1887–1956) entwickelt hat.

**Spezifikation:** Es ist ein Schaubild (Auflösung $1024 \times 1024$ Bildpunkte, 1024 Graustufen) der Dichteverteilung in einem Material (Gehirn) aus einer großen Menge an Einzelmessungen mit Hilfe von Röntgenstrahlen zu berechnen. Das Meßverfahren erfolgt in etwa gemäß dem Schaubild aus Abbildung 2.5: Es werden Röntgenstrahlen durch den auszumessenden Körperteil geschickt, wobei für jede Richtung Eingangsintensität $I_0$ und Ausgangsintensität $I$ bestimmt werden.

**Modell:** Es wird ein mathematisches Modell der Dichteverteilung im Körperteil gebildet. Die Dichteverteilung wird als Funktion

$$\varphi : \mathbb{R}^2 \to \mathbb{R}_+$$

dargestellt mit $\varphi \equiv 0$ außerhalb des Körperteils. Danach soll mit Hilfe dieser Funktion ein graphisches Modell gebildet werden, das die Dichte im Körperteil in Form $C$ unterschiedlicher Graustufen (Farben) darstellt (z.B. schwarz bedeute $\varphi = 0$, weiß bedeute $\varphi =$ maximal). Die Auflösung des Bildes soll zumindest $N \times N$ Bildpunkte betragen, wobei nur ein minimaler Teil des Bildes schwarz sein soll.

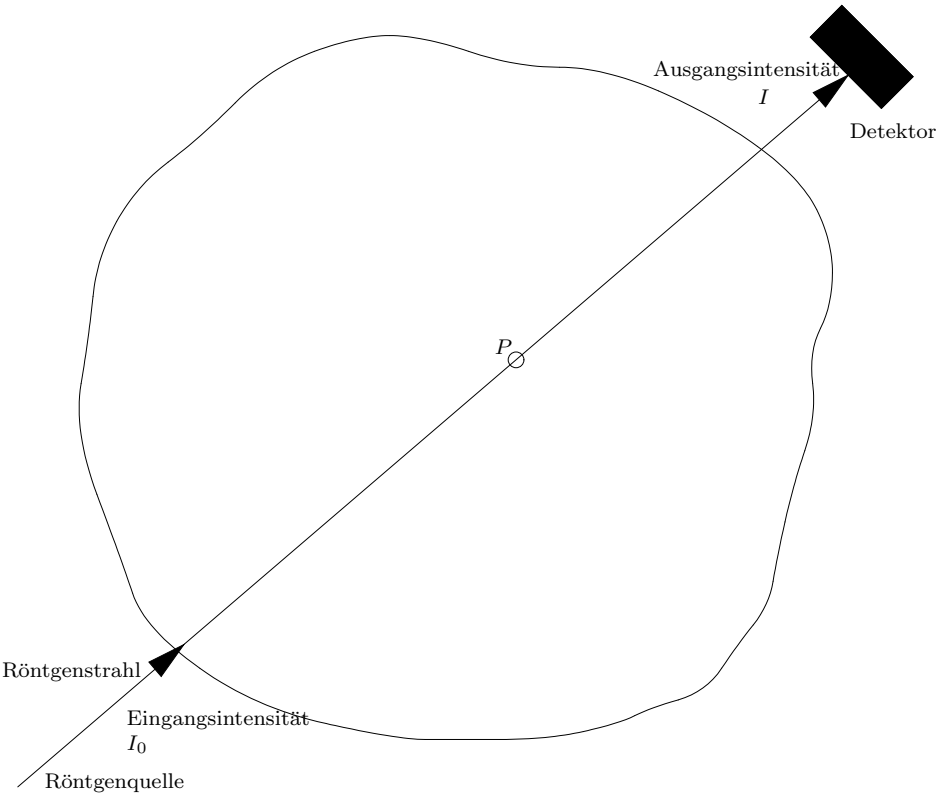



Abbildung 2.5. Meßverfahren der Computertomographie

Der Röntgenstrahl verläuft entlang einer Geraden in Richtung $y$ durch den Punkt $z$ und hat eine Eingangsintensität $I_0(y, z)$ und eine Ausgangsintensität $I(y, z)$. Die Gerade hat die Parameterdarstellung

$$g : x = z + \lambda \cdot y, \quad \lambda \in (-\infty, \infty).$$

Auf dem Weg durch den Körperteil wird der Strahl vom Gewebe teilweise absorbiert. Die Intensitätsabnahme erfolgt gemäß dem Absorptionsgesetz

$$I = I_0 \cdot e^{-\int_{-\infty}^{\infty} \varphi(z+\lambda y)\, d\lambda}.$$

Dies führt zu der Beziehung

$$-\int_{-\infty}^{\infty} \varphi(z+\lambda y)\, d\lambda = \log I(y,z) - \log I_0(y,z) =: \hat{\varphi}(y,z), \tag{17}$$

welche zu einer Integralgleichung für die unbekannte Funktion $\varphi$ führt. Ist nun $\mathcal{E}$ eine Ebene, die normal zur Strahlrichtung $y$ liegt und $z \in \mathcal{E}$. Läßt man $z$ variieren, dann kann man die Integralbeziehung als Projektion der Funktion $\varphi$ auf die Ebene $\mathcal{E} \perp y$ deuten. Diese Zuordnung, welche einer Funktion $\varphi$ die Menge aller Projektionen auf Hyperebenen zuordnet heißt *Radontransformation*. Die Aufgabe ist also, aus diesen Projektionen die Funktion $\varphi$ zu berechnen. J. Radón hat 1917 bewiesen, daß die Radontransformation umkehrbar ist, daß man also tatsächlich die Funktion $\varphi$ eindeutig aus allen diesen Integralgleichungen zurückgewinnen kann. Diese umgekehrte Radontransformation gilt es also zu berechnen.

Diese Transformation im $\mathbb{R}^2$ und $\mathbb{R}^3$ und eine Formel für ihre Umkehrung war der eigentliche Satz von J. Radón. In der modernen Variante ist die Radontransformation ein Operator

$$\hat{}: C_c^\infty(\mathbb{R}^n) \to C^\infty(\mathbb{P}^n),$$

wobei mit $\mathbb{P}^n$ der Raum aller Hyperebenen in $\mathbb{R}^n$ bezeichnet sei. Die Transformation ist dann definiert als

$$\hat{f}(\xi) := \int_\xi f(x)\, dm(x),$$

wobei mit $dm(x)$ das normale Flächenintegral bezeichnet sei. Mit den Methoden von J. Radón kann man dann eine Umkehrformel für diese Transformation beweisen.

**Entwurf:** Eine praktische Überlegung führt nun zu dem Schluß, daß man unmöglich alle Projektionen (überabzählbar viele!) bestimmen kann. Man muß sich also auf eine endliche Anzahl beschränken. Andererseits genügt es aber am Ende die Funktion als Bild graphisch zu repräsentieren mit einer Auflösung von $N \times N$ Pixel. Wir können $\varphi$ also durch eine Treppenfunktion $\hat{\varphi}$ approximieren, die konstant auf der Fläche ist, die einem Pixel $P$ entspricht, und sich dort „z.B. in weniger als einer Graustufe“ von $\varphi$ unterscheidet:

$$\left| \int_P \varphi(x) - \hat{\varphi}(x), dx \right| < \frac{\varphi_{\max}}{C}.$$

In diesem Fall kann man das gesamte Problem diskretisieren. Man legt einen Raster über den Körperteil wie in Abbildung 2.6 und nimmt an, daß die Pixel so klein sind, daß ein Strahl sie entweder in ihrer gesamten Länge durchläuft oder aber überhaupt nicht berührt.

Numeriert man Pixel von links oben nach rechts unten durch, so kann man dem Pixel $P_i$ die Dichte $\hat{\varphi}|_{P_i} \equiv: x_i$ zuweisen. Die Absorption im Bereich, der durch ein Pixel $P_k$ repräsentiert wird, berechnet sich zu

$$\Delta \log I = -x_k \ell,$$

wobei $\ell$ die Länge ist, die der Strahl im Pixel zurücklegt. Wegen der Kleinheit der Pixel und der Tatsache, daß nur Dichten relativ zu der Maximaldichte dargestellt werden, setzt man die Länge $\ell$ zu 0, wenn der Strahl den Bildpunkt nicht (oder fast nicht) trifft und zu 1, wenn der Strahl den Pixel schneidet. Das führt schließlich für jeden einzelnen Röntgenstrahl $R_j$ zu einer Gleichung der Form

$$g_j := \log I(j) - \log I_0(j) = -\sum_{k=1}^{N^2} a_{jk} x_k,$$

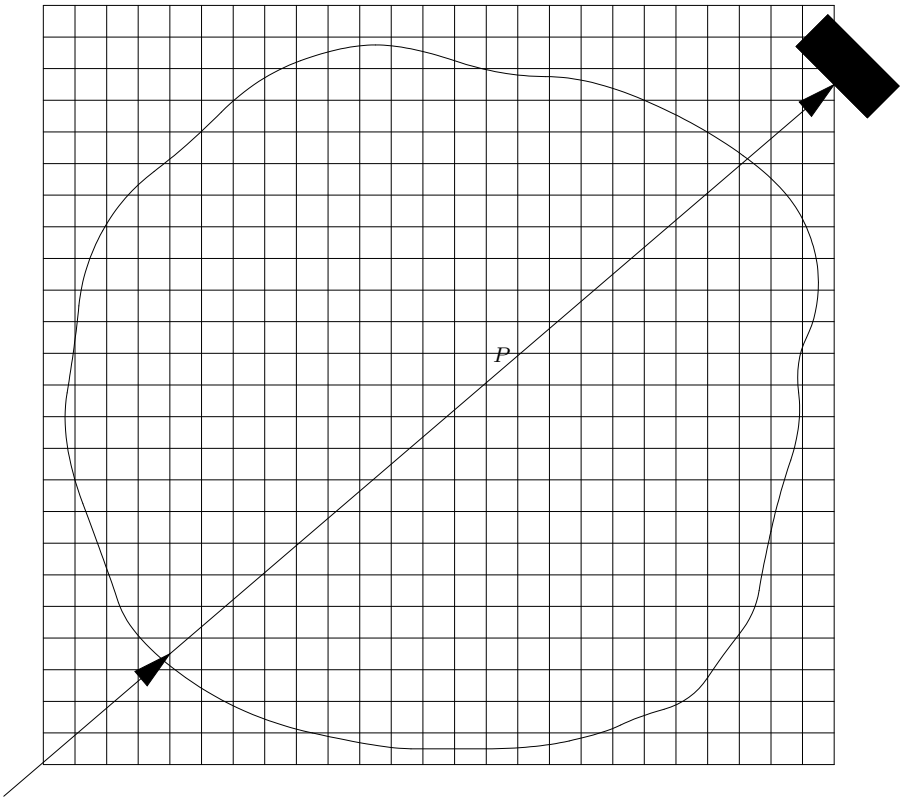



ABBILDUNG 2.6. Diskretes Modell der Computertomographie

wobei die einzelnen $a_{jk} \in \{0, 1\}$. Faßt man die $x_k$ und die $g_j$ zu Vektoren $x$ bzw. $g$ und die $a_{jk}$ zu einer Matrix $A$ zusammen, so erhält man das lineare Gleichungssystem

$$g = A \cdot x.$$

Dieses Gleichungssystem enthält $N^2$ Variablen und so viele Gleichungen $M$ wie gemessene Strahlen. Klarerweise sollte $M \geq N$ sein, damit man erwarten kann, die Werte $x_i$ zu berechnen. Eine Frage ist, sollte $M = N$ gelten (damit die Lösungsätze der linearen Algebra gelten) oder soll man $M > N$ wählen und sich etwas dazu überlegen? Es zeigt sich, daß auch aufgrund der vereinfachenden Annahmen von oben eine bessere Genauigkeit erzielt wird, wenn man $M > N$ wählt, also ein überbestimmtes Gleichungssystem bildet. Methoden, solche Gleichungssysteme zu behandeln, werden wir in Kapitel 4 kennenlernen. Bei sehr hohen Bildauflösungen, andererseits verzichtet man manchmal auch darauf $M \geq N$ zu wählen. Dann wird das System unterbestimmt, und man muß wieder nachdenken, was mit Lösung gemeint sein könnte. Auch unterbestimmte Gleichungssysteme werden in Kapitel 4 untersucht.

**Festlegung der Parameterwerte:** Die Zahl $N$ der Pixel pro Reihe und Spalte liegt bei modernen Computertomographen zwischen 512 und 1024, was zu Gleichungssystemen in ca. 250000 bis 1000000 Variablen führt. Die Anzahl der Farben $C$ liegt zwischen 256 und 16384. Die Zahl $M$ der gemessenen Strahlen ist zwischen $0.5N^2$ und $2.5N^2$.

**Beobachtungen:**

- Die Matrix $A$ kann für jeden Computertomographen ein einziges Mal vor den Messungen bestimmt werden, da von einem Steuerprogramm immer die gleichen Strahlenverläufe erzeugt werden.
- In jeder Zeile des Gleichungssystems sind nur wenige $a_{ik}$ von Null verschieden (ca. $2N$ bis $3N$ von den $N^2$ vielen). Solch eine Matrix heißt dünn besetzt. Es gibt spezielle

Methoden zur Lösung dünn besetzter linearer Gleichungssysteme (Iterationsverfahren), siehe Kapitel 10.

Für Lehramtsstudenten ist es in diesem Zusammenhang interessant, daß man einige dieser Verfahren bereits Schülern ab der 6. bis 7. Klasse nahebringen kann. Dazu empfiehlt es sich z.B. den Artikel [**Reichel, Zöchling 1990**] zu lesen.

**2.6. Tourismus: Lawinenvorhersage.** Am 23. Februar 1999 verschüttete eine Lawine Teile des tiroler Ortes Galtür, eine der größten Lawinenkatastrophen, die je in den Alpen geschehen waren. Tagelang wurde darüber diskutiert, ob das Unglück hätte verhindert werden können, ob die Lawinenkommission in fahrlässiger Weise die Gefahr unterschätzt hatte.

Diese Fragen werden wahrscheinlich niemals restlos geklärt werden können. Tatsache ist jedoch, daß die Vorhersage von Lawinen eine äußerst schwierige Sache ist. Die momentane Abschätzung der Lawinenwahrscheinlichkeit für einzelne Hänge durch die Einteilung in vier Klassen ist fehleranfällig und wird von Menschen aufgrund von Erfahrungswerten vorgenommen.

Seit zwei Jahrzehnten gibt es in der Schweiz ein Projekt zur Erstellung eines automatisierten Vorhersage- und Warnsystems, das Erfahrungswerte in mathematisch/physikalische Zusammenhänge fassen und ein Modell für Lawinen bilden soll. Nach den Untersuchungen hängen Zeit und Ort eines Lawinenabganges sowie dessen Umfang von zahllosen Faktoren ab, die zum Teil nur sehr schwer meßbar sind (z.B. gewisse Veränderungen in der Form der Schneekristalle in verschiedenen Tiefen des Schneebelages). Es ist aber auch bekannt, daß sich über ähnlichen Böden, bei ähnlichen Temperaturverläufen (über Tage und Wochen hinweg), bei ähnlichen Steilheiten und Rauhheiten der Hänge und bei ähnlichen Niederschlagsprofilen (auch über Tage hinweg), auch ähnliche Lawinenhäufigkeiten und -größen ergeben.

Man versucht also, das Problem durch einen Klassifikationsalgorithmus (siehe Kapitel 8) zu lösen, indem man die verschiedenen Temperatur-, Niederschlags- und Hangprofile in wenige Klassen einteilt, für die die Lawinenhäufigkeit ähnlich genug ist. Dann muß man nur noch für alle interessanten Hänge die Hangprofile erstellen und die Temperatur- und Niederschlagskurven mitprotokollieren, um eine Art Lawinenwarnstufe (etwas feiner unterteilt als die üblichen Stufen I–IV) auf automatisierte Weise zu erhalten. Klassifikationsalgorithmen sind z.B. ein mathematischer Weg, Erfahrungswerte zusammenzufassen und Zusammenhänge zu erkennen, die zuvor vielleicht übersehen wurden.

**2.7. Unterhaltung: 3D engine.** Bei der neuesten Generation der Computerspiele stehen hervorragende 3D–Graphikdarstellungen im Vordergrund. Die Aufgabe solch einen 3D engine zu programmieren zerfällt dabei in einige Aufgaben. Zuerst muß man die drei dimensionale virtuelle Welt des Spiels in Blickrichtung auf den zwei dimensionalen Bildschirm projizieren und dabei verdeckte Kanten löschen. Das ist ein Problem aus der linearen Algebra. Danach muß man die Oberflächen der projizierten Körper modellieren und Licht und Schatten berechnen. Zu diesem Zweck benötigt man mehrdimensionale Interpolation (meist Triangulierungen, siehe Teil 2 Kapitel 14) und wieder lineare Algebra. Der wesentliche Punkt bei den Algorithmen, die dabei verwendet werden, ist Geschwindigkeit. Der Mensch ist viel besser im Erkennen von „Rucken“ in der Erzeugung bewegter Bilder als im Erkennen von Darstellungsfehlern. Die meisten Algorithmen kommen mit Matrix–Vektor Multiplikationen aus und verwenden eine Art „vorgefertigtes Ray–Tracing“ zur Darstellung von Licht und Schatten.

Will man wirklich fotorealistische Bilder erzeugen, ist mehr an Zeit- und Rechenaufwand von Nöten (Jurassic Park,...), die Methoden zur Ausgestaltung der Bilder bleiben aber im Prinzip gleich und Beschränken sich auf mehrdimensionale Interpolation und lineare Algebra.

**2.8. Astronomie: Veränderliche Sterne.** Zur Analyse von Sternen, die im Verlauf der Zeit ihre Helligkeit verändern, verwendet man Zeitreihen (über die Zeit verteilte Meßwerte). Um Aussagen über den Grund der Helligkeitsschwankungen machen zu können, muß diese Zeitreihe genau analysiert werden. Nachdem es viele verschiedene Effekte gibt, die zu einer Helligkeitsveränderung eines Sterns beitragen können (ein Zwillings- oder Mehrlingssternsystem, Planeten, exotische Begleiter wie Neutronensterne oder Schwarze Löcher, Schwankungen im Fusionsprozeß,...), ist es wichtig, aus einer Zeitreihe möglichst viel an Information zu extrahieren und mit Modellen von Sternen und Planetensystemen zu vergleichen.

Diese Aufgabe läuft wieder auf ein Datenanalyseproblem hinaus, das mit Methoden des Kapitels 8 oder in eingeschränkterer Form mit Methoden des Kapitels 5 behandelt werden kann. Alles was über die Analyse der Zeitreihe selbst hinausgeht (Stern- und Planetensystemmodelle) ist für die Methoden des Teils 1 zu komplex und kann zum Teil erst mit Verfahren aus Teil 2 behandelt werden.

## 3. Andere Anwendungen

Andere Anwendungsbeispiele, deren Modelle auf mathematische Probleme führen, die mit den Methoden aus dem ersten Teil nicht gelöst werden können aber zum Teil mittels der Verfahren, die im zweiten Teil vorgestellt werden, behandelt werden können, werden in den folgenden Unterabschnitten kurz vorgestellt. Die Modelle werden allerdings nicht vollständig angegeben. Das wird für einige interessante Modelle im zweiten Teil getan.

**3.1. Astrophysik: Sterne.** Wie ist ein Stern aufgebaut? Wodurch entstehen Sonnenflecken und Protuberanzen? Wie entwickeln sich Sterne und was wird aus ihnen, wenn der Fusionsbrennstoff ausgeht? Fragen dieser Art werden mit Sternmodellen untersucht, die meist auf partiellen Differentialgleichungen (siehe Teil 2 Kapitel 20) und Integralgleichungen (kann man mit ähnlichen Methoden behandeln) beruhen.

**3.2. Biologie, Medizin: Epidemologie.** Wie schnell breiten sich Krankheiten aus? Wie viele Menschen werden wahrscheinlich betroffen sein? Ist ein rasches Eingreifen der Gesundheitsbehörden nötig und sinnvoll? Fragen dieser Art führen auf dynamische Systeme, die üblicherweise mit Hilfe nicht-linearer gewöhnlicher Differentialgleichungen und Differenzengleichungen modelliert werden. Gewöhnliche Differentialgleichungen werden in Teil 2 Kapitel 19 behandelt, während Differenzengleichungen weitgehend im Computer als rekursive Prozesse simuliert werden.

**3.3. Biologie und Wirtschaft: Fischpopulationen.** Auch auf gewöhnliche Differentialgleichungen und Differenzengleichungen führt die Untersuchung von Räuber–Beute–Modellen und Fischpopulationen. Leider hält sich niemand an die Resultate dieser Modelle, was mittlerweile zur Gefährdung der meisten vom industriellen Fischfang verfolgten Arten mit sich gebracht hat.

**3.4. Graphik, Textverarbeitung: Fonts.** Will man Schriften skalierbar auf dem Computer und auf Ausdrucken darstellen, ist eine Pixel-unabhängige Beschreibung der einzelnen Buchstaben und Zeichen notwendig. Das geschieht meist auf der Grundlage von Bèzier–Kurven, die in Teil 2 Kapitel 14 genauer beschrieben werden.

**3.5. Film, Graphik und Werbung: Photorealistische Darstellung.** Wie bereits weiter oben erwähnt, benötigt man mehrere Zutaten, um Bilder und Filme fotorealistisch darzustellen. Neben mehrdimensionaler Interpolation (Form) und linearer Algebra (Licht, Projektion) benötigt man auch noch gute Modelle für Oberflächen von Gegenständen (Texturen). Während die ersten beiden saubere mathematische Beschreibungen besitzen, sind alle

Modelle, die eine ausreichend schnelle Berechnung der Lichteffekte von Texturen zulassen, rein auf Erfahrungswerten aufgebaut.

**3.6. Informatik: Kryptographie.** Die Verschlüsselung von Daten ist in der Zeit von Internet, e-business und e-cash nicht mehr nur Sache der Geheimdienste sondern beinflußt das Leben jedes einzelnen immer mehr (Bankomat- und Kreditkarten!).

Gute Verschlüsselungsalgorithmen (RSA, IDEA, Diffie-Hellman) beruhen auf zahlentheoretischen Zusammenhängen und endlichen Körpern. Diese diskreten Algorithmen und Modelle werden in keinem Teil der numerischen Mathematik behandelt, führen aber mitunter zu ähnlichen Problemen und werden in der einen oder anderen Vorlesung über Zahlentheorie oder Angewandte Mathematik behandelt.

**3.7. Meteorologie: Wettervorhersage.** Wie wird das Wetter morgen? Und übermorgen? Täglich erfahren wir in den Nachrichten mehr oder weniger gute Vorhersagen für das Wetter der nächsten Tage. All das beruht auf Wettermodellen, meist mittels dynamischer Systeme auf Grundlage partieller Differentialgleichungen modelliert. Die Güte und Komplexität der Modelle hat in den letzten zwei Jahrzehnten so stark zugenommen, daß die Vorhersagegenauigkeit von knapp 70% auf über 85% gestiegen ist (die einfachste Vorhersage: „Morgen wird das Wetter so wie heute“, hat übrigens eine Trefferquote von etwa 74%).

**3.8. Meteorologie: Klimamodelle, Ozonloch, Treibhauseffekt.** Ozonloch und Treibhauseffekt führen immer noch zu Diskussionen unter den Meteorologen und Klimaforschern, obwohl sich langsam Trends abzuzeichnen beginnen. Der Grund für diese Uneinigkeit ist die enorme Komplexität des Problems, das nur mit gekoppelten partiellen Differential- und Integralgleichungen beschrieben werden kann, die Tendenz zu chaotischem Verhalten haben. Auch ist die Güte der Modelle nur schwer zu bestimmen (es gibt keine Experimente unter Laborbedingungen). Die Größe der Probleme erzwingt darüber hinaus noch mathematische Vereinfachungen, deren Auswirkungen nicht vollständig abgeschätzt werden können. Klimatologie bringt sowohl die numerische Mathematik als auch die Informatik und Computertechnologie an ihre Grenzen und darüber hinaus.

**3.9. Pharmazie: Proteinfaltung.** Die Kartographie der menschlichen Gene wurde ja zu Beginn des neuen Jahrtausends abgeschlossen, doch sie ist in ihrem heutigen Zustand etwa so brauchbar wie eine völlig verzerrte schwarz–weiß Kopie eines Straßenplanes von Wien ohne Straßennamen, oder noch weniger. Der Grund ist, daß man nur die Nukleinbasensequenz des Genoms entschlüsselt hat. Alle biologischen Aktivitäten werden aber von Proteinen gesteuert. Der Zusammenhang besteht darin, daß die Abfolge der Basen (A,C,T,G) die Abfolge der 20 verschiedenen Aminosäuren in den Proteinen (zwischen 15 und 30000 Aminosäuren lang) codiert.

Leider reicht das Wissen über die bloße Abfolge der Aminosäuren in einem Protein (die Primärstruktur) nicht aus, um biologische Effekte des Proteins abschätzen zu können. Dazu muß man die dreidimensionale Anordnung der Aminosäuren und ihrer Atome im Raum (pro Aminosäure etwa 10 Atome) kennen. Nachdem es **sehr schwierig** ist, die Struktur eines Proteins zu messen (eine darauf spezialisierte Gruppe von Chemikern schafft 1-3 Proteine im Jahr), ist es notwendig ein mathematisches Modell für Proteine zu schaffen. Diese Modelle sind meist Potentialmodelle, die das Energieniveau einer bestimmten Konfiguration beschreiben, und es ist ein noch *ungelöstes Problem*, welches Potential die beste Beschreibung bietet. Hat man ein Potential, muß man noch diejenige Konfiguration der Aminosäuren und ihrer Atome herausfinden, die das niedrigste Energieniveau aufweist. Das ist ein *globales Optimierungsproblem*, das etwa mit den Methoden aus Teil 2 Kapitel 18 behandelt werden kann.

**3.10. Physik: Gasdynamik.** Die Beschreibung des Verhaltens realer Gase führt auf hyperbolische Erhaltungssätze (spezielle partielle Differentialgleichungen), über deren theoretisches Lösungsverhalten nicht sehr viel bekannt ist. Trotzdem kann man numerische Berechnungen machen, die verwendbare Vorhersagen für den realen Fall zulassen.

**3.11. Physik: Halbleiter, Transportprobleme.** Auch die Beschreibung von Halbleiterbausteinen führt auf partielle Differentialgleichungen, sogenannte Drift–Diffusionsgleichungen, die mit numerischen Methoden analog zu Teil 2 Kapitel 20 untersucht werden können.

**3.12. Unterhaltung: Digitales Fernsehen, Videoverschlüsselung.**

**3.13. Verkehr: Crash-Tests.** Wie viel Energie wirkt bei einem Unfall auf die Insassen? Welche Teile der Karrosserie brechen? Diese Fragen werden mit Hilfe von Modellen, die aus partiellen Differentialgleichungen bestehen, untersucht.

**3.14. Verkehr: Aerodynamik, Flugzeugbau.** Die optimale Gestaltung von Flugzeugflügeln in Bezug auf Sicherheit und Energieverbrauch benötigt eine Kombination verschiedenster mathematischer Richtungen. Partielle Differentialgleichungen beschreiben die Strömungsverhältnisse am Flügel. Faktorenanalyse (Singulärwertzerlegung,...) versucht die wesentlichen Parameter (Form, Materialeigenschaften,...) festzustellen, und mittels globaler Optimierung wird die vorteilhafteste Parameterkonfiguration bestimmt.

**3.15. Wirtschaft: Buchung von Flugzeugsitzen.** Wenn ich Buchungen entgegennehme, wie viele Sitze soll ich überbuchen (doppelt buchen) damit die Wahrscheinlichkeit weniger als 2% ist, daß wirklich mehr Fluggäste auftauchen als Sitze im Flugzeug vorhanden sind? Beim heutigen Konkurrenzkampf in der Luftfahrt ist das eine der wesentlichen Fragen. Buche ich zu wenige, bleiben Sitze frei, buche ich aber zu oft zu viel, verliere ich die Reputation und es entstehen hohe Zusatzkosten (Freiflüge, Hotelrechnungen,...). Stochastische Prozesse, Differentialgleichungen, und in weiterer Folge globale stochastische Optimierungsprobleme sind die mathematischen Modelle, zu denen dieses Problem führt. Besonders letztere gehören zum Schwierigsten, was die numerische Mathematik im Moment zu bieten hat.

KAPITEL 3

# Numerische Lineare Algebra I: Lineare Gleichungssysteme

## 1. Grundlagen

Dieser Abschnitt dient der Abklärung der Schreibweisen und der grundlegenden Begriffe, die in den Abschnitten über lineare Algebra verwendet werden.

**1.1. Problemstellung.** In diesem und dem nächsten Kapitel werden Verfahren zur Untersuchung der Lösbarkeit und zur Lösung von Gleichungssystemen der Form

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ \vdots \qquad\qquad & \qquad \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n &= b_m \end{aligned} \tag{18}$$

Gleichungen dieser Art treten einerseits als Hauptprobleme in manchen Modellen auf (Computertomographie, siehe Kapitel 2 Abschnitt 2.5), andererseits aber als Nebenprobleme in einer ganzen Reihe numerischer Verfahren (Splineinterpolation, Lösung von Differentialgleichungen,...).

Mit den Methoden der linearen Algebra kann man das Problem (18) in Matrix–Vektor Schreibweise umformulieren und endet so bei der im späteren Verlauf verwendeten Form

$$A \cdot x = b, \tag{19}$$

wobei wir die einzelnen Teile zur *Koeffizientenmatrix* $A = (a_{ij}) \in \mathbb{K}^{n\times m}$, dem *Variablenvektor* $x = (x_i) \in \mathbb{K}^n$ und dem *Konstantenvektor* $b = (b_j) \in \mathbb{K}^m$ zusammenfassen. $\mathbb{K}$ sei im folgenden entweder $\mathbb{R}$ oder $\mathbb{C}$.

In diesem Kapitel werden die Probleme mit $m = n$ behandelt, also die Fälle mit quadratischer Matrix $A$. Die übrigen Fälle, die *unterbestimmten linearen Gleichungssysteme* ($m < n$) und die *überbestimmten linearen Gleichungssysteme* ($m > n$) werden im Kapitel 4 untersucht.

**1.2. Schreibweisen, Typen von Matrizen.**

**Definition 1.2.1.** *$\mathbb{K}^n$ sei der $n$-dimensionale Vektorraum über dem Körper $\mathbb{K}$, wobei $\mathbb{K}$ für $\mathbb{R}$ oder $\mathbb{C}$ stehe. Für $a \in \mathbb{C}$ bezeichne $\overline{a}$ die konjugiert komplexe Zahl und für $a \in \mathbb{R}$ gelte $\overline{a} := a$. Das* innere Produkt *zweier Vektoren aus $\mathbb{K}^n$ sei definiert als*

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

**Definition 1.2.2.** *Im folgenden sei $\mathbb{K}^{m\times n}$ als der Vektorraum der $m \times n$–Matrizen ($m$ Zeilen, $n$ Spalten) über dem Körper $\mathbb{K}$ definiert. Mit der Matrixmultiplikation bildet der Vektorraum der quadratischen Matrizen $\mathbb{K}^{n\times n}$ eine* Algebra *(d.h. $\mathbb{K}^{n\times n}$ ist mit $+$ und $\cdot$ ein Ring zusätzlich zur Vektorraumstruktur). Sei $A \in \mathbb{K}^{m\times n}$, dann schreiben wir $A_{ij} = (A)_{ij}$ für das Element von $A$, das in der $i$–ten Zeile und der $j$–ten Spalte sitzt. $i$ heiße der* Zeilenindex, *$j$ der* Spaltenindex *des Elements.*

*Mit $A^\top$ sei die* transponierte Matrix

$$(A^\top)_{ij} = A_{ji},$$

*und mit $A^*$ die* (hermitesch) konjugierte Matrix

$$(A^*)_{ij} = \overline{A_{ji}}$$

*bezeichnet.*

*Ist $A$ eine quadratische Matrix, so kann man zusätzlich die Schreibweisen $A^{-1}$ für die inverse Matrix, falls sie existiert,* $\operatorname{tr} A$ *für die* Spur *von $A$,*

$$\operatorname{tr} A := \sum_{i=1}^{n} A_{ii},$$

*und* $\det A$ *für die* Determinante *von $A$ einführen.*

Im folgenden werden wir einige grundlegende Typen von Matrizen aufzählen, die im weiteren Verlauf der Vorlesung noch gebraucht werden.

**1.2.1. Diagonalmatrix.** Eine Matrix der Form

$$D := \begin{pmatrix} d_{11} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & d_{nn} \end{pmatrix} =: \operatorname{diag}(d_{11}, \ldots, d_{nn})$$

heißt *Diagonalmatrix*. Für $D$ gilt $D_{ij} \neq 0$ für $i \neq j$. Wird eine Matrix $A$ oder ein Vektor $v$ mit $D$ multipliziert, so entspricht diese Operation einer *Skalierung*. Im Fall einer Matrix entspricht eine Multiplikation von links ($DA$) einer Zeilenskalierung und eine Multiplikation von rechts ($AD$) einer Spaltenskalierung. Z.B.

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} \cdot \operatorname{diag}(d_1, d_2, \ldots, d_n) = \begin{pmatrix} d_1 a_{11} & d_2 a_{12} & \cdots & d_n a_{1n} \\ d_1 a_{21} & d_2 a_{22} & \cdots & d_n a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ d_1 a_{m1} & d_2 a_{m2} & \cdots & d_n a_{mn} \end{pmatrix}.$$

Der Diagonalteil einer quadratischen Matrix $A$ sei definiert als

$$\operatorname{Diag}(A) := \operatorname{diag}(A_{11}, \ldots, A_{nn}).$$

Die Menge aller $n \times n$–Diagonalmatrizen bildet eine Algebra. Sie sei mit $\Sigma^n$ bezeichnet.

**1.2.2. Nullmatrix.** Die Matrix

$$\mathbb{O} := \begin{pmatrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{pmatrix}$$

heißt die *Nullmatrix*. Sie ist das neutrale Element bzgl. der Addition von Matrizen. Gleichzeitig ist sie der Nullvektor des Raumes $\mathbb{K}^{m \times n}$.

**1.2.3. Einheitsmatrix.** Das neutrale Element der Matrixmultiplikation quadratischer Matrizen ist die *Einheitsmatrix*

$$\mathbb{I} := \operatorname{diag}(1, \ldots, 1).$$

Es gilt klarerweise $(\mathbb{I})_{ii} = 1$ und $(\mathbb{I})_{ij} = 0$ für $i \neq j$. Die Spalten von $\mathbb{I}$ sind die *Einheitsvektoren* $e_i$ $(i = 1, \ldots, n)$ des Raumes $\mathbb{K}^n$. Für reguläre Matrizen gilt $A \cdot A^{-1} = A^{-1} \cdot A = \mathbb{I}$.

**1.2.4. Permutationsmatrix.** Eine quadratische Matrix $P$ wird *Permutationsmatrix* genannt, wenn in jeder ihrer Zeilen und Spalten die Zahl 1 genau einmal auftaucht und alle anderen Einträge gleich 0 sind. Der Name ist dadurch gerechtfertigt, daß die Multiplikation eines Vektors $v$ mit $P$ eine Permutation der Komponenten von $v$ bewirkt. Multiplikation einer Matrix $A$ mit $P$ von links entspricht einer Vertauschung der Zeilen; Multiplikation von rechts entspricht einer Vertauschung der Spalten.

Die Menge aller Permutationsmatrizen bildet eine Gruppe, die *Permutationsgruppe* $\mathfrak{S}^n$.

**1.2.5. Monomiale Matrix.** Eine *Monomiale Matrix* ist eine Verallgemeinerung einer Permutationsmatrix. Bei ihr ist in jeder Zeile und Spalte genau ein Element ungleich Null. Jede monomiale Matrix $M$ läßt sich schreiben als Produkt einer Permutationsmatrix $P$ und einer Diagonalmatrix $D$, $M = PD$ oder $M = DP$.

Die Menge aller monomialen Matrizen bildet eine Gruppe $\mathrm{Mon}(n)$.

**1.2.6. Rang–1 Matrix.** Eine *Rang–1 Matrix* läßt sich in der Form

$$A = xy^* := (\overline{y_1}x, \overline{y_2}x, \ldots, \overline{y_n}x)$$

mit Vektoren $x, y \neq 0$ schreiben. In einer Rang–1 Matrix sind je zwei Spalten von $A$ sind linear abhängig) schreiben.

Es gilt $\mathrm{tr}(xy^*) = \langle x, y\rangle$.

**1.2.7. Tridiagonale Matrix, Bandmatrix.** Eine quadratische Matrix der Gestalt

$$T := \begin{pmatrix} * & * & 0 & 0 & \cdots & 0 & 0 & 0 \\ * & * & * & 0 & \cdots & 0 & 0 & 0 \\ 0 & * & * & * & \cdots & 0 & 0 & 0 \\ 0 & 0 & * & * & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & * & * & 0 \\ 0 & 0 & 0 & 0 & \cdots & * & * & * \\ 0 & 0 & 0 & 0 & \cdots & 0 & * & * \end{pmatrix},$$

heißt *tridiagonale Matrix*. Es gilt $T_{ij} = 0$ für $|i - j| > 1$. Die Menge der tridiagonalen Matrizen bildet eine Algebra $\mathrm{Trid}(n)$.

Sind noch weitere Subdiagonalen ungleich Null, so bezeichnet man diese etwas allgemeinere Matrix als *Bandmatrix*. Bei einer Bandmatrix gilt also $T_{ij} = 0$ für $|i - j| \geq K$. In diesem Fall heißt $K$ die *Bandbreite* von $T$. In manchen Fällen unterscheidet man noch die *obere Bandbreite* und die *untere Bandbreite*.

**1.2.8. Dreiecksmatrix.** Zwei Haupttypen von *Dreiecksmatrizen* sind definiert:

Eine *obere Dreiecksmatrix* ist eine quadratische Matrix der Form

$$R := \begin{pmatrix} * & * & * & \cdots & * & * & * \\ 0 & * & * & \cdots & * & * & * \\ 0 & 0 & * & \cdots & * & * & * \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & * & * & * \\ 0 & 0 & 0 & \cdots & 0 & * & * \\ 0 & 0 & 0 & \cdots & 0 & 0 & * \end{pmatrix},$$

es gilt also $R_{ij} = 0$ für $i > j$.

Eine *untere Dreiecksmatrix* hat hingegen die Gestalt

$$L := \begin{pmatrix} * & 0 & 0 & \cdots & 0 & 0 & 0 \\ * & * & 0 & \cdots & 0 & 0 & 0 \\ * & * & * & \cdots & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ * & * & * & \cdots & * & 0 & 0 \\ * & * & * & \cdots & * & * & 0 \\ * & * & * & \cdots & * & * & * \end{pmatrix},$$

und daher $L_{ij} = 0$ für $i < j$.

Sowohl die oberen als auch die unteren Dreiecksmatrizen bilden Algebren, die mit $\mathcal{R}(n)$ bzw. $\mathcal{L}(n)$ bezeichnet werden.

Eine spezielle Teilklasse der Dreiecksmatrizen bilden die *normierten Dreiecksmatrizen*, die $L_{ii} = 1$ bzw. $R_{ii} = 1$ erfüllen. Die Mengen dieser Matrizen bilden Gruppen, für welche die Notationen $\mathcal{R}_{\text{norm}}(n)$ und $\mathcal{L}_{\text{norm}}(n)$ eingeführt werden.

**1.2.9. Hessenbergmatrix.** Kreuzungen zwischen Tridiagonal- und Dreiecksmatrizen sind die *Hessenbergmatrizen*, die besonders bei der Berechnung von Eigenwerten (Kapitel 9) eine große Rolle spielen. Es gibt obere und untere Hessenbergmatrizen, und die oberen Varianten haben die Gestalt

$$H := \begin{pmatrix} * & * & * & \cdots & * & * & * \\ * & * & * & \cdots & * & * & * \\ 0 & * & * & \cdots & * & * & * \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & * & * & * \\ 0 & 0 & 0 & \cdots & * & * & * \\ 0 & 0 & 0 & \cdots & 0 & * & * \end{pmatrix}.$$

Es sind also obere Dreiecksmatrizen mit einer zusätzlichen Nebendiagonale unterhalb der Hauptdiagonale.

**1.2.10. Unitäre Matrix, Orthogonale Matrix.** Eine *unitäre Matrix* ist eine Matrix $A \in \mathbb{C}^{n\times n}$ charakterisiert durch die Gleichung

$$A^*A = AA^* = \mathbb{I}.$$

Alle solche Matrizen sind klarerweise invertierbar, und es gilt $|\det A| = 1$. Die unitären $n \times n$ Matrizen bilden eine Gruppe bezüglich der Matrixmultiplikation, die unitäre Gruppe $U(n)$. Fordert man zusätzlich zur charakterisierenden Gleichung noch $\det A = 1$, so erhält man die Menge der *speziellen unitären Matrizen*, die eine Untergruppe von $U(n)$ bildet und mit $SU(n)$ bezeichnet wird.

Ist $A \in \mathbb{R}^{n\times n}$, so gilt $A^* = A^\top$, und eine Matrix, die

$$A^\top A = AA^\top = \mathbb{I}$$

erfüllt, heißt *orthogonale Matrix*. Wieder gilt $|\det A| = 1$ und die Matrizen bilden wieder eine Gruppe, die orthogonale Gruppe $O(n, \mathbb{K})$. Die Teilmenge der *speziellen orthogonalen Matrizen*, der orthogonalen Matrizen mit $\det A = 1$, bildet wieder eine Untergruppe, die spezielle orthogonale Gruppe $SO(n, \mathbb{K})$.

**1.2.11. Spiegelungsmatrix.** Die Matrix

$$S := \mathbb{I} - \frac{2}{\langle z, z\rangle} zz^*$$

definiert eine Spiegelung an der Hyperebene durch 0 senkrecht zu $z \in \mathbb{K}^n \setminus \{0\}$. Wegen $S^2 = \mathbb{I}$ und $S^* = S$ ist $S$ eine unitäre Matrix. Ein Satz der Geometrie besagt, daß sich jede unitäre Matrix als Produkt von *Spiegelungsmatrizen* schreiben läßt.

**1.2.12. Hermitesche Matrix, Symmetrische Matrix.** Gilt für eine Matrix $A \in \mathbb{K}^{n\times n}$ die Gleichung

$$A^* = A,$$

so wird $A$ *hermitesche Matrix* genannt. Gilt stattdessen die Gleichung

$$A^\top = A$$

oder ist speziell $A \in \mathbb{R}^{n\times n}$, so nennt man $A$ eine *symmetrische Matrix*. Die Mengen der hermiteschen und der symmetrischen Matrizen bilden Vektorräume $\mathrm{Sym}(n)$ und $\mathrm{Herm}(n)$. Für eine hermitesche (reelle symmetrische) Matrix gilt

$$\langle Ax, y\rangle = \langle x, Ay\rangle.$$

**1.2.13. Positiv (Negativ) (semi–)definite Matrix.** Eine Matrix $A$ heißt *positiv definit*, falls

$$x^*Ax \in \mathbb{R} \quad \text{und} \quad x^*Ax > 0$$

für alle $x \neq 0$ gilt. Insbesondere gilt $Ax \neq 0$, also ist $A$ regulär. $A$ heißt *positiv semidefinit*, falls

$$x^*Ax \in \mathbb{R} \quad \text{und} \quad x^*Ax \geq 0$$

für alle $x$ gilt.

*Negativ definit* und *negativ semidefinit* seien analog definiert mit $>$ und $\geq$ durch $<$ bzw. $\leq$ ersetzt.

Matrizen, die weder positiv noch negativ semidefinit sind, heißen *indefinit*.

**1.2.14. Dünnbesetzte Matrix.** Eine letzte Klasse von Matrizen kann nicht so eindeutig definiert werden wie die vorhergehenden Klassen. Trotzdem ist sie sehr wichtig für numerische Anwendungen. Wenn eine Matrix so aufgebaut ist, daß ein großer Teil der Einträge gleich 0 ist, dann spricht man von einer *dünnbesetzten Matrix*. Matrizen dieser Art kann man sparsamer abspeichern, und es existieren spezielle Algorithmen für Matrixmultiplikation, Eigenwerte,..., die weniger Rechenoperationen verbrauchen, indem alle Operationen, in denen 0 auftritt vereinfacht bzw. weggelassen werden.

**1.3. Normen.** Im Gegensatz zu $\mathbb{R}$ gibt es in $\mathbb{C}$ und in Vektorräumen keine natürliche Ordnungsrelation. Um also die „Größe“ eines Vektors bestimmen zu können, muß ein neuer Begriff eingeführt werden.

**Definition 1.3.1.** *Sei $V$ ein Vektorraum. Eine Abbildung $\|\quad\| : V \to \mathbb{R}^0_+$ heißt* Norm, *wenn sie die folgenden drei Eigenschaften hat:*

**N1:** $\|x\| > 0$ *für* $x \neq 0$,
**N2:** $\|\alpha x\| = |\alpha|\|x\|$ *für* $\alpha \in \mathbb{K}$,
**N3:** $\|x + y\| \leq \|x\| + \|y\|$, *die* Dreiecksungleichung.

Am $\mathbb{K}^n$ gibt es eine ganze Reihe von Normen. Die wichtigsten sind

**Euklidische Norm:**

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

**Maximumsnorm:**

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

**Summennorm:**

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

Alle diese Normen gehören zur Familie der Höldernormen, die definiert ist für $1 \leq p < \infty$:

$$\|x\|_p := \sqrt[p]{\sum_{i=1}^{n} |x_i|^p}.$$

Die Maximumsnorm entsteht durch Grenzübergang für $p \to \infty$.

Jede Norm definiert durch die Vorschrift

$$d(x, y) := \|x - y\|$$

eine *Metrik* auf dem zugrunde liegenden Vektorraum $V$, also eine Abbildung $d : V \times V \to \mathbb{R}_0^+$ mit den Eigenschaften

**M1:** $d(x, x) = 0$ für alle $x \in V$,
**M2:** $d(x, y) = 0 \implies x = y$,
**M3:** $d(x, y) = d(y, x)$,
**M4:** $d(x, y) + d(y, z) \geq d(x, z)$ für alle $x, y, z \in V$, die *Dreiecksungleichung.*

Die Menge

$$D_\varepsilon := \{x \in V | \quad \|x\| < \varepsilon\}$$

heißt die zur Norm gehörende offene *$\varepsilon$–Kugel.* Die 1–Kugel wird auch als *Einheitskugel* bezeichnet. Die Menge

$$B_\varepsilon := \{x \in V | \quad \|x\| \leq \varepsilon\}$$

heißt die zur Norm gehörende abgeschlossene *$\varepsilon$–Kugel.* Für die 2–Norm, die $\infty$–Norm und die 1–Norm auf $\mathbb{K}^n$ sehen die Einheitskugeln wie Kugeln, bzw. Würfel, bzw. auf der Spitze stehende Würfel aus.

Eine Norm $\| \quad \|_M$ auf dem Vektorraum $\mathbb{K}^{m\times n}$ der linearen Abbildungen vom normierten Raum $(\mathbb{K}^n, \| \quad \|_1)$ in den normierten Raum $(\mathbb{K}^m, \| \quad \|_2)$ heißt *Operatornorm*, falls zu den Normeigenschaften noch

**N4':** $\|Ax\|_2 \leq \|A\|_M \|x\|_1$

für alle $A \in \mathbb{K}^{m\times n}$ und alle $x \in \mathbb{K}^n$ gilt.

Eine Norm auf der Algebra $\mathbb{K}^{n\times n}$ heißt *Algebranorm*, wenn sie zusätzlich zu den Eigenschaften **N1**, **N2** und **N3** noch

**N4:** $\|A \cdot B\| \leq \|A\|\|B\|$ für $A, B \in \mathbb{K}^{n\times n}$ erfüllt.

Betrachtet man die Elemente von $\mathbb{K}^{m\times n}$ als lineare Abbildungen $A : (\mathbb{K}^n, \| \quad \|_x) \to (\mathbb{K}^m, \| \quad \|_y)$, so kann man *die zu den beiden Normen gehörige Operatornorm* (die zugehörige Matrixnorm) definieren als

$$\|A\|_M := \sup_{\|x\|_x=1} \|Ax\|_y = \sup_{x\neq 0} \frac{\|Ax\|_y}{\|x\|_x}.$$

Diese Norm ist natürlich eine Operatornorm, in einem gewissen Sinn sogar die bestmögliche.

Im Spezialfall, daß die quadratische Matrix $A$ eine Abbildung von $(\mathbb{K}^n, \| \quad \|)$ auf sich selbst repräsentiert, reduziert sich das zu der Definition

$$\|A\|_M := \sup_{\|x\|=1} \|Ax\| = \sup_{x\neq 0} \frac{\|Ax\|}{\|x\|}.$$

In diesem Fall zeigt ein leichtes Übungsbeispiel, daß die Operatornorm $\|A\|_M$ eine Algabranorm ist.

**Proposition 1.3.2.** *Sei $\| \quad \|$ eine Norm auf $\mathbb{K}^n$, sei $x \in \mathbb{K}^n$ und seien $A \in \mathbb{K}^{n\times n}$. Für die zu $\| \quad \|$ gehörende Matrixnorm gilt dann*

$$\|Ax\| \leq \|A\|_M \|x\|$$
$$\|\mathbb{I}\| = 1.$$

*Die zu den Höldernormen für $p = 1, 2, \infty$ gehörenden Matrixnormen sind:*

**Spektralnorm:**

$\|A\|_2 = \sup\{\sqrt{\langle Ax, Ax\rangle} | \langle x, x\rangle = 1\}$ = *Betrag des betragsgrößten Eigenwertes von $A$.*

**Zeilensummennorm:**

$$\|A\|_\infty = \max_{i=1,\ldots,n} \sum_{k=1}^{n} |A_{ik}|$$

**Spaltensummennorm:**

$$\|A\|_1 = \max_{k=1,\ldots,n} \sum_{i=1}^{n} |A_{ik}|$$

Beweis. Folgt aus den Definitionen. □

**Definition 1.3.3.** *Wird im Verlauf der Vorlesung eine Abbildung $f : \mathbb{K} \to \mathbb{K}$ auf ein Element $x \in \mathbb{K}^n$ oder eine Matrix $A \in \mathbb{K}^{n\times m}$ angewendet, so sei folgende Vereinbarung getroffen:*

$$f(x) \in \mathbb{K}^n \quad \textit{und} \quad (f(x))_i := f(x_i),$$
$$f(A) \in \mathbb{K}^{n\times m} \quad \textit{und} \quad (f(A))_{ij} := f(A_{ij});$$

*das heißt die Funktion werde komponentenweise angewendet. Dies gilt im speziellen für die Funktionen $|\quad|$ und* sgn.

*Eine ähnliche Definition gelte für Vergleichsoperationen. Sei $\prec$ eine der Vergleichsoperationen aus der Menge $\{<, >, \leq, \geq, \ll, \gg\}$. Dann gelte für $x, y \in \mathbb{K}^n$ und $A, B \in \mathbb{K}^{n\times m}$, $C \in \mathbb{K}^{m\times k}$*

$$x \prec y :\Longleftrightarrow \forall i : x_i \prec y_i,$$
$$A \prec B :\Longleftrightarrow \forall i, j : A_{ij} \prec B_{ij}.$$

**Proposition 1.3.4.** *Für $x, y \in \mathbb{K}^n$ und $A, B \in \mathbb{K}^{n\times m}$ gilt*

$$|x \pm y| \leq |x| + |y|, \qquad |Ax| \leq |A||x|,$$
$$|A \pm B| \leq |A| + |B|, \qquad |AC| \leq |A||C|.$$

Beweis. Alle Resultate folgen aus einfachen Anwendungen der Dreiecksungleichung. □

Mit Hilfe von Normen läßt sich eine hinreichende Bedingung für die Regularität einer Matrix beweisen.

**Proposition 1.3.5.** *Sei $A \in \mathbb{K}^{n\times n}$; gilt $\|\mathbb{I} - A\| \leq \alpha < 1$, so ist $A$ regulär und $\|A^{-1}\| \leq \frac{1}{1-\alpha}$.*

Beweis. Der Beweis erfolgt indirekt. Angenommen, $A$ ist nicht regulär. Dann existiert ein $0 \neq x \in \mathbb{K}^n$ mit $Ax = 0$. Dieses $x$ erfüllt

$$\|x\| = \|x - Ax\| = \|(\mathbb{I} - A)x\| \leq \|\mathbb{I} - A\|\|x\| \leq \alpha\|x\|,$$

und damit gilt $\|x\| = 0$, Widerspruch.

Die Rechnung

$$\|A^{-1}\| \leq \|A^{-1} - \mathbb{I}\| + \|\mathbb{I}\| \leq \|A^{-1}\|\|\mathbb{I} - A\| + 1 \leq \alpha\|A^{-1}\| + 1$$

zeigt, daß $\|A^{-1}\| \leq \frac{1}{1-\alpha}$. □

Mit Hilfe von Normen definiert man auch noch zwei weitere Klassen von Matrizen:

**1.3.1. H-Matrix.** Eine Matrix $A$ heißt *H-Matrix*, falls es reguläre Diagonalmatrizen $D_1$ und $D_2$ gibt mit $\|\mathbb{I} - D_1 A D_2\| \leq \alpha < 1$. Diese Matrizen, speziell die beiden folgenden Unterklassen, treten bei der Lösung elliptischer Differentialgleichungen auf.

**1.3.2. Diagonaldominante Matrix.** Diese Matrizen erfüllen das folgende Kriterium, das *schwache Zeilensummenkriterium*

$$|A_{ii}| \geq \sum_{k \neq i} |A_{ik}|, \quad \forall i.$$

Sind diese Ungleichungen strikt erfüllt, dann sind diagonaldominante Matrizen auch H-Matrizen.

**1.3.3. M-Matrix.** Eine *M-Matrix* ist definiert als H-Matrix $A$, die die Vorzeichenverteilung

$$A_{ii} \geq 0, \quad A_{ik} \leq 0, \quad i \neq k$$

aufweist und es $D_1$ und $D_2$ gibt mit $\|\mathbb{I} - D_1 A D_2\| \leq \alpha < 1$. Diese Matrizen treten speziell bei der Lösung elliptischer Differentialgleichungen auf.

**1.4. Ergebnisse über die Lösbarkeit linearer Gleichungssysteme.** Aus der linearen Algebra sind folgende Ergebnisse bekannt

**Theorem 1.4.1.** *Es sei $A \in \mathbb{K}^{n\times n}$ und $b \in \mathbb{K}^n$. Das lineare Gleichungssystem $Ax = b$ hat genau dann eine eindeutige Lösung, wenn $A$ regulär ist. Dies ist wiederum genau dann der Fall, wenn* $\det A \neq 0$, *bzw. wenn* $\operatorname{rg} A = n$. *In diesem Fall ist die Lösung gegeben durch*

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

Eine weitere Möglichkeit zur Berechnung der Lösung basiert auf der Berechnung einiger Determinanten.

**Theorem 1.4.2** (Die Cramersche Regel)**.** *Die Lösung des linearen Gleichungssystems $Ax = b$ läßt sich auch berechnen mittels der Formel*

$$x_i = \frac{\det A^{(i)}}{\det A} = \frac{1}{\det A} \operatorname{Adj}(A) b,$$

*wobei $A^{(i)}$ diejenige Matrix sei, die entsteht, wenn man die $i$-te Spalte von $A$ durch den Vektor $b$ ersetzt. Die Matrix* $\operatorname{Adj}(A)$ *ist die zu $A$ adjungierte Matrix. Sie ist definiert durch*

$$\operatorname{Adj}(A)_{jk} = (-1)^{j+k} \det S_{kj}(A),$$

*wo $S_{kj}(A)$, die $(k,j)$–Streichungsmatrix von $A$ bezeichnet. Das ist die $(n-1)\times(n-1)$ Matrix, die entsteht, wenn man in $A$ die $k$–te Zeile und die $j$–te Spalte streicht.*

Unglücklicherweise ist der Aufwand zur Berechnung einer Determinante sehr hoch (er ist $O(n!)$, wenn $n$ die Dimension der Matrix $A$ ist). Dies macht die Cramersche Regel zu einem numerisch unbrauchbaren Berechnungsverfahren. Beginnend mit dem folgenden Abschnitt werden wir Verfahren kennenlernen, deren Aufwand bedeutend geringer ist.

## 2. Direkte Verfahren

Das wohl bekannteste Verfahren zur Berechnung der Lösung eines linearen Gleichungssystemes stammt von C.F. Gauß (1777–1855) und wird deshalb Gaußscher Algorithmus genannt. Es ist das Eliminationsverfahren, das im heutigen Lehrplan auch in Schulen gelehrt wird. Um zu stabilen und gutartigen Verfahren zu kommen, ist es trotz des hohen Bekanntheitsgrades des Gaußschen Algorithmus notwendig, ihn noch einmal genau zu untersuchen.

**2.1. Der Gaußsche Algorithmus.** Beginnen wir die Untersuchungen mit den am einfachsten zu lösenden linearen Gleichungssystemen, denen mit dreieckiger Koeffizientenmatrix. Sei $A = L$ eine untere oder $A = R$ eine obere Dreiecksmatrix. Dann gilt

**Proposition 2.1.1.** *Das lineare Gleichungssystem $Ax = b$ hat genau dann eine eindeutige Lösung, wenn*

$$\forall i \in \{1, \ldots, n\} : A_{ii} \neq 0.$$

Beweis. Wegen Theorem 1.4.1 wissen wir, daß das lineare Gleichungssystem genau dann eindeutig lösbar ist, wenn $\det A \neq 0$. Nachdem $A$ eine Dreiecksmatrix ist, gilt aber

$$\det A = \prod_{i=1}^{n} A_{ii}.$$

Dieses Produkt ist aber genau dann von 0 verschieden, wenn alle Diagonalelemente von $A$ von Null verschieden sind. □

Ein Algorithmus zur Lösung solch eines „dreieckigen“ Systems ergibt sich beinahe von selbst. Sei $R$ eine obere Dreiecksmatrix, deren Diagonalelemente sämtlich von Null verschieden sind. Schreibt man das Gleichungssystem $Rx = b$ in Komponenten aus, so erhält man

$$\begin{aligned} R_{11}x_1 + R_{12}x_2 + \cdots + R_{1n}x_n &= b_1 \\ &\vdots \\ R_{ii}x_i + \cdots + R_{in}x_n &= b_i \\ &\vdots \\ R_{nn}x_n &= b_n. \end{aligned}$$

Wegen $R_{nn} \neq 0$ kann man $x_n$ aus der letzten Gleichung berechnen $x_n = b_n/R_{nn}$. Danach kann man rekursiv die jeweils vorhergehenden $x_i$ ausrechnen, da jedes der $R_{ii} \neq 0$. Diese Vorgehensweise ist im folgenden Algorithmus zusammengefaßt.

**Algorithmus 2.1.2. Lösen von Gleichungssystemen mit oberer Dreiecksmatrix**

**for** $i = n$ **to** $1$ **step** $-1$ **do**
  **if** $R_{ii} = 0$ **stop** *;* $R$ `ist singulär`
  $x_i = \dfrac{1}{R_{ii}}\left(b_i - \displaystyle\sum_{k=i+1}^{n} R_{ik}x_k\right)$
**done**

Für untere Dreiecksmatrizen $L$ ist die Vorgehensweise und damit der Algorithmus analog.

**Algorithmus 2.1.3. Lösen von Gleichungssystemen mit unterer Dreiecksmatrix**

**for** $i = 1$ **to** $n$ **do**
  **if** $L_{ii} = 0$ **stop** *;* $L$ `ist singulär`
  $x_i = \dfrac{1}{L_{ii}}\left(b_i - \displaystyle\sum_{k=i+1}^{n} L_{ik}x_k\right)$
**done**

Zählt man die Operationen, die nötig sind, um ein Gleichungssystem mit einer $n \times n$ Dreiecksmatrix zu lösen, so sieht man, daß es im $k$-ten Schritt $k-1$ Multiplikationen, $k-1$ Additionen und eine Division zur Berechnung von $x_k$ ($x_{n-k}$) sind. Das gibt bei $n$ Schritten eine Gesamtsumme von $n(n-1)/2$ Multiplikationen, $n(n-1)/2$ Additionen und $n$ Divisionen. Auf modernen Computersystemen sind alle diese Operationen in etwa gleich aufwendig, was zu einem Aufwand von $n^2$ Operationen zur Lösung eines „dreieckigen“ Gleichungssystemes

führt. (Im allgemeinen legt man sich nicht darauf fest, welche Operationen wieviel aufwendiger sind als welche anderen und berechnet auch nicht die genaue Anzahl sondern bedient sich der Landau–Symbole zur Darstellung des Aufwandes eines Algorithmus. In diesem Fall wäre das $O(n^2)$.)

Für allgemeines $A$ zielt der Gaußsche Algorithmus darauf ab, das Gleichungssystem in obere Dreiecksgestalt zu transformieren, indem man einfache lineare Transformationen anwendet, und es so zu lösen. Dies geschieht dadurch, daß man systematisch Nullen unter der Hauptdiagonale erzeugt. Dazu addiert man zu den Zeilen, in denen man 0 erzeugen will, Vielfache einer Arbeitszeile.

**Beispiel 2.1.4.** *Seien*

$$A = \begin{pmatrix} 2 & 1 & 1 & 0 \\ 4 & 3 & 3 & 1 \\ 8 & 7 & 9 & 5 \\ 6 & 7 & 9 & 8 \end{pmatrix}, \quad b = \begin{pmatrix} 1 \\ 3 \\ 2 \\ 7 \end{pmatrix}.$$

*Dann berechnen wir*

$$A' = \begin{pmatrix} 2 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 3 & 5 & 5 \\ 0 & 4 & 6 & 8 \end{pmatrix}, \quad b' = \begin{pmatrix} 1 \\ 1 \\ -2 \\ 4 \end{pmatrix},$$

*und die Gleichungssysteme $Ax = b$ und $A'x = b'$ haben dieselbe Lösung. $A'$ ist entstanden, indem geeignete Vielfache der ersten Zeile zu den anderen Zeilen addiert wurden. Für $b'$ wurden die Zeilenumformungen von $A$ auf die einzelnen Komponenten von $b$ angewendet. Diese Operation kann man in Matrixschreibweise zusammenfassen als*

$$A' = L_1 A = \begin{pmatrix} 1 & & & \\ -2 & 1 & & \\ -4 & & 1 & \\ -3 & & & 1 \end{pmatrix} \begin{pmatrix} 2 & 1 & 1 & 0 \\ 4 & 3 & 3 & 1 \\ 8 & 7 & 9 & 5 \\ 6 & 7 & 9 & 8 \end{pmatrix} = \begin{pmatrix} 2 & 1 & 1 & 0 \\ & 1 & 1 & 1 \\ & 3 & 5 & 5 \\ & 4 & 6 & 8 \end{pmatrix}, \quad \textit{und} \quad b' = L_1 b.$$

*Diese Prozedur wird fortgesetzt:*

$$L_2 L_1 A = \begin{pmatrix} 1 & & & \\ & 1 & & \\ & -3 & 1 & \\ & -4 & & 1 \end{pmatrix} \begin{pmatrix} 2 & 1 & 1 & 0 \\ & 1 & 1 & 1 \\ & 3 & 5 & 5 \\ & 4 & 6 & 8 \end{pmatrix} = \begin{pmatrix} 2 & 1 & 1 & 0 \\ & 1 & 1 & 1 \\ & & 2 & 2 \\ & & 2 & 4 \end{pmatrix},$$

*und*

$$L_3 L_2 L_1 A = \begin{pmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & -1 & 1 \end{pmatrix} \begin{pmatrix} 2 & 1 & 1 & 0 \\ & 1 & 1 & 1 \\ & & 2 & 2 \\ & & 2 & 4 \end{pmatrix} = \begin{pmatrix} 2 & 1 & 1 & 0 \\ & 1 & 1 & 1 \\ & & 2 & 2 \\ & & & 2 \end{pmatrix} := R,$$

*die gesuchte obere Dreiecksmatrix. Mit Hilfe dieser Matrixdarstellung kann man jetzt $A$ wie folgt zerlegen:*
*Zuerst beachte man, daß sich jede der unteren Dreiecksmatrizen $L_i$ von der Einheitsmatrix nur in einer Spalte unterscheidet. Matrizen dieser Form nennt man* Frobeniusmatrizen. *Die Inversen dieser Matrizen berechnet man, indem man einfach das Vorzeichen der Elemente neben der Hauptdiagonalen ändert. D.h.*

$$L_1^{-1} = \begin{pmatrix} 1 & & & \\ 2 & 1 & & \\ 4 & & 1 & \\ 3 & & & 1 \end{pmatrix}.$$

*Das Produkt der Matrizen $L_i$ errechnet man auch sehr leicht; die Produktmatrix ist eine untere Dreiecksmatrix, die als Nebendiagonaleinträge genau die Einträge der einzelnen Matrizen $L_i$ besitzt.*

$$L := L_1^{-1}L_2^{-1}L_3^{-1} == \begin{pmatrix} 1 & & & \\ 2 & 1 & & \\ 4 & 3 & 1 & \\ 3 & 4 & 1 & 1 \end{pmatrix},$$

*und zusammen mit den obigen Rechnungen folgt $A = LR$. Mit Hilfe dieser* Dreieckszerlegung *von $A$ kann man auch das lineare Gleichungssystem $Ax = b$ lösen. Da $Ax = LRx = b$ gilt, kann man mit Hilfe zweier „dreieckiger Systeme“ das Problem lösen: $Lu = b$ und danach $Rx = u$ berechnet die Lösung $x$.*

Dieses Beispiel zum Anlaß nehmend formulieren wir den naiven Gaußschen Eliminationsalgorithmus

**Algorithmus 2.1.5. Naiver Gaußscher Algorithmus**

(1) *Berechne eine Dreieckszerlegung $A = LR$,*
(2) *Löse $Lu = b$,*
(3) *Löse $Rx = u$.*

**Bemerkung 2.1.6.** *Wichtig ist zu erkennen, daß zur Lösung mehrerer linearer Gleichungssysteme mit derselben Koeffizientenmatrix $A$ Schritt 1. nicht wiederholt werden muß. Dies führt im allgemeinen zu erheblicher Rechenzeitersparnis.*

Der genaue Algorithmus zur Berechnung der Matrizen $L$ und $R$ läuft dann folgendermaßen ab:

**Algorithmus 2.1.7. LR–Zerlegung ohne Pivotierung**

$R = A$, $L = \mathbb{I}$
**for** $k = 1$ **to** $n - 1$ **do**
  **for** $j = k + 1$ **to** $n$ **do**
    $\ell_{jk} = r_{jk}/r_{kk}$
    $r_{j,k:n} = r_{j,k:n} - \ell_{jk} r_{k,k:n}$
  **done**
**done**

*wobei $r_{j,k:n}$ in MATLAB–ähnlicher Notation den Vektor*

$$r_{j,k:n} := \begin{pmatrix} r_{jk} \\ r_{j\,k+1} \\ \vdots \\ r_{jn} \end{pmatrix}$$

*bezeichne.*

Nun zur Aufwandsabschätzung: Wieviele *Flops* (Floating point operations = Gleitkommaoperationen) werden bei einer Dreieckszerlegung verbraucht? Der Hauptaufwand ist in der inneren Schleife die Vektoroperation. Sie benötigt $2(n - k + 1)$ Operationen für eine Skalarmultiplikation und eine Vektorsubtraktion. Dies wird innerhalb einer Doppelschleife für $k = 1, \ldots, n - 1$ auf die Spalten $k + 1, \ldots, n$ angewendet. D.h. der Aufwand für die $LR$–Zerlegung nach dem naiven Gaußschen Algorithmus ist $O(n^3)$ (genauer ist er asymptotisch $\approx \frac{2}{3}n^3$).

Unglücklicherweise ist der naive Gaußsche Algorithmus, der bislang präsentiert wurde unbrauchbar zur Lösung linearer Gleichungssysteme. Dies hat zwei Gründe, oder einen Grund, je nach Standpunkt. Erstens ist der Algorithmus nicht gutartig, und zweitens funktioniert er nicht für alle regulären Matrizen.

**Beispiel 2.1.8.** *Sei*

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

*Im ersten Schritt versucht der Algorithmus durch* 0 *zu dividieren und scheitert. Das liegt aber nicht nur am Algorithmus, sondern bereits an der Fragestellung. Die Matrix A besitzt nämlich keine Zerlegung in Dreiecksmatrizen der Form A = LR. (Gäbe es eine solche, so müßten sowohl* $L_{11}$ *als auch* $R_{11}$ *ungleich* 0 *sein, weil A regulär ist. Dann wäre aber* $A_{11} = L_{11}R_{11} \neq 0$*, ein Widerspruch.)*

**Beispiel 2.1.9.** *Stört man die Matrix A aus Beispiel 2.1.8 ein wenig*

$$A = \begin{pmatrix} 10^{-20} & 1 \\ 1 & 1 \end{pmatrix},$$

*so besitzt A eine Dreieckszerlegung mit*

$$L = \begin{pmatrix} 1 & 0 \\ 10^{20} & 1 \end{pmatrix} \quad R = \begin{pmatrix} 10^{-20} & 1 \\ 0 & 1 - 10^{20} \end{pmatrix}.$$

*Jetzt kommen wir jedoch in der Bereich der Probleme mit der Gleitkommaarithmetik. Angenommen* $\mathsf{eps} = 10^{-16}$*, ein üblicher Bereich bei der Verwendung von double precision Arithmetik. Dann wird die Zahl* $1 - 10^{20}$ *nicht exakt repräsentiert sondern gerundet. Nehmen wir an, daß diese gerundete Zahl genau* $-10^{20}$ *ist. Dann sind die Dreiecksmatrizen in Gleitkommaform*

$$\tilde{L} = \begin{pmatrix} 1 & 0 \\ 10^{20} & 1 \end{pmatrix} \quad \tilde{R} = \begin{pmatrix} 10^{-20} & 1 \\ 0 & -10^{20} \end{pmatrix},$$

*deren Rundungsfehler auf den ersten Blick tolerierbar erscheinen mögen. Die Matrix* $\tilde{R}$ *unterscheidet sich von R nur wenig relativ zu* $\|R\|$*. Wenn wir jedoch das Produkt* $\tilde{L}\tilde{R}$ *errechnen, wird das Problem offensichtlich:*

$$\tilde{L}\tilde{R} = \begin{pmatrix} 10^{-20} & 1 \\ 1 & 0 \end{pmatrix}.$$

*Diese Matrix ist nicht im geringsten nahe bei A, weil sich* $A_{22} = 1$ *von dem errechneten* $(\tilde{L}\tilde{R})_{22} = 0$ *doch bedeutend unterscheidet. Berechnen wir mit Hilfe dieser Dekomposition z.B. die Lösung eines linearen Gleichungssystems, vielleicht für* $b = (1, 0)^\top$*, so erhalten wir mit der Zerlegung* $\tilde{L}\tilde{R}\tilde{x} = b$ *das Ergebnis* $\tilde{x} = (0, 1)^\top$*, doch ist die korrekte Lösung* $x = (-1, 1)^\top$*. Obwohl der Algorithmus die Zerlegungsmatrizen L und R gutartig berechnet hat, hat es das Ergebnis des linearen Gleichungssystems* $Ax = b$ *nicht stabil berechnet.*

Den Ursachen dieses Versagens gehen wir im nächsten Abschnitt auf den Grund. Zusätzlich werden wir einen Algorithmus entwickeln, der die meisten der obigen Schwierigkeiten überwindet.

**2.2. Pivotverfahren.** Im letzten Abschnitt haben wir gesehen, daß der Gaußsche Eliminationsalgorithmus in seiner naiven Form nicht gutartig ist. Es hat sich jedoch gezeigt, daß durch eine Permutation der Zeilen von $A$ die Instabilität unter Kontrolle gehalten werden kann:

**Beispiel 2.2.1.** *Betrachten wir noch einmal die Matrix A und den Vektor b aus Beispiel 2.1.9. Vertauschen wir die Zeilen von A und die Komponenten von b, so erhalten wir das „neue“ aber äquivalente Gleichungssystem*

$$\begin{pmatrix} 1 & 1 \\ 10^{-20} & 1 \end{pmatrix} x = \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

*Führen wir jetzt den naiven Gaußschen Algorithmus durch, so erhalten wir (wieder mit rundungsfehlerbehafteter Rechnung) die Zerlegung $A \approx \tilde{L}\tilde{R}$ mit*

$$\tilde{L} = \begin{pmatrix} 1 & 0 \\ 10^{-20} & 1 \end{pmatrix} \quad \tilde{R} = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}.$$

*Das Produkt dieser beiden Matrizen ergibt gerundet*

$$\tilde{L}\tilde{R} \approx \begin{pmatrix} 1 & 1 \\ 10^{-20} & 1 \end{pmatrix}.$$

*Die Rundungsfehler heben sich in diesem Fall auf, und auch die Berechnung von $\tilde{x}$ aus den beiden dreieckigen Systemen ergibt $\tilde{x} = (-1, 1)^\top$, was mit dem richtigen Ergebnis übereinstimmt.*

Die günstigsten Permutationen werden durch Wahl der *Pivotelemente* bestimmt, eine Standardverbesserung des naiven Gaußschen Algorithmus, die seit den Fünfzigerjahren in Verwendung ist.

Dazu beginnen wir mit einer Beobachtung: Im Schritt $k$ des Eliminationsalgorithmus werden Vielfache der $k$-ten Zeile von den Zeilen $k+1, \ldots, n$ der Matrix $A$ subtrahiert, um Nullen im $k$-ten Eintrag dieser Zeilen zu erzeugen. Dabei spielen die Zeile $k$ die Spalte $k$ und besonders das Element $A_{kk}$ eine besondere Rolle. Dieses Element $A_{kk}$ wird mit *Pivot* bezeichnet. Von jedem Element der Teilmatrix $A_{k+1:n,k:n}$ wird das Produkt eines Eintrages in Spalte $k$ mit einem Element in Zeile $k$ subtrahiert, dividiert durch $A_{kk}$:

$$\begin{pmatrix} * & * & \cdots & * & \cdots & * & * \\ & * & \cdots & * & \cdots & * & * \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots \\ & & \cdots & \mathbf{A_{kk}} & \cdots & * & * \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots \\ & & \cdots & * & \cdots & * & * \\ & & \cdots & * & \cdots & * & * \end{pmatrix} \quad \rightarrow \quad \begin{pmatrix} * & * & \cdots & * & \cdots & * & * \\ & * & \cdots & * & \cdots & * & * \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots \\ & & \cdots & A_{kk} & \cdots & * & * \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots & \vdots \\ & & \cdots & \mathbf{0} & \cdots & * & * \\ & & \cdots & \mathbf{0} & \cdots & * & * \end{pmatrix}$$

Es gibt jedoch keinen Grund, warum gerade die $k$-te Zeile und die $k$-te Spalte für den Eliminationsschritt verwendet werden müssen. Man könnte genauso leicht Nullen in der Spalte $k$ erzeugen, indem man Vielfache irgendeiner Zeile $i$ mit $k < i \leq n$ von den anderen Zeilen subtrahiert. In diesem Fall wäre dann das Element $A_{ik}$ der Pivot. Genausogut könnte man auch in Spalte $j$ mit $k < j \leq n$ Nullen erzeugen statt in Spalte $k$. Damit würde $A_{ij}$ zum Pivot. Man kann also jedes beliebige Element der Teilmatrix $A_{k:n,k:n}$, das ungleich Null ist, als Pivotelement verwenden. Die Vertauschung von Zeilen und Spalten und die Wahl eines anderen Pivotelements als $A_{kk}$ ist das, was üblicherweise als *Pivotverfahren* bezeichnet wird.

Vom mathematischen Standpunkt ist das sehr gut, denn es löst die Probleme, daß für manche Matrizen der Algorithmus scheitert, weil er versucht durch Null zu dividieren, wenn $A_{kk} = 0$. Andererseits zeigt es sich, daß es aus Stabilitätsgründen sinnvoll ist, auch dann das Pivotverfahren zu verwenden, wenn $A_{kk} \neq 0$.

**2.2.1. Spaltenpivotsuche.** Nachdem man beim Pivotverfahren im $k$-ten Schritt die Wahl zwischen $(n-k)^2$ möglichen Pivotelementen hat, benötigt man also im Verlauf des Algorithmus $O(n^3)$ Operationen zur Auswahl der Pivotelemente, was signifikant zum Aufwand des Gaußschen Verfahrens beiträgt. Daher beschäftigt man sich in einem ersten Schritt nicht mit allen Elementen der Submatrix $A_{k:n,k:n}$ sondern nur mit denjenigen in der Spalte $k$. Man vertauscht also im Verlauf des Algorithmus nur Zeilen der zu bearbeitenden Matrix $A$. Solche Zeilenvertauschungen lassen sich nach Abschnitt 1.2.4 als Produkt mit einer Permutationsmatrix von links beschreiben.

Betrachtet man das Beispiel 2.1.9 für die Instabilität des naiven Gaußschen Algorithmus, so erkennt man, daß die Quelle für die Instabilität vom Faktor der Größenordnung $10^{20}$ in $L$ herrührt. Das bedingt nämlich, daß die Normen von $L$ und $R$ sehr groß werden, was die Rundungsfehler stark verstärkt. Man muß also während der Dreieckszerlegung darauf achten, daß die Faktoren $L$ und $R$ nicht zu groß werden (dies ist in Beispiel 2.2.1 passiert). Dazu führen wir in einem ersten Schritt folgende Verbesserung des naiven Gaußschen Algorithmus ein. Das so entstehende Verfahren wird in der Literatur $LR$–Zerlegung mit Spaltenpivotsuche (oder einfach $LR$–Zerlegung, bzw. Gaußscher Algorithmus) genannt.

Wie bereits erwähnt, kann jedes Element der Teilmatrix $A_{k:n,k:n}$ als Pivotelement verwendet werden, soferne es ungleich Null ist. Glücklicherweise ist es meist nicht nötig, alle diese Elemente zu durchsuchen; im allgemeinen reicht es völlig aus, eine der Komponenten von $A_{k,k:n}$, also des unteren Teils der $k$–ten Spalte, auszuwählen. Um die Komponenten von $L$ klein zu halten, wählt man dasjenige Element, mit maximalem Absolutbetrag.

$$\begin{pmatrix} * & * & * & * & * \\ & * & * & * & * \\ & * & * & * & * \\ & \mathbf{A}_{ik} & * & * & * \\ & * & * & * & * \end{pmatrix} \overset{P_1}{\to} \begin{pmatrix} * & * & * & * & * \\ & \mathbf{A}_{ik} & * & * & * \\ & * & * & * & * \\ & * & * & * & * \\ & * & * & * & * \end{pmatrix} \overset{L_1}{\to} \begin{pmatrix} * & * & * & * & * \\ & A_{ik} & * & * & * \\ & \mathbf{0} & * & * & * \\ & \mathbf{0} & * & * & * \\ & \mathbf{0} & * & * & * \end{pmatrix},$$

wobei $P_1$ eine Permutationsmatrix und $L_1$ eine Frobeniusmatrix ist. Der aus diesem Schema entstehende Algorithmus kann wieder als Produkt von Matrizen geschrieben werden:

$$L_{n-1}P_{n-1}\cdots L_2P_2L_1P_1A = R. \tag{20}$$

**Beispiel 2.2.2.** *Kehren wir zum Beispiel 2.1.4 zurück, um die Vorgänge zu illustrieren.*

$$A = \begin{pmatrix} 2 & 1 & 1 & 0 \\ 4 & 3 & 3 & 1 \\ 8 & 7 & 9 & 5 \\ 6 & 7 & 9 & 8 \end{pmatrix}$$

*Die Spaltenpivotsuche liefert die Acht in der dritten Reihe als größtes Element, daher werden im ersten Schritt die dritte und die erste Zeile vertauscht.*

$$\begin{pmatrix} & & 1 & \\ & 1 & & \\ 1 & & & \\ & & & 1 \end{pmatrix} \begin{pmatrix} 2 & 1 & 1 & 0 \\ 4 & 3 & 3 & 1 \\ 8 & 7 & 9 & 5 \\ 6 & 7 & 9 & 8 \end{pmatrix} = \begin{pmatrix} 8 & 7 & 9 & 5 \\ 4 & 3 & 3 & 1 \\ 2 & 1 & 1 & 0 \\ 6 & 7 & 9 & 8 \end{pmatrix}$$

*Nach der Vertauschung folgt der erste Eliminationsschritt, der genau wie im Verfahren ohne Pivotsuche funktioniert.*

$$\begin{pmatrix} 1 & & & \\ -\frac{1}{2} & 1 & & \\ -\frac{1}{4} & & 1 & \\ -\frac{3}{4} & & & 1 \end{pmatrix} \begin{pmatrix} 8 & 7 & 9 & 5 \\ 4 & 3 & 3 & 1 \\ 2 & 1 & 1 & 0 \\ 6 & 7 & 9 & 8 \end{pmatrix} = \begin{pmatrix} 8 & 7 & 9 & 5 \\ & -\frac{1}{2} & -\frac{3}{2} & -\frac{3}{2} \\ & -\frac{3}{4} & -\frac{5}{4} & -\frac{5}{4} \\ & \frac{7}{4} & \frac{9}{4} & \frac{17}{4} \end{pmatrix}$$

*Die anderen Vertauschungs- und Eliminationsschritte funktionieren analog:*

$$\begin{pmatrix} 1 & & & \\ & & & 1 \\ & & 1 & \\ & 1 & & \end{pmatrix}\begin{pmatrix} 8 & 7 & 9 & 5 \\ & -\frac{1}{2} & -\frac{3}{2} & -\frac{3}{2} \\ & -\frac{3}{4} & -\frac{5}{4} & -\frac{5}{4} \\ & \frac{7}{4} & \frac{9}{4} & \frac{17}{4} \end{pmatrix} = \begin{pmatrix} 8 & 7 & 9 & 5 \\ & \frac{7}{4} & \frac{9}{4} & \frac{17}{4} \\ & -\frac{3}{4} & -\frac{5}{4} & -\frac{5}{4} \\ & -\frac{1}{2} & -\frac{3}{2} & -\frac{3}{2} \end{pmatrix}$$

$$\begin{pmatrix} 1 & & & \\ & 1 & & \\ & \frac{3}{7} & 1 & \\ & \frac{2}{7} & & 1 \end{pmatrix}\begin{pmatrix} 8 & 7 & 9 & 5 \\ & \frac{7}{4} & \frac{9}{4} & \frac{17}{4} \\ & -\frac{3}{4} & -\frac{5}{4} & -\frac{5}{4} \\ & -\frac{1}{2} & -\frac{3}{2} & -\frac{3}{2} \end{pmatrix} = \begin{pmatrix} 8 & 7 & 9 & 5 \\ & \frac{7}{4} & \frac{9}{4} & \frac{17}{4} \\ & & -\frac{2}{7} & \frac{4}{7} \\ & & -\frac{6}{7} & -\frac{2}{7} \end{pmatrix}$$

$$\begin{pmatrix} 1 & & & \\ & 1 & & \\ & & & 1 \\ & & 1 & \end{pmatrix}\begin{pmatrix} 8 & 7 & 9 & 5 \\ & \frac{7}{4} & \frac{9}{4} & \frac{17}{4} \\ & & -\frac{2}{7} & \frac{4}{7} \\ & & -\frac{6}{7} & -\frac{2}{7} \end{pmatrix} = \begin{pmatrix} 8 & 7 & 9 & 5 \\ & \frac{7}{4} & \frac{9}{4} & \frac{17}{4} \\ & & -\frac{6}{7} & -\frac{2}{7} \\ & & -\frac{2}{7} & \frac{4}{7} \end{pmatrix}$$

$$\begin{pmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & -\frac{1}{3} & 1 \end{pmatrix}\begin{pmatrix} 8 & 7 & 9 & 5 \\ & \frac{7}{4} & \frac{9}{4} & \frac{17}{4} \\ & & -\frac{6}{7} & -\frac{2}{7} \\ & & -\frac{2}{7} & \frac{4}{7} \end{pmatrix} = \begin{pmatrix} 8 & 7 & 9 & 5 \\ & \frac{7}{4} & \frac{9}{4} & \frac{17}{4} \\ & & -\frac{6}{7} & -\frac{2}{7} \\ & & & \frac{2}{3} \end{pmatrix}.$$

Mit Hilfe dieses Schemas berechnet man allerdings keine $LR$–Faktorisierung von $A$ (glücklicherweise, denn wir haben schon zuvor gesehen, daß das nicht immer möglich ist). Tatsächlich haben wir nur eine ähnliche Zerlegung berechnet. Kehren wir zur Matrixdarstellung (20) zurück. Um etwas ähnliches wie eine Dreieckszerlegung zu erhalten, müssen wir das Produkt $L_iP_i$ genauer untersuchen. Eine kurze Rechnung zeigt, daß

$$L_{n-1}P_{n-1}\cdots L_2P_2L_1P_1 = L'_{n-1}\cdots L'_2L'_1P_{n-1}\cdots P_2P_1,$$

wenn man definiert

$$L'_i := \Big(\prod_{j=i+1}^{n-1} P_j^{-1}\Big)^{-1} L_i \Big(\prod_{j=i+1}^{n-1} P_j^{-1}\Big) = \Big(\prod_{j=i+1}^{n-1} P_j\Big)^{-1} L_i \Big(\prod_{j=i+1}^{n-1} P_j\Big).$$

Ein „glücklicher Zufall“ bedingt dabei, daß $L'_i$ wieder eine untere Dreiecksmatrix der gleichen Gestalt wie $L_i$ ist (das wird dadurch verursacht, daß $P_i^{-1} = P_i$ gilt und Rechts- und Linksmultiplikation mit derselben Permutationsmatrix eine symmetrische Vertauschung der Zeilen und Spalten vornimmt. Zuerst vertauscht die Linksmultiplikation die Zeilen, was die Einser auf der Hauptdiagonalen „in Unordnung bringt“. Die Rechtsmultiplikation vertauscht mit Hilfe von Spaltenvertauschungen genau die beiden Einser wieder auf die Hauptdiagonale zurück. Außerdem werden nur Zeilen und Spalten unterhalb der $\mathbb{I}$ in der linken oberen Ecke von $L_i$ betroffen.)

**Beispiel 2.2.3.** *Eine Rechnung als Beispiel:*

$$\begin{pmatrix} 1 & & & \\ & 1 & & \\ & & & 1 \\ & & 1 & \end{pmatrix}\begin{pmatrix} 1 & & & \\ & 1 & & \\ & \frac{3}{7} & 1 & \\ & \frac{2}{7} & & 1 \end{pmatrix}\begin{pmatrix} 1 & & & \\ & 1 & & \\ & & & 1 \\ & & 1 & \end{pmatrix} = \begin{pmatrix} 1 & & & \\ & 1 & & \\ & \frac{2}{7} & & 1 \\ & \frac{3}{7} & 1 & \end{pmatrix}\begin{pmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & & 1 \end{pmatrix} =$$

$$= \begin{pmatrix} 1 & & & \\ & 1 & & \\ & \frac{2}{7} & 1 & \\ & \frac{3}{7} & & 1 \end{pmatrix}.$$

Nachdem die unteren Dreiecksmatrizen eine Algebra und die Permutationsmatrizen eine Gruppe bilden, ist

$$L := {L'_1}^{-1} {L'_2}^{-1} \cdots {L'_{n-1}}^{-1}$$

wieder eine untere Dreiecksmatrix und

$$P := P_{n-1} \cdots P_2 P_1$$

wieder eine Permutationsmatrix. Das führt zur Darstellung

$$PA = LR,$$

der Dreieckszerlegung mit Spaltenpivotsuche. Diese Zerlegung wird in der Literatur üblicherweise auch als *die LR–Zerlegung* der Matrix $A$ bezeichnet.

Dieses Verfahren liefert auch einen konstruktiven Beweis für den folgenden Satz

**Theorem 2.2.4** ($LR$–Zerlegung)**.** *Zu jeder regulären quadratischen Matrix $A \in \mathbb{K}^{n\times n}$ gibt es eine Permutationsmatrix $P \in \mathfrak{S}^n$, eine untere Dreiecksmatrix $L \in \mathcal{L}_{norm}(n)$ und eine obere Dreiecksmatrix $R \in \mathcal{R}(n)$ mit*

$$PA = LR.$$

Beweis. Es ist nur noch zu beweisen, daß immer ein Pivotelement existiert, das verschieden von Null ist. Der Beweis erfolgt wieder indirekt. Angenommen, das wäre nicht der Fall. Dann ist im $k$-ten Schritt

$$\max_{i=k,\ldots,n} |(L_{k-1}P_{k-1}\cdots L_1P_1A)_{ik}| = 0.$$

Die Matrix $A' := L_{k-1}P_{k-1}\cdots L_1P_1A$ sieht also etwa folgendermaßen aus:

$$A' = \left(\begin{array}{cccc|cccc} * & * & \cdots & * & * & * & \cdots & * \\ 0 & * & \cdots & * & * & * & \cdots & * \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & * & * & * & \cdots & * \\ \hline 0 & 0 & \cdots & 0 & \mathbf{0} & * & \cdots & * \\ 0 & 0 & \cdots & 0 & \mathbf{0} & * & \cdots & * \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & \mathbf{0} & * & \cdots & * \end{array}\right) =: \left(\begin{array}{c|c} B & C \\ \hline \mathbb{O} & D \end{array}\right)$$

$A'$ ist eine Blockdreiecksmatrix, und es gilt $\det A' = \det B \det D$. Nachdem aber die erste Spalte von $D$ der Nullvektor ist, gilt $\det D = 0$ und damit $\det A' = 0$. Weil aber nun alle $L_i$ und $P_i$ invertierbar sind, gilt auch $\det A = 0$, was im Widerspruch zur Regularität von $A$ steht. □

Was bleibt, ist den Algorithmus genau aufzuschreiben:

**Algorithmus 2.2.5.** *LR–Zerlegung mit Spaltenpivotsuche*

$R = A$, $L = \mathbb{I}$, $P = \mathbb{I}$
**for** $k = 1$ **to** $n-1$ **do**
  *Wähle $i \geq k$ so, daß $|r_{ik}| = \max_{j=k,\ldots,n} |r_{jk}|$*
  $r_{k,k:n} \leftrightarrow r_{i,k:n}$ `; Vertausche die beiden Zeilen`
  $\ell_{k,1:k-1} \leftrightarrow \ell_{i,1:k-1}$
  $p_{k,1:n} \leftrightarrow p_{i,1:n}$
  **for** $j = k+1$ **to** $n$ **do**
    $\ell_{jk} = r_{jk}/r_{kk}$
    $r_{j,k:n} = r_{j,k:n} - \ell_{jk} r_{k,k:n}$
  **done**
**done**

Der Rechenaufwand dieses Algorithmus ist derselbe wie der für die Gaußelimination ohne Pivotsuche $O(n^3)$, da die $O(n^2)$ Vergleichsoperationen, die für die Suche nach den Pivotelementen nötig sind, nicht ins Gewicht fallen. An Speicherbedarf fällt zusätzlich zu $A$ und $b$ noch die Matrix $P$ an, die aber nicht im Matrixformat abgespeichert werden muß. Da $P$ eine Permutation $\sigma$ beschreibt, kann $P$ auch als Vektor $p$ abgespeichert werden mit

$$p_k := \sigma(k).$$

Der zusätzliche Speicherbedarf ist also $n$ ganze Zahlen.

Was noch übrig bleibt ist, zu untersuchen, was sich für die Lösung von Gleichungssystemen ergibt, wenn man die $LR$–Zerlegung der Matrix mit Spaltenpivotsuche verwendet. Man erhält

$$Ax = b \iff PAx = Pb \iff LRx = Pb.$$

Man muß also nur den permutierten Vektor $Pb$ berechnen bevor man mit dem Vorwärts- und Rückwärtseinsetzen beginnt.

**Algorithmus 2.2.6.** *Lösung eines linearen Gleichungssystemes mit Spaltenpivotsuche*

(1) *Berechne $PA = LR$, die $LR$–Faktorisierung von $A$.*
(2) *Löse $Lu = Pb$*
(3) *Löse $Rx = u$.*

Der Rechenaufwand beträgt wieder $O(n^3)$ Operationen.

**2.2.2. Totalpivotsuche.** Wählt man nicht das betragsgrößte Element der $k$–ten Spalte als Pivotelement sondern das betragsgrößte Element der gesamten Teilmatrix $A_{k:n,k:n}$, so erhält man das Verfahren der Totalpivotsuche (vollständige Pivotsuche). In der Praxis wird dieses Verfahren nur sehr selten angewendet, da der sehr geringe Zuwachs an Stabilität den hohen Aufwand nicht rechtfertigt; denn zur Auswahl der Pivotelemente sind $O(n^3)$ Vergleichsoperationen notwendig, was wesentlich zum Gesamtaufwand des Algorithmus beiträgt.

Der Vollständigkeit halber soll die Mathematik, die hinter dem Verfahren steckt, jedoch kurz beschrieben werden. In jedem Schritt der Totalpivotsuche werden nicht nur Zeilen sondern auch Spalten von $A$ vertauscht, was sich als Matrixprodukt mit zusätzlichen Permutationsmatrizen $Q_i$ darstellen läßt:

$$L_{n-1}P_{n-1}\cdots L_2P_2L_1P_1AQ_1Q_2\cdots Q_{n-1} = R.$$

Der gleiche Trick wie bei der Spaltenpivotsuche erlaubt es uns, die Zerlegung als

$$PAQ = LR$$

zu schreiben, wobei $L$ und $P$ wie dort definiert werden und $Q$ einfach das Produkt der $Q_i$ sei. Als Algorithmus aufgeschrieben stellt sich alles folgendermaßen dar:

**Algorithmus 2.2.7.** *$LR$–Zerlegung mit Totalpivotsuche*

$R = A$, $L = \mathbb{I}$, $P = \mathbb{I}$, $Q = \mathbb{I}$
**for** $k = 1$ **to** $n - 1$ **do**
  *Wähle $i, j \geq k$ so, daß $|r_{ij}| = \max_{s=k,\ldots,n}^{t=k,\ldots,n} |r_{st}|$*
  $r_{k,k:n} \leftrightarrow r_{i,k:n}$ ; `Vertausche die beiden Zeilen`
  $r_{k:n,k} \leftrightarrow r_{k:n,j}$ ; `Vertausche die beiden Spalten`
  $\ell_{k,1:k-1} \leftrightarrow \ell_{i,1:k-1}$
  $p_{k,1:n} \leftrightarrow p_{i,1:n}$
  $q_{1:n,k} \leftrightarrow p_{1:n,j}$
  **for** $j = k + 1$ **to** $n$ **do**
    $\ell_{jk} = r_{jk}/r_{kk}$
    $r_{j,k:n} = r_{j,k:n} - \ell_{jk}r_{k,k:n}$

**done**
**done**

Auch für $Q$ benötigt man keine gesamte Matrix sondern nur einen weiteren Vektor $q$ von ganzen Zahlen. Der Algorithmus zur Lösung des linearen Gleichungssystems wird auch ein wenig komplizierter:

$$Ax = b \iff PAx = Pb \iff PAQQ^{-1}x = Pb \iff LRQ^{-1}x = Pb.$$

$Q^{-1}$ wäre übrigens leicht zu berechnen, da jede Permutationsmatrix eine orthogonale Matrix ist, und daher $Q^{-1} = Q^\top$ gilt.

**Algorithmus 2.2.8.** *Lösung eines linearen Gleichungssystemes mit Totalpivotsuche*

(1) *Berechne $PAQ = LR$, die LR–Faktorisierung mit Totalpivotsuche von $A$.*
(2) *Löse $Lu = Pb$*
(3) *Löse $Rv = u$.*
(4) *Berechne $x = Qv$.*

Der Rechenaufwand beträgt wieder $O(n^3)$ Operationen, doch ist die vernachlässigte Konstante deutlich größer als für die $LR$–Zerlegung mit Spaltenpivotsuche.

**2.3. Nachiteration.** Die Genauigkeit der Lösung eines linearen Gleichungssystemes, die mit dem Spaltenpivotverfahren berechnet wurde, kann mit relativ geringem Aufwand verbessert werden. Wir haben ja nicht die wahre Lösung $x$ des Gleichungssystemes $Ax = b$ bestimmt sondern nur eine rundungsfehlerbehaftete Näherungslösung $\tilde{x}$. Die genaue Fehleranalyse werden wir dann im Abschnitt 3 vornehmen. Wenn wir folgenden Algorithmus durchführen

**Algorithmus 2.3.1.** *Nachiteration*

(1) *Berechne den Residuenvektor $r := b - A\tilde{x}$.*
(2) *Löse $Lu = Pr$.*
(3) *Löse $Rz = u$.*
(4) *Setze $x_{neu} = \tilde{x} + z$.*

Bei exakter Rechnung hätten wir dann $Ax_{\text{neu}} = A\tilde{x} + Az = (b - r) + r = b$. Leider ergibt der Algorithmus, wenn er in naiver Weise in Gleitkommaarithmetik umgesetzt wird, ein $x_{\text{neu}}$, das in keiner Weise genauer ist als der ursprünglich berechnete Wert $\tilde{x}$. Der Grund dafür ist darin zu suchen, daß der berechnete Wert $\tilde{r} = \text{gl}(b - A\tilde{x})$ aufgrund von Auslöschung nur eine erbärmliche Genauigkeit besitzt. Daher besitzt das aus $\tilde{r}$ in Schritt 3 berechnete $\tilde{z}$ auch nur eine sehr geringe Genauigkeit; daher verbessert die Korrektur in Schritt 4 die Lösung $\tilde{x}$ nicht. In [**Skeel 1980**] wurde im Rahmen einer Fehleranalyse gezeigt, daß der Iterationsschritt ein in der Genauigkeit verbessertes $x_{\text{neu}}$ erzeugt, wenn die Zahl

$$\tau := \big(\||A||A^{-1}|\|_\infty\big)\frac{\max_i(|A||x|)_i}{\min_i(|A||x|)_i}$$

klein ist (was klein bedeutet, hängt von der verwendeten Gleitkommaarithmetik ab). In diesem Fall erfüllt $x_{\text{neu}}$

$$(A + E)x_{\text{neu}} = b \tag{21}$$

mit sehr kleinem Fehlerterm $E$. Eine Lösung $\tilde{x}$, die mit Hilfe des Spaltenpivotverfahren berechnet wurde, erfüllt allerdings auch schon eine Gleichung dieser Art.

Um in diesem Fall mit Hilfe von Algorithmus 2.3.1 eine Genauigkeitsverbesserung zu erzielen, muß $r$ mit Gleitkommaarithmetik höherer Genauigkeit berechnet werden. Dazu muß aber auch $A$ in höherer Genauigkeit verwendet werden (das ist sinnvoll, wenn $A$ ausgehend von bekannten Datenbeständen gerundet wird). Hat man beispielsweise $\tilde{x}$ mit einfacher Genauigkeit (single precision) berechnet, so bietet sich an, $r$ mit doppelter Genauigkeit (double

precision) zu bestimmen. Benötigt man höhere Präzision und steht keine extended precision Arithmetik zur Verfügung, so kann man sich eines Softwarepaketes zur Rechnung mit noch höherer Genauigkeit bedienen, bzw. die Rechenfehler minimieren, indem man etwa fehlerkorrigierende Summation (Kahan-Babuška–Summation, siehe Kapitel 1 Beispiel 5.2.9) verwendet. Die Geschwindigkeit der arithmetischen Operationen fällt dabei nicht so sehr ins Gewicht, da der Gesamtaufwand eines Nachiterationsschrittes $O(n^2)$ beträgt.

Das Verfahren, den Algorithmus 2.3.1 zu verwenden, wenn Schritt 1 mit höherer Genauigkeit berechnet wird, heißt Nachiteration mit gemischter Genauigkeit. Als Faustregel für die erreichte Präzision kann man sagen, daß nach $k$ Iterationsschritten $x$ mindestens $\min(d, k(d-q))$ korrekte Stellen hat, wobei $\mathsf{eps} \approx 10^{-d}$ und $\kappa_\infty(A) \approx 10^q$ ($\kappa_\infty(A)$, eine Konditionszahl für $A$, wird in Abschnitt 3.1.2 eingeführt und hängt mit der Kondition des Problemes $Ax = b$ zusammen). Grob gesagt erzeugt die Nachiteration eine Lösung, die korrekt ist bis zur vollen (einfachen) Genauigkeit, wenn $\mathsf{eps}\,\kappa_\infty(A) < 1$. Auf Systemen mit sehr kurzer Mantisse kann das Verfahren die Anzahl der effektiv lösbaren linearen Gleichungssysteme signifikant vergrößern ohne alle Matrizen und Vektoren mit höherer Genauigkeit abspeichern zu müssen.

Zusätzlich ist es sehr wichtig, sich die Möglichkeit der Nachiteration im Gedächtnis zu halten im Zusammenhang mit einigen Verfahren zur Lösung von Gleichungssystemen, die wegen der speziellen Gestalt der Matrix $A$ (Bandstruktur, Dünnbesetztheit) zur Berechnung von $P$, $L$ und $R$ spezielle Pivotstrategien verwenden, um für $L$ und $R$ auch eine spezielle Struktur zu erreichen. Bei diesen kann es passieren, daß $\tilde{x}$ keine Gleichung der Form (21) mit besonders kleinem $E$ erfüllt. In diesem Fall kann mit Hilfe der Nachiteration die Genauigkeit der Lösung stufenweise verbessert werden, auch wenn man zur Berechnung von $\tilde{r}$ nicht Gleitkommaarithmetik höherer Genauigkeit verwendet. Der geringe Aufwand $O(n^2)$, der meist nicht ins Gewicht fällt, und der Erfolg rechtfertigen bei weitem den höheren Implementationsaufwand.

## 3. Fehleranalyse

Wie schon in Kapitel 1 erwähnt, ist es die Pflicht des numerischen Mathematikers, einen entwickelten Algorithmus auf seine Gutartigkeit zu testen. Wie sich herausstellen wird ist gerade dies beim Gaußschen Eliminationsverfahren äußerst schwierig, in jedem Fall schwieriger als bei beinahe allen anderen Algorithmen der numerischen linearen Algebra.

Für die Analyse, auch für die der später noch kommenden Algorithmen, sind jedoch noch einige Begriffe notwendig.

**3.1. Regularitätsmaße.** Es ist aus der linearen Algebra wohl bekannt, daß eine Matrix $A$ genau dann regulär ist, wenn ihre Determinante $\det A \neq 0$ erfüllt. Dieses Kriterium ist jedoch leider für numerische Zwecke nicht sehr brauchbar. Denn die Rundungsfehler, die während der Berechnung der Determinante einer singulären Matrix entstehen, führen meist dazu, daß der tatsächlich berechnete Wert von Null verschieden ist. Es wird also Regularität vorgetäuscht obwohl in Wirklichkeit Singularität vorliegt; es wird die eindeutige Lösbarkeit eines linearen Gleichungssystemes vorgespiegelt obwohl eventuell keine oder unendlich viele Lösungen existieren.

Unterscheidet sich eine Matrix $A$ nur sehr wenig von einer singulären Matrix, so kann durch Rundungsfehlereinflüsse genauso der umgekehrte Effekt eintreten. Während der $LR$–Zerlegung treten dann sehr kleine Pivotelemente auf oder sogar Pivotelemente, die Null sind. In beiden Fällen bricht der Algorithmus ab, da entweder durch Null dividiert wird oder ein Überlauffehler entsteht.

Diese Betrachtungen unterstreichen, daß es wichtig für die Beurteilung numerische Berechnungen ist zu wissen, *wie regulär* eine Matrix ist. Mit der Hilfe dieser *Regularitätsmaße*

können wir dann abschätzen, wie groß die auftretenden Fehler sein dürfen damit das Ergebnis noch brauchbar bleibt.

**3.1.1. Determinante.** Das **unbrauchbarste** Regularitätsmaß ist die Determinante. Sie trifft nur ja/nein Aussagen über die Regularität:

$$\det A \neq 0 \quad \Longleftrightarrow \quad A \text{ regulär.}$$

Außerdem ist die Größe der Determinante kein Hinweis auf den Grad an Regularität. Es gilt ja bekannterweise

$$\det(\lambda A) = \lambda^n \det A,$$

doch ein Vielfaches einer Matrix ist nicht mehr oder weniger regulär als die Matrix selbst. Andererseits ist für $\lambda = 2$ und $n = 50$ der Faktor $\lambda^n > 10^{15}$. Die Determinante ist also offensichtlich nicht geeignet als Regularitätsmaß.

**3.1.2. Konditionszahl.** Beginnen wir die Untersuchung über Regularität von Matrizen mit dem Studium des Problems der Matrixmultiplikation. Wie ist die Kondition des Problems $y = Ax$ aus gegebenem $A$ und $x$ zu berechnen?

Nach Kapitel 1 Abschnitt 5 wissen wir, daß aus $\varphi(x) = Ax$ die Gleichung

$$\Delta y \doteq D\varphi(x)\Delta x = A\Delta x$$

folgt, die die Verstärkung der absoluten Fehler der Eingangsdaten beschreibt. Fassen wir die Konditionszahlen (die Fehlerverstärkungsfaktoren für die relativen Fehler) zu einer einzigen Konditionszahl $\kappa$ zusammen, so finden wir, daß für beliebig gewählte Normen gelten muß

$$\frac{\|\Delta y\|}{\|y\|} \leq \kappa \frac{\|\Delta x\|}{\|x\|}$$

für $x, y \neq 0$. Durch Umformen finden wir, daß

$$\kappa = \frac{\|x\|}{\|\varphi(x)\|} \cdot \|D\varphi(x)\|_M$$

eine Konditionszahl ist, wenn $\| \quad \|_M$ eine mit den zuvor gewählten Normen verträgliche Matrixnorm ist. Im speziellen Fall der Matrixmultiplikation folgt

$$\kappa = \frac{\|x\|}{\|Ax\|} \cdot \|A\|.$$

Diese Konditionszahl hängt von $A$ und $x$ ab, bietet aber relativ gute Abschätzungen für die Fehlerverstärkungsfaktoren. Nehmen wir zusätzlich an, daß die Matrix $A$ quadratisch und regulär ist. Dann können wir die Abhängigkeit von $x$ aus den obigen Abschätzungen entfernen:

$$\kappa = \frac{\|A^{-1}y\|}{\|y\|} \cdot \|A\| \leq \|A^{-1}\| \, \|A\|,$$

und es gibt Vektoren $x$, für die Gleichheit gilt (wählt man die 2–Norm, dann gilt das für jedes nichtverschwindende Vielfache eines rechtssingulären Vektors zum kleinsten Singulärwert von $A$ — siehe Kapitel 4 Abschnitt 2.3).

Berechnen wir nun die Konditionszahl für das inverse Problem, der Lösung des linearen Gleichungssystemes $Ax = y$. Für reguläres $A$ ist das äquivalent zur Multiplikation von $y$ mit der inversen Matrix $A^{-1}$. Nachdem die Konditionszahl $\kappa$ von oben unabhängig von $x$ und $y$ und symmetrisch in $A$ und $A^{-1}$ ist, erkennen wir sofort, daß $\kappa$ auch eine Konditionszahl für das inverse Problem ist.

Dies rechtfertigt die folgende Definition.

**Definition 3.1.1.** *Sei $A$ eine Matrix und $\|\ \|$ eine Matrixnorm. Dann können wir die (zu $\|\ \|$ gehörige)* Konditionszahl *von $A$ definieren als*

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

*Falls wir uns auf eine bestimmte Norm beziehen, z.B. $\|\ \|_2$, dann versehen wir auch $\kappa$ mit einem Index ($\kappa_2$). Für singuläre Matrizen setzen wir $\kappa(A) := \infty$.*

Mit dieser Festlegung können wir auch die folgende Proposition formulieren, die die Herleitungen von oben zusammenfaßt.

**Proposition 3.1.2.** *Sei $A \in \mathbb{K}^{n\times n}$ regulär und betrachten wir die Gleichung $Ax = y$. Das Problem $y$ aus $x$ zu berechnen hat Konditionszahl*

$$\kappa = \|A\| \cdot \frac{\|x\|}{\|y\|} \le \|A\|\|A^{-1}\|$$

*bezüglich Fehlern in $x$. Das Problem $x$ aus $y$ zu berechnen hat Konditionszahl*

$$\kappa = \|A^{-1}\| \cdot \frac{\|y\|}{\|x\|} \le \|A\|\|A^{-1}\|$$

*bezüglich Fehlern in $y$. In beiden Ungleichungen existieren $x$ und $y$, sodaß Gleichheit gilt.*

Die Kondition ist bei Beachtung einiger Regeln auch als Regularitätsmaß gut geeignet. Im Gegensatz zur Determinante erfüllt die Konditionszahl $\kappa(\lambda A) = \kappa(A)$ für alle $\lambda \in \mathbb{K}$.

Die Definition für singuläre Matrizen ist auch nicht unbegründet wie die folgende Proposition zeigt.

**Proposition 3.1.3.** *Sei $B \in \mathbb{K}^{n\times n}$ singulär mit $\operatorname{rg} B = n-1$. Sei $A_n$ eine Folge von regulären Matrizen mit $A_n \to B$. Dann gilt $\kappa(A_n) \to \infty$.*

Beweis. Es gilt $A^{-1} = \operatorname{Adj}(A)/\det(A)$ (aus der linearen Algebra). Ist $\operatorname{rg} B = n-1$ dann ist $\operatorname{Adj}(B) \neq 0$. Aus diesen Fakten folgt

$$\kappa(A_n) = \frac{\|\operatorname{Adj}(A_n)\|\|A_n\|}{|\det A_n|} \to \infty,$$

da der Zähler ungleich Null bleibt, der Nenner aber gegen Null strebt. □

Ferner gilt für die Konditionszahl noch

$$\kappa(A) \ge \|AA^{-1}\| = \|\mathbb{I}\| = 1.$$

Nachdem die Einheitsmatrix eine sehr reguläre Matrix ist, die Konditionszahl 1 hat, liegt die Annahme nahe, daß einerseits sehr reguläre Matrizen kleine Konditionszahlen nahe bei eins, fast singuläre Matrizen andererseits sehr große Konditionszahlen haben.

Folgendes Resultat, das aus den Definitionen und Resultaten zuvor folgt, sei zusammenfassend noch einmal erwähnt

**Proposition 3.1.4.** *Es gilt für die tatsächlich Lösung $x$ und die berechnete Näherungslösung $\tilde{x}$ eines linearen Gleichungssystemes $Ax = b$*

$$\frac{\|\Delta x\|}{\|x\|} \le \kappa(A) \cdot \frac{\|b - A\tilde{x}\|}{\|b\|}$$

**Beispiel 3.1.5.** *Sei*

$$A = \begin{pmatrix} \frac{1}{200} & 1 \\ 1 & 1 \end{pmatrix} \quad \text{und daher} \quad A^{-1} = \frac{200}{199}\begin{pmatrix} -1 & 1 \\ 1 & -\frac{1}{200} \end{pmatrix}.$$

*Es gilt $\|A\|_\infty = 2$ und $\|A^{-1}\|_\infty = \frac{400}{199}$. Die Konditionszahl von $A$ ist also $\kappa_\infty(A) = \frac{800}{199} \approx 4$.*

*Skalieren wir $A$ mit einer Diagonalmatrix $D$ um, so ändert das das Verhalten der Matrix im Gaußschen Algorithmus nicht im geringsten (sofern durch die Wahl der Skalierung nicht die Wahl des Pivotelementes beeinflußt wird). Wie sieht es mit der Konditionszahl aus?*

*Sei* $D = \operatorname{diag}(200, 1)$. *Dann sind*

$$DA = \begin{pmatrix} 1 & 200 \\ 1 & 1 \end{pmatrix} \quad \textit{und} \quad (DA)^{-1} = \frac{1}{199} \begin{pmatrix} 1 & -200 \\ -1 & 1 \end{pmatrix},$$

$\|DA\|_\infty = 201$, $\|(DA)^{-1}\|_\infty = 201/199$ *und* $\kappa_\infty(A) = 40401/199 \approx 200$*; die Umskalierung hat als die Konditionszahl um einen Faktor* 50 *vergrößert.*

Das obige Beispiel zeigt, daß die Konditionszahl nur dann als Maß für die Regularität von $A$ verwendet werden kann, wenn man zuvor die Skalierung von $A$ festlegt. Die folgende von van der Sluis in [**van der Sluis 1969**] bewiesene Proposition gibt einen Hinweis auf die zu verwendende Skalierung

**Proposition 3.1.6.** *Für alle regulären Diagonalmatrizen $D$ wird $\kappa_\infty(DA)$ minimal bei $D = \operatorname{diag}(1/\sum_{k=1}^n |A_{1k}|, \ldots, 1/\sum_{k=1}^n |A_{nk}|)$, also wenn $\sum_{k=1}^n |(DA)_{ik}| = 1$ für $i = 1, \ldots, n$.*

Beweis. Sei o.B.d.A. $A$ so skaliert, daß $\sum_{k=1}^n |A_{ik}| = 1$ für alle $i$ gilt. Für eine beliebige reguläre Diagonalmatrix $D$ finden wir dann

$$\|DA\|_\infty = \max_{i=1,\ldots,n} (|D_{ii}| \sum_{k=1}^n |A_{ik}|) = \max_{i=1,\ldots,n} |D_{ii}| = \|D\|_\infty,$$

und daher

$$\begin{aligned} \kappa_\infty(A) = \|A^{-1}\|_\infty \|A\|_\infty = \|(DA)^{-1} D\|_\infty \leq \|(DA)^{-1}\|_\infty \|D\|_\infty = \\ = \|(DA)^{-1}\|_\infty \|DA\|_\infty = \kappa_\infty(DA). \end{aligned}$$

□

Matrizen $A$, die

$$\sum_{k=1}^n |A_{ik}| \approx 1, \quad \text{für } i = 1, \ldots, n$$

erfüllen, heißen *equilibriert*. Für solche Matrizen ist die Konditionszahl ein nützliches Maß für die Regularität.

**3.1.3. Singularitätsabstand.** Das „beste“ Maß für die Regularität einer Matrix ist der *Singularitätsabstand*

$$\operatorname{sing}(A) := \inf\{\delta \geq 0 | \exists B \text{ singulär mit } |B - A| \leq \delta |A|\}.$$

Läßt man also in $A$ komponentenweise Fehler der Größenordnung $\operatorname{sing}(A)$ zu kann man eine singuläre Matrix finden, für kleinere Fehler jedoch nicht.

Eigenschaften von $\operatorname{sing}(A)$:

- $0 \leq \operatorname{sing}(A) \leq 1$,
- $\operatorname{sing}(A) = 0$ genau dann, wenn $A$ singulär ist,
- $\operatorname{sing}(D_1 A D_2) = \operatorname{sing}(A)$ für beliebige reguläre Diagonalmatrizen $D_1$ und $D_2$; das heißt sing ist skalierungsunabhängig.

Unglücklicherweise ist der Singularitätsabstand sehr schwierig zu berechnen. Man kann jedoch einen Zusammenhang mit der Konditionszahl beweisen.

**Proposition 3.1.7.** *Falls $A$ regulär ist, gilt* $\operatorname{sing}(A) \geq 1/\kappa_\infty(A)$.

BEWEIS. Sei $|B - A| \leq \delta|A|$. Ist $B$ singulär, dann ist auch $A^{-1}B$ singulär, und damit gilt wegen Proposition 3.1.6 $\|\mathbb{I} - A^{-1}B\|_\infty \geq 1$ und deshalb

$$1 \leq \|\mathbb{I} - A^{-1}B\|_\infty = \|A^{-1}(A - B)\|_\infty \leq \|A^{-1}\|_\infty \|B - A\|_\infty \leq \|A^{-1}\|_\infty \|\delta|A|\|_\infty = \delta\kappa_\infty(A).$$

Aus dieser Abschätzung folgt für alle $\delta$ die Ungleichung $\delta \geq 1/\kappa_\infty(A)$ und daraus folgt die Behauptung. □

**3.1.4. Konditionsschätzung.** Ein großer Nachteil der Definition der Konditionszahl $\kappa(A)$ ist, daß man die Norm der Inversen von $A$ kennen muß. Diese ohne die Inverse selbst zu berechnen ist schwierig. Das Berechnen der Inversen aber ist aufwendiger als die Lösung eines linearen Gleichungssystemes. Eines der schnellsten Verfahren zur Bestimmung der Inversen $A^{-1}$ besteht darin, eine $LR$–Zerlegung von $A$ durchzuführen und für jeden Einheitsvektor $e_i$ das lineare Gleichungssystem $Ax^{(i)} = e_i$ zu lösen. Die $x^{(i)}$ sind dann die Spalten von $A^{-1}$. Der Aufwand für diese Berechnung ist zusätzlich zu den $O(n^3)$ der $LR$–Faktorisierung noch einmal $n \cdot O(n^2) = O(n^3)$ für die Lösung der Gleichungssysteme. Daher ist es manchmal besser, die Konditionszahl von $A$ nicht exakt zu berechnen sondern nur abzuschätzen.

Die Konditionszahl $\kappa_\infty(A)$ für die Zeilensummennorm bestimmen wir näherungsweise wie folgt:
Sei $s_i$ die Summe der Beträge der $i$–ten Zeile der Matrix $A^{-1}$. Aus der Definition von $\| \quad \|_\infty$ folgt dann, daß ein Index $i_0$ existiert mit

$$\|A^{-1}\|_\infty = \max_{i=1,\ldots,n} s_i = s_{i_0}.$$

Die $i$–te Zeile von $A^{-1}$ ist

$$(f^{(i)})^* = (e_i)^* A^{-1}.$$

Falls eine $LR$–Faktorisierung von $A$ vorliegt, so läßt sich $f^{(i)}$ als Lösung des konjugiert transponierten Systems

$$A^* f^{(i)} = e_i$$

berechnen:

(1) Löse $R^* y = e_i$,
(2) Löse $L^* z = y$,
(3) Setze $f^{(i)} = P^\top z$.

Könnten wir den Index $i_0$ erraten, so wäre die Berechnung von $\kappa_\infty(A)$ in einfachster Weise beendet. Für einen beliebigen Index erhalten wir eine untere Schranke für die Konditionszahl von $A$. Sei $e = (1, \ldots, 1)^\top$ und $v$ ein beliebiger Vektor mit $|d| = e$. Dann gilt

$$|(A^{-1}d)|_i = (e_i)^* |A^{-1}d| \leq (e_i)^* |A^{-1}||d| = |f^{(i)}|^* e = s_i. \tag{22}$$

Die Berechnung der $|A^{-1}d|_i$ ist weniger aufwendig als die Berechnung der $s_i$. Es ist naheliegend, für den Index $i$ den Index der größten Komponente von $A^{-1}d$ zu wählen. Für $d$ setzt man am besten

$$d := \operatorname{sgn}((A^*)^{-1}e).$$

In diesem Fall erhalten wir, falls $A^{-1}$ spaltenweise konstante Vorzeichen hat, den exakten Wert $s_i = \|A^{-1}\|_\infty$. Der Grund dafür ist, daß dann die $i$–te Komponente $((A^*)^{-1}e)_i$ das Vorzeichen der $i$–ten Spalte von $A^{-1}$ trägt, und wegen $d_i = \operatorname{sgn}(((A^*)^{-1}e)_i)$ gilt in (22) Gleichheit.

Alle diese Überlegungen führen zum folgenden Algorithmus zur Abschätzung der Konditionszahl $\kappa_\infty(A)$.

**Algorithmus 3.1.8.** *Konditionsschätzung*

(1) *Löse $A^* v = e$ und setze $d = \operatorname{sgn}(v)$.*
(2) *Löse $Aw = d$ und suche $j$ mit $w_j = \max_i |w_i|$.*

(3) *Löse* $A^*f = e_j$ *und bestimme* $s = \langle |f|, e\rangle$.

*Dann gilt* $\|A^{-1}\|_\infty \geq s$ *und es gilt oft Gleichheit. Bei Untersuchungen mit Zufallsmatrizen mit* $A_{ik} \in [-1, 1]$ *findet man Gleichheit in mehr als 60% der Fälle. In über 99% aller Fälle gilt zumindest* $\|A^{-1}\|_\infty \leq 3s$.

Der Aufwand für die Konditionsschätzung ist bei vorliegender $LR$–Zerlegung nur $O(n^2)$ zusätzliche Operationen. Es existieren auch andere Verfahren, die z.B. auch in den LAPACK Routinen `???con` (siehe 5) implementiert sind.

**3.2. Datenstörung.** Bevor wir uns auf die Analyse der Rundungsfehler im Gaußschen Algorithmus stürzen, ein kleiner Vorgeschmack. In vielen Anwendungsproblemen kommt es vor, daß die Koeffizienten des Systemes nicht exakt bekannt sind. In einem ersten Schritt interessieren wir uns für die Ungenauigkeit der Lösung, die daraus erwächst. Der folgende Satz, gibt Auskunft über die Auswirkungen von Ungenauigkeiten in der Matrix $A$.

**Theorem 3.2.1** (Prager/Oettli)**.** *Seien* $A \in \mathbb{K}^{n\times n}$, $b, \tilde{x} \in \mathbb{K}^n$. *Für beliebige nichtnegative Matrizen* $\Delta A \in \mathbb{R}^{n\times n}$ *und Vektoren* $\Delta b \in \mathbb{R}^n$ *sind folgende Aussagen äquivalent:*

(1) *Es gibt* $\tilde{A} \in \mathbb{K}^{n\times n}$ *und* $\tilde{b} \in \mathbb{K}^n$ *mit*

$$\tilde{A}\tilde{x} = \tilde{b}, \quad |\tilde{A} - A| \leq \Delta A, \quad \textit{und} \quad |\tilde{b} - b| \leq \Delta b.$$

(2) *Es gilt die Ungleichung*

$$|b - A\tilde{x}| \leq \Delta b + \Delta A|\tilde{x}|$$

*für das Residuum.*

Beweis. In [**Prager, Oettli 1964**]. □

Der Satz von Prager und Oettli erlaubt es also, von der Kleinheit des Residuums auf die Brauchbarkeit einer Lösung zu schließen, soferne man eine Lösung $\tilde{x}$ von $Ax = b$ dann als brauchbar definiert, wenn sie ein in der Nähe liegendes Gleichungssystem $\tilde{A}\tilde{x} = \tilde{b}$ löst.

Besitzen z.B. alle Komponenten von $A$ und $b$ dieselbe relative Genauigkeit $\varepsilon$, d.h.

$$\Delta A = \varepsilon|A|, \quad \Delta b = \varepsilon|b|,$$

so ist $\tilde{x}$ Lösung eines nahe liegenden Gleichungssystemes $\tilde{A}\tilde{x} = \tilde{b}$ mit $|A - \tilde{A}| \leq \Delta A$ und $|b - \tilde{b}| \leq \Delta b$, falls

$$|A\tilde{x} - b| \leq \varepsilon(|b| + |A||\tilde{x}|).$$

Diese Ungleichung liefert sofort das kleinste $\varepsilon$, für das ein berechnetes $\tilde{x}$ noch als brauchbare Lösung akzeptiert werden kann.

**3.3. Rundungsfehler beim Gaußschen Algorithmus und verwandten Verfahren.** Daß selbst beim Spaltenpivotverfahren Probleme auftreten können, sei an den beiden folgenden Beispielen belegt:

**Beispiel 3.3.1.** *Aus Beispiel 2.1.9 ist schon bekannt, daß der Gaußsche Algorithmus ohne Pivotsuche auf die Matrix*

$$A = \begin{pmatrix} 10^{-20} & 1 \\ 1 & 1 \end{pmatrix}$$

*angewendet nicht gutartig ist. Betrachten wir als nächstes die Matrix*

$$\operatorname{diag}(10^{20}, 1) \cdot A = \begin{pmatrix} 1 & 10^{20} \\ 1 & 1 \end{pmatrix}.$$

*Wenden wir darauf das Spaltenpivotverfahren an, so ist im ersten Schritt die Wahl des Elements $A_{11}$ als Pivotelement zulässig. Die Matrix $L_1P_1A$ sieht danach aber ähnlich aus wie die Matrix $R$ in Beispiel 2.1.9:*

$$R = L_1P_1A = \begin{pmatrix} 1 & 10^{20} \\ 0 & 1-10^{20} \end{pmatrix}$$

*mit den gleichen Rundungsfehlerproblemen wie dort.*

**Beispiel 3.3.2.** *Betrachten wir als nächstes die Matrix*

$$A = \begin{pmatrix} 1 & & & & \cdots & 1 \\ -1 & 1 & & & \cdots & 1 \\ -1 & -1 & 1 & & \cdots & 1 \\ -1 & -1 & -1 & 1 & \cdots & 1 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ -1 & -1 & -1 & -1 & \cdots & 1 \end{pmatrix}.$$

*Beim ersten Schritt passiert keine Vertauschung durch Pivotsuche. Die Elemente in der letzten Spalte werden jedoch verdoppelt. Eine weitere Verdoppelung passiert im nächsten Schritt, und so weiter. Die am Ende erzeugte Matrix $R$ ist*

$$R = \begin{pmatrix} 1 & & & \cdots & 1 \\ & 1 & & \cdots & 2 \\ & & 1 & \cdots & 4 \\ & & & \cdots & 8 \\ & & & \ddots & \vdots \\ & & & & 2^{n-1} \end{pmatrix}.$$

*Ihre $\infty$–Norm ist $2^{n-1}$, die untere Dreiecksmatrix $L$ dagegen hat $\infty$–Norm $n$, die gleiche wie die Matrix $A$:*

$$L = \begin{pmatrix} 1 & & & & & \\ -1 & 1 & & & & \\ -1 & -1 & 1 & & & \\ -1 & -1 & -1 & 1 & & \\ \vdots & \vdots & \vdots & \vdots & \ddots & \\ -1 & -1 & -1 & -1 & \cdots & 1 \end{pmatrix}.$$

Betrachten wir die beiden Beispiele, so erscheint es als ob das Spaltenpivotverfahren kein gutartiger Algorithmus sei.

Die Probleme im ersten Beispiel können wir noch korrigieren, indem wir uns die Bemerkungen über Konditionszahlen und equilibrierte Matrizen noch einmal ansehen und dafür sorgen, daß die Matrix $A$ equilibriert ist bevor das Spaltenpivotverfahren durchgeführt wird. Dies führt dann zum folgenden veränderten Spaltenpivotverfahren zur Berechnung einer $LR$–Zerlegung der Matrix $A$. Man muß dazu die Matrix nicht wirklich umskalieren. Es genügt, dies nur für die Wahl des Pivotelementes zu tun.

**Algorithmus 3.3.3.** *LR–Zerlegung mit equilibrierter Spaltenpivotsuche*

$R = A$, $L = \mathbb{I}$, $P = \mathbb{I}$
**for** $k = 1$ **to** $n-1$ **do**
  *Wähle $i \geq k$ so, daß* $\frac{|r_{ik}|}{\sum_{r=1}^{n} |r_{ir}|} = \max_{j=k,\ldots,n} \frac{|r_{jk}|}{\sum_{s=1}^{n} |r_{js}|}$
  $r_{k,k:n} \leftrightarrow r_{i,k:n}$ ; `Vertausche die beiden Zeilen`
  $\ell_{k,1:k-1} \leftrightarrow \ell_{i,1:k-1}$
  $p_{k,1:n} \leftrightarrow p_{i,1:n}$
  **for** $j = k+1$ **to** $n$ **do**

$\ell_{jk} = r_{jk}/r_{kk}$
$r_{j,k:n} = r_{j,k:n} - \ell_{jk} r_{k,k:n}$
**done**
**done**

Die Probleme, die aus Beispiel 3.3.2 erwachsen, bedürfen allerdings einer weitaus gründlicheren Untersuchung.

Nachdem die Anwendung einer Permutation keine numerisch bedenkliche Operation ist, genügt es, die Fehlerrechnung für die Matrix $PA$ durchzuführen, bzw. o.B.d.A. anzunehmen, daß keine Vertauschungen bei der Bearbeitung von $A$ nötig sind.

Beginnen wir zuerst mit der Untersuchung der auftretenden Fehler im Rahmen der $LR$–Zerlegung. Seien im folgenden $\tilde{L}$ und $\tilde{R}$ die bei der Zerlegung der Matrix $A \in \mathbb{K}^{n\times n}$ berechneten Faktoren. Deren Produkt unterscheidet sich von $A$ durch eine Fehlermatrix $H$:

$$\tilde{L}\tilde{R} = A + H,$$

für die man folgende Fehlerabschätzung erhält:

**Theorem 3.3.4.** *Seien $A$, $\tilde{L}$, $\tilde{R}$ und $H$ wie im Absatz zuvor. Dann gilt*

$$|H| \dot{\leq} 3(n-1)\,\mathsf{eps}\big(|A| + |\tilde{L}||\tilde{R}|\big),$$

*wobei $\dot{\leq}$ bedeute in erster Näherung kleiner gleich (d.h. unter Vernachlässigung von Termen $O(\mathsf{eps}^2)$).*

Beweis. Der Beweis erfolgt durch Induktion über $n$. Für $n = 1$ ist der Satz offensichtlich wahr, da $a = 1 \cdot a$ ohne Rundungsfehler zerlegbar ist.

Also angenommen, daß der Satz für alle $(n-1)\times(n-1)$ Gleitkommamatrizen gilt. Sei nun

$$A = \begin{pmatrix} \alpha & w^* \\ v & B \end{pmatrix}$$

mit $B \in \mathbb{K}^{(n-1)\times(n-1)}$ und $v, w \in \mathbb{K}^{n-1}$. Der erste Eliminationsschritt führt die folgenden Rechnungen durch:

$$\tilde{z} = \mathrm{gl}\big(\tfrac{1}{\alpha}\, v\big)$$
$$\tilde{A}_1 = \mathrm{gl}(B - \tilde{z}w^*).$$

Nachdem in der Berechnung von $\tilde{z}$ nur einzelne elementare Operationen vorkommen, haben wir nach den Voraussetzungen an die Computerarithmetik

$$\tilde{z} = \tfrac{1}{\alpha}v + f \quad \text{mit} \quad |f| \leq \mathsf{eps}\,\frac{|v|}{|\alpha|}.$$

Auch für $\tilde{A}_1$ gilt eine entsprechende Abschätzung

$$\tilde{A}_1 = B - \tilde{z}w^* + F \quad \text{mit} \quad |F| \dot{\leq} 2\,\mathsf{eps}\big(|B| + |\tilde{z}||w|^\top\big), \tag{23}$$

wie sich leicht nachrechnen läßt, da hier für jedes berechnete Element jeweils zwei elementare Operationen hintereinander ausgeführt werden.

Die erste Zeile bleibt bei den weiteren Schritten der $LR$–Zerlegung unverändert, und der Algorithmus fährt mit der $LR$–Zerlegung der Matrix $\tilde{A}_1$ fort. Diese $(n-1)\times(n-1)$–Matrix erfüllt aber die Induktionsvoraussetzung. Die Näherungsfaktoren $\tilde{L}_1$ und $\tilde{R}_1$ für $\tilde{A}_1$ erfüllen also

$$\tilde{L}_1\tilde{R}_1 = \tilde{A}_1 + H_1 \tag{24}$$

$$|H_1| \dot{\leq} 3(n-2)\,\mathsf{eps}\big(|\tilde{A}_1| + |\tilde{L}_1||\tilde{R}_1|\big). \tag{25}$$

Setzt man die Resultate zusammen, so erhält man

$$\tilde{L}\tilde{R} = \begin{pmatrix} 1 & 0 \\ \tilde{z} & \tilde{L}_1 \end{pmatrix} \begin{pmatrix} \alpha & w^* \\ 0 & \tilde{R}_1 \end{pmatrix} = \\ = A + \begin{pmatrix} 0 & 0 \\ \alpha f & H_1 + F \end{pmatrix} = A + H.$$

Aus der Gleichung (23) folgt, daß

$$|\tilde{A}_1| \dot{\leq} (1 + 2\,\mathsf{eps})\big(|B| + |\tilde{z}||w|^\top\big),$$

und daher haben wir wegen (24) und (25)

$$|H_1 + F| \dot{\leq} 3(n-1)\,\mathsf{eps}\big(|B| + |\tilde{z}||w|^\top + |\tilde{L_1}||\tilde{R_1}|\big).$$

Schließlich folgt aus $|\alpha f| \leq \mathsf{eps}\,|v|$

$$|H| \dot{\leq} 3(n-1)\,\mathsf{eps}\left(\begin{pmatrix} |\alpha| & |w|^\top \\ |v| & |B| \end{pmatrix} + \begin{pmatrix} 1 & 0 \\ |\tilde{z}| & |\tilde{L_1}| \end{pmatrix} \begin{pmatrix} |\alpha| & |w|^\top \\ 0 & |\tilde{R_1}| \end{pmatrix}\right),$$

die gesuchte Abschätzung. □

Auf den ersten Blick erscheint diese Abschätzung die Gutartigkeit des Algorithmus zu beweisen. Der Fehler der Zerlegung ist ein Vielfaches der Maschinengenauigkeit $\mathsf{eps}$, das von $|A|$ abhängt. Doch auch ein zweiter Term steckt im Multiplikator von $\mathsf{eps}$. Es ist dies der Term $|\tilde{L}||\tilde{R}|$. Das Spaltenpivotverfahren erzeugt eine Matrix $\tilde{L}$ mit $|\tilde{L}| = O(\mathbb{I})$, doch die Matrix $\tilde{R}$ ist nicht kontrollierbar, wie das Beispiel 3.3.2 zeigt. Um dieses Verhalten noch weiter zu analysieren, definieren wir

**Definition 3.3.5.** *Der* Wachstumsfaktor *für die LR–Zerlegung von A sei*

$$\rho(A) := \frac{\max_{i,j} |\tilde{R}_{ij}|}{\max_{i,j} |A_{ij}|}.$$

Wenn $\rho(A)$ von Größenordnung 1 ist, so ist nichts passiert und der Algorithmus war stabil. Wenn $\rho(A)$ andererseits sehr groß ist, dann kann man Instabilität erwarten. Beispiel 3.3.2 zeigt, daß $\rho(A) = 2^{n-1}$ werden kann. Umgekehrt ist es aber nicht allzu schwierig zu beweisen, daß das auch schon der schlimmste Fall ist.

Die Frage ist jetzt: „Warum ist der Gaußsche Eliminationsalgorithmus, genauer das Spaltenpivotverfahren, dann so berühmt und populär, wenn es doch nicht stabil ist?“ Die Antwort kommt (wie ist es in der numerischen Mathematik auch anders zu erwarten) aus der Praxis:

> Obwohl es Beispiele wie 3.3.2 gibt, die Wachstumsfaktoren besitzen, die exponentiell von der Dimension abhängen, erweist sich das Spaltenpivotverfahren in der Praxis als äußerst stabil. Faktoren $R$, die sehr viel größer als $A$ sind, treten scheinbar in den Anwendungen nie auf. Es ist kein Beispiel bekannt, bei dem in der Praxis exponentielle Instabilität beobachtet worden wäre.

Trotzdem, warum kann man verantworten, einen Algorithmus zu verwenden, bei dem diese Art von Instabilität auftreten kann und ihm in der Praxis vertrauen? Die Beantwortung dieser Frage ist eine Sache der Statistik. Ausgeprägte Untersuchungen an Zufallsmatrizen haben gezeigt, daß schon Matrizen, deren Wachstumsfaktor $\rho(A) > \sqrt{n}$ erfüllt, äußerst selten sind. In den allermeisten Fällen liegt er sehr weit unter dieser Grenze. Es gibt zwar einige heuristische Argumente warum das so sein muß, doch was der wahre Grund für dieses Verhalten ist, ist ein noch **ungelöstes Problem**. Empirische Wahrscheinlichkeitsverteilungen für die Wachstumsfaktoren für Zufallsmatrizen der Dimensionen 8, 16, 32 und 64 aufgetragen auf einer halblogarithmischen Skala kann man in Abbildung 3.1 sehen. Eine Darstellung der

Wachstumsfaktoren von etwa 1000 Zufallsmatrizen verschiedenster Dimensionen kann man in Abbildung 3.2 finden.

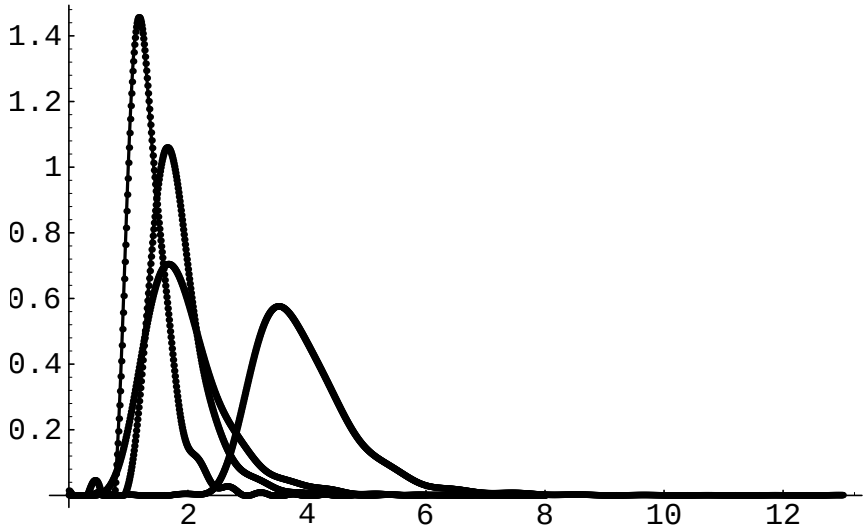



ABBILDUNG 3.1. Eine glatte Dichteschätzung basierend auf empirischen Wahrscheinlichkeitsverteilungen für die Wachstumsfaktoren von Zufallsmatrizen der Dimensionen 8, 16, 32 und 64, basierend auf Samples der Größe $10^4$ für jede dieser Dimensionen. Die Dichte scheint exponentiell mit $\rho(A)$ zu fallen, während sie mit zunehmender Dimension kleiner und breiter wird und ihr Peak Richtung $+\infty$ wandert.

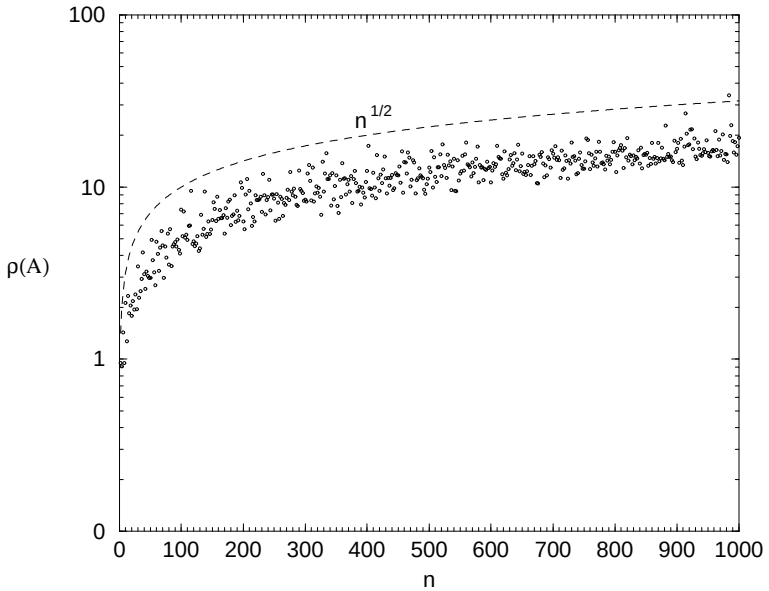



ABBILDUNG 3.2. Wachstumsfaktoren für 500 Zufallsmatrizen verschiedener Dimensionen mit unabhängig normalverteilten Eintragungen. Es gibt nur 1 Matrix, deren Wachstumsfaktor die „imaginäre Grenze“ $\sqrt{n}$ überschreitet.

Nachdem die Fehleranalyse der $LR$–Zerlegung abgeschlossen ist, bleibt für eine vollständige Analyse des Gaußschen Algorithmus noch übrig, die Fehlerverstärkung in der Auflösung

der dreieckigen Gleichungssysteme zu untersuchen. Was passiert also, wenn wir die berechneten $\tilde{L}$ und $\tilde{R}$ anstelle der wirklichen Faktoren $L$ und $R$ zur Lösung des Gleichungssystemes heranziehen? Diese Frage beantwortet der folgende Satz 3.3.7.

Zuvor untersuchen wir aber noch, was bei einem (Vorwärts–) Rückwärtseinsetzschritt gemäß Algorithmus 2.1.3 (2.1.2) passiert.

**Proposition 3.3.6.** *Sei $R \in \mathcal{R}(n)$ ($\in \mathcal{L}(n)$) eine obere (untere) Dreiecksmatrix und $\tilde{x}$ die nach Algorithmus 2.1.2 (2.1.3) berechnete Lösung von $Rx = b$. Dann ist $\tilde{x}$ Lösung eines benachbarten Gleichungssystemes $(R + F)\tilde{x} = b$ mit*

$$|F| \dot{\leq} n\,\mathsf{eps}\,|R|.$$

Beweis. Der Beweis erfolgt mittels vollständiger Induktion nach $n$, der Dimension der Matrix $R$.

Der Induktionsanfang für $n = 1$ ist einfach: Es gilt $\tilde{x}_1 = \mathrm{gl}(b_1/R_{11})$, und die Eigenschaften der Gleitkommaarithmetik liefern sofort die Abschätzung

$$\tilde{x}_1 = \frac{b_1}{R_{11}}(1 + \varepsilon_1), \quad |\varepsilon_1| \leq \mathsf{eps}\,.$$

Für die benötigte Abschätzung müssen wir nur noch beobachten, daß mit $\varepsilon_1' = -\varepsilon_1/(1+\varepsilon_1)$ gilt

$$\tilde{x}_1 = \frac{b_1}{R_{11}(1 + \varepsilon_1')}, \quad |\varepsilon_1'| \leq \mathsf{eps} + O(\mathsf{eps}^2) \dot{\leq} \mathsf{eps}\,.$$

Nehmen wir nun an, wir hätten den Satz bereits bewiesen für Matrizen bis zur Größe $(n-1) \times (n-1)$. Schreiben wir das Gleichungssystem $Rx = b$ an als

$$\begin{pmatrix} \beta & v \\ 0 & R_1 \end{pmatrix} \begin{pmatrix} x_1 \\ y \end{pmatrix} = \begin{pmatrix} b_1 \\ c \end{pmatrix}$$

mit $(n-1) \times (n-1)$–Dreiecksmatrix $R_1$. Multiplizieren wir aus, so erhalten wir die beiden Gleichungen

$$R_1 y = c, \quad \beta x_1 + vy = b_1.$$

Aus der ersten berechnen wir eine Näherungslösung $\tilde{y}$ für $y$. Diese Lösung erfüllt nach Induktionsvoraussetzung ein Gleichungssystem

$$(R_1 + F_1)\tilde{y} = c, \quad |F_1| \dot{\leq} (n-1)\,\mathsf{eps}\,|R_1|.$$

Untersuchen wir nun die erste Gleichung:

$$\begin{aligned}
\tilde{x}_1 &= \mathrm{gl}\left(\frac{b_1 - v\tilde{y}}{\beta}\right) \\
&= \frac{\mathrm{gl}(b_1 - v\tilde{y})}{\beta}(1 + \varepsilon_1), \quad |\varepsilon_1| \leq \mathsf{eps} \\
&= \frac{\mathrm{gl}(b_1 - v\tilde{y})}{\beta(1 + \varepsilon_1')}, \quad |\varepsilon_1'| \dot{\leq} \mathsf{eps} \\
&= \frac{(b_1 - \mathrm{gl}(v\tilde{y}))(1 + \varepsilon_2)}{\beta(1 + \varepsilon_1')}, \quad |\varepsilon_2| \leq \mathsf{eps} \\
&= \frac{b_1 - \mathrm{gl}(v\tilde{y})}{\beta(1 + \varepsilon_1')(1 + \varepsilon_2')}, \quad |\varepsilon_2'| \dot{\leq} \mathsf{eps} \\
&= \frac{b_1 - \mathrm{gl}(v\tilde{y})}{\beta(1 + 2\varepsilon_3)}, \quad |\varepsilon_3| \dot{\leq} \mathsf{eps}\,.
\end{aligned}$$

Untersuchen wir den Ausdruck im Zähler weiter:

$$b_1 - \mathrm{gl}(v\tilde{y}) = b_1 - \left(\sum_{j=1}^{n-1} \mathrm{gl}(v_j\tilde{y}_j)(1+\eta_j)\right)$$

mit

$$1+\eta_j = \prod_{k=j}^{n-1}(1+\varepsilon_{4k}) = (1+(n-j)\varepsilon_{5j}), \quad |\varepsilon_{5j}| \dot{\leq} \mathsf{eps}\,.$$

Setzen wir das oben ein, so erhalten wir

$$\begin{aligned} b_1 - \mathrm{gl}(v\tilde{y}) &= b_1 - \left(\sum_{j=1}^{n-1}(v_j\tilde{y}_j)(1+\varepsilon_{6j})(1+(n-j)\varepsilon_{5j})\right), \quad |\varepsilon_{6j}| \leq \mathsf{eps} \\ &= b_1 - \left(\sum_{j=1}^{n-1} y_j v_j (1+(n-j+1)\varepsilon_{7j})\right), \quad |\varepsilon_{7j}| \dot{\leq} \mathsf{eps}\,. \end{aligned}$$

Insgesamt ergibt sich aus dieser Rechnung die Formel

$$\tilde{x}_1 = \frac{b_1 - \left(\sum_{j=1}^{n-1} y_j v_j (1+(n-j+1)\varepsilon_{7j})\right)}{\beta(1+2\varepsilon_3)}.$$

Beobachtet man nun, daß $\beta = R_{11}$ und $v_j = R_{1\,j+1}$, so sehen wir, daß

$$\begin{pmatrix} R_{11}(1+2\varepsilon_3) & R_{1j}(1+(n-j+2)\varepsilon_{7j}) \\ 0 & R_1 + F_1 \end{pmatrix} \begin{pmatrix} \tilde{x}_1 \\ \tilde{y} \end{pmatrix} = \begin{pmatrix} b_1 \\ c \end{pmatrix}$$

gilt. Aus dieser Gleichung und der Tatsache, daß alle $\varepsilon_\cdot$ vom Betrag in erster Ordnung kleiner als $\mathsf{eps}$ sind, folgt die Behauptung:

$$\left(\begin{pmatrix} R_{11} & R_{1j} \\ 0 & R_{kj} \end{pmatrix} + \underbrace{\begin{pmatrix} 2\varepsilon_3 R_{11} & (n-j+2)\varepsilon_{7j} R_{1j} \\ 0 & F_1 \end{pmatrix}}_{:=F}\right) \tilde{x} = b,$$

denn

$$|F| \dot{\leq} n\,\mathsf{eps}\,|R|$$

wegen der offensichtlichen Abschätzungen in der ersten Zeile und der Induktionsvoraussetzung über $F_1$. □

**Theorem 3.3.7.** *Seien $\tilde{L}$ und $\tilde{R}$ die berechneten LR–Faktoren zur Matrix $A \in \mathbb{K}^{n\times n}$ (nach dem Spaltenpivotverfahren oder dem vollständigen Pivotverfahren). Seien $\tilde{u}$ die berechnete Lösung zu $\tilde{L}\tilde{u} = b$ und $\tilde{x}$ die berechnete Lösung zu $\tilde{R}x = \tilde{u}$. Dann erfüllt $\tilde{x}$ ein Gleichungssystem der Form $(A+E)\tilde{x} = b$ mit*

$$|E| \dot{\leq} n\,\mathsf{eps}\big(3|A| + 5|\tilde{L}||\tilde{R}|\big).$$

Beweis. Nach Proposition 3.3.6 gelten

$$\begin{aligned} (\tilde{L}+F)\tilde{u} &= b \qquad & |F| &\dot{\leq} n\,\mathsf{eps}\,|\tilde{L}| \\ (\tilde{R}+G)\tilde{x} &= b \qquad & |G| &\dot{\leq} n\,\mathsf{eps}\,|\tilde{R}|, \end{aligned}$$

und daher

$$(\tilde{L}+F)(\tilde{R}+G)\tilde{x} = (\tilde{L}\tilde{R} + F\tilde{U} + \tilde{L}G + FG)\tilde{x} = b.$$

Theorem 3.3.4 impliziert

$$\tilde{L}\tilde{R} = A + H$$

mit

$$|H| \dot{\leq} 3(n-1)\,\mathsf{eps}\big(|A| + |\tilde{L}||\tilde{R}|\big).$$

Setzen wir nun

$$E = H + F\tilde{R} + \tilde{L}G + FG,$$

so erhalten wir $(A+E)\tilde{x} = b$ und die Abschätzung

$$\begin{aligned} |E| &\dot{\leq} |H| + |F||\tilde{R}| + |\tilde{L}||G| \\ &\dot{\leq} 3n\,\mathsf{eps}\big(|A| + |\tilde{L}||\tilde{R}|\big) + 2n\,\mathsf{eps}\big(|\tilde{L}||\tilde{R}|\big). \end{aligned}$$

Zusammengefaßt ist das die gesuchte Fehlerschranke. □

Wie nicht zu erwarten, tritt auch in Theorem 3.3.7 der Term $|\tilde{L}||\tilde{R}|$ auf, der den Eliminationsalgorithmus latent instabil gestaltet. Doch auch hier zählt das (unbewiesene) statistische Argument von vorhin, das bedingt, daß in der Praxis Instabilitäten dieser Art nicht auftreten.

## 4. Systeme mit speziellen Eigenschaften

In manchen Fällen, für spezielle Klassen von Matrizen, lassen sich lineare Gleichungssysteme einerseits stabiler und andererseits mit weniger Aufwand behandeln. Außerdem treten die im folgenden besprochenen Klassen von Matrizen in den Anwendungen und in anderen Bereichen der Numerik häufig auf, sodaß sich eine gesonderte Untersuchung dieser Matrixtypen lohnt.

**4.1. Hermitesche positiv definite Matrizen: Die Cholesky–Zerlegung.** Sei $A \in \mathrm{Herm}(n)$ (bzw. für reelle $A$ sei $A \in \mathrm{Sym}(n)$) und positiv definit — siehe 1.2.12 und 1.2.13. Dann kann man versuchen, eine Dreieckszerlegung zu konstruieren, die von der Form

$$A = LL^*$$

ist. In diesem Abschnitt werden wir untersuchen, ob das immer funktioniert und welche Stabilitätseigenschaften diese Zerlegung besitzt. Zuvor müssen wir jedoch noch die Eigenschaften von hermitesch positiv definiten Matrizen analysieren.

**Proposition 4.1.1.** *Sei $A \in \mathrm{Herm}(n)$ eine hermitesche positiv definite Matrix. Dann gilt*

1. *Für jede Matrix $B \in \mathbb{K}^{n\times m}$, $n \geq m$ mit $\operatorname{rg} B = m$ gilt $B^*AB$ ist hermitesch positiv definit.*
2. *Alle Eigenwerte von $A$ sind reell und positiv.*
3. *Die Determinante von $A$ ist reell und positiv.*
4. *Jede Hauptuntermatrix von $A$ ist wieder hermitesch positiv definit.*
5. *Eigenvektoren zu verschiedenen Eigenwerten sind orthogonal.*
6. *Eine Matrix ist genau dann positiv definit, wenn alle Hauptminoren positiv sind.*

Beweis. (1) $(B^*AB)^* = B^*A^*B = B^*AB$ und daher ist die Matrix hermitesch. Sie ist positiv definit, weil für jeden Vektor $x \neq 0$ auch $Bx \neq 0$ gilt ($B$ hat vollen Rang), und daher ist $x^*(B^*AB)x = (Bx)^*A(Bx) > 0$, weil $A$ positiv definit ist.

(2) Sei $x \neq 0$ und $Ax = \lambda x$. Dann gilt $0 < x^*Ax = \lambda x^*x = \lambda|x|_2^2$. Daher ist $\lambda$ positiv.

(3) Die Matrix $A$ läßt sich, da sie hermitesch ist, orthogonal diagonalisieren. $D = O^\top AO$ mit einer orthogonalen Matrix $O$ und der Diagonalmatrix $D = \operatorname{diag}(\lambda_1, \ldots, \lambda_n)$ gebildet aus den Eigenwerten von $A$. Also gilt $\det A = \det O \det D \det O^\top = \det D = \prod_{i=1}^n \lambda_i > 0$ nach 2.

(4) Sei $B \in \mathbb{K}^{n\times m}$ so gewählt, daß in jeder Spalte genau eine und in jeder Zeile höchstens eine Eins vorkommt; alle anderen Einträge seien Null. Dann besteht $B^*AB$ genau aus der Hauptuntermatrix von $A$, die aus den $m$ Zeilen und Spalten besteht für die in den entsprechenden Zeilen von $B$ eine Eins vorkommt. Nach 1 ist diese Matrix hermitesch positiv definit.

(5) Seien $\lambda_1$ und $\lambda_2$ zwei verschiedene Eigenwerte von $A$ und $x_1$ und $x_2$ dazugehörige Eigenvektoren. Dann gilt

$$\lambda_2\langle x_1, x_2\rangle = \langle x_1, Ax_2\rangle = \langle Ax_1, x_2\rangle = \overline{\lambda_1}\langle x_1, x_2\rangle = \lambda_1\langle x_1, x_2\rangle$$

und damit $(\lambda_2 - \lambda_1)\langle x_1, x_2\rangle = 0$. Weil $\lambda_2 \neq \lambda_1$ gilt, ist $\langle x_1, x_2\rangle = 0$.

(6) Daß die Hauptminoren einer hermiteschen positiv definiten Matrix positiv sind, folgt aus 3 und 4. Die Umkehrung ist ein bekannter Satz der linearen Algebra, das *Hauptminorenkriterium*.

□

Wenden wir uns nun dem Problem zu, die Matrix $A$ in Dreiecksfaktoren zu zerlegen. Beginnen wir damit, zu untersuchen, wie der erste Schritt einer Gaußelimination aussähe, würden wir ihn auf die Matrix $A$ anwenden, und nehmen wir dabei an, daß $A_{11} = 1$:

$$A = \begin{pmatrix} 1 & v^* \\ v & \bar{A} \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ v & \mathbb{I} \end{pmatrix} \begin{pmatrix} 1 & v^* \\ 0 & \bar{A} - vv^* \end{pmatrix}.$$

Der Gaußsche Algorithmus würde jetzt im nächsten Schritt Nullen in der zweiten Spalte erzeugen. Wollen wir allerdings eine symmetrische Zerlegung erreichen, dann wäre es günstig, bereits jetzt wieder zu einer symmetrischen Form zurückzukehren und statt in der zweiten Spalte in der ersten Zeile Nullen zu generieren:

$$\begin{pmatrix} 1 & v^* \\ 0 & \bar{A} - vv^* \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & \bar{A} - vv^* \end{pmatrix} \begin{pmatrix} 1 & v^* \\ 0 & \mathbb{I} \end{pmatrix}.$$

Man bemerke, daß die obere Dreiecksmatrix, die von rechts multiplizierend die erste Zeile zu Null setzt, genau die adjungierte Matrix zu der unteren Dreiecksmatrix ist, die die Nullen in der ersten Spalte erzeugt. Zusammengefaßt ergibt sich die Rechnung

$$A = \begin{pmatrix} 1 & 0 \\ v & \mathbb{I} \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & \bar{A} - vv^* \end{pmatrix} \begin{pmatrix} 1 & v^* \\ 0 & \mathbb{I} \end{pmatrix}.$$

Die Grundidee der Cholesky–Zerlegung ist, diesen Prozeß (etwas verallgemeinert) fortzusetzen bis die mittlere Matrix in die Einheitsmatrix verwandelt ist.

Sei $A$ jetzt beliebig hermitesch positiv definit. Dann gilt $A_{11} > 0$ nach Proposition 4.1.1.6 und wir können einen ähnlichen Reduktionsschritt wie zuvor durchführen, wenn wir jeweils einen Faktor $\alpha := \sqrt{A_{11}}$ zu jeder der dreieckigen Transformationsmatrizen dazuschlagen:

$$\begin{aligned} A &= \begin{pmatrix} A_{11} & v^* \\ v & \bar{A} \end{pmatrix} = \\ &= \begin{pmatrix} \alpha & 0 \\ \frac{1}{\alpha}v & \mathbb{I} \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & \bar{A} - \frac{1}{A_{11}}vv^* \end{pmatrix} \begin{pmatrix} \alpha & \frac{1}{\alpha}v^* \\ 0 & \mathbb{I} \end{pmatrix} = L_1 A_1 L_1^*. \end{aligned}$$

Ist die Teilmatrix $\bar{A} - \frac{1}{A_{11}}vv^*$ positiv definit, so kann das Verfahren wiederholt werden. Das ist aber klar, weil $\bar{A} - \frac{1}{A_{11}}vv^*$ die untere $(n-1)\times(n-1)$ Hauptuntermatrix der Matrix $L_1^{-1}AL_1^{-1*}$ ist, welche nach Proposition 4.1.1.1 und 4.1.1.4 wieder hermitesch positiv definit ist. Im nächsten Schritt wird also eine Matrix $L_2$ gebildet, sodaß in $L_1L_2A_2L_2^*L_1^*$ alle Einträge

der ersten beiden Zeilen und Spalten (mit Ausnahme der Diagonalelemente) gleich Null sind. Das Verfahren wird fortgesetzt und es endet mit einer Zerlegung der Gestalt

$$A = \underbrace{L_1 L_2 \dots L_n}_{L} \mathbb{I} \underbrace{L_n^* \dots L_2^* L_1^*}_{L^*} = LL^*,$$

wobei $L_{ii} > 0$ gilt für alle $i$, und $L$ ist eine untere Dreiecksmatrix. Diese Abhandlung fassen wir als Ergebnis im folgenden Satz zusammen:

**Theorem 4.1.2** (Cholesky–Zerlegung)**.** *Jede hermitesche positiv definite Matrix $A \in \mathrm{Herm}(n)$ besitzt eine eindeutige Zerlegung der Form*

$$A = LL^* \quad mit \quad L_{ii} > 0,$$

*mit einer unteren Dreiecksmatrix $L$. Diese Zerlegung heißt die* Cholesky–Zerlegung *von $A$. Ist $A$ reell, so ist auch $L$ reell.*

Beweis. Der oben vorgestellte Algorithmus konstruiert solch eine Zerlegung, und aus den Aussagen von Proposition 4.1.1 folgt, daß alle berechneten Quadratwurzeln aus positiven Zahlen gezogen werden und daß niemals durch Null dividiert wird. □

Bei der Implementation der Cholesky–Faktorisierung wird aus Symmetriegründen immer nur die Hälfte der Matrix $A$ benötigt, welche im Lauf der Berechnung mit der Matrix $L$ überschrieben werden kann. Es wird auch nur die Hälfte der Rechenoperationen gebraucht. Im folgenden Algorithmus sei die $n \times n$–Matrix $A$ durch ihre untere Hälfte repräsentiert.

**Algorithmus 4.1.3.** *Cholesky–Faktorisierung*

$L = A$
**for** $k = 1$ **to** $n$ **do**
  **for** $j = k + 1$ **to** $n$ **do**
    $L_{j:n,j} = L_{j:n,j} - L_{j:n,k} L_{jk} / L_{kk}$
  **done**
  $L_{k:n,k} = L_{k:n,k} / \sqrt{L_{kk}}$
**done**

Wieder ist der Hauptaufwand des Algorithmus in der innersten Schleife zu suchen. Im $j$–ten inneren Schritt ist der Aufwand etwa $2(n - j)$ Flops. Insgesamt ergibt das

$$\sum_{k=1}^{n} \sum_{j=k+1}^{n} 2(n-j) \sim 2 \sum_{k=1}^{n} \sum_{j=1}^{k} j \sim \sum_{k=1}^{n} k^2 \sim \tfrac{1}{3} n^3 \text{ Flops.}$$

Die Cholesky–Faktorisierung benötigt damit etwa halb so viele Operationen wie die $LR$–Zerlegung, was durch die Ausnützung der Symmetrie nicht weiter verwunderlich ist.

Sie bietet aber noch einen weiteren großen Vorteil gegenüber dem Spaltenpivotverfahren. Die Cholesky–Zerlegung ist immer stabil. Durch die Symmetrie der Zerlegung wird die Matrix $L$ nie sehr groß. Man kann zum Beispiel zeigen, daß $\|L\|_2 = \sqrt{\|A\|_2}$. In den anderen $p$–Normen unterscheidet sich $\|L\|$ niemals um mehr als $\sqrt{n}$ von $\sqrt{\|A\|}$. Die Stabilität wird darüberhinaus erreicht ohne Pivotsuche. Das hängt damit zusammen, daß der betragsgrößte Eintrag jeder hermiteschen positiv definiten Matrix immer auf der Hauptdiagonalen liegt. Eine Fehleranalyse des Cholesky–Verfahrens führt zu dem folgenden Satz.

**Theorem 4.1.4.** *Sei $A \in \mathrm{Herm}(n)$ eine hermitesch positiv definite Matrix, und sei eine Cholesky–Zerlegung von $A$ mittels Algorithmus 4.1.3 berechnet. Für alle* eps*, die klein genug sind, läuft der Algorithmus garantiert zu Ende, und die berechnete Matrix $\tilde{L}$ erfüllt*

$$\tilde{L}\tilde{L}^* = A + E \quad mit \quad \frac{\|E\|}{\|A\|} = O(\mathsf{eps}).$$

*Der relative Fehler in der Berechnung der Zerlegung ist also von der Größenordnung der Maschinengenauigkeit* eps.

BEWEIS. Z.B. in [**Wilkinson 1968**] □

Man beachte, daß der Satz nur eine Aussage über die Güte der Zerlegung, also über das Produkt $\tilde{L}\tilde{L}^*$ trifft. Über die Matrix $\tilde{L}$ selbst ist in diesem Theorem keine Aussage getroffen. Es gilt nämlich, daß der relative Fehler in $L$

$$\frac{\|\tilde{L} - L\|}{\|L\|} = O(\kappa(A)\,\mathsf{eps})$$

von der Kondition $\kappa(A)$ der Matrix $A$ abhängt. Nur das Produkt $LL^*$ erfüllt die nette Stabilitätseigenschaft. Im schlecht konditionierten Fall heben sich beim Ausmultiplizieren die Rundungsfehler gegenseitig automagisch auf.

Die gute Fehlerschranke erlaubt es auch, lineare Gleichungssysteme auf stabile Weise zu lösen:

**Theorem 4.1.5.** *Die berechnete Lösung $\tilde{x}$ eines linearen Gleichungssystemes $Ax = b$ mit hermitesch positiv definierter Koeffizientenmatrix $A$ erfüllt*

$$(A + \Delta A)\tilde{x} = b, \quad \textit{wobei} \quad \frac{\|\Delta A\|}{\|A\|} = O(\mathsf{eps})$$

*für ein $\Delta A \in \mathbb{K}^{n\times n}$.*

BEWEIS. Das folgt aus Theorem 4.1.4 und den Ergebnissen über dreieckige Gleichungssysteme. □

**4.1.1. Die modifizierte Cholesky–Zerlegung, $LDL$–Zerlegung.** Wenn $A$ hermitesch aber nicht positiv definit ist, so kann die Cholesky–Zerlegung nicht mehr funktionieren, da es dann einen Schritt gibt, für den $A_{ii} <= 0$, sodaß entweder das Wurzelziehen oder das Dividieren fehlschlägt. Möchte man in diesem Fall eine ähnliche Zerlegung wie die Cholesky–Faktorisierung machen, so hat man grundsätzlich zwei Möglichkeiten:

$$A = LL^* - D \quad \text{Modifizierte Cholesky–Zerlegung}$$
$$A = LDL^* \quad LDL\text{–Zerlegung.}$$

**Die $LDL$–Zerlegung.** In diesem Fall wird in den symmetrischen Eliminationsschritten $A$ nicht zur Einheitsmatrix reduziert sondern zu einer Diagonalmatrix, und die Matrix $L$ ist dann eine normierte Dreiecksmatrix. Es muß allerdings, ähnlich wie beim Gaußschen Algorithmus, keine Zerlegung dieser Form existieren. Zusätzlich muß man wie dort, auch um numerische Stabilität zu erreichen, ein Pivotverfahren verwenden. Um die Hermitizität nicht aufzugeben, müssen dabei aber die Permutationsmatrizen von links und von rechts multipliziert werden. Das Verfahren (ein Diagonalpivotverfahren) führt dann zu einer Zerlegung der Form

$$PAP^\top = LDL^*$$

mit $P \in \mathfrak{S}(n)$ eine Permutationsmatrix und $L \in \mathcal{L}_{\text{norm}}(n)$ eine normierte untere Dreiecksmatrix. Der Vorteil dieser Zerlegung besteht erstens darin, daß durch die Symmetrie die exponentielle Instabilität des Gaußschen Algorithmus verschwindet, da alle möglicherweise großen Werte in $D$ landen. Die Faktoren $L$ haben ja dieselbe Größenordnung wie $A$. Das Auflösen eines diagonalen Gleichungssystemes mit $D = \operatorname{diag}(d_1, \ldots, d_n)$ ist unabhängig von den Größenordnungen der $d_i$ gutartig. Der Speicher– und Rechenbedarf bei der $LDL$–Zerlegung ist durch die Symmetrie von $A$ etwa halb so groß wie für eine $LR$-Zerlegung. Die Diagonalelemente von $A$ können mit der Matrix $D$ überschrieben werden, die Elemente von $L$ können, da $L$ normiert ist, unterhalb der Hauptdiagonalen von $A$ abgespeichert werden.

Wegen der Symmetrieeigenschaften von $A$ muß man das obere Dreieck von $A$ nicht im Speicher ablegen. Unglücklicherweise muß nicht jede hermitesche Matrix $A$ eine $LDL$–Zerlegung besitzen, wie man an dem Beispiel

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

sehen kann. Wenn also von einem bestimmten Moment an alle Hauptdiagonalelemente, die noch zur Auswahl stehen gleich Null sind, dann bricht der Zerlegungsalgorithmus ab. In diesem Fall muß man ein Offdiagonalelement als Pivot wählen. Das zerstört aber die Hermitizität der Zerlegung, und das Verfahren führt zu einer normalen $LR$–Faktorisierung.

**Die modifizierte Cholesky–Faktorisierung.** Bei diesem Verfahren wird zu $A$ eine Diagonalmatrix $D$ addiert, die $A + D$ positiv definit macht. Um eine *modifizierte Cholesky–Faktorisierung* $A = LL^* - D$ mit $D > 0$ zu berechnen, gibt es nicht nur einen einzigen vernünftigen Algorithmus. Abhängig von der Anwendung sind verschiedene Methoden zur Konstruktion von $D$ sinnvoll. Das einfachste Verfahren ist, während der Ausführung der Cholesky–Faktorisierung $D$ mitzukonstruieren. Das Verfahren bietet auch einen einfachen Test für die positiv Definitheit einer hermiteschen Matrix.

**Algorithmus 4.1.6.** *Modifizierte Cholesky–Faktorisierung*
$L = A$, $D = \mathbb{O}$, $\varepsilon =$ *gewünschter* Singularitätsabstand von $A + D$
*ist_pdef* = `TRUE ; Ist` $A$ `positiv definit?`
**for** $k = 1$ **to** $n$ **do**
  **if** $L_{kk} \leq 0$ **then**
    $D_{kk} = -L_{kk} + \varepsilon$
    $L_{kk} = \varepsilon$
    *ist_pdef* = `FALSE`
  **endif**
  **for** $j = k + 1$ **to** $n$ **do**
    $L_{j:n,j} = L_{j:n,j} - L_{j:n,k} L_{jk} / L_{kk}$
  **done**
  $L_{k:n,k} = L_{k:n,k} / \sqrt{L_{kk}}$
**done**

Dieser Algorithmus ist nicht für jede Matrix stabil, und es gibt bessere Methoden zur Konstruktion einer modifizierten Cholesky–Zerlegung, doch das geht über den Rahmen dieser Vorlesung hinaus.

**4.1.2. unvollstaendige Cholesky–Zerlegung.** Eigentlich nicht in dieses Kapitel sondern zu den Vorkonditionierverfahren aus Kapitel 9 gehört die unvollständige Cholesky–Zerlegung. Das Problem beginnt damit, daß für dünnbesetzte Matrizen der Choleskyfaktor $L$ im allgemeinen nicht mehr dünn besetzt ist. Besteht Speicherknappheit ist es oft nur mehr möglich eine Matrix $\hat{L}$ zu konstruieren, die ähnlich wenige Nicht-Null-Elemente enthält wie $A$. Natürlich ist die Matrix $M = \hat{L}\hat{L}^*$ meist deutlich von $A$ verschieden, und daher kann man mit $\hat{L}$ auch nicht direkt zur Lösung von Gleichungssystemen verwenden. Sehr gut verwendbar ist $\hat{L}$ allerdings zur Vorkonditionierung von Iterationsverfahren, wie in Kapitel 9 besprochen.

Zum Abschluß des Abschnittes über die Cholesky–Faktorisierung sei noch bemerkt, daß alle Theoreme und Algorithmen anstelle für untere Dreiecksmatrizen $L$ auch für obere Dreiecksmatrizen $R$ formuliert werden können, und somit auch eine Zerlegung der Form

$$A = R^* R$$

existiert mit der Beziehung $L^* = R$. Die dazu nötigen Änderungen in den obigen Abschnitten seien als Übungsaufgabe dem Einzelnen überlassen.

**4.2. Tridiagonale Matrizen.** Für tridiagonale Matrizen ist das Spaltenpivotverfahren zur Lösung eines Gleichungssystemes besonders gut geeignet. Einerseits werden Matrizen $L$ mit einem Band unterhalb der Hauptdiagonale und $R$ mit höchstens zwei Bändern oberhalb der Hauptdiagonale erzeugt, und andererseits ist der Algorithmus in diesem Fall besonders gutartig. Es gilt nämlich

$$\rho(A) \leq 2$$

und damit folgender Satz.

**Theorem 4.2.1.** *Sei $T$ eine tridiagonale Matrix, und sei $\tilde{x}$ die mit Hilfe des Spaltenpivotverfahrens berechnete Lösung des Gleichungssystemes $Tx = b$. Dann erfüllt $\tilde{x}$ ein benachbartes Gleichungssystem $(T + F)\tilde{x} = b$ mit*

$$|F| \leq 13n \,\mathsf{eps}\, |T|.$$

Der totale Rechenaufwand zur Auflösung eines tridiagonalen Gleichungssystemes beträgt $9(n-1)$ Flops und ist damit **linear** in der Dimension der Matrix!

Berechnet man eine Cholesky–Zerlegung einer hermiteschen positiv definiten Tridiagonalmatrix, dann hat die entstehende Matrix $L$ nur ein Band unterhalb der Hauptdiagonalen und der Rechenaufwand ist wiederum $O(n)$.

**4.3. Bandmatrizen.** Gaußelimination mit Spaltenpivotverfahren kann darauf spezialisiert werden, um Bandstruktur in der Koeffizientenmatrix $A$ auszunützen. Die Bandstrukturen von $L$ und $R$ sind allerdings nicht ganz einfach zu beschreiben. Zeilenvertauschungen zerstören nämlich die Nullen oberhalb der Bänder und erweitern damit die Bandbreite gemäß dem folgenden Satz.

**Theorem 4.3.1.** *Angenommen $A \in \mathbb{K}^{n\times n}$ ist nichtsingulär und hat obere und untere Bandbreiten $p$ und $q$. Wenn mittels Spaltenpivotverfahren $L$ und $R$ berechnet werden, dann hat $R$ obere Bandbreite $p + q$. In jeder Spalte von $L$ sind höchstens $p + 1$ Elemente ungleich Null.*

Das Vergrößern der Bandstruktur wird oft als Grund dafür genommen, so lange wie möglich ohne Zeilenvertauschung auszukommen und nur dann ein Pivotelement außerhalb der Diagonale zu wählen, wenn das Diagonalelement gleich Null ist oder sehr kleinen Absolutbetrag hat. Der entstehende Algorithmus ist dann zwar nicht stabil aber der Speicherbedarf bleibt gering. Mit Hilfe der Nachiteration kann aber, wenn die Kondition von $A$ gut genug ist, die Genauigkeit der Lösung im Nachhinein verbessert werden.

Bei der Cholesky–Faktorisierung hermitesch positiv definiter Bandmatrizen tritt das Problem der Bandbreitenvergrößerung nicht auf, da keine Pivotsuche nötig ist. Der Cholesky–Faktor $L$ hat genausoviele Bänder unterhalb der Hauptdiagonalen wie $A$. Beide Matrizen benötigen gleich viel Speicher, und der Rechenaufwand ist $O(np^2)$, wenn $p$ die Bandbreite von $A$ ist. Der Algorithmus selbst unterscheidet sich von Algorithmus 4.1.3 nur dadurch, daß Multiplikationen für Elemente $A_{ij}$ mit $|i - j| > m$ nicht ausgeführt werden.

**4.4. Dünnbesetzte Matrizen.** Für Gleichungssysteme mit dünnbesetzten Matrizen ist das Spaltenpivotverfahren meist nicht einsetzbar. Grund ist der Speicherbedarf der Faktoren $L$ und $R$. Sind die Matrizen groß genug, dann ist die Minimierung des Speicherbedarfs wichtiger als die Stabilität der Zerlegung. Die Wahl der Permutationsmatrix $P$ kann auf die Größe des Speicherbedarfs großen Einfluß haben, wie das folgende Extrembeispiel zeigt.

**Beispiel 4.4.1.** *Sei $A$ die Pfeilmatrix*

$$A = \begin{pmatrix} 1 & * & * & \cdots & * \\ * & 2 & & & \\ * & & 3 & & \\ \vdots & & & \ddots & \\ * & & & & n \end{pmatrix}.$$

*Ist $A$ symmetrisch, dann kann man die Cholesky–Faktorisierung berechnen. Der Choleskyfaktor $L$ ist dann voll besetzt (ebenso die Faktoren $L$ und $R$ einer LR-Zerlegung für unsymmetrisches $A$):*

$$L = \begin{pmatrix} * & & & & \\ * & * & & & \\ * & * & * & & \\ \vdots & \vdots & \vdots & \ddots & \\ * & * & * & \cdots & * \end{pmatrix}.$$

*Wählt man vor der Zerlegung allerdings die Permutation*

$$PAP^\top = \begin{pmatrix} n & & & & * \\ & \ddots & & & \vdots \\ & & 3 & & * \\ & & & 2 & * \\ * & \cdots & * & * & 1 \end{pmatrix},$$

*so sieht der Choleskyfaktor (oder $L$ bzw. $R$) viel dünner besetzt aus. Er benötigt ebensoviel Speicher wie $A$:*

$$L = \begin{pmatrix} * & & & & \\ & * & & & \\ & & * & & \\ & & & \ddots & \\ * & * & * & \cdots & * \end{pmatrix}.$$

Diejenigen Elemente, die in $L$ ungleich Null in $A$ aber gleich Null sind, werden mit *Fill-In* bezeichnet. Diesen Fill-In zu minimieren gilt es in allgemeinen Verfahren zur Faktorisierung dünnbesetzter Matrizen. Dabei verwandelt man die Matrix $A$ in eine Gestalt, die der unteren Pfeilform möglichst ähnlich ist (multifrontale Elimination). Der Weg dazu ist Graphentheorie und geht über den Rahmen dieser Vorlesung hinaus.

Meist werden Gleichungssysteme mit dünnbesetzten Matrizen jedoch nicht direkt durch Zerlegungen gelöst sondern mit iterativen Verfahren behandelt. Diese werden im Kapitel 9 besprochen.

## 5. Software

Die in diesem Unterkapitel gegebenen Erklärungen sind als Hinweise gedacht. Genaue Beschreibungen sprengen den Bereich des Skriptums und können unter den jeweiligen Funktionsnamen in den Handbüchern gefunden werden.

**5.1. LAPACK.** In diesem Abschnitt werden die zu allen besprochenen Algorithmen gehörenden LAPACK Routinen aufgelistet. LAPACK Routinen können am einfachsten aus FORTRAN Programmen aufgerufen werden; jedoch auch aus C und C++ sind diese Unterprogramme verwendbar.

Die Namen der LAPACK Funktionen sind folgendermaßen aufgebaut:

*pmmfff*,

wobei $p$ für die Genauigkeit steht und entscheidet, ob reell oder komplex gerechnet wird. Hier steht `s` für single precision reell, `d` für double precision reell, `c` für single precision komplex und `z` für double precision komplex. Die beiden folgenden Buchstaben $mm$ stehen für den Matrixtyp, für den die Funktion ausgeführt werden soll. Die Bedeutung kann Tabelle 3.2 entnommen werden. Schließlich bezeichnet die zwei– oder dreibuchstabige Abkürzung *fff* die

| $mm$ | Matrixtyp |
|---|---|
| `ge` | Allgemeine Matrix (auch rechteckig) |
| `gg` | Allgemeine Matrix bei allgemeinen Eigenwertproblemen |
| `ge` | Allgemeine Bandmatrix |
| `ge` | Allgemeine Tridiagonalmatrix |
| `he` | Hermitesche Matrix |
| `ge` | Hermitesche Matrix (gepackt abgespeichert – nur ein Dreieck) |
| `hs` | Untere Hessenberg–Matrix |
| `ge` | Untere Hessenberg–Matrix bei allgemeinen Eigenwertproblemen |
| `ge` | Orthogonale Matrix |
| `ge` | Orthogonale Matrix (gepackt abgespeichert) |
| `po` | Hermitesche positiv definite Matrix |
| `pp` | Hermitesche positiv definite Matrix (gepackt abgespeichert) |
| `pb` | Hermitesche positiv definite Bandmatrix |
| `pt` | Hermitesche positiv definite Tridiagonalmatrix |
| `sy` | Hermitesche (symmetrische) Matrix |
| `sp` | Hermitesche (symmetrische) Matrix (gepackt abgespeichert) |
| `sb` | Hermitesche (symmetrische) Bandmatrix |
| `st` | Hermitesche (symmetrische) Tridiagonalmatrix |
| `tr` | Dreiecksmatrix oder Block–Dreiecksmatrix |
| `tp` | Dreiecksmatrix (gepackt abgespeichert) |
| `tg` | Dreiecksmatrix bei allgemeinen Eigenwertproblemen |
| `tb` | Dreiecksbandmatrix |
| `un` | Unitäre Matrix |
| `up` | Unitäre Matrix (gepackte Speicherung) |
| `bd` | Bidiagonalmatrix |

TABELLE 3.2. LAPACK Matrixtypen

Funktion, die ausgeführt werden soll. Die Routinen, die für dieses Kapitel relevant sind, werden im folgenden aufgeführt.

**5.1.1. Auflösen von dreieckigen Gleichungssystemen.** Dazu dienen die Funktionen $p$`tptrs`. Die genauen Aufrufparameter kann man im LAPACK–Handbuch finden, aber die Funktionen lassen dem Benutzer die Wahl zwischen oberen und unteren Dreiecksmatrizen, ob er das Problem $Ax = b$ oder das Problem $A^*x = b$ lösen will, und bieten weiters

die Möglichkeit, mehrere lineare Gleichungen mit der gleichen Koeffizientenmatrix aber mit verschiedenen rechten Seiten in einem Aufruf zu lösen.

**5.1.2. Berechnung der $LR$–Zerlegung einer Matrix.** Die Funktionen *pmm*`trf` berechnen eine $LR$–Zerlegung einer Matrix mit Hilfe des Spaltenpivotverfahrens. Der Funktion übergeben wird eine Matrix einer bestimmten Gestalt, die meist mit den Faktoren $L$ und $R$ überschrieben wird. Die Permutationsmatrix wird als Vektor von ganzen Zahlen zurückgegeben.

**5.1.3. Berechnung der Cholesky–Zerlegung einer Matrix.** Für Berechnung der Cholesky–Faktorisierung symmetrischer oder hermitescher Matrizen sind ebenfalls die Funktionen *pmm*`trf` zuständig. Falls $A$ nicht positiv definit ist, kann alternativ auch eine $LDL$–Zerlegung bestimmt werden. Auch hier wird die Eingabematrix mit dem berechneten Faktor überschrieben.

**5.1.4. Auflösen eines Gleichungssystemes.** LAPACK stellt zur Lösung von linearen Gleichungssystemen zwei verschiedene Typen von Funktionen zur Verfügung.

**Simple driver:** Diese einfach strukturierten Funktionen (*pmm*`sv`) lösen Gleichungssysteme intern mit einer Form der Dreieckszerlegung, die dem jeweiligen Matrixtyp am besten angepaßt erscheint. Alle bieten die Möglichkeit, gleichzeitig mehrere Systeme mit derselben Koeffizientenmatrix zu lösen, und liefern zusätzlich zur Lösung nur noch eventuelle Abbruchinformationen, die auf numerische Singularität der Koeffizientenmatrix hindeuten.

**Expert driver:** Die weitergehenden mit mehr Aufrufparametern versehenen und komplizierter zu bedienenden Funktionen *pmm*`svx` liefern zusätzlich zur Lösung noch Informationen, die man zur Fehlerrechnung verwenden kann, wie Fehlerschranken für die Lösung und eine Schätzung der Konditionszahl der Koeffizientenmatrix. Außerdem werden Nachiterationsschritte verwendet, um die Lösung bis zur Grenzgenauigkeit zu verbessern.

Weitergehende Funktionsklassen berechnen Lösungen zu linearen Gleichungssystemen aus zuvor bestimmten Dreieckszerlegungen. Dies sind die Funktionen *pmm*`trs`.

**5.1.5. Konditionszahlen.** Schätzwerte für die Konditionszahl einer Matrix kann man nicht nur aus den Expert drivers für Gleichungslösung gewinnen sondern auch durch einfache Funktionsaufrufe direkt erhalten. Dazu dienen die LAPACK Routinen *pmm*`con`. Bei diesen Funktionen kann man sich entscheiden, welche der beiden Normen $\kappa_\infty(A)$ oder $\kappa_1(A)$ geschätzt werden soll. Die Norm $\kappa_2(A)$ kann direkt aus einer Singulärwertzerlegung (Kapitel 4 Abschnitt 2.3) berechnet werden. Die Routinen zur Berechnung von Singulärwertzerlegungen werden in Kaptitel 4 forgestellt.

**5.2. MATLAB.** Auch in MATLAB sind die meisten Algorithmen implementiert. Die Funktionsnamen und einige Erklärungen werden in diesem Abschnitt gegeben.

Mit Hilfe des MATLAB–Aufrufes

```
[L,R,P] = lu(A)
```

wird die Matrix `A` $LR$–faktorisiert. Die Matrizen $L$, $R$ und $P$ halten danach die Dreiecksfaktoren und die Permutationsmatrix. Verwendet man in MATLAB den Aufruf

```
[L,R] = lu(A)
```

so liefert MATLAB keine naive $LR$–Zerlegung, wie man vielleicht annehmen könnte. Es erzeugt eine pivotisierte Zerlegung in eine obere Dreiecksmatrix $R$ und eine „psychologisch“ untere Dreiecksmatrix $L$, die das Produkt aus der Permutationsmatrix und der unteren Dreiecksmatrix ist.

Cholesky–Zerlegungen werden in MATLAB mit Hilfe der Funktion

```
L = chol(A)
```

berechnet, und das Lösen linearer Gleichungssysteme ist besonders einfach durch den Operator \.

```
x = A\b
```

löst das lineare Gleichungssystem `Ax=b` mit Hilfe einer geeigneten Matrixzerlegung. Konditionszahlen schließlich können mit der Funktion `cond` bestimmt werden.

KAPITEL 4

# Numerische Lineare Algebra II: Über– und unterbestimmte lineare Gleichungssysteme

## 1. Grundlagen

In diesem Abschnitt behandeln wir lineare Gleichungssysteme mit rechteckiger Koeffizientenmatrix $A \in \mathbb{K}^{n\times m}$. Zuerst werden wir den wesentlich wichtigeren Fall des überbestimmten Systemes ($n > m$) untersuchen. Zwei wichtige neue Matrixdekompositionen werden zur Behandlung des Problemes notwendig sein.

**1.1. Problemstellung, Ausgleichsprobleme, Fitten.** Sei $A \in \mathbb{K}^{n\times m}$ und $b \in \mathbb{K}^n$. Ist $A$ nicht quadratisch, dann ist a priori nicht klar, was unter *der Lösung* des linearen Gleichungssystemes $Ax = b$ zu verstehen sein soll. Ist nämlich $n > m$, so kann die Lösungsmenge des Gleichungssystemes leer sein, da $\dim \operatorname{im}(A) = m$ und eine Lösung nur dann existiert, wenn $b \in \operatorname{im}(A)$. Ist hingegen $n < m$, so existieren im allgemeinen unendlich viele Lösungen, und es bleibt zu entscheiden, welche als *die* Lösung gelten soll. Um die beiden Probleme zu entscheiden, ist es günstig, Anwendungsprobleme zu untersuchen, bei denen überbestimmte Systeme auftreten.

Eines dieser Anwendungsprobleme ist in Kapitel 2 Abschnitt 2.5 vorgestellt worden, die Computertomographie. In diesem Abschnitt ist ein Gleichungssystem konstruiert worden, das eigentlich eine Lösung haben müßte, wären keine Vereinfachungen und keine Meßfehler bei der Bestimmung der Koeffizientenmatrix und der rechten Seite gemacht worden. Formuliert man das Gleichungssystem etwas um, so erhält man

$$r = A - b = 0.$$

Ist das Gleichungssystem also erfüllt, so verschwindet das Residuum. Hat das Gleichungssystem hingegen keine Lösung, so existiert kein $x$, sodaß das Residuum verschwindet. In diesem Fall ist es vernünftig zu verlangen, daß das Residuum möglichst klein gemacht werden soll.

**Definition 1.1.1.** *Der Vektor $x \in \mathbb{K}^m$ heißt Lösung des überbestimmten Gleichungssystemes $Ax = b$ mit $A \in \mathbb{K}^{n\times m}$, $b \in \mathbb{K}^n$, falls*

$$\|Ax - b\| = \min_{y\in\mathbb{K}^m} \|Ay - b\|,$$

*also wenn das Residuum in einer gewählten Norm minimal ist.*

*Solch eine Minimallösung heißt* Tschebyscheff–Approximation *für die $\infty$–Norm $\| \quad \|_\infty$ und* Kleinste–Quadrate–Lösung *für die 2–Norm $\| \quad \|_2$.*

Die Verwendung der 2–Norm ist üblicher, da dann die Funktion $\|Ax - b\|_2^2$, die zu minimieren ist, differenzierbar ist und als quadratische Funktion eine lineare Ableitungsfunktion besitzt.

Tschebyscheff Approximation wird z.B. durch iterierte gewichtete Kleinste–Quadrate–Approximation bestimmt, wobei bei jedem Iterationsschritt die jeweils betragsgrößte Komponente zunehmend stärker gewichtet wird.

**1.1.1. Skalierung.** Bevor man Optimierungsprobleme untersucht, ist es nötig, über die richtige Art zu skalieren zu sprechen. Untersuchen wir die folgenden Beispiele:

**Beispiel 1.1.2.** *Betrachten wir ein Gleichungssystem mit einer* $2 \times 1$*–Matrix.*

$$A = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \quad b = \begin{pmatrix} 2 \\ 3 \end{pmatrix}$$

*Die optimale Lösung ist* $x = \frac{5}{2}$*. In diesem Fall gilt* $\|r\|_2 = \|Ax - b\|_2 = 1/\sqrt{2}$*. Wenn wir allerdings das Gleichungssystem* $Ax = b$ *umskalieren, sagen wir, wir multiplizieren die erste Zeile mit* 100*, dann ändert das an der Bedeutung des Gleichungssystemes* $Ax = b$ *nichts. Bearbeiten wir allerdings das Optimierungsproblem, so verändert sich die Lösung:*

$$A = \begin{pmatrix} 100 \\ 1 \end{pmatrix}, \quad b = \begin{pmatrix} 200 \\ 3 \end{pmatrix},$$

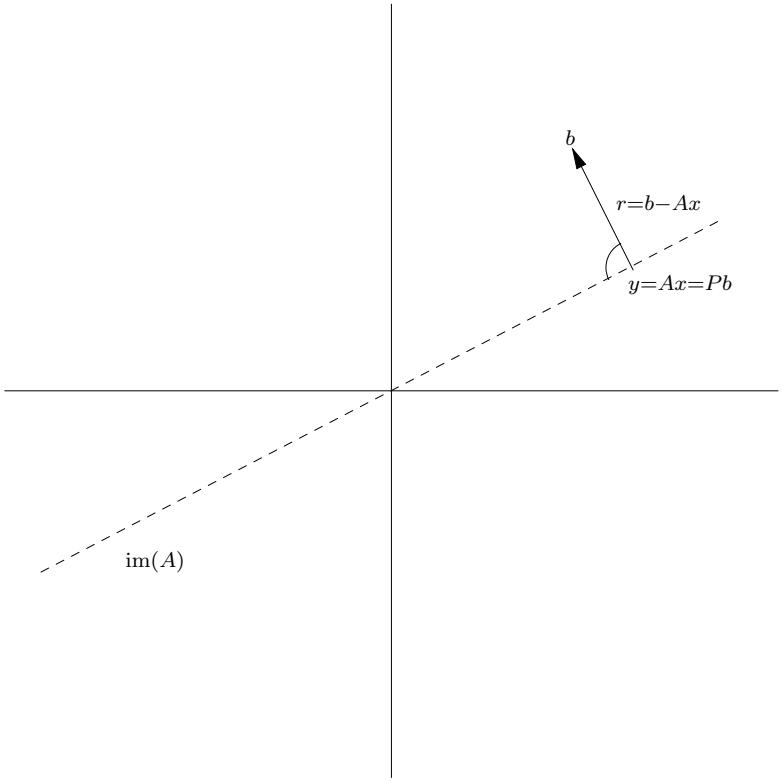
*und* $x \approx 2.0001$*. Stärker gewichtete Gleichungen werden also besser erfüllt. Man muß daher vor der Berechnung alle Gleichungen so skalieren, daß die erwarteten Fehler in etwa dieselbe Größenordnung besitzen.*

## 2. Lineare überbestimmte Systeme

Angenommen, wir haben das System richtig skaliert und untersuchen das überbestimmte Gleichungssystem $Ax = b$ mit $A \in \mathbb{K}^{n \times m}$, $b \in \mathbb{K}^n$, $n > m$ und $\operatorname{rg}(A) = m$. Wie besprochen suchen wir die Kleinste–Quadrate–Lösung $x$ zu diesem System, das heißt wir lösen

$$\min_{x \in \mathbb{K}^m} \|Ax - b\|_2^2.$$

Das ist äquivalent zur Formulierung, die wir vorhin gewählt haben, da $\| \quad \|_2$ nicht negativ und $x \mapsto x^2$ eine monoton wachsende Funktion ist. Aus einer zweidimensionalen Skizze kann man erahnen, was im allgemeinen Fall gilt:




Offensichtlich ist $r$ dann minimal, wenn es orthogonal auf das Bild $\operatorname{im}(A)$ von $A$ steht. Im obigen Bild bezeichnet $P \in \mathbb{K}^{n \times n}$ die Orthogonalprojektion auf $\operatorname{im}(A)$.

**2.1. Normalengleichungen.** Diese Untersuchungen führen direkt zum ersten Lösungsansatz. Wir untersuchen zuerst, wie man die Orthogonalprojektion $P$ berechnen kann: Projiziert man $v$ mittels $P$ orthogonal auf $y \in \mathrm{im}(A)$, so erfüllt $y$ für alle $a \in \mathrm{im}(A)$

$$\langle a, y - v\rangle = a^*(y - v) = 0.$$

Speziell gilt das für eine Basis von $\mathrm{im}(A)$, das sind aber gerade die Spalten von $A$. Weil $y \in \mathrm{im}(A)$ liegt, kann man es schreiben als $y = Ax$. Setzen wir all dies zusammen, so erhalten wir für jede Spalte $a_j$ von $A$

$$a_j^*(Ax - v) = 0,$$

oder zusammengefaßt

$$A^*(Ax - v) = 0, \quad \text{oder} \quad A^*Ax = A^*v.$$

Hat $A$ vollen Rang (wie vorausgesetzt), dann ist $A^*A$ regulär, und wir können $x$ berechnen aus

$$x = (A^*A)^{-1}A^*v,$$

und schließlich ist die Orthogonalprojektion $y$ auf $v$

$$y = A(A^*A)^{-1}A^*v = Pv.$$

Der Orthogonalprojektor auf $\mathrm{im}(A)$ läßt sich also schreiben als

$$P = A(A^*A)^{-1}A^*v.$$

Faßt man alle diese Beobachtungen zusammen, so vermutet man, daß sich die Kleinste–Quadrate–Lösung des überbestimmten Gleichungssystemes $Ax = b$ schreiben lassen sollte als

$$A^*Ax = A^*b.$$

Das ist in der Tat so, wie der folgende Satz zeigt.

**Theorem 2.1.1.** *Sei $A \in \mathbb{K}^{n\times m}$, $b \in \mathbb{K}^n$, und habe $A$ vollen Rang (somit ist $A^*A$ regulär). Dann nimmt $\|Ax - b\|_2$ sein Minimum genau für die Lösung $\hat{x}$ der* Normalengleichungen

$$A^*A\hat{x} = A^*b$$

*an.*

Beweis. Sei $r = b - Ax$, $\hat{r} = b - A\hat{x}$. Weil $\hat{x}$ eine Lösung der Normalengleichungen ist, gilt $A^*\hat{r} = 0$. Daraus wiederum folgt

$$\langle \hat{r} - r, \hat{r}\rangle = \langle A(x - \hat{x}), \hat{r}\rangle = \langle x - \hat{x}, A^*\hat{r}\rangle = 0.$$

Jetzt sind wir in der Lage die 2–Normen zu bestimmen:

$$\|\hat{r}\|_2^2 + \|\hat{r} - r\|_2^2 = \langle \hat{r}, \hat{r}\rangle + \langle \hat{r} - r, \hat{r} - r\rangle = \langle \hat{r}, \hat{r}\rangle - \langle \hat{r} - r, r\rangle =$$
$$= \langle r, r\rangle + \langle \hat{r}, \hat{r} - r\rangle = \|r\|_2^2.$$

Aus dieser Gleichung ist sofort ersichtlich, daß $\|r\|_2 \geq \|\hat{r}\|_2$, und es gilt Gleichheit genau dann, wenn $\hat{r} = r$. Setzt man dies in die Definitionen von $r$ und $\hat{r}$ ein, so erhält man

$$A^*A(x - \hat{x}) = A^*A(r - \hat{r}) = 0,$$

und weil $A^*A$ regulär ist, folgt daß das Minimum bei $x = \hat{x}$ angenommen wird. □

Wie löst man die Normalengleichungen? Zuerst bemerkt man folgendes Resultat:

**Proposition 2.1.2.** *Sei $A \in \mathbb{K}^{n\times m}$, $n \geq m$. Dann gilt*

(1) *$A$ und $A^*A$ haben denselben Rang.*
(2) *Hat $A$ maximalen Rang, so ist $A^*A$ hermitesch positiv definit.*

BEWEIS. (1) Wenn $Ax = 0$, so ist auch $A^*Ax = 0$. Ist umgekehrt $A^*Ax = 0$, so ist $0 = \langle x, A^*Ax\rangle = \langle Ax, Ax\rangle$, und daher ist $Ax = 0$. Damit haben $A$ und $A^*A$ denselben Kern, daher denselben Rang.

(2) $A^*A$ ist offensichtlich hermitesch. Wenn $\mathrm{rg}(A) = n$, so ist für $x \neq 0$ auch $Ax \neq 0$, also $0 < \langle Ax, Ax\rangle = x^*A^*Ax$.

□

Damit hat man das überbestimmte Problem auf einen Fall aus Kapitel 3 zurückgeführt, und man kann einen ersten Algorithmus zur Lösung des überbestimmten Gleichungssystemes $Ax = b$ angeben:

**Algorithmus 2.1.3.** *Normalengleichungen*

(1) *Bilde* $A^*A$.
(2) *Berechne eine Cholesky–Faktorisierung* $A^*A = LL^*$.
(3) *Löse* $Lu = A^*b$.
(4) *Löse* $L^*x = u$.

Der Rechenaufwand für diesen Algorithmus liegt fast vollständig in den ersten beiden Punkten. Das Matrixprodukt benötigt $mn^2$ Flops, und die Cholesky–Zerlegung trägt weitere $n^3/3$ Flops bei. Alles gesamt ergibt $O(n^2(m + n/3))$.

Unglücklicherweise ist dieses Verfahren nicht sehr gut zur Berechnung von $x$ geeignet, da die Matrix $A^*A$ oft schlecht konditioniert ist. Genaue Abschätzungen folgen später in Abschnitt 2.5.

**2.2. QR–Zerlegung.** Nachdem wir eine Orthogonalprojektion suchen, kann man in einer weiteren Variante versuchen, die Spalten der Matrix $A$ zu orthonormalisieren. Daß das immer geht, weiß man aus der linearen Algebra. Auch ein Algorithmus, das *Orthogonalisierungsverfahren von Gram–Schmidt*, wird dort angegeben. Dabei stellt man die Spaltenvektoren $a_i$ von $A$ als Linearkombination von orthonormalen Vektoren $q_j$ dar. Diese Vektoren werden so konstruiert, daß $a_i$ immer nur von $q_j$, $j \leq i$ abhängt. Man erhält in Matrixschreibweise also eine Gleichung der Form

$$A = \hat{Q}\hat{R},$$

wobei $\hat{Q} \in \mathbb{K}^{n\times m}$ eine Matrix ist, deren Spalten orthonormal sind, und $\hat{R} \in \mathbb{K}^{m\times m}$ eine obere Dreiecksmatrix mit positiven Diagonalelementen ist. Eine Dekomposition dieser Form heißt *reduzierte QR–Zerlegung* von $A$.

Ergänzt man die Spalten von $\hat{Q}$ zu einer Orthonormalbasis von $\mathbb{K}^n$ und sammelt alle Vektoren in einer unitären Matrix $Q \in U(n)$, so kann man $A$ auch schreiben als

$$A = QR,$$

wobei $R \in \mathbb{K}^{n\times m}$ eine verallgemeinerte obere Dreiecksmatrix (eine $m \times m$–Dreiecksmatrix plus $n - m$ Nullzeilen am unteren Ende) mit positiven Hauptdiagonaleinträgen ist. Diese vergrößerte Zerlegung heißt *(volle) QR–Zerlegung* von $A$.

Für die beiden Typen von $QR$–Zerlegungen gilt folgender Satz:

**Theorem 2.2.1.** *Jede Matrix $A \in \mathbb{K}^{n\times m}$ hat eine (volle) QR–Zerlegung, äquivalent eine reduzierte QR–Zerlegung. Hat $A$ vollen Rang, so ist die reduzierte QR–Zerlegung eindeutig, wenn $R_{ii} > 0$ vorausgesetzt wird.*

BEWEIS. Beginnen wir mit der Existenz: Der Beweis ist konstruktiv, mit Hilfe des Orthonormalisierungsverfahrens von Gram–Schmidt. Die Konstruktion erfolgt rekursiv. Seien die $a_i$ die Spalten von $A$ und die $q_j$ die Spalten von $\hat{Q}$. Der Vektor $q_1$ wird aus $a_1$ einfach

durch Normierung berechnet:

$$q_1 = \frac{a_1}{r_{11}}, \quad \text{mit} \quad |r_{11}| = \|a_1\|.$$

Danach müssen die Vektoren $a_i$ zusätzlich zur Normierung so korrigiert werden, daß sie orthogonal auf die schon bestimmten Vektoren $q_j$, $j < i$ stehen:

$$q_i = \frac{1}{r_{ii}}\left(a_i - \sum_{j=1}^{i-1}\langle q_j, a_i\rangle q_j\right).$$

Setzen wir $r_{k\ell} = \langle q_k, a_\ell\rangle$ für $k \neq \ell$, so können wir die letzte Gleichung schreiben als

$$q_i = \frac{1}{r_{ii}}\left(a_i - \sum_{j=1}^{i-1} r_{ji} q_j\right) \quad \text{mit} \quad r_{ii} = \left\|a_i - \sum_{j=1}^{i-1} r_{ji} q_j\right\|.$$

Die entstehende Matrix $\hat{R} = (r_{ij})$ ist dann eine obere Dreiecksmatrix mit $R_{ii} > 0$ für alle $i$. Dadurch entsteht durch Umformen und Zusammenfassen der Spaltenvektoren zu Matrizen $\hat{R}$ bzw. $\hat{Q}$ eine reduzierte $QR$–Zerlegung:

$$\underbrace{\left(\begin{array}{c|c|c|c} a_1 & a_2 & \cdots & a_m \end{array}\right)}_{A} = \underbrace{\left(\begin{array}{c|c|c|c} q_1 & q_2 & \cdots & q_m \end{array}\right)}_{\hat{Q}} \cdot \underbrace{\begin{pmatrix} r_{11} & r_{12} & r_{13} & \cdots & r_{1m} \\ & r_{22} & r_{23} & \cdots & r_{2m} \\ & & r_{33} & \cdots & r_{2m} \\ & & & \ddots & \vdots \\ & & & & r_{mm} \end{pmatrix}}_{\hat{R}}$$

Die volle $QR$–Zerlegung kann dadurch berechnet werden, daß man die Spaltenvektoren von $\hat{Q}$ zu einer Orthonormalbasis ergänzt und mit den zusätzlichen Vektoren $\hat{Q}$ zu einer Matrix $Q$ ergänzt. Die Matrix $\hat{R}$ wird durch Erweiterung mit Nullzeilen zu $R$ verändert.

Wenn die Matrix vollen Rang hat, dann sind alle Elemente $R_{ii}$, durch die dividiert wird, ungleich Null, und die Zerlegung ist eindeutig. Sollte das nicht der Fall sein, so wählt man in diesem Schritt irgendeinen Vektor $q_i$, der orthonormal auf die bereits berechneten $q_j$ steht, und fährt mit dem Gram–Schmidt Prozeß fort. □

Mit Hilfe dieser Zerlegung können wir jetzt den Orthonormalprojektor auf das Bild von $A$ erneut berechnen. Sei wieder vorausgesetzt, daß $A$ vollen Rang hat.

$$\begin{aligned} P &= A(A^*A)^{-1}A^* = \\ &= \hat{Q}\hat{R}(\hat{R}^*\hat{Q}^*\hat{Q}\hat{R})^{-1}\hat{R}^*\hat{Q}^* = \\ &= \hat{Q}\hat{R}(\hat{R}^*\hat{R})^{-1}\hat{R}^*\hat{Q}^* = \\ &= \hat{Q}\hat{R}\hat{R}^{-1}\hat{R}^{*-1}\hat{R}^*\hat{Q}^* = \\ &= \hat{Q}\hat{Q}^*, \end{aligned}$$

wobei die zweite Gleichung gilt, da $\hat{Q}^*\hat{Q} = \mathbb{I}$ und die dritte Gleichung, da $\hat{R}$ invertierbar ist. Man sieht also, daß sich die Gestalt von $P$ stark vereinfacht. Versucht man nun wieder die Normalengleichungen zu lösen, so erhält man

$$A^*Ax = \hat{R}^*\hat{R}x = \hat{R}^*\hat{Q}^*b = A^*b,$$

und da $\hat{R}$ invertierbar ist, folgt $x$ ist Lösung der Normalengleichungen genau dann, wenn es Lösung der Gleichung

$$\hat{R}x = \hat{Q}^*b$$

ist.

Betrachtet man all dies und setzt man voraus, daß man eine reduzierte $QR$–Zerlegung von $A$ berechnen kann, so erhält man folgenden Algorithmus zur Lösung des Kleinste–Quadrate–Problems:

**Algorithmus 2.2.2.** *Kleinste–Quadrate–Lösung mit QR–Zerlegung*

(1) *Berechne die reduzierte QR–Zerlegung* $A = \hat{Q}\hat{R}$.
(2) *Löse* $\hat{R}x = \hat{Q}^* b$.

Der Aufwand wird vollständig durch die $QR$–Zerlegung bestimmt. Verwendet man das Housholder–Schmidt Verfahren (siehe 2.2.1), so ist der Aufwand $\sim 2mn^2 + \frac{2}{3}n^3$.

Das Gram–Schmidt Verfahren kann man zwar sehr gut für mathematische Beweise einsetzen, doch in Abschnitt 2.5 werden wir sehen, daß die Berechnungsmethode nicht gutartig ist. Zwei gutartige Methoden werden wir in den folgenden Unterabschnitten kennenlernen. Zuvor sei das Gram–Schmidt Verfahren jedoch noch als Algorithmus zusammengefaßt:

**Algorithmus 2.2.3.** *Orthogonalisierungsverfahren von Gram–Schmidt*

**for** $j = 1$ **to** $m$ **do**
  $v_j = a_j$
  **for** $i = 1$ **to** $j - 1$ **do**
    $r_{ij} = q_i^* a_j$
    $v_j = v_j - r_{ij} q_i$
  **done**
  $r_{jj} = \|v_j\|_2$
  $q_j = v_j / r_{jj}$
**done**

Wie schon erwähnt ist dieser Algorithmus instabil. Durch Umstellen der Schleifen kann man den Algorithmus etwas stabilisieren. Der entstehende Algorithmus ist als modifiziertes Gram–Schmidt Verfahren bekannt. Es ist ein wenig stabiler als das Original, jedoch unbrauchbar verglichen mit den unten vorgestellten Verfahren.

**Algorithmus 2.2.4.** *Modifiziertes Gram–Schmidt Verfahren*

**for** $i = 1$ **to** $m$ **do**
  $v_i = a_i$
**done**
**for** $i = 1$ **to** $m$ **do**
  $r_{ii} = \|v_i\|_2$
  $q_i = v_i / r_{ii}$
  **for** $j = i + 1$ **to** $m$ **do**
    $r_{ij} = q_i^* v_j$
    $v_j = v_j - r_{ij} q_i$
  **done**
**done**

Der Leser überzeuge sich selbst, daß die Algorithmen 2.2.3 und 2.2.4 äquivalent sind.

**Bemerkung 2.2.5.** *Übrigens läßt sich die QR–Zerlegung auch hervorragend zur Lösung von quadratischen linearen Gleichungssystemen verwenden:*

$$b = Ax = QRx \Longrightarrow Q^* b = Rx,$$

*und damit ergibt sich folgender Algorithmus:*

**Algorithmus 2.2.6.** *(Lösung eines linearen Gleichungssystemes mit QR–Zerlegung)*

(1) *Berechne die QR–Zerlegung von A.*
(2) *Löse* $Rx = Q^* b$.

*Der Algorithmus ist im allgemeinen stabiler als der Gaußsche Algorithmus, wenn das Householder–Schmidt Verfahren zur Berechnung der Faktoren $Q$ und $R$ verwendet wird. Der Aufwand ist mit $\frac{8}{3}n^3$ allerdings deutlich größer als der Aufwand beim Spaltenpivotverfahren.*

**2.2.1. Housholder–Schmidt Verfahren.** Analysieren wir die Methode von Gram–Schmidt etwas genauer, so können wir erkennen, daß die Spalten von $\hat{Q}$ und die Zeilen von $\hat{R}$ in Algorithmus 2.2.3 schrittweise berechnet werden können: Im ersten Schritt erhalten wir

$$\left(\begin{array}{c|c|c|c} & & & \\ a_1 & a_2 & \cdots & a_m \\ & & & \end{array}\right) \begin{pmatrix} \frac{1}{r_{11}} & -\frac{r_{12}}{r_{11}} & \cdots & -\frac{r_{1m}}{r_{11}} \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{pmatrix} = \left(\begin{array}{c|c|c|c} & & & \\ q_1 & a_2^{(2)} & \cdots & a_m^{(2)} \\ & & & \end{array}\right).$$

Mathematisch ist das äquivalent mit dem Produkt mit einer elementaren oberen Dreiecksmatrix $R_1$:

$$AR_1 = A^{(2)}.$$

Nach diesem Schritt ist die erste Spalte $A_1^{(2)} = q_1$. Dann formt man die Matrix $A^{(2)}$ in einem weiteren Schritt um, in dem man die zweite Spalte von $A^{(2)}$ zu $q_2$ transformiert, dabei die erste Spalte aber unverändert läßt:

$$\left(\begin{array}{c|c|c|c} & & & \\ q_1 & a_2^{(2)} & \cdots & a_m^{(2)} \\ & & & \end{array}\right) \begin{pmatrix} 1 & & & \\ & \frac{1}{r_{22}} & \cdots & -\frac{r_{2m}}{r_{22}} \\ & & \ddots & \\ & & & 1 \end{pmatrix} = \left(\begin{array}{c|c|c|c} & & & \\ q_1 & q_2 & \cdots & a_m^{(3)} \\ & & & \end{array}\right).$$

Fährt man mit diesem Verfahren fort, so erhält man am Ende

$$A \underbrace{R_1 R_2 \cdots R_n}_{\hat{R}^{-1}} = \hat{Q}.$$

Das Gram–Schmidt Verfahren ist also ein Orthogonalisierungsverfahren mittels Dreiecksmatrizen. Die mathematische Methode ähnelt damit der des Gaußschen Algorithmus. Die Ursache für die Instabilität dieses Verfahrens liegt darin, daß die Größe der $R_{ij}$ stark variieren kann. Alle diese Größenunterschiede führen in späteren Rechenschritten zu Auslöschungen.

Die Methode von Householder geht grundsätzlich anders vor. Die zugrunde liegende Idee ist, die Hauptlast der Transformation auf die unitären Matrizen zu legen und mit Hilfe solcher Matrizen $A$ in Dreiecksgestalt zu transformieren:

$$\underbrace{Q_m \cdots Q_2 Q_1}_{Q^*} A = R.$$

Damit konstruiert man eine volle $QR$–Zerlegung:

$$A = \underbrace{Q_1^* Q_2^* \cdots Q_m}_{Q} R = QR.$$

Das Grundprinzip ist wieder ein Iterationsverfahren, in dessen Verlauf Nullen in den Spalten von $A$ erzeugt werden.

$$\begin{pmatrix} * & * & * & \cdots & * \\ * & * & * & \cdots & * \\ * & * & * & \cdots & * \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ * & * & * & \cdots & * \\ * & * & * & \cdots & * \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ * & * & * & \cdots & * \end{pmatrix} \overset{Q_1}{\to} \begin{pmatrix} * & * & * & \cdots & * \\ \mathbf{0} & * & * & \cdots & * \\ \mathbf{0} & * & * & \cdots & * \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & * & * & \cdots & * \\ \mathbf{0} & * & * & \cdots & * \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & * & * & \cdots & * \end{pmatrix} \overset{Q_2}{\to} \begin{pmatrix} * & * & * & \cdots & * \\ & * & * & \cdots & * \\ & \mathbf{0} & * & \cdots & * \\ & \vdots & \vdots & \ddots & \vdots \\ & \mathbf{0} & * & \cdots & * \\ & \mathbf{0} & * & \cdots & * \\ & \vdots & \vdots & \ddots & \vdots \\ & \mathbf{0} & * & \cdots & * \end{pmatrix} \to \cdots$$

$$\cdots \overset{Q_m}{\to} \begin{pmatrix} * & * & * & \cdots & * \\ & * & * & \cdots & * \\ & & * & \cdots & * \\ & & & \ddots & \vdots \\ & & & & * \\ & & & & \mathbf{0} \\ & & & & \vdots \\ & & & & \mathbf{0} \end{pmatrix}.$$

Was noch bleibt ist, die Matrizen $Q_i$ zu konstruieren. Im Standardverfahren setzt man die Matrizen an als

$$Q_i = \begin{pmatrix} \mathbb{I} & 0 \\ 0 & U \end{pmatrix},$$

wobei $\mathbb{I}$ eine $(i-1)\times(i-1)$–Einheitsmatrix und $U$ eine $(n-i+1)\times(n-i+1)$ unitäre Matrix sind. Die Matrix $U$ muß dann Nullen in der $i$–ten Spalte von $A$ erzeugen. Das Householder–Schmidt Verfahren wählt dazu eine spezielle Spiegelungsmatrix $U$, eine Householder Spiegelung. Die Idee bei der Konstruktion ist, eine geeignete Spiegelungshyperebene $H$ so zu wählen, daß der $i$–te Spaltenvektor $x$ auf $\|x\|e_1$ abgebildet wird:

$$x = \begin{pmatrix} * \\ * \\ * \\ \vdots \\ * \end{pmatrix} \overset{U}{\to} Ux = \begin{pmatrix} \|x\| \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} = \|x\|e_1,$$

siehe Abbildung 4.1. In Kapitel 3 Abschnitt 1.2.11 haben wir gesehen, daß die Matrix $S = \mathbb{I} - \frac{2}{\langle v,v\rangle} vv^*$ eine Spiegelung an der Hyperebene orthogonal zu $v$ beschreibt. Aus Abbildung 4.1 sehen wir, daß $H$ die Hyperebene sein sollte, die normal auf $v = \|x\|e_1 - x$ steht. Unglücklicherweise haben wir dabei die Sache zu sehr vereinfacht. Es gibt nämlich viele Möglichkeiten, $x$ auf die Gerade $\lambda e_1$ zu spiegeln. Jedes Bild mit $|\lambda| = \|x\|$ ist möglich. Das sind genau die Vektoren $\mu\|x\|e_1$ mit $|\mu| = 1$. Aus Gründen der numerischen Stabilität ist es am günstigsten (um Auslöschungseffekte zu vermeiden) einen Wert für $\mu$ zu wählen, für den der Abstand zwischen $x$ und $\mu\|x\|e_1$ nicht zu klein ist. Ist $x_1 = |x_1|e^{i\alpha}$, dann setzt man $\mu = -e^{i\alpha}$ — im reellen Fall bedeutet das $\mu = -\operatorname{sgn}(x_1)$. Wenn $x_1 = 0$, dann setzt man $\mu = 1$. Wir können jetzt den Algorithmus zur $QR$–Zerlegung zusammenfassen:

**Algorithmus 2.2.7.** *$QR$–Zerlegung mittels Householder–Schmidt Verfahren*

**for** $k = 1$ **to** $m$ **do**

$x = A_{k:n,k}$

$v_k = e^{i\arg(x_1)}\|x\|_2 e_1 + x$

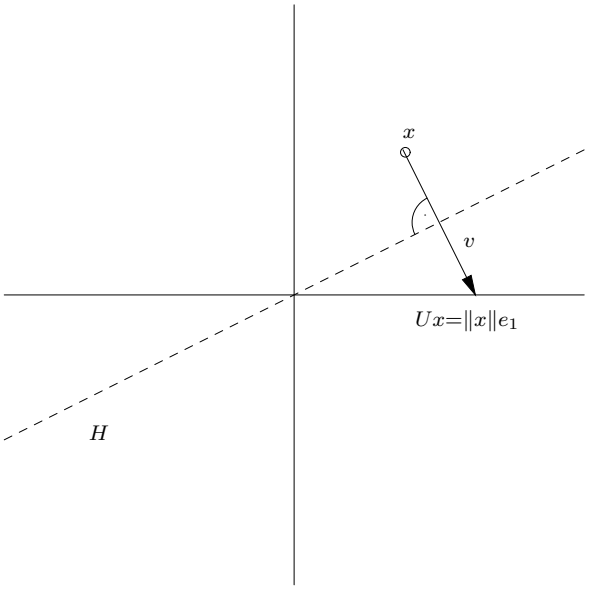



ABBILDUNG 4.1. Householder Spiegelung

$v_k = v_k/\|v_k\|_2$
$A_{k:n,k:m} = A_{k:n,k:m} - 2v_k(v_k^* A_{k:n,k:m})$
**done**

Man beachte, daß in diesem Algorithmus die Matrix $Q = Q_1^* \cdots Q_m^*$ nicht berechnet wird. Der Grund dafür ist, daß die Bestimmung von $Q$ zusätzlichen Aufwand kostet, aber man in den meisten Anwendungen ebenso gut das Produkt $Q_1 \cdots Q_m$ (kein Druckfehler — die Matrizen $Q_i$ sind hermitesch) verwenden kann. Um Matrix–Vektor Produkte $Q^*x$ bzw. $Qy$ zu berechnen, benötigt man nur die gespeicherten Vektoren $v_k$:

**Algorithmus 2.2.8.** *Berechnung von $Q^*x$*
**for** $k = 1$ **to** $m$ **do**
$x_{k:n} = x_{k:n} - 2v_k(v_k^* x_{k:n})$
**done**

und

**Algorithmus 2.2.9.** *Berechnung von $Qy$*
**for** $k = m$ **to** $1$ **step** $-1$ **do**
$y_{k:n} = y_{k:n} - 2v_k(v_k^* y_{k:n})$
**done**

Diese Algorithmen können auch dazu verwendet werden, um die Matrix $Q$ selbst zu berechnen. Dazu verwendet man z.B. Algorithmus 2.2.9, um die Spalten $Qe_i$ von $Q$ sukzessive zu bestimmen.

Bleibt noch, den Rechenaufwand für Algorithmus 2.2.7 festzustellen. Der dominierende Faktor ist die Berechnung

$$A_{k:n,k:m} = A_{k:n,k:m} - 2v_k(v_k^* A_{k:n,k:m}).$$

Sie kann zerlegt werden in eine Schleife von Vektoroperationen der Form

$$A_{k:n,i} = A_{k:n,i} - 2v_k(v_k^* A_{k:n,i}),$$

die jede $4(n-k+1)-1 \sim 4(n-k+1)$ elementare Operationen verbraucht. All das $m-k+1$ mal für jedes $k = 1, \ldots, m$. Asymptotisch ergibt das

$$\sim 2nm^2 - \tfrac{2}{3}m^3 \quad \text{Flops.}$$

Einen Beweis für diese Behauptung kann man geometrisch finden: Man repräsentiert je vier Flops durch einen kleinen Würfel der Kantenlänge 1. Dann benötigt eine der Vektoroperationen im Schritt $k$ einen Quader der Ausmaße

$$(n-k+1) \times 1 \times 1.$$

Wenn man die gesamte innere Schleife zur Matrixoperation zusammenfaßt, mißt ein Quader der Ausmaße

$$(n-k+1) \times (m-k+1) \times 1$$

den Aufwand. Faßt man alle diese Quader für $k = 1, \ldots, m$ zusammen, so erhält man den Körper aus Abbildung 4.2. Um den asymptotischen Aufwand des Algorithmus zu berechnen,

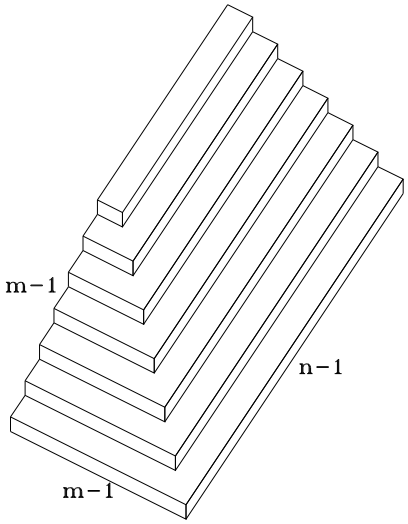



Abbildung 4.2. Gesamtaufwand des Householder–Schmidt Verfahrens

überlegt man sich, gegen welchen Körper diese Repräsentation des Aufwandes konvergiert für $n \to \infty$ und $m \to \infty$ (bei gleichzeitiger Reskalierung des vier Flops repräsentierenden Einheitswürfels), das Ergebnis sieht man in Abbildung 4.3. Der Rauminhalt des Grenzwertkörpers ist die Summe aus einer Pyramide und einem Prisma, wie es in im rechten Teil

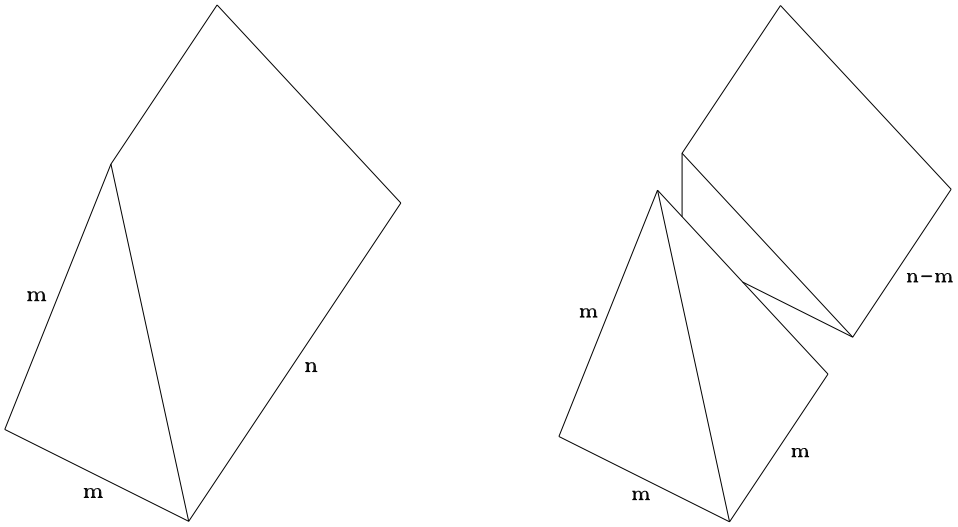



Abbildung 4.3. Asymptotischer Gesamtaufwand des Householder–Schmidt Verfahrens

von Abbildung 4.3 dargestellt ist. Deren Volumina betragen

$$\tfrac{1}{2}(n-m)m^2, \quad \text{und} \quad \tfrac{1}{3}m^3.$$

Insgesamt ergibt sich also für den Rechenaufwand

$$\sim 2nm^2 - \tfrac{2}{3}m^3.$$

**2.2.2. Givens Rotationen.** Bei der Verwendung von Householder–Spiegelungen kann man Nullen im großen Rahmen erzeugen. Möchte man jedoch selektiver vorgehen, gibt es einen anderen Typ von unitären Matrizen, die sich anbieten, die *Givens Rotationen*. Diese Matrizen sind Rang–2 Korrekturen der Identität von der Gestalt

$$G(i,j,\theta) = \begin{pmatrix} 1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & c & \cdots & s & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & -s & \cdots & c & \cdots & 0 \\ \vdots & \ddots & \vdots & \ddots & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{pmatrix},$$

wobei $s = \sin(\theta)$ und $c = \cos(\theta)$. Die Abbildung, die dieser Matrix entspricht, ist eine Drehung um den Winkel $\theta$ in der $(i,j)$–Ebene.

Setzt man $y = G(i,j,\theta)x$, so ist $y_k = 0$, wenn

$$c = \frac{x_i}{\sqrt{x_i^2 + x_j^2}}, \quad s = \frac{-x_j}{\sqrt{x_i^2 + x_j^2}}.$$

Diese Formeln sind allerdings nicht stabil. Daher ist es günstiger, zur Berechnung von $s$ und $c$ den folgenden Algorithmus zu verwenden:

**Algorithmus 2.2.10.** *Berechnung der Givens Rotation*

**if** $b = 0$ **then**
  $c = 1$; $s = 0$
**else**
  **if** $|x_j| > |x_i|$ **then**
    $\tau = -x_i/x_j$
    $s = 1/\sqrt{1+\tau^2}$
    $c = s\tau$
  **else**
    $\tau = -x_j/x_i$
    $c = 1/\sqrt{1+\tau^2}$
    $s = c\tau$
  **endif**
**endif**

Die Berechnung benötigt 5 Flops und eine Quadratwurzel. Bei der Linksmultiplikation $G(i,j,\theta)^*A$ werden nur die $i$–te und die $j$–te Zeile von $A$ verändert. Es genügt also, nur diese beiden Zeilen neu zu berechnen, was einem Aufwand von $6m$ Flops entspricht. Analog verändert Rechtsmultiplikation mit einer Givens Rotation nur zwei Spalten von $A$ und benötigt $6n$ Flops. Zur Abspeicherung einer Givens Rotation benötigt man zusätzlich zu den beiden Indices $i$ und $j$ nur noch *eine* Gleitkommazahl $\gamma$. Nachdem $s^2 + c^2 = 1$ gilt, kann man $\gamma$ folgendermaßen definieren.

**if** $c = 0$ **then**
  $\gamma = 1$
**elseif** $|s| > |c|$ **then**
  $\gamma = \operatorname{sgn}(c)s/2$
**else**
  $\gamma = 2\operatorname{sgn}(s)/c$
**endif**

Man speichert im Prinzip also den kleineren Wert von $s$ und $c$ mit der Zusatzidee, auszunützen, daß $|c| < 1$ und $|s| < 1$ gelten und man daher weiß, daß man $s$ gespeichert hat, wenn $|\gamma| < 1$, und man sich $c$ gemerkt hat, wenn $|\gamma| > 1$. Der Grund für das Abspeichern des kleineren Wertes ist die Instabilität von $\sqrt{1-x^2}$, wenn $x$ nahe bei 1 liegt. Der Algorithmus zur Zurückgewinnung von $s$ und $c$ sieht dann so aus:

**if** $\gamma = 1$ **then**
  $c = 0$
  $s = 1$
**elseif** $|\gamma| < 1$ **then**
  $s = 2\gamma$
  $c = \sqrt{1-s^2}$
**else**
  $c = 2/\gamma$
  $s = \sqrt{1-c^2}$
**endif**

Es ist dadurch zwar nicht zu unterscheiden, ob die Matrix $\begin{pmatrix} c & s \\ -s & c \end{pmatrix}$ oder die Matrix $\begin{pmatrix} -c & -s \\ s & -c \end{pmatrix}$ abgespeichert worden ist, doch beide Matrizen haben die gleichen Eigenschaften, wenn es darum geht, Elemente zu Null zu setzen.

Mit Hilfe der Givensrotationen kann man natürlich auf offensichtliche Weise eine $QR$–Zerlegung berechnen, indem man schrittweise alle Elemente unterhalb der Hauptdiagonalen von $A$ auf Null transformiert, z.B. spaltenweise:

**Algorithmus 2.2.11.** *$QR$–Zerlegung mit Givens Rotationen*

**for** $j = 1$ **to** $n$ **do**
  **for** $i = n$ **to** $j + 1$ **step** $-1$ **do**
    Berechne $c$ und $s$ für $G(i-1, i, \theta)$ nach
      Algorithmus 2.2.10 für $x = A_{j:n,j}$
    $A_{i-1:i,j:m} = G(i-1, i, \theta)A_{i-1:i,j:m}$.
  **done**
**done**

Der Algorithmus benötigt $3m^2n - m^3$ Flops, was etwa eineinhalb mal so viel wie für die Householder Spiegelungen ist. Die numerische Stabilität ist vergleichbar. Der Vorteil der Givensrotationen ist, daß man spezielle Matrixstrukturen besser ausnützen kann.

**Beispiel 2.2.12.** *Ein Beispiel für den Vorteil von Givens Rotationen bei speziellen Problemen ist die QR–Zerlegung einer Matrix der Gestalt*

$$H = \begin{pmatrix} * & * & * & * & \cdots & * & * \\ * & * & * & * & \cdots & * & * \\ & * & * & * & \cdots & * & * \\ & & * & * & \cdots & * & * \\ & & & \ddots & \ddots & \vdots & \vdots \\ & & & & \ddots & * & * \\ & & & & & * & * \end{pmatrix},$$

*einer* oberen Hessenbergmatrix. *Diese Matrizen treten bei der Bestimmung von Eigenwerten und der Singulärwertzerlegung auf. Um die QR–Zerlegung solch einer Matrix auszurechnen, muß man nur die Elemente unterhalb der Hauptdiagonalen auf Null transformieren. Das ist aber mit Givensrotationen leicht zu realisieren: Im ersten Schritt berechnet man*

$$G(1,2,\theta_1)H = \begin{pmatrix} * & * & * & * & \cdots & * & * \\ & * & * & * & \cdots & * & * \\ & * & * & * & \cdots & * & * \\ & & * & * & \cdots & * & * \\ & & & \ddots & \ddots & \vdots & \vdots \\ & & & & \ddots & * & * \\ & & & & & * & * \end{pmatrix},$$

*und im nächsten Schritt*

$$G(2,3,\theta_2)G(1,2,\theta_1)H = \begin{pmatrix} * & * & * & * & \cdots & * & * \\ & * & * & * & \cdots & * & * \\ & & * & * & \cdots & * & * \\ & & * & * & \cdots & * & * \\ & & & \ddots & \ddots & \vdots & \vdots \\ & & & & \ddots & * & * \\ & & & & & * & * \end{pmatrix},$$

*und so weiter bis*

$$G(m-1,m,\theta_m)\ldots G(2,3,\theta_2)G(1,2,\theta_1)H = \begin{pmatrix} * & * & * & * & \cdots & * & * \\ & * & * & * & \cdots & * & * \\ & & * & * & \cdots & * & * \\ & & & * & \cdots & * & * \\ & & & & \ddots & \vdots & \vdots \\ & & & & & * & * \\ & & & & & & * \end{pmatrix},$$

*Man benötigt also nur $m-1$ Givensrotationen für die QR–Zerlegung dieser Matrix, der Aufwand ist also $O(m^2)$, also wesentlich weniger als bei Verwendung von Householder–Reflexionen, die bereits im ersten Schritt die Struktur von $H$ zerstören würden.*

Was noch bleibt, ist eine Bemerkung zur *schnellen Givens Transformation*: Wenn man bei der Berechnung der Givens Rotationen darauf verzichten möchte, Quadratwurzeln zu ziehen, die auch auf modernen Computersystemen einen erheblichen Mehraufwand gegenüber einer Addition oder Multiplikation bedeuten, dann stößt man bald auf eine Abwandlung der Givens Rotationen. Diese veränderten Transformationen basieren auf der Idee, auf die Unitarität der Matrizen (fast) zu verzichten.

Bei der schnellen Givens Transformation wird die Matrix $Q$ der Zerlegung, die Produkt von Givens Rotationen ist, dargestellt als Paar von Matrizen $(M, D)$, wobei $M^*M = D = \operatorname{diag}(d_1, \ldots, d_n)$ gilt und jedes $d_i$ positiv ist; das heißt $M$ ist „unitär bis auf Skalierung“. Definiert man nämlich $Q = MD^{-1/2}$, so ist $Q$ wegen $Q^*Q = (MD^{-1/2})^*(MD^{-1/2}) = D^{-1/2}DD^{-1/2} = \mathbb{I}$ unitär. Ist jetzt $Y$ eine $m \times m$–Matrix mit $Y^*DY = D_{\text{neu}}$ eine andere Diagonalmatrix, so erfüllt die Matrix $M_{\text{neu}} = MY$

$$M_{\text{neu}}^* M_{\text{neu}} = D_{\text{neu}}.$$

Man kann also die Repräsentation $(M, D)$ mit einfachen Rechnungen zu $(M_{\text{neu}}, D_{\text{neu}})$ transformieren. Um die Matrizen so zu konstruieren, daß sie geeignet Elemente zu Null setzen, muß man nur der Matrix $Y$ diese Eigenschaft verpassen und dafür sorgen, daß sie $D$ wieder auf eine Diagonalmatrix abbildet.

Nachdem wie für die Givens Rotationen nur eine $2 \times 2$–Teilmatrix von der Identität verschieden ist, genügt es das ganze Verfahren für $2 \times 2$–Matrizen und zweikomponentige Vektoren zu entwickeln. Setzt man dann die Elemente dieser $2 \times 2$–Matrizen an die richtigen Stellen in der großen Matrix (analog zu den üblichen Givens Rotationen), dann erhält man die Transformationsmatrizen der schnellen Givens Transformation. Sei also $D = \operatorname{diag}(d_1, d_2)$ mit $d_i > 0$. Setzen wir $Y$ an als

$$Y_1 = \begin{pmatrix} \overline{b_1} & 1 \\ 1 & a_1 \end{pmatrix}.$$

Dann ist

$$Y_1^* D Y_1 = \begin{pmatrix} d_2 + |b_1|^2 d_1 & d_1 b_1 + d_2 a_1 \\ d_1 \overline{b_1} + d_2 \overline{a_1} & d_1 + |a_1|^2 d_2 \end{pmatrix},$$

wenn man somit erzwingen möchte, daß $Y_1^* D Y_1$ diagonal ist, so erhält man die Gleichung

$$b_1 d_1 + a_1 d_2 = 0.$$

Ist $x = (x_1, x_2)^*$ der Vektor, bei dem $x_2$ zu Null gemacht werden soll und ist $x_2 \neq 0$, dann erhält man aus

$$Y_1^* x = \begin{pmatrix} b_1 x_1 + x_2 \\ x_1 + \overline{a_1} x_2 \end{pmatrix} = y$$

die weitere Gleichung

$$\overline{a_1} x_2 = -x_1,$$

vorausgesetzt $y_2$ soll gleich 0 sein. Nachdem $x_2 \neq 0$ ist kann man die Gleichungen auflösen und erhält

$$Y_1 = \begin{pmatrix} -\frac{x_1 d_2}{x_2 d_1} & 1 \\ 1 & -\frac{\overline{x_1}}{\overline{x_2}} \end{pmatrix}$$

und

$$Y_1^* D Y_1 = \operatorname{diag}(d_2(1+\gamma_1), d_1(1+\gamma_1)) =: D_1$$
$$Y_1^* x = \begin{pmatrix} x_2(1+\gamma_1) \\ 0 \end{pmatrix}$$

mit $\gamma_1 = \frac{|x_1|^2}{|x_2|^2} \frac{d_2}{d_1}$.

Analog, wenn man $x_1 \neq 0$ annimmt und $Y_2$ ansetzt als

$$Y_2 = \begin{pmatrix} 1 & a_2 \\ \overline{b_2} & 1 \end{pmatrix}$$

mit $a_2 = -\frac{\overline{x_2}}{\overline{x_1}}$ und $b_2 = -a_2\frac{d_1}{d_2}$, dann ergibt sich

$$Y_2^* D Y_2 = \operatorname{diag}(d_1(1+\gamma_2), d_2(1+\gamma_2)) =: D_2$$
$$Y_2^* x = \begin{pmatrix} x_1(1+\gamma_2) \\ 0 \end{pmatrix}$$

mit $\gamma_2 = \frac{|x_2|^2}{|x_1|^2}\frac{d_1}{d_2}$.

Die Matrix $G = D^{1/2} Y_i D_i^{-1/2}$ ist unitär und so konstruiert, daß die zweite Komponente von $G^*(D^{-1/2})x$ Null ist (für $i = 1, 2$, wenn $Y_i$ existiert). Nachdem $\gamma_1\gamma_2 = 1$ kann man $Y_i$ so wählen, daß $(1+\gamma_i) \leq 2$ gilt. Matrizen $M$, die aus $2\times 2$–Matrizen der Form $Y_1$ gebildet werden, heißen schnelle Givenstransformationen vom Typ 1, diejenigen die aus $2\times 2$–Matrizen der Form $Y_2$ konstruiert sind, werden schnelle Givenstransformationen vom Typ 2 genannt.

Linksmultiplikation mit $M_i$ benötigt nur halb so viele Operationen wie Linksmultiplikation mit einer gewöhnlichen Givens Rotation. Außerdem tritt im gesamten Prozeß des Zu-Null-Machens keine Quadratwurzel auf. Die schnellen Givenstransformationen werden bei der $QR$–Zerlegung genauso angewendet wie normale Givens Rotationen. Der einzige Unterschied ist, daß durch die Faktoren $(1+\gamma)$ die Elemente von $M$ und $D$ exponentiell wachsen können. Das führt zwar nicht zu Stabilitätsproblemen wie man mit Fehleranalyse nachweisen kann. Trotzdem muß man sich regelmäßig davon überzeugen, daß die Elemente nicht schon so weit gewachsen sind, daß die Gefahr eines Überlauffehlers auftritt. In diesem Fall muß man dann die Matrizen $M$ und $D$ reskalieren. Dabei muß man allerdings wieder Quadratwurzeln berechnen.

**Algorithmus 2.2.13.** *$QR$–Zerlegung mit schnellen Givenstransformationen*

```
[a, b, t] = function fast.givens(x, D)
        if x_2 ≠ 0 then
          a = −conj(x_1)/conj(x_2)
          b = −a d_2/d_1
          γ = −conj(a) b
          if γ ≤ 1 then
            t = 1 ; Typ der Transformation
            τ = d_1
            d_1 = (1 + γ) d_2
            d_2 = (1 + γ) τ
          else
            t = 2 ; Typ der Transformation
            a = 1/a; b = 1/b
            γ = 1/γ
            d_1 = (1 + γ) d_1
            d_2 = (1 + γ) d_2
          endif
        else
          t = 2
          a = 0
          b = 0
        endif
end
; Nach der Definition der Funktion fast.givens, die 2 × 2
; schnelle Givens Transformationen berechnet, folgt jetzt
; der Hauptteil des Algorithmus
```

$d = (1, \ldots, 1)$
**for** $j = 1$ **to** $m$ **do**
  **for** $i = n$ **to** $j + 1$ **step** $-1$ **do**
    $[a, b, t] = \mathsf{fast.givens}(A_{i-1:i,j}, d_{i-1:i})$
    **if** $t = 1$ **then**

$$A_{i-1:i,j:m} = \begin{pmatrix} \overline{b} & 1 \\ 1 & a \end{pmatrix} A_{i-1:i,j:m}$$

    **else**

$$A_{i-1:i,j:m} = \begin{pmatrix} 1 & a \\ \overline{b} & 1 \end{pmatrix} A_{i-1:i,j:m}$$

    **endif**
  **done**
**done**

Zum Verständnis des obigen Algorithmus beachte man noch, daß wenn $(M, D)$ eine Schnelle–Givens–Darstellung von $Q$ ist mit $M^*A = T$ für eine obere Dreiecksmatrix $T$, dann sind $Q = MD^{-1/2}$ und $R = D^{-1/2}T$ die Faktoren der $QR$–Transformation von $A$:

$$QR = MD^{-1/2}D^{-1/2}T = MD^{-1}M^*A = MM^{-1}M^{*-1}M^*A = A.$$

Der Algorithmus benötigt $2m^2n - \frac{2}{3}m^3$ Flops. Allerdings ist es, wie oben bemerkt, nötig, Überläufe zu vermeiden und $A$, $M$ zu reskalieren. Vorteile hat dieser Algorithmus vor allem bei der $QR$–Zerlegung von Bandmatrizen. Wenn $A$ obere Bandbreite $p$ und untere Bandbreite $q$ hat, so hat $R$ obere Bandbreite $p + q$. $QR$–Zerlegung mit Givens Rotationen benötigt $O(mp(p + q))$ Flops und $O(mp)$ Quadratwurzeln. Sind $p, q \ll m$, so machen die Quadratwurzeln einen bedeutenden Anteil des Gesamtaufwandes aus. Bei der Verwendung schneller Givens Transformationen kann man auf diese Quadratwurzeln verzichten.

$QR$–Faktorisierung mit Hilfe der Householder Spiegelungen oder der Givens Rotationen ist viel stabiler als das Gram–Schmidt Verfahren und die Normalengleichungen bei der Lösung des Kleinste–Quadrate–Problems. Wie wir jedoch in Abschnitt 2.5 sehen werden, versagen alle diese Verfahren, wenn die Matrix $A$ beinahe singulär (numerisch singulär) ist. Ein noch etwas aufwendigeres Verfahren, das im nächsten Abschnitt vorgestellt wird, kann dabei Abhilfe schaffen - die Singulärwertzerlegung.

**2.3. Singulärwertzerlegung.** Eine dritte Methode, das Kleinste–Quadrate–Problem zu lösen, die oftmals zwar eine noch aufwendigere Rechnung erfordert als das Householder–Schmidt Verfahren oder die Givensrotationen, die aber auch dann stabil ist, wenn die Matrix $A$ fast singulär ist, basiert auf einer weiteren Art $A$ in ein Produkt einfacherer Matrizen zu zerlegen.

Die Idee der Singulärwertzerlegung geht von einer einfachen geometrischen Tatsache aus:

> Das Bild der Einheitssphäre $S$ (bzgl. $\| \quad \|_2$ — $S := \{x \in \mathbb{K}^m | \ \|x\|_2 = 1\}$) unter einer linearen Abbildung $A : \mathbb{K}^m \to \mathbb{K}^n$ ist ein Hyperellipsoid (mehrdimensionale Verallgemeinerung einer Ellipse).

Ein Hyperellipsoid (siehe Abbildung 4.4) entsteht durch Strecken einer Einheitskugel um Faktoren $\sigma_1, \ldots, \sigma_m$ in Richtung orthogonaler Vektoren $u_1, \ldots, u_m \in \mathbb{K}^n$. Nehmen wir o.B.d.A. die $u_i$ als normierte Vektoren an. Dann sind die Vektoren $\sigma_i u_i$ die Haupthalbachsen des Hyperellipsoides. Nachdem $\sigma_i u_i$ auf dem Hyperellipsoid liegt, existiert ein normierter Vektor $v_i \in \mathbb{K}^m$ mit $Av_i = \sigma_i u_i$. Wenn $\mathrm{rg}(A) = r$, dann sind genau $r$ dieser $\sigma_i \neq 0$. Alle anderen verschwinden.

**Definition 2.3.1.** *Sei $A \in \mathbb{K}^{n\times m}$. Dann heißen die Längen $\sigma_i$ der Haupthalbachsen des Bildhyperellipsoides die* singulären Werte *von $A$. Nach Konvention werden sie der Größe*

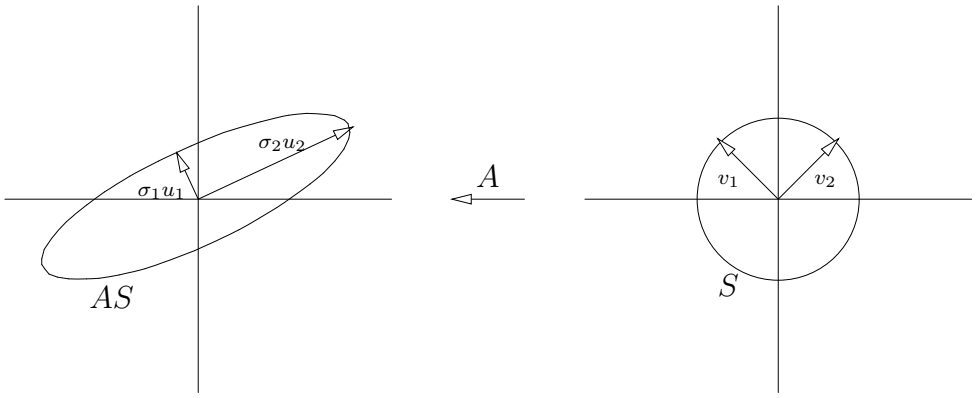



ABBILDUNG 4.4. singuläre Werte und Vektoren im $\mathbb{R}^2$

*nach angeordnet:*

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n.$$

*Die normierten Richtungsvektoren der Haupthalbachsen $u_i$ heißen die* links singulären Vektoren *von $A$ und die Urbilder $v_i$ der Vektoren $\sigma_i u_i$ heißen die* rechts singulären Vektoren *von $A$.*

Stellt man die obigen Fakten in Matrixschreibweise dar, so erhält man

$$AV = \hat{U}\hat{\Sigma},$$

wobei $\hat{U} \in \mathbb{K}^{n\times m}$ die Matrix gebildet aus den Spaltenvektoren $u_i$ und $V \in \mathbb{K}^{m\times m}$ die Matrix geformt aus den Spaltenvektoren $v_j$ bezeichnet. Weiters ist $\Sigma = \operatorname{diag}(\sigma_1, \ldots, \sigma_m)$. Es wird sich im weiteren herausstellen, daß $V$ unitär ist, und daher können wir die Gleichung von rechts mit $V^*$, der Inversen von $V$, multipizieren und erhalten

$$A = \hat{U}\hat{\Sigma}V^*,$$

die *reduzierte Singulärwertzerlegung* von $A$. Verfahren wir nun genauso wie bei der reduzierten $QR$–Zerlegung, so erweitern wir $\hat{U}$, eine Matrix, deren Spalten orthonormal sind, zu einer unitären Matrix $U \in U(n)$ und $\hat{\Sigma}$ zu einer verallgemeinerten Diagonalmatrix $\Sigma \in \mathbb{K}^{n\times m}$ durch Hinzufügen von Nullzeilen. Nach dieser Erweiterung erhalten wir die Gleichung

$$A = U\Sigma V^*,$$

die *(volle) Singulärwertzerlegung* von $A$.

Nachdem wir bis jetzt zwei Behauptungen noch nicht belegt haben, nämlich, daß das Bild der Einheitssphäre $S$ unter jeder linearen Abbildung ein Hyperellipsoid ist und daß die Matrix $V$ unitär ist, fehlt noch ein wasserdichter Beweis für die Existenz der Singulärwertzerlegung. Der folgende Satz räumt diese Probleme aus:

**Theorem 2.3.2.** *Jede Matrix $A \in \mathbb{K}^{n\times m}$ besitzt eine Singulärwertzerlegung*

$$A = U\Sigma V^*$$

*mit $U \in U(n)$, $V \in U(m)$ und $\Sigma \in \mathbb{K}^{n\times m}$ eine verallgemeinerte Diagonalmatrix ($\Sigma_{ij} = 0$ für $i \neq j$) mit*

$$\Sigma = \operatorname{diag}(\sigma_1, \ldots, \sigma_m), \quad \textit{mit} \quad \sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_m.$$

*Die singulären Werte sind eindeutig bestimmt, und wenn die Matrix quadratisch ist und die $\sigma_i$ paarweise verschieden sind, dann sind die rechts und links singulären Vektoren $u_i$ und $v_j$ eindeutig bestimmt bis auf Skalare $\lambda$ mit $|\lambda| = 1$.*

BEWEIS. Die größte Ausdehnung von $AS$ ist gegeben durch $\|A\|_2$. Sei also $\sigma_1 := \|A\|_2$. Nachdem

$$\|A\|_2 = \sup_{\|x\|_2=1} \|Ax\|_2$$

und weil $S$ kompakt ist, wird das Supremum angenommen. Also existiert ein $v_1$ mit $\|v_1\|_2 = 1$ und $\|Av_1\|_2 = \sigma_1$. Wir setzen $u_1 := Av_1$. Seien $u_i^{(1)} \in \mathbb{K}^n$ und $v_j^{(1)} \in \mathbb{K}^m$ Ergänzungen auf Orthonormalbasen, und konstruieren wir die Matrizen $U_1 \in U(n)$ und $V_1 \in U(m)$ aus den Spaltenvektoren $u_1$, $u_i^{(1)}$ bzw. $v_1$, $v_j^{(1)}$. Dann erhalten wir

$$U_1^* A V_1 = S = \begin{pmatrix} \sigma_1 & w^* \\ 0 & B \end{pmatrix}. \tag{26}$$

Nun folgt ein Induktionsargument. Eine $1 \times (n-m)$–Matrix $A$ bzw. eine $(m-n) \times 1$–Matrix $A$ hat offensichtlich eine Singulärwertzerlegung.

In Gleichung (26) ist $B$ eine $(n-1) \times (m-1)$–Matrix, hat nach Induktionsvoraussetzung also eine Singulärwertzerlegung

$$B = U_2 \Sigma' V_2.$$

Weiters gilt

$$\left\| \begin{pmatrix} \sigma_1 & w^* \\ 0 & B \end{pmatrix} \begin{pmatrix} \sigma_1 \\ w \end{pmatrix} \right\|_2 \geq \sigma_1^2 + w^* w = \sqrt{\sigma_1^2 + \langle w, w \rangle} \left\| \begin{pmatrix} \sigma_1 \\ w \end{pmatrix} \right\|_2,$$

und daher ist

$$\|S\|_2 \geq \sqrt{\sigma_1^2 + \langle w, w \rangle}.$$

Nachdem $U_1$ und $V_1$ unitär sind, gilt aber $\|S\|_2 = \|A\|_2 = \sigma_1$ und daher $\sqrt{\sigma_1^2 + \langle w, w \rangle} \geq \sigma_1 \geq \sqrt{\sigma_1^2 + \langle w, w \rangle}$, woraus wiederum $\langle w, w \rangle = 0 \Rightarrow w = 0$ folgt. Faßt man alle diese Resultate zusammen, so erhält man

$$A = U_1 \begin{pmatrix} \sigma_1 & 0 \\ 0 & B \end{pmatrix} V_1^* = \underbrace{U_1 \begin{pmatrix} 1 & 0 \\ 0 & U_2 \end{pmatrix}}_{U} \underbrace{\begin{pmatrix} \sigma_1 & 0 \\ 0 & \Sigma' \end{pmatrix}}_{\Sigma} \underbrace{\begin{pmatrix} 1 & 0 \\ 0 & V_2^* \end{pmatrix} V_1^*}_{V^*},$$

eine Singulärwertzerlegung von $A$.

Für den zweiten Teil des Beweises nehmen wir an, daß $A$ quadratisch ist und daß die $\sigma_i$ paarweise verschieden sind. In diesem Fall ist $\sigma_1$ eindeutig bestimmt durch die Eigenschaft $\sigma_1 = \|A\|_2$. Sei nun $w$ ein von $v_1$ linear unabhängiger Vektor mit $\|w\|_2 = 1$ und $\|Aw\|_2 = \sigma_1$. Führen wir nun einen Schritt des Gram–Schmidt Verfahrens durch:

$$v_2 = \frac{w - \langle v_1, w \rangle v_1}{\|w - \langle v_1, w \rangle v_1\|_2}.$$

Nachdem $\|A\|_2 = \sigma_1$ gilt, folgt $\|Av_2\|_2 \leq \sigma_1$. In dieser Ungleichung muß aber Gleichheit gelten, da nach dem Satz von Pythagoras $w = cv_1 + sv_2$ mit $|c|^2 + |s|^2 = 1$ und daher $\|Aw\|_2 < \sigma_1$ gelten würde. Definieren wir $\mathbb{K}^{m-1} \ni y := (V_1^* v_2)_{2:m}$, dann finden wir $\|y\|_2 = 1$ und $\|By\|_2 = \sigma_1$. Daher ist der singuläre Wert $\sigma_1$ nicht eindeutig, was einem Widerspruch zur Annahme der paarweisen Verschiedenheit der $\sigma_i$ gleichkommt. Daher ist der singuläre Vektor $v_1$ (und damit auch $u_1$) eindeutig bis auf Vielfache vom Betrag eins. Für alle anderen singulären Vektoren folgt die Eindeutigkeit aus einem Induktionsargument. □

Die Existenz der Singulärwertzerlegung trifft eine bedeutende und starke Aussage.

> Jede Matrix ist eine Diagonalmatrix, vorausgesetzt man verwendet im Urbildraum und im Bildraum geeignete Orthonormalbasen, die links bzw. rechts singulären Vektoren.

Aus der linearen Algebra ist noch eine andere ähnliche Zerlegung dieser Art bekannt, die Eigenwertzerlegung. Für (manche) quadratische Matrizen $A \in \mathbb{K}^{n\times n}$ existiert eine Matrix $T \in \mathbb{K}^{n\times n}$ mit

$$A = T\Lambda T^{-1},$$

wobei $\Lambda$ die Diagonalmatrix ist, die aus den Eigenwerten gebildet wird. Wie hängen nun die Singulärwertzerlegung und die Egenwertzerlegung zusammen? Die Antwort ist: Eigentlich gar nicht. Im Gegenteil, es gibt einige fundamentale Unterschiede:

- Die Singulärwertzerlegung verwendet zwei unterschiedliche Basen (die links bzw. rechts singulären Vektoren), während die Eigenwertzerlegung im Urbild– und im Bildbereich dieselbe Basis (die Eigenvektoren) verwendet.
- Die Singulärwertzerlegung verwendet Orthonormalbasen während die Eigenwertzerlegung als Basis die normierten Eigenvektoren verwendet, die aber nicht orthogonal aufeinander stehen müssen.
- Nicht alle quadratischen Matrizen besitzen eine Eigenwertzerlegung, doch alle Matrizen, sogar rechteckige, besitzen eine Singulärwertzerlegung.

Die Eigenwertzerlegung wird vor allem verwendet, wenn Potenzen oder allgemeinere Funktionen der Matrix $A$, wie $A^5$, $e^A$ oder $\sin(A)$ benötigt werden. Die Singulärwertzerlegung benötigt man vor allem zur Untersuchung der Eigenschaften von $A$ selbst oder von $A^{-1}$.

Fassen wir einige Eigenschaften der Singulärwertzerlegung zusammen:

**Proposition 2.3.3.** *Sei $A \in \mathbb{K}^{n\times m}$, $p := \min(n, m)$, $r$ sei der Index des kleinsten nichtverschwindenden singulären Wertes von $A$*

$$\sigma_1 \geq \cdots \geq \sigma_r \geq 0 = \sigma_{r+1} = \sigma_{r+2} = \cdots = \sigma_p,$$

*dann gilt*

(1) $\operatorname{rg}(A) = r$
(2) $\operatorname{im}(A) = \langle u_1, \ldots, u_r\rangle$ *und* $\ker(A) = \langle v_{r+1}, \ldots, v_m\rangle$.
(3) *Die nichtverschwindenden singulären Werte von $A$ sind die Quadratwurzeln der nichtverschwindenden Eigenwerte von $A^*A$ oder $AA^*$.*
(4) *Wenn $A$ hermitesch ist, dann sind die singulären Werte von $A$ die Absolutbeträge der Eigenwerte.*
(5) *Für $n = m$ gilt* $|\det(A)| = \prod_{i=1}^{n} \sigma_i$.
(6) $\kappa_2(A) = \frac{\sigma_1}{\sigma_n}$.

Beweis. Die meisten Behauptungen folgen beinahe direkt aus den Definitionen und der Singulärwertzerlegung.

(1) Sei $A = U\Sigma V^*$. Nachdem $U$ und $V$ unitär sind, also bijektiven Abbildungen entsprechen, gilt $\operatorname{rg}(A) = \operatorname{rg}(\Sigma)$. Der Rang der verallgemeinerten Diagonalmatrix $\Sigma$ ist aber die Anzahl der nichtverschwindenden Diagonalelemente. Demnach ist $\operatorname{rg}(\Sigma) = r$.
(2) Auch diese Behauptung folgt aus der Singulärwertzerlegung $A = U\Sigma V^*$. Es gilt $\operatorname{im}(\Sigma) = \langle e_1, \ldots, e_r\rangle$ und $\ker(\Sigma) = \langle e_{r+1}, \ldots, e_m\rangle$. Nachdem $\ker(A) = V^{-1}(\ker(\Sigma))$ und $\operatorname{im}(A) = U \operatorname{im}(\Sigma)$ gelten, folgt die Behauptung.
(3) $A^*A$ ist hermitesch und daher unitär äquivalent zur Diagonalmatrix, die aus den Eigenwerten gebildet wurde. Aus der Singulärwertzerlegung haben wir aber auch

$$A^*A = (U\Sigma V^*)^*(U\Sigma V^*) = V\Sigma^* U^* U\Sigma V^* = V(\Sigma^*\Sigma)V^*.$$

Daher ist $A^*A$ auch unitär äquivalent zu $\Sigma^*\Sigma$, also hat $\Sigma^*\Sigma$ dieselben Eigenwerte wie $A^*A$. Die nichtverschwindenden Eigenwerte von $\Sigma^*\Sigma$ sind aber $\sigma_1^2, \ldots, \sigma_p^2$. Klarerweise haben $A^*A$ und $AA^*$ dieselben nichtverschwindenden Eigenwerte.

(4) Wenn $A$ hermitesch ist, dann existiert die Eigenwertzerlegung mit unitärer Transformationsmatrix $Q$ und der reellen Diagonalmatrix $\Lambda$ gebildet aus den Eigenwerten. Dann haben wir jedoch

$$A = Q\Lambda Q^* = Q|\Lambda| \operatorname{sgn}(\Lambda) Q^*.$$

$Q$ und $Q \operatorname{sgn}(\Lambda)$ sind unitär, und $|\Lambda|$ ist eine Diagonalmatrix mit positiven Einträgen. Sei jetzt $P$ diejenige Permutationsmatrix, für die $P|\Lambda|P^*$ jene Diagonalmatrix ist, in der die Diagonalelemente absteigend nach der Größe sortiert sind. Setzen wir $U = QP^*$, $V = Q \operatorname{sgn}(\Lambda) P^*$ und $\Sigma = P|\Lambda|P^*$, dann gilt

$$A = QP^*P|\Lambda|P^*P \operatorname{sgn}(\Lambda) Q^* = U\Sigma V^*,$$

und wir sehen, daß die singulären Werte von $A$ (die Einträge von $\Sigma$) die Beträge der Eigenwerte von $A$ (die Einträge von $|\Lambda|$) sind.

(5) $|\det A| = |\det(U\Sigma V^*)| = |\det U||\det \Sigma||\det V^*| = 1|\det \Sigma|1 = \prod_{i=1}^n \sigma_i$.

(6) Wir wissen schon, daß $\|A\|_2 = \sigma_1$. Die Singulärwertzerlegung von $A^{-1}$ ist

$$A^{-1} = (VP)(P\Sigma^{-1}P)(UP)^*,$$

wobei $P$ die Permutationsmatrix $P_{ij} = 1$ für $i + j = n + 1$ ist, also die Permutation, die die Reihenfolge der Diagonalelemente von $\Sigma$ umkehrt. Der größte Singulärwert von $A^{-1}$ ist also $1/\sigma_n$, daher gilt $\|A^{-1}\|_2 = 1/\sigma_n$.

□

Die Singulärwertzerlegung bietet zusätzlich noch die Möglichkeit, Auskünfte über den Grad an Regulariät von $A$ zu gewinnen. Wenn wir die Matrixgleichung $A = U\Sigma V^*$ ausmultiplizieren, so gewinnen wir die Gleichung

$$A = \sum_{i=1}^{r} \sigma_i u_i v_i^*,$$

das heißt $A$ wird als Summe von $r$ Rang–1 Matrizen dargestellt. Das ist nichts außergewöhnliches, denn man kann jede Matrix auf vielfältige Arten als Summe von Rang–1 Matrizen darstellen (als Summe der Zeilen oder der Spalten,...), doch diese Darstellung hat eine spezielle Eigenschaft, die sie für alle möglichen *numerischen Untersuchungen* auszeichnet. Diese Gleichung gibt nämlich an, wie weit $A$ von Matrizen entfernt ist, die niedrigeren Rang haben, wie der folgende Satz bestätigt.

**Theorem 2.3.4.** *Für $0 \le s \le r$ sei*

$$A_s := \sum_{i=1}^{s} \sigma_i u_i v_i^*$$

*die $s$–te Partialsumme der ausmultiplizierten Singulärwertzerlegung. Dann gilt*

$$\|A - A_s\|_2 = \inf_{\substack{B \in \mathbb{K}^{n\times m} \\ \operatorname{rg}(B) \le s}} \|A - B\|_2 = \sigma_{s+1},$$

*wobei wir $\sigma_{p+1} = 0$ setzen. Das bedeutet, daß der Abstand zu den Rang–$s$ Matrizen genau der singuläre Wert $\sigma_{s+1}$ ist.*

Beweis. Der Beweis erfolgt indirekt. Angenommen, es gibt eine Matrix $B$ vom Rang höchstens $s$ mit $\|A-B\| < \sigma_{s+1}$. Dann existiert ein $(m-s)$–dimensionaler Teilraum $W \in \mathbb{K}^m$, sodaß $Bw = 0$ für alle $w \in W$ und daher

$$\|Aw\|_2 = \|(A - B)w\|_2 \le \|A - B\|_2 \|w\|_2 < \sigma_{s+1}\|w\|_2.$$

Andererseits gilt für alle $v \in V_{s+1} := \langle v_1, \dots, v_{s+1} \rangle$

$$\|Av\|_2 \ge \sigma_{s+1} v.$$

Nachdem $\dim W = m - s$ und $\dim V_{s+1} = s + 1$, muß es einen nichttrivialen Vektor $x \in W \cap V_{s+1}$ geben. Dieser müßte dann aber

$$\sigma_{s+1}\|x\|_2 \leq \|Ax\|_2 < \sigma_{s+1}\|s\|_2$$

erfüllen, ein Widerspruch. Also ist $\inf_{\mathrm{rg}(B)\leq s} \|A - B\|_2 \geq \sigma_{s+1}$. Wegen

$$\|A - A_s\| = \sigma_{s+1}$$

folgt die Behauptung. □

In den vorhergehenden Abschnitten haben wir öfters Voraussetzungen der Art: „Sei $A$ eine Matrix von vollem Rang“,... gehabt. Nun ist die Bestimmung des Ranges von $A$ im Zusammenhang mit dem Auftreten von Rundungsfehlern keine Trivialität. Oft läßt sich in numerischen Algorithmen auch kein Unterschied zwischen Matrizen, die nicht vollen Rang haben, und Matrizen, die sehr nahe an solchen sind, machen. Das führt zur folgenden Definition

**Definition 2.3.5.** *Sei $A \in \mathbb{K}^{n\times m}$. Der* $\varepsilon$–Rang von $A$ *ist definiert als*

$$\mathrm{rg}(A, \varepsilon) = \inf_{\|A-B\|_2\leq\varepsilon} \mathrm{rg}(B).$$

*$A$ heißt* numerisch von geringerem Rang, *falls* $\mathrm{rg}(A, \varepsilon) < \min(m, n)$ *für* $\varepsilon = \mathsf{eps}\,\|A\|_2$.

Mit Hilfe der Singulärwertzerlegung und Theorem 2.3.4 kann man für beliebiges $\varepsilon$ den $\varepsilon$–Rang von $A$ bestimmen und feststellen, ob $A$ numerisch von vollem Rang ist oder nicht. Ist $\sigma_{\min(m,n)} > \mathsf{eps}\,\|A\|_2$, so ist $A$ numerisch von vollem Rang.

Kehren wir nun zurück zum eigentlichen Inhalt dieses Abschnittes, den überbestimmten linearen Gleichungssystemen. Beginnen wir wieder mit den Normalengleichungen und verwenden wir die reduzierte Singulärwertzerlegung:

$$\begin{aligned} A^*Ax = A^*b &\iff (\hat{U}\hat{\Sigma}V^*)^*(\hat{U}\hat{\Sigma}V^*)x = (\hat{U}\hat{\Sigma}V^*)^*b \\ &\iff V\hat{\Sigma}\hat{U}^*\hat{U}\hat{\Sigma}V^*x = V\hat{\Sigma}\hat{U}^*b \\ &\iff V\hat{\Sigma}^2V^*x = V\hat{\Sigma}\hat{U}^*b \\ &\iff \hat{\Sigma}V^*x = \hat{U}^*b. \end{aligned}$$

Man erhält also folgenden Algorithmus zur Lösung des Kleinste–Quadrate–Problems:

**Algorithmus 2.3.6.** *Lösung des Kleinste–Quadrate–Problems mit Singulärwertzerlegung*

(1) *Bestimme die reduzierte Singulärwertzerlegung $A = \hat{U}\hat{\Sigma}V^*$.*
(2) *Löse das diagonale Gleichungssystem $\hat{\Sigma}u = \hat{U}^*b$.*
(3) *Setze $x = Vu$.*

Der Aufwand ist hauptsächlich bestimmt durch den Rechenaufwand der zur Bestimmung der reduzierten Singulärwertzerlegung nötig ist. In Kapitel 10 werden wir sehen, daß der Aufwand dafür etwa

$$\sim 2nm^2 + 11m^3$$

Flops beträgt. Für $m \gg n$ ist der Aufwand in etwa gleich dem, der zur Lösung des Problems mit Hilfe der $QR$–Zerlegung benötigt wird.

**2.3.1. Berechnung.** Entsprechend dem Aufbau der anderen Abschnitte sollte hier ein Algorithmus zur Berechnung der Singulärwertzerlegung folgen. Doch so vielfältig die Verwendbarkeit dieser Zerlegung, so zentral ihre Existenz ist, so schwierig ist ihre Berechnung. Erst in den Kapiteln 9 und 10 werden wir im Rahmen der Eigenwertberechnung genug Wissen angesammelt haben, um das Problem der Berechnung der Singulärwertzerlegung anpacken zu können. Bis dahin werden wir einfach so tun als könnten wir die Zerlegung ohne Schwierigkeiten durchführen.

**2.4. Die Pseudoinverse einer Matrix.** Ein Konzept, das vor allem der mathematischen Untersuchung der Kleinste–Quadrate–Probleme dient, ist die *Pseudoinverse* einer Matrix. Betrachten wir noch einmal die Normalengleichungen

$$A^*Ax = A^*b.$$

Hat $A$ vollen Rang, so kann man $x$ berechnen als $x = (A^*A)^{-1}A^*b$. Die Abbildung $A^\dagger := (A^*A)^{-1}A^* \in \mathbb{K}^{m\times n}$ vermittelt also eine lineare Abbildung, die einem Konstantenvektor $b \in \mathbb{K}^n$ die Kleinste–Quadrate–Lösung $x \in \mathbb{K}^m$ zuordnet. Nachdem im Fall $m = n$ die Matrizen $A^\dagger$ und $A^{-1}$ übereinstimmen und außerdem $A^\dagger A = \mathbb{I}$ gilt, nennt man $A^\dagger$ die *Pseudoinverse* von $A$. Jeder Algorithmus, den man zur Lösung des Kleinste–Quadrate–Problemes heranzieht, kann auch dazu dienen, die Pseudoinverse der Koeffizientenmatrix zu berechnen.

Die Matrix $A^\dagger$ dient auch zur Definition der Konditionszahl einer rechteckigen Matrix, die uns bei der Fehleranalyse in Abschnitt 2.5 noch gute Dienste leisten wird. In Analogie zur Definition für quadratische Matrizen setzt man

**Definition 2.4.1.** *Die* Konditionszahl *der Matrix $A \in \mathbb{K}^{n\times m}$ zur Matrixnorm $\|\quad\|$ ist gegeben durch*

$$\kappa(A) := \|A\|\|A^\dagger\|,$$

*falls $A$ vollen Rang hat und*

$$\kappa(A) := \infty,$$

*falls $A$ nicht vollen Rang hat. Falls eine bestimmte Matrixnorm (z.B. $\|\quad\|_2$) gewählt wird, so versehen wir die Konditionszahl mit einem dazupassenden Index (z.B. $\kappa_2(A)$).*

Aus der Singulärwertzerlegung folgt für $A \in \mathbb{K}^{n\times m}$

$$\kappa_2(A) = \frac{\sigma_1}{\sigma_m}.$$

**2.5. Fehleranalyse.** Um die Untersuchungen und das Hauptbeispiel (entnommen aus [**Trefethen, Bau 1997**]) in diesem Abschnitt zu motivieren, betrachten wir zu Beginn eine Anwendung, die irgendwo zwischen den Inhalten der Kapitel 5 und 8 liegt:

**Beispiel 2.5.1.** *Gegeben seien* 11 *Punkte im $\mathbb{R}^2$, Meßwerte zu denen eine Funktion geschätzt werden soll, die das Verhalten des gemessenen Phänomens möglichst gut beschreiben soll. Vereinfachend nehmen wir einmal an, daß wir ein Polynom suchen. Andere, oft besser geeignete, Funktionsklassen werden wir in den Kapiteln 5 und 8 kennenlernen.*

*Ein Satz, der direkt aus dem Fundamentalsatz der Algebra folgt, besagt, daß es genau ein Polynom höchstens* 10*–ten Grades gibt, auf dessen Graphen alle* 10 *gegebenen Punkte liegen. Dieses* Interpolationspolynom *ist in Abbildung 4.5 dargestellt. Seien für eine kurze mathematische Untersuchung $(x_1, y_1), \ldots, (x_n, y_n) \in \mathbb{K}^2$ die gegebenen Datenpunkte und*

$$p(x) = \sum_{i=0}^{n-1} a_i x^i$$

*das Interpolationspolynom. Der Zusammenhang zwischen den $a_i$ und den $x_i$ bzw. $y_i$ kann durch ein lineares Gleichungssystem ausgedrückt werden (auch wenn man so im allgemeinen*

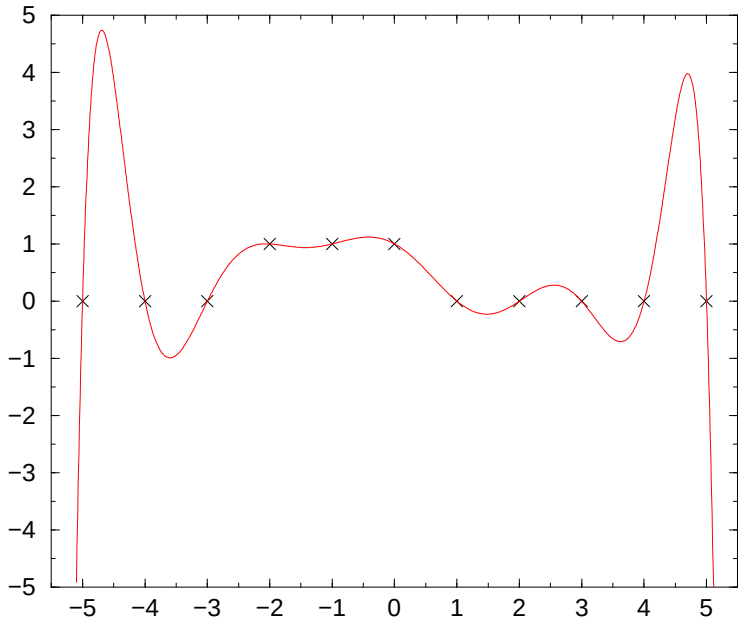



ABBILDUNG 4.5. Interpolationspolynom vom Grad 10

*Interpolationspolynome nicht berechnet) mit einem speziellen Typ von Koeffizientenmatrix, einer* Vandermonde Matrix

$$\begin{pmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \cdots & x_2^{n-1} \\ 1 & x_3 & x_3^2 & \cdots & x_3^{n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^{n-1} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{pmatrix}.$$

*Die Determinante der Vandermonde Matrix ist* $\prod_{i \neq j}(x_i - x_j)$*; daher ist die Koeffizientenmatrix invertierbar, wenn die* $x_i$ *paarweise verschieden sind.*

*Betrachten wir die Abbildung 4.5, so erkennen wir unschwer, daß das Interpolationspolynom die Form der Daten nur schlecht repräsentiert. Das ist ein typischer Effekt der Polynominterpolation, und üblicherweise wird die Übereinstimmung eher schlechter als besser, wenn mehr Datenpunkte verwendet werden.*

*Es gibt jedoch die Möglichkeit, die Übereinstimmung in der Form zu verbessern, sogar wenn man die Art und Anzahl der Datenpunkte nicht verändert. Der Trick ist, den Grad des Polynomes zu reduzieren. Mit diesem Polynom kann man die Datenpunkte zwar nicht mehr interpolieren sondern nur noch approximieren, doch im Falle eines Meßprozesses macht das nicht wirklich viel aus. Sei also* $p$ *ein Polynom* $(m-1)$*–ten Grades mit* $m < n$*. Stellt man wie zuvor das Gleichungssystem auf, das dem Interpolationsproblem entspricht, dann erhält man*

$$\begin{pmatrix} 1 & x_1 & \cdots & x_1^{m-1} \\ 1 & x_2 & \cdots & x_2^{m-1} \\ 1 & x_3 & \cdots & x_3^{m-1} \\ \vdots & \vdots & & \vdots \\ 1 & x_n & \cdots & x_n^{m-1} \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{m-1} \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{pmatrix},$$

*eine Art Vandermonde System mit rechteckiger Koeffizientenmatrix, ein überbestimmtes Gleichungssystem. Löst man für dieses Gleichungssystem das Kleinste–Quadrate–Problem, so*

*suchen wir damit das Polynom* $(m-1)$*–ten Grades, für das*

$$\sum_{i=1}^{m} |p(x_i) - y_i|^2,$$

*die Summe der Quadrate der Abweichungen an den Datenpunkten, minimal wird. In Abbildung 4.6 ist das Approximationspolynom* 7*–ten Grades dargestellt, das zu denselben* 11

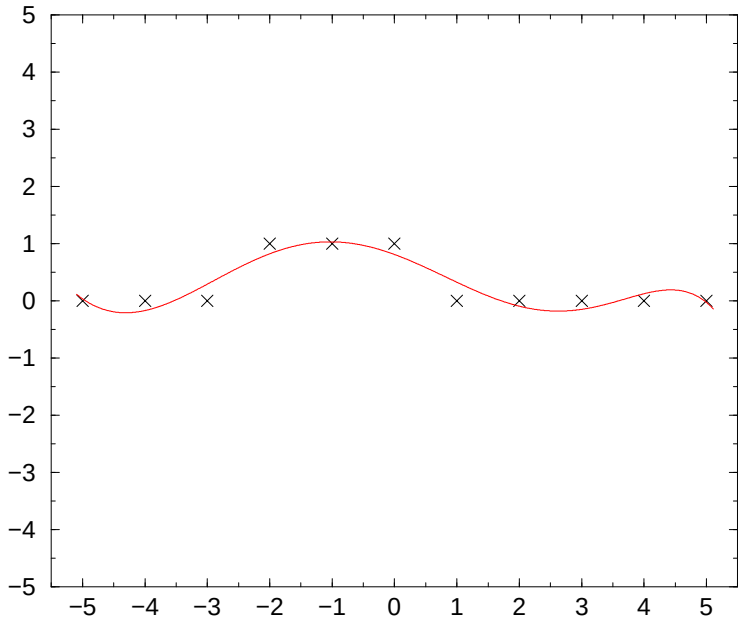



ABBILDUNG 4.6. Approximationspolynom 7–ten Grades

*Datenpunkten gehört wie das Interpolationspolynom aus Abbildung 4.5. Man sieht, daß es die Form der Daten viel besser wiederspiegelt als das Interpolationspolynom.*

*Im folgenden sei die Fehleranalyse motiviert durch den Versuch, die Abbildung* $e^{\sin(4t)}$ *auf dem Intervall* $[0,1]$ *durch ein Polynom* 14*–ten Grades anzunähern. Dazu seien* 100 *Datenpunkte gleichförmig über das Intervall* $[0,1]$ *verteilt vorgegeben:* $\tau_i = i/100$ *für* $i = 0,\ldots,99$. $A := (\tau_i^j)_{i=0,\ldots,99}^{j=0,\ldots,14} \in \mathbb{R}^{100\times 15}$ *sei die Koeffizientenmatrix und* $b := (e^{\sin(\tau_i)})_{i=0,\ldots,99}$ *der Vektor der rechten Seiten. Löst man das System mit den diversen Verfahren, die wir in diesem Abschnitt kennengelernt haben, so erhalten wir folgende relativen Fehler für den höchsten Koeffizienten* $a_{14}$ *bei Berechnung mit doppelter Genauigkeit:*

| Algorithmus | $(\tilde{a_{14}} - a_{14})/a_{14}$ |
|---|---|
| Lösung der Normalengleichungen | $0.607 \cdot 10^{0}$ |
| $QR$–Zerlegung nach dem modifizierten Gram–Schmidt Verfahren | $0.293 \cdot 10^{-2}$ |
| $QR$–Zerlegung der Matrix $(A\|b)$ nach dem modifizierten GS Verfahren | $0.565 \cdot 10^{-7}$ |
| $QR$–Zerlegung nach dem Householderverfahren mit expliziter Berechnung von $Q$ | $0.315 \cdot 10^{-6}$ |
| $QR$–Zerlegung der Matrix $(A\|b)$ nach dem Householderverfahren mit impliziter Berechnung von $\mathbb{Q}^* b$ | $0.315 \cdot 10^{-6}$ |
| Singulärwertzerlegung | $0.177 \cdot 10^{-7}$ |

*Die einzelnen Approximationspolynome sind in Abbildung 4.7 dargestellt. Sie unterscheiden sich im Bereich, in dem angepaßt (gefittet) wird, kaum von der Funktion und voneinander. Außerhalb des Bereiches machen sich die Rechenfehler jedoch deutlich bemerkbar: Die einzelnen Approximationspolynome weisen bedeutende Unterschiede auf.*

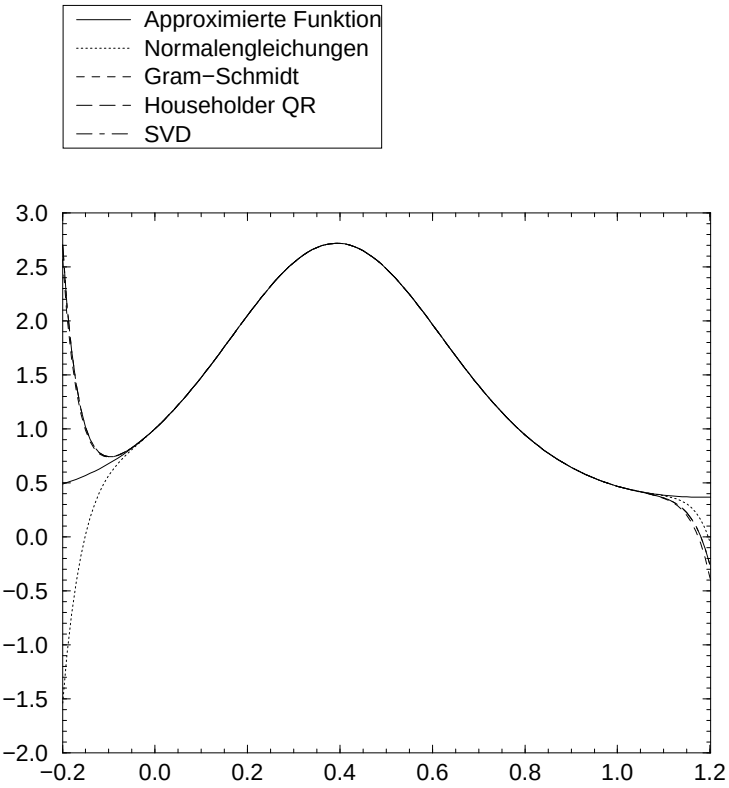



ABBILDUNG 4.7. Approximationspolynome 14–ten Grades zur Funktion $e^{\sin(4t)}$

Um Gutartigkeit zu untersuchen, müssen wir zuerst einmal den unvermeidbaren Fehler abschätzen. Nachdem das Kleinste–Quadrate–Problem stark mit $\| \ \|_2$–Approximationen verknüpft ist, ist es natürlich bei dieser Untersuchung die euklidische Norm in den Mittelpunkt zu stellen. Wie bei allen Untersuchungen der numerischen linearen Algebra wird die Konditionszahl (hier $\kappa_2(A)$) der involvierten Matrizen im Mittelpunkt stehen.

Drei hauptsächliche Größen werden benötigt, um den unvermeidbaren Fehler einer Kleinste–Quadrate–Approximation zu bestimmen:

(1) Die Konditionszahl $\kappa_2(A)$ der Koeffizientenmatrix $A$;
(2) Der Winkel $\theta = \arccos(\frac{\|Ax\|_2}{\|b\|_2})$, der die Güte der Approximation (die relative Größe des Residuums) mißt;
(3) Die Güte der Approximation von $\|Ax\|_2$ durch $\|A\|_2$ und $\|x\|_2$: $\eta := \frac{\|A\|_2 \, \|x\|_2}{\|Ax\|_2}$.

Es gelten folgende Beziehungen:

$$1 \leq \kappa_2(A) \leq \infty, \quad 0 \leq \theta \leq \tfrac{\pi}{2}, \quad 1 \leq \eta \leq \kappa_2(A).$$

Mit Hilfe dieser Größen können wir die Abhängigkeiten von $x$, $A$ und $b$ im folgenden Satz zusammenfassen.

**Theorem 2.5.2.** *Sei $b \in \mathbb{K}^n$, $A \in \mathbb{K}^{n\times m}$ und sei $x \in \mathbb{K}^m$ die Lösung des Kleinste–Quadrate–Problems. Dann sind die relativen $\|\quad\|_2$ Konditionszahlen gegeben durch*

$$\frac{\|\Delta x\|_2}{\|x\|_2} \leq \underbrace{\frac{\kappa_2(A)}{\eta\cos\theta}}_{\kappa_{b\mapsto x}} \frac{\|\Delta b\|_2}{\|b\|_2}$$

*und*

$$\frac{\|\Delta x\|_2}{\|x\|_2} \leq \underbrace{(\kappa_2(A) + \frac{\kappa_2(A)^2\tan\theta}{\eta})}_{\kappa_{A\mapsto x}} \frac{\|\Delta A\|_2}{\|A\|_2}.$$

*Im Spezialfall $m = n$ gilt $\theta = 0$ und damit reduzieren sich die Abschätzungen auf die üblichen Konditionszahlen $\kappa_2(A)/\eta$ und $\kappa_2(A)$.*

Beweis. Zur Bestimmung der Konditionszahlen transformieren wir das Problem in die einfachste Darstellung, die wir bislang kennengelernt haben. Sei $A = U\Sigma V^*$ die Singulärwertzerlegung von $A$. Nachdem alle Abweichungen in der 2–Norm gemessen werden, spielen unitäre Transformationen keine Rolle. Wir können also o.B.d.A. annehmen, daß $A$ eine verallgemeinerte Diagonalmatrix ist:

$$A = \begin{pmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_m \\ & & & \end{pmatrix} = \begin{pmatrix} B \\ 0 \end{pmatrix}.$$

Der orthogonale Projektor $P$ auf $\mathrm{im}(A)$ ist dann trivialerweise gegeben durch $P = \begin{pmatrix} \mathbb{I} & 0 \\ 0 & 0 \end{pmatrix}$ eine $n \times n$–Matrix mit einer $m \times m$–Einheitsmatrix in der linken oberen Ecke. Auch die Pseudoinverse von $A$ läßt sich einfach darstellen: $A^\dagger = \begin{pmatrix} B^{-1} & 0 \end{pmatrix}$. Daraus ergibt sich dank dem Zusammenhang $x = A^\dagger b$ die Abschätzung

$$\kappa_{b\mapsto x} = \frac{\|A^\dagger\|_2\|b\|_2}{\|x\|_2} = \|A^\dagger\|_2\frac{\|b\|_2}{\|Pb\|_2}\frac{\|Pb\|_2}{\|b\|_2} = \|A^\dagger\|_2\frac{1}{\cos\theta}\frac{\|A\|_2}{\eta} = \frac{\kappa_2(A)}{\eta\cos\theta},$$

wobei wir verwendet haben, daß $x$ Lösung des Kleinste–Quadrate–Problems ist und daher $Ax = Pb$ gilt.

Die Untersuchungen von Änderungen in $A$ gestaltet sich weitaus schwieriger. Eine Veränderung von $A$ ändert nämlich sowohl $\mathrm{im}(A)$ als auch die singulären Werte von $A$. Die Zusammenhänge sind im Gegensatz zu den Störungen in $b$ nichtlinear. Untersuchen wir zu Beginn die Abhängigkeit von $Pb$ von $A$. Diese Abhängigkeit ist von $\mathrm{im}(A)$ allein bestimmt. Wie ändert sich also $\theta$ wenn $A$ verändert wird?

Sei $\Delta A$ eine Änderung der (Diagonal-) Matrix $A$. Dann wird bei gegebener $\|\Delta A\|_2$ die maximale Drehung $\Delta\alpha$ ausgelöst, wenn man $\Delta A := \|\Delta A\|_2 u_n e_n^*$. Diese Störung von $A$ bewirkt eine Verdrehung von $\mathrm{im}(A)$ um einen Winkel von $\tan(\Delta\alpha) = \frac{\|\Delta A\|_2}{\sigma_n}$. Wegen $\Delta\alpha \dot{\leq} \tan(\Delta\alpha)$ haben wir

$$\Delta\alpha \dot{\leq} \frac{\|\Delta A\|_2}{\sigma_n} = \frac{\|\Delta A\|_2}{\|A\|_2}\kappa_2(A). \tag{27}$$

Die nächste Frage ist, was passiert bei Verdrehung von $\mathrm{im}(A)$ um den Winkel $\Delta\alpha$ mit dem Vektor $Pb$ (siehe zur folgenden Erklärung Abbildung 4.8). Wir wissen, daß $b - Pb$ orthogonal zu $Pb$ ist, und daher folgt aus dem Satz von Thales, daß egal wie $\mathrm{im}(A)$ verdreht wird $Pb$ immer auf der Sphäre $S_{\frac{1}{2}\|b\|_2}(\frac{1}{2}b)$ liegt ($S_r(x) := \{y \in \mathbb{K}^n | \, \|y - x\|_2 = r\}$). Der Vektor $Pb$ ist

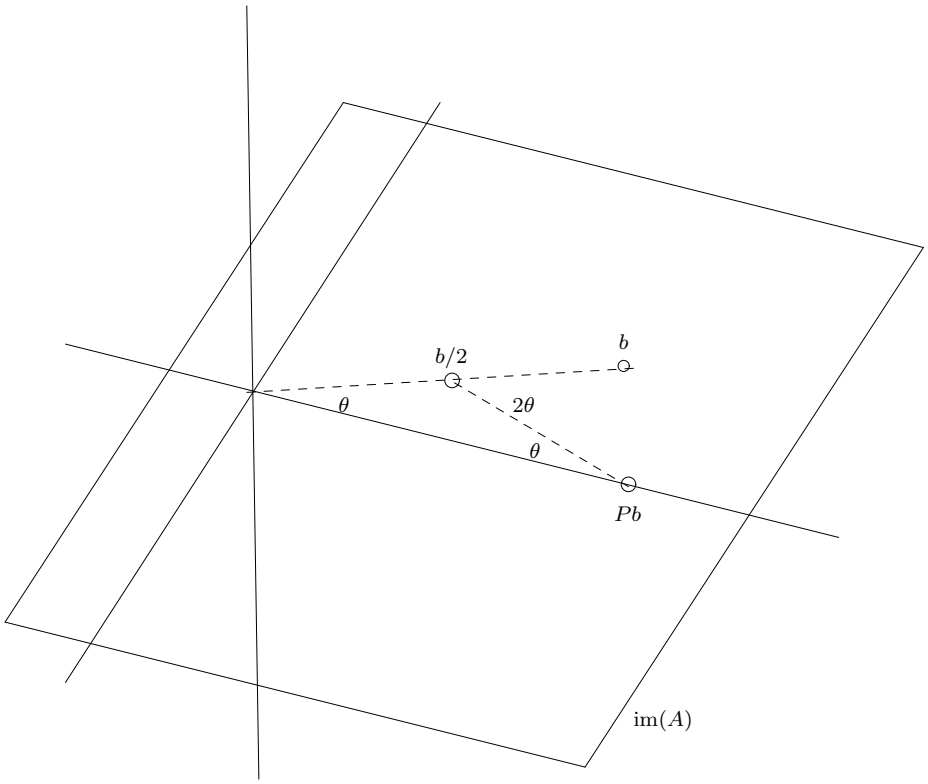



ABBILDUNG 4.8. Zusammenhang zwischen $Pb$, $b$ und $\mathrm{im}(A)$.

Basis eines gleichschenkeligen Dreiecks mit Basiswinkel $\theta$. Wird $\mathrm{im}(A)$ um den Winkel $\Delta\alpha$ verdreht, so wird auch der Winkel $\theta$ zwischen $b$ und $\mathrm{im}(A)$ um höchstens $\Delta\alpha$ vergrößert. Die Veränderung $\Delta(Pb)$ ist die Basis eines gleichschenkeligen Dreiecks mit Zentralwinkel $\leq 2\Delta\alpha$ und Schenkellänge $\frac{1}{2}\|b\|_2$. Demnach gilt

$$\|\Delta(Pb)\|_2 \leq \|b\|_2 \sin(\Delta\alpha) \dot{\leq} \|b\|_2 \Delta\alpha, \tag{28}$$

und wegen der Abschätzung (27) folgt

$$\frac{\|\Delta(Pb)\|_2}{\|Pb\|_2} \dot{\leq} \frac{\kappa_2(A)}{\cos\theta} \frac{\|\Delta A\|_2}{\|A\|_2}. \tag{29}$$

Nun fehlt zur Vollständigkeit der Abschätzungen noch der Rest der Zusammenhänge zwischen $x$ und $A$. Sei also

$$\Delta A = \begin{pmatrix} \Delta A_1 \\ \Delta A_2 \end{pmatrix} = \begin{pmatrix} \Delta A_1 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ \Delta A_2 \end{pmatrix}$$

mit der $m \times m$–Matrix $\Delta A_1$ und der verbleibenden $(n-m) \times m$–Matrix $\Delta A_2$, und seien $A$ und $b$ analog zerlegt. Veränderungen des Typs $\Delta A_1$ sind leicht zu analysieren, da von ihnen $\mathrm{im}(A)$ nicht verändert wird. Daher wird der Einfluß solch einer Störung durch die Konditionszahl von $A_1$ bestimmt:

$$\frac{\|\Delta x\|_2}{\|x\|_2} \leq \kappa_2(A_1) \frac{\|\Delta A_1\|_2}{\|A\|_2} = \kappa_2(A) \frac{\|\Delta A\|_2}{\|A\|_2}. \tag{30}$$

Störungen der Form $\Delta A_2$ verdrehen $\mathrm{im}(A)$ ohne aber die Abbildung innerhalb von $\mathrm{im}(A)$ zu verändern. Das entspricht einer Verdrehung von $b_1$ ohne $A_1$ zu ändern. Die Konditionszahl dieser Veränderung haben wir schon bei der Berechnung von $\kappa_{b \mapsto x}$ im ersten Teil des Beweises

bestimmt:

$$\frac{\|\Delta x\|_2}{\|x\|_2} \leq \frac{\kappa_2(A)}{\eta} \frac{\|\Delta b_1\|_2}{\|b_1\|_2}. \tag{31}$$

Um die Abschätzungen zu beenden müssen wir noch den Zusammenhang zwischen $\Delta b_1$ und $\Delta A_2$ herausarbeiten. $\Delta b_1$ reagiert nur auf Veränderungen parallel zu $\mathrm{im}(A)$. Diese hängen aber direkt mit der Veränderung $\Delta(Pb)$ zusammen, die wir schon bestimmt haben. Die Veränderung $\Delta(Pb)$ liegt nicht parallel zu $\mathrm{im}(A)$ sondern in einem Winkel von $\pi/2 - \theta$ zu $\mathrm{im}(A)$. Daher gilt $\|\Delta b_1\|_2 = \|\Delta(Pb)\| \sin\theta$, und nach der Abschätzung (28) haben wir

$$\|\Delta b_1\| \dot{\leq} (\|b\|_2 \Delta\alpha) \sin\theta \dot{\leq} \|b\|_2 \frac{\|\Delta A\|_2}{\|A\|_2} \kappa_2(A) \sin\theta.$$

Weil $\|b_1\|_2 = \|b\|_2 \cos\theta$ gilt, folgt

$$\frac{\|\Delta b_1\|_2}{\|b_1\|_2} \dot{\leq} \frac{\|\Delta A\|_2}{\|A\|_2} \kappa_2(A) \tan\theta,$$

und zusammen mit (31) gelangen wir schließlich zu

$$\frac{\|\Delta x\|_2}{\|x\|_2} \dot{\leq} \frac{\kappa_2(A)\tan\theta}{\eta} \frac{\|\Delta A_2\|_2}{\|A\|_2},$$

was zu (30) addiert schließlich die behauptete Abschätzung ergibt. □

Nachdem wir nun endlich die Kondition des Problemes festgestellt haben, können wir den Ergebnissen von Beispiel 2.5.1 auf den Grund gehen. Bestimmen wir zuerst die drei Kenngrößen:

$$\kappa_2(A) = 0.22718 \cdot 10^{11}, \quad \theta = 0.37461 \cdot 10^{-5}, \quad \eta = 0.21036 \cdot 10^{6}.$$

Die Matrix $A$ ist also besonders schlecht konditioniert, und die Kleinheit des Winkels $\theta$ bedeutet, daß $e^{\sin(4t)}$ sehr gut durch ein Polynom 14–ten Grades angenähert werden kann. $\eta$ liegt etwa in der Mitte zwischen den möglichen Werten 1 und $\kappa_2(A)$. Wir können aus den Formeln von Theorem 2.5.2 sofort die Einflußzahlen

$$\kappa_{b\mapsto x} = 0.11 \cdot 10^{6}, \quad \kappa_{A\mapsto x} = 0.32 \cdot 10^{11}$$

berechnen. Jetzt können wir auch die relativen Fehler der von den einzelnen Algorithmen berechneten Lösungen bestimmen.

**2.5.1. Normalengleichungen.** Das Ergebnis, das die Normalengleichungen liefern, ist katastrophal schlecht. Man kann die Instabilität förmlich riechen. Trotzdem werden auch heute die Normalengleichungen häufig zur Lösung eines Kleinste–Quadrate–Problemes herangezogen. Der Grund ist meist Geschwindigkeit. Für $m \gg n$ benötigt diese Methode etwa den halben Aufwand einer $QR$–Zerlegung ($\sim m^2 n$ Flops). Der Grund für die Instabilität ist wie folgt: Nachdem die Cholesky–Zerlegung ein stabiler Algorithmus ist, gilt, daß die berechnete Lösung $\tilde{x}$ von $A^*Ax = A^*b$ ein benachbartes Gleichungssystem $(A^*A + F)\tilde{x} = A^*b$ erfüllt mit $\|H\|_2/\|A^*A\|_2 = O(\mathsf{eps})$. Dann folgt aus der Theorie über quadratische lineare Gleichungssysteme

$$\frac{\|x - \tilde{x}\|_2}{\|x\|_2} = O(\kappa_2(A^*A)\,\mathsf{eps}) = O(\kappa_2(A)^2\,\mathsf{eps}).$$

Die Kondition des Problemes ist aber $\kappa_2(A) + \frac{\kappa_2(A)^2 \tan\theta}{\eta}$. Die Normalengleichungen liefern also nur dann eine vertretbar gute Lösung, wenn entweder $\tan(A) \approx 1$ oder $\eta \ll \kappa_2(A)$. Falls andererseits $\kappa_2(A)$ sehr groß ist und entweder $\tan(A) \approx 0$ oder $\eta \approx \kappa_2(A)$ ist, so liefern die Normalengleichungen meist schlechte Lösungen. Wir haben folgende Proposition

**Proposition 2.5.3.** *Die Lösung des Kleinste–Quadrate–Problems unter Verwendung der Normalengleichungen ist instabil. Stabilität ist nur dann gegeben, wenn* $\tan\theta/\eta$ *groß oder* $\kappa_2(A)$ *klein ist.*

Im Beispielfall ist $\kappa_2(A)^2 \approx 0.5 \cdot 10^{21}$. Bei der verwendeten doppelten Rechengenauigkeit, die etwa Rundung auf 16 Stellen entspricht, ist es nicht verwunderlich, daß relative Fehler von Größenordnung 1 auftreten.

**2.5.2. Gram–Schmidt Verfahren.** Hier muß man den Grund für das schlechte Ergebnis im Test an einer anderen Stelle als zuvor suchen. Zuerst erkennen wir, daß die Rundungsfehler um einen Faktor $10^{14}$ verstärkt worden sind, also etwa einen Faktor 10000 über der durch die schlechte Konditionierung bedingten zu erwartenden Verstärkung liegt. Der Grund dafür liegt darin, daß das Gram–Schmidt Verfahren Matrizen $Q$ erzeugt, deren Spalten nicht orthogonal sind. Dies ist jedoch ein zentraler Punkt bei der Umformung der Normalengleichungen.

**Proposition 2.5.4.** *Die Lösung des Kleinste–Quadrate–Problems mit QR–Zerlegung, die nach dem (modifizierten) Gram–Schmidt Verfahren erzeugt wurde, ist instabil wegen Verlust der Orthogonalität in* $Q$*.*

Man kann diese Probleme jedoch umgehen, indem man verwendet, daß wenigstens das Produkt $\tilde{Q}\tilde{R}$, der berechneten Matrizen mit $A$ sehr gut übereinstimmt. Man kann also die Normalengleichungen

$$(\tilde{Q}^*\tilde{Q})Rx = \tilde{Q}^*b$$

für $Rx$ aufstellen und $x$ durch Rückwärtssubstitution berechnen. Üblicherweise ist das berechnete $\tilde{Q}$ gut konditioniert, und daher sind die Normalengleichungen stabil lösbar. Der Mehraufwand ist jedoch bedeutend und sollte daher vermieden werden.

Eine weitere Möglichkeit ist, den Vektor $b$ gleich mitzuorthonormalisieren und daher die Matrix $(A|b)$ mit dem modifizierten Gram–Schmidt Verfahren $QR$ zu zerlegen $(A|b) = \mathcal{Q}\mathcal{R}$. Dann erhält man den Vektor $Q^*b$ als die letzte Spalte von $\mathcal{R}$ und $\tilde{R}$ als die ersten $m$ Spalten von $\mathcal{R}$. Löst man nun $\tilde{R}x = Q^*b$ mit den so berechneten Komponenten, so erhält man einen stabilen Algorithmus.

**Proposition 2.5.5.** *Der Algorithmus zur Berechnung der Kleinste–Quadrate–Lösung mit Hilfe des modifizierten Gram–Schmidt Verfahrens ist gutartig, wenn* $Q^*b$ *und* $R$ *implizit durch die QR–Zerlegung der erweiterten Matrix* $(A|b)$ *berechnet werden.*

**2.5.3. Householder Verfahren.** Die berechnete Lösung im Beispiel hat einen relativen Fehler von $0.315 \cdot 10^{-6}$, was einer Fehlerverstärkung um den Faktor $10^{10}$ entspricht. Das war durch die schlechte Konditionierung des Problemes auch etwa zu erwarten. Man sieht auch, daß es keinen Unterschied macht, ob man $Q$ berechnet oder $Q^*b$ iterativ aus der Zerlegung bestimmt.

**Proposition 2.5.6.** *Sei* $\tilde{x}$ *die Kleinste–Quadrate–Lösung des Problems* $Ax = b$ *berechnet mit dem Householder Verfahren. Dann hat* $\tilde{x}$ *folgende Eigenschaft:*

$$\|(A+H)\tilde{x} - b\|_2 = \min, \quad \frac{\|H\|_2}{\|A\|_2} = O(\mathsf{eps}).$$

*für eine Fehlermatrix* $H \in \mathbb{K}^{n\times m}$*. Dabei macht es keinen Unterschied, ob* $Q$ *berechnet wird oder ob* $Q^*b$ *implizit bestimmt wird mittels Algorithmus 2.2.8.*

Ein analoges Resultat gilt für die Berechnung der $QR$–Zerlegung mittels Givens Rotationen.

**2.5.4. Singulärwertzerlegung.** Im Beispiel liefert die Singulärwertzerlegung die genaueste Lösung mit einem relativen Fehler von lediglich $0.177 \cdot 10^{-7}$, was noch unter dem erwarteten Fehler liegt. Wie fast alle Algorithmen, die auf der Singulärwertzerlegung basieren, ist auch die Lösung des Kleinste–Quadrate–Problemes auf diese Weise ein gutartiger Algorithmus. Es gilt eine Proposition analog zu 2.5.6.

## 3. Lineare unterbestimmte Systeme

Linear unterbestimmte Systeme und überbestimmte Systeme, bei denen $A$ nicht vollen Rang hat, treten nicht allzu häufig auf. Meist verlangt man dann zusätzlich, um eine der unendlich vielen möglichen Lösungen auszuwählen, die Minimalität von $\|x\|_2$. Solche Probleme kommen üblicherweise aus zwei unterschiedlichen Quellen.

- $A$ hat eigentlich vollen Rang, aber der eps–Rang von $A$ ist nicht maximal, das heißt $A$ ist *numerisch von geringerem Rang.*
- $A$ hat durch Konstruktion niedrigen Rang und eigentlich haben wir es mit einem einfachen Optimierungsproblem mit linearen Gleichungsnebenbedingungen zu tun.

**3.1. Berechnung.** Der eigentlich einzige stabile Algorithmus zur Lösung des Problems basiert auf der Singulärwertzerlegung.

**Theorem 3.1.1.** *Sei $A \in \mathbb{K}^{n\times m}$ mit Rang $\operatorname{rg}(A) = r < m$. Sei $A = U\Sigma V^*$ die Singulärwertzerlegung von $A$. Dann ist*

$$x := \sum_{i=1}^{r} \frac{\langle u_i, b\rangle}{\sigma_i} v_i$$

*der Vektor in $\mathbb{K}^m$ mit minimaler 2–Norm $\|x\|_2$, der $\|Ax - b\|_2$ minimiert. Es gilt*

$$\|Ax - b\|_2 = \sqrt{\sum_{i=r+1}^{n} \langle u_i, b\rangle}.$$

Beweis. Für jedes $x \in \mathbb{K}^m$ gilt

$$\|Ax - b\|_2^2 = \|(U^*AV)(V^*x) - U^*b\|_2^2 = \|\Sigma w - U^*b\|_2^2 =$$
$$= \sum_{i=1}^{r} (\sigma_i w_i - \langle u_i, b\rangle)^2 + \sum_{i=r+1}^{n} \langle u_i, b\rangle$$

mit $w = U^*x$. Offensichtlich gilt für eine Lösung $x$ des Kleinste–Quadrate–Problems $w = (u_i^* b)/\sigma_i$ für $i = 1, \ldots, r$. Setzen wir $w_j = 0$ für $j \geq r + 1$, dann hat das entstehende $x$ minimale Norm. □

Der Satz gibt auch Auskunft über Minimalnormlösungen eines echten unterbestimmten linearen Gleichungssystemes ($A \in \mathbb{K}^{n\times m}$ mit $n < m$). Dann ist $\operatorname{rg}(A) = n$, und jede Lösung $x$ des Gleichungssystemes ist eine Kleinste–Quadrate–Lösung mit $\|Ax - b\|_2 = 0$. Wieder ist

$$x := \sum_{i=1}^{n} \frac{\langle u_i, b\rangle}{\sigma_i} v_i$$

die Lösung mit minimaler Norm.

Dies ist ein typisches Beispiel, wie man eine Lösung eines Problemes $A$ erhalten kann, indem man ein anderes Problem $B$, dessen Lösung man bereits kennt so verallgemeinert, daß das neue Problem $C$ auch eine Verallgemeinerung vom ursprünglichen Problem $A$ ist.

Eine weitere Methode zur Behandlung des Kleinste–Quadrate–Problemes, wo $A$ nicht vollen Rang hat, ist durch $QR$–Zerlegung mit Spaltenpivotsuche. Dies ergibt einen schnelleren

Algorithmus, der jedoch nicht so stabil ist wie die Singulärwertzerlegung. Dieses Verfahren wird z.B. in [**Golub, Van Loan 1996**, 5.5.6] beschrieben.

## 4. Software

Auch hier seien wieder Hinweise auf LAPACK und MATLAB gegeben.

**4.1. LAPACK.** Die genauen Beschreibungen der LAPACK Routinen kann man dem LAPACK Handbuch entnehmen:

Einerseits existieren Funktionen zur Erzeugung von Householder Matrizen und von Givens Rotationen:

| LAPACK | Beschreibung |
|---|---|
| `?larfg` | Erzeugt eine Householder Spiegelung |
| `?larf` | Berechnet das Produkt einer Householder Spiegelung mit einer Matrix |
| `?larfx` | Kleine $n$–Householder Spiegelung mal Matrix |
| `?larft` | Berechne Block Householder Spiegelung |
| `?larfb` | Block Householder Spiegelung mal Matrix |
| `?lartg` | Erzeugt eine Givens Rotation |
| `?largv` | Erzeugt einen Vektor von Givens Rotationen |
| `?lartv` | Wendet einen Vektor von Givens Rotationen auf ein Vektorpaar an |
| `?lasr` | Wendet eine Folge von Givens Rotationen auf eine Matrix an |

Andererseits stellt LAPACK auch Möglichkeiten zur Verfügung, Zerlegungen direkt zu berechnen:

| LAPACK | Beschreibung |
|---|---|
| `?geqrf` | Berechnet eine $QR$–Zerlegung |
| `?geqpf` | Berechnet eine $QR$–Zerlegung mit Spaltenpivotsuche |
| `?ormqr` | Multipliziert $Q$ in faktorisierter Form mit einer Matrix (reell) |
| `?unmqr` | Multipliziert $Q$ in faktorisierter Form mit einer Matrix (komplex) |
| `?orgqr` | Erzeugt die Matrix $Q$ (reell) |
| `?ungqr` | Erzeugt die Matrix $Q$ (komplex) |
| `?gesvd` | Bestimmt die Singulärwertzerlegung einer Matrix |

Außerdem kann LAPACK unter interner Verwendung der geeigneten Verfahren gleich das gesamte Kleinste–Quadrate–Problem lösen:

| LAPACK | Beschreibung |
|---|---|
| `?gels` | Minimiert $\|AX - B\|_2$ für Matrix $A$ von vollem Rang |
| `?gelss` | Minimiert $\|AX - B\|_2$ für beliebige Matrix $A$ unter Verwendung der Singulärwertzerlegung |
| `?geequ` | Equilibriert eine Matrix |

**4.2. MATLAB.** Die Funktion

```
[Q,R] = qr(A)
```

berechnet eine volle $QR$–Zerlegung von $A$ (meist nach dem Householder Verfahren). Eine noch etwas stabilere Variante, die $QR$–Zerlegung mit Spaltenpivotsuche erhält man durch Aufruf von

```
[Q,R,P] = qr(A)
```

Die Permutationsatrix $P$ ist so gewählt, daß $QR = PA$ gilt und außerdem die Diagonalelemente von $R$ monoton fallend sind.

Möchte man reduzierte $QR$–Zerlegungen berechnen, so verändert man die Funktionsaufrufe ein wenig:

```
[Q,R] = qr(A,0)
[Q,R,P] = qr(A,0)
```

Die Pseudoinverse $A^\dagger$ von $A$ erhält man durch Aufruf der Funktion

```
Api = pinv(A)
```

Die zweite wichtige Zerlegung, die Singulärwertzerlegung wird mit der Funktion

```
[U,Si,V] = svd(A)
```

bestimmt (`svd` ist die Abkürzung von „*singular value decomposition*“, dem englischen Ausdruck für Singulärwertzerlegung). Wieder berechnet der oben angegebene Funktionsaufruf die volle Zerlegung, und analog zur $QR$–Zerlegung erzeugt

```
[U,Si,V] = svd(A,0)
```

die reduzierte Variante. Benötigt man nicht alle sondern nur dir $K$ größten Singulärwerte, so kann man diese schneller mit der Funktion

```
S = svds(A,K)
```

finden, die eine sortierte Liste dieser $K$ größten Singulärwerte zurückliefert.

Um den Rang einer Matrix zu bestimmen, kann man auf die Funktion

```
n = rank(A)
```

zurückgreifen. Den $\varepsilon$–Rang von $A$ kann man ebenfalls mit dieser Funktion bestimmen, indem man die Toleranz $\varepsilon$ als zusätzlichen Parameter angibt:

```
n = rank(A,eps)
```

KAPITEL 5

# Interpolation I: Eindimensionaler Fall

## 1. Grundlagen

Wie wir schon in Kapitel 2 in den Anwendungsbeispielen 2.8, 3.4 und 3.5 gesehen haben, ist es öfters wünschenswert, durch vorgegebene Punkte Kurven oder Flächen zu legen. Dabei denkt man sich, oder es folgt aus einer Theorie, daß die gegebenen Punkte in Wirklichkeit auf einer durchgehenden Kurve liegen, die möglichst gut approximiert werden soll. Im einfachsten Fall, den wir in diesem Kapitel besprechen, geht es darum, eine eindimensionale Funktion zu approximieren.

Manchmal ist es über das Approximationsproblem hinaus auch interessant, den Verlauf einer „durch Punkte gegebenen Funktion“ außerhalb des durch die Punkte überdeckten Bereichs zu schätzen, also den Verlauf zu extrapolieren.

Das Problem, das wir in diesem Kapitel behandeln wollen, läßt sich folgendermaßen definieren.

**Definition 1.0.1.** *Sei $\varphi(x) : \mathbb{R} \to \mathbb{R}$ eine FunktiOn. Wir sagen $\varphi$* interpoliert *die Punkte $(x_j, f_j)$, $j = 0, \ldots, n$ mit $x_i \neq x_k$ für $i \neq k$, falls für alle $j$*

$$\varphi(x_j) = f_j$$

*gilt. Die Punkte $(x_j, f_j)$ heißen die* Stützstellen*, die Zahlen $x_j$ nennen wir* Stützabszissen *und die $f_j$* Stützordinaten*.*

*Sei $\mathcal{F}$ eine Klasse reeller Funktionen $\Phi(x, a_0, \ldots, a_n)$, die von $n+1$ Parametern $a_i$ abhängen. Bei der Lösung eines* Interpolationsproblemes *zu vorgegebenen Stützstellen $(x_j, f_j)$ geht es darum, jenes $\Phi \in \mathcal{F}$ (also die Parameterwerte $a_i$) zu finden, das die Punkte $(x_j, f_j)$ interpoliert. Wir sprechen von einem* linearen Interpolationsproblem*, falls die Parameter $a_i$ linear in $\Phi$ auftreten, also*

$$\Phi(x, a_0, \ldots, a_n) = a_0\Phi_0(x) + \cdots + a_n\Phi_n(x)$$

*gilt. Andernfalls sprechen wir von einem* nichtlinearen Interpolationsproblem*.*

Zu den linearen Interpolationsproblemen gehören die Interpolation durch

- Polynome,
- trigonometrische Funktionen,
- Splines.

Das einzige nichtlineare Interpolationsproblem, das wir besprechen werden, ist die Interpolation durch

- rationale Funktionen.

**1.1. Polynome.** Die älteste Form der Interpolation ist die Interpolation durch Polynome. Die Klasse $\mathcal{F}$ von Interpolationsfunktionen ist dann $\mathbb{R}^{(n)}[x]$, die Menge der Polynome höchstens $n$–ten Grades, also Funktionen der Gestalt

$$\Phi(x, a_0, \ldots, a_n) = \sum_{j=0}^{n} a_j x^j.$$

Diese Art der Interpolation geht in die Zeit von Newton, Lagrange zurück, und es gibt sehr viel Material und viele verschiedenartige Algorithmen zur Behandlung dieser Interpolationsprobleme. Nachdem Polynome jedoch sehr „starre“ Funktionen sind, die die (optische) Form der Punktmenge nur sehr schlecht repräsentieren, wird die Polynominterpolation in letzter Zeit nicht mehr so häufig verwendet wie in der Vergangenheit. Als Beispiel diene noch einmal das folgende Interpolationspolynom 10–ten Grades durch die mit × gekennzeichneten Punkte, dargestellt in Abbildung 5.1. Die Oszillationsphänomene am Ende des interpolierten

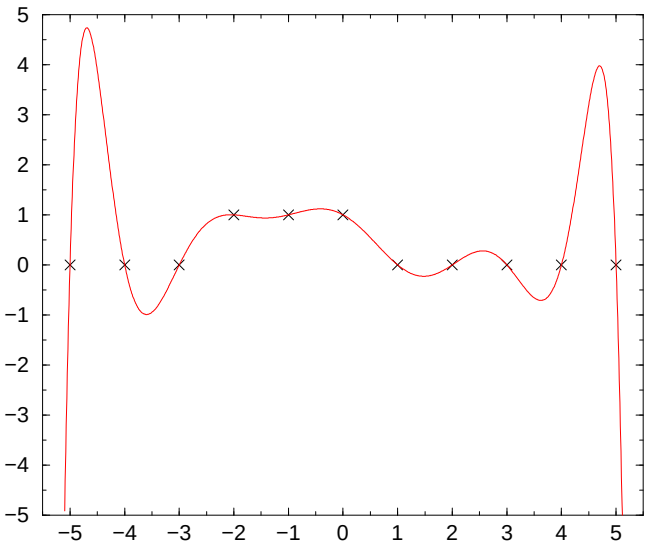



ABBILDUNG 5.1. Interpolationspolynom 10–ten Grades

Bereiches sind typisch für Interpolationspolynome. Erhöht man die Anzahl der Stützstellen (und damit den Grad des Polynomes) verbessert sich dieses Verhalten nicht. Im Gegenteil, es wird meist noch schlimmer. Daher weicht man heute meist auf andere Funktionsklassen aus, die die „Form“ der Punktmengen besser repräsentieren. Da jedoch die meisten dieser Funktionsklassen mehr oder weniger auf Polynomen aufbauen, ist es zumindest nötig, sich die Grundlagen der Polynominterpolation anzueignen.

Alles beginnt mit einem Existenz- und Eindeutigkeitssatz:

**Theorem 1.1.1.** *Zu beliebigen $n+1$ Stützstellen $(x_j, f_j)$, $j = 0, \dots, n$ existiert genau ein Polynom $P \in \mathbb{R}^{(n)}[x]$ höchstens $n$–ten Grades, das die Punkte $(x_j, f_j)$ interpoliert.*

BEWEIS. **Eindeutigkeit:** Angenommen, $P$ und $Q$ seien zwei Polynome höchstens $n$–ten Grades, die das Interpolationsproblem lösen. Dann gilt

$$(P - Q)(x_i) = P(x_i) - Q(x_i) = f_i - f_i = 0$$

für alle $i$. Das Polynom $P - Q$ ist als Differenz zweier Polynome höchstens $n$–ten Grades selbst wieder ein Polynom höchstens $n$–ten Grades. $P - Q$ hat aber nach der obigen Rechnung $n+1$ Nullstellen, ist somit das Nullpolynom. Daher folgt $P \equiv Q$, was die Eindeutigkeit beweist.

**Existenz:** Der Existenzbeweis erfolgt konstruktiv. Wir lösen zuerst die einfacheren Interpolationsprobleme

$$L_i(x_k) = \delta_{ik}, \quad \delta_{ik} = \begin{cases} 1 & i = k, \\ 0 & \text{sonst}, \end{cases}$$

also für die Stützstellen $(x_j, \delta_{ij})$. Die Interpolationspolynome für diese Stützstellen heißen *Lagrangesche Interpolationspolynome* und lassen sich explizit angeben:

$$\begin{aligned} L_i(x) &:= \frac{(x-x_0)\dots(x-x_{i-1})(x-x_{i+1})\dots(x-x_n)}{(x_i-x_0)\dots(x_i-x_{i-1})(x_i-x_{i+1})\dots(x_i-x_n)} \\ &= \frac{\omega(x)}{(x-x_i)\omega'(x_i)} \end{aligned}$$

mit $\omega(x) = \prod_{k=0}^{n}(x-x_k)$. Wegen des ersten Beweisteiles sind die $L_i$ eindeutig bestimmt. Die Lösung des allgemeinen Interpolationsproblemes läßt sich dann einfach aus diesen elementaren Bausteinen berechnen:

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

Es gilt

$$P(x_k) = \sum_{i=0}^{n} f_i L_i(x_k) = \sum_{i=0}^{n} f_i \delta_{ik} = f_k.$$

Diese Darstellung heißt die *Lagrangesche Darstellung* des Interpolationspolynoms oder die *Lagrangesche Interpolationsformel*. □

**Beispiel 1.1.2.** *Für $n = 2$ seien die drei Punkte $(0,1)$, $(1,3)$ und $(3,2)$ gegeben. Dann sind die elementaren Polynome gegeben als*

$$L_0(x) = \frac{(x-1)(x-3)}{(0-1)(0-3)}, \quad L_1(x) = \frac{(x-0)(x-3)}{(1-0)(1-3)}, \quad L_2(x) = \frac{(x-0)(x-1)}{(3-0)(3-1)}.$$

*$P$ ergibt sich dann zu*

$$P(x) = L_0(x) + 3L_1(x) + 2L_2(x) = \tfrac{1}{6}(-5x^2 + 17x + 6).$$

$P(2) = \frac{1}{6}(-5 \cdot 2^2 + 17 \cdot 2 + 6) = \frac{10}{3}$.

Unglücklicherweise ist die Lagrangesche Darstellung zwar sehr brauchbar für Beweise und theoretische Untersuchungen, doch völlig ungeeignet für numerische Berechnungen. Sowohl Berechnung als auch Auswertung sind viel zu aufwendig. Die beiden folgenden Algorithmen bauen die Lösung des Problemes für $n+1$ Punkte schrittweise aus den Lösungen für weniger Punkte auf.

**1.1.1. Der Algorithmus von Neville.** Möchte man den Wert des Interpolationspolynomes nur an einer bestimmten Stelle berechnen (oder an wenigen Stellen), so kann man dafür den Algorithmus von Neville verwenden. Sei dazu

$$P_{i_0 i_1 \dots i_k}(x)$$

das Interpolationspolynom aus $\mathbb{R}^{(k)}[x]$, das das Interpolationsproblem

$$P_{i_0 i_1 \dots i_k}(x_{i_r}) = f_{i_r}$$

löst für $r = 0, \dots, k$. Man kann diese $P$ dann rekursiv berechnen

$$P_i(x) \equiv f_i \tag{32}$$

$$P_{i_0 i_1 \dots i_k}(x) = \frac{(x-x_{i_0})P_{i_1 \dots i_k}(x) - (x-x_{i_k})P_{i_0 \dots i_{k-1}}(x)}{x_{i_k} - x_{i_0}}. \tag{33}$$

Der Algorithmus von Neville berechnet aus dem folgenden Tableau den Wert des Interpolationspolynoms an einer fixen Stelle $y$ gemäß dieser Rekursionsformel:

| | $k=0$ | 1 | 2 | 3 |
|---|---|---|---|---|
| $x_0$ | $f_0 = P_0(y)$ | | | |
| | | $P_{01}(y)$ | | |
| $x_1$ | $f_1 = P_1(y)$ | | $P_{012}(y)$ | |
| | | $P_{12}(y)$ | | $P_{0123}(y)$ |
| $x_2$ | $f_2 = P_2(y)$ | | $P_{123}(y)$ | |
| | | $P_{23}(y)$ | | |
| $x_3$ | $f_3 = P_3(y)$ | | | |

Dabei werden die Elemente $P_{i_0\dots i_k}$ aus ihren linken Nachbarn und den passenden Stützabszissen gemäß der Rekursionsformel (33) berechnet.

**Beispiel 1.1.3.** *Den Wert $P(2)$ für das Interpolationsproblem aus Beispiel 1.1.2 können wir gemäß dem Neville Algorithmus errechnen:*

| | | | |
|---|---|---|---|
| 0 | $P_0(2) = 1$ | | |
| | | $P_{01}(2) = 5$ | |
| 1 | $P_1(2) = 3$ | | $P_{012}(2) = \frac{10}{3}$ |
| | | $P_{12}(2) = \frac{5}{2}$ | |
| 3 | $P_2(2) = 2$ | | |

Der Algorithmus ist sehr effizient, wenn man das Interpolationspolynom nur an einer bestimmten Stelle auswerten möchte. Er ist nicht besonders gut geeignet, wenn man die Koeffizienten des Polynoms berechnen möchte oder den Wert des Polynoms auch nur an mehreren Stellen finden möchte.

Eine etwas ausführlichere Abhandlung findet man z.B. in [**Stoer 1994a**, 2.1.2] oder in [**Schwarz 1986**, 3.5].

**1.1.2. Die Newtonsche Interpolationsformel. Dividierte Differenzen.** Wenn man eine der Aufgaben lösen möchte, für die der Nevillsche Algorithmus nicht geeignet ist, also die Koeffizienten des Polynomes zu bestimmen oder das Polynom an mehreren Stellen auszuwerten, dann bewährt sich der *Newtonsche Algorithmus*.

Newton schlägt den folgenden Ansatz für das Interpolationspolynom vor:

$$P_{01\dots n}(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + \cdots + a_n(x - x_0)\dots(x - x_{n-1}).$$

Dieser Ansatz resultiert aus einer Verallgemeinerung der „Darstellung mit verschobenem Zentrum“

$$p(x) = \sum_{i=0}^{n} a_i(x - c)^i$$

eines Polynomes $p$. Die Auswertung eines so dargestellten Polynoms erfolgt recht effizient mit einer Verallgemeinerung des Horner Schemas gemäß der Formel

$$P(y) = a_0 + (x - x_0)(a_1 + (x - x_1)(a_2 + \dots (x - x_{n-2})(a_{n-1} + (x - x_{n-1})a_n)\dots)).$$

Ferner können die Koeffizienten $a_i$ durch Einsetzen rekursiv bestimmt werden. Setzen wir in $P_{01\dots n}$ den Wert $x_0$ ein, so erhalten wir sofort die Gleichung

$$a_0 = f_0.$$

Fahren wir sukzessiv mit den Werten $x_1, \ldots, x_n$ fort, so finden wir die Beziehungen

$$\begin{aligned} f_1 &= P(x_1) = a_0 + a_1(x_1 - x_0) \\ f_2 &= P(x_2) = a_0 + a_1(x_2 - x_0) + a_2(x_2 - x_0)(x_2 - x_1) \\ &\vdots \end{aligned}$$

Verwenden wir zusätzlich die rekursive Beziehung (33), so können wir die Gleichungen weiter vereinfachen. Beobachten wir folgendes Faktum: $P_{i_0 i_1 \ldots i_{k-1}}$ und $P_{i_0 i_1 \ldots i_k}$ unterscheiden sich durch ein Polynom höchstens $k$–ten Grades, das die $k$ Nullstellen $x_{i_0}, \ldots, x_{i_{k-1}}$ besitzt. Sei mit

$$f_{i_0 i_1 \ldots i_k}$$

der Koeffizient von $x^k$ des Polynoms $P_{i_0 i_1 \ldots i_k}$ bezeichnet. Dann gilt

$$P_{i_0 i_1 \ldots i_k}(x) = P_{i_0 i_1 \ldots i_{k-1}}(x) + f_{i_0 i_1 \ldots i_k}(x - x_{i_0}) \ldots (x - x_{i_{k-1}}).$$

Aus der rekursiven Beziehung (33) folgt dann eine rekursive Gleichung für die $f_{i_0 i_1 \ldots i_k}$:

$$f_{i_0 i_1 \ldots i_k} = \frac{f_{i_1 \ldots i_k} - f_{i_0 \ldots i_{k-1}}}{x_{i_k} - x_{i_0}}.$$

Wegen dieses Zusammenhanges nennt man die $f_{i_0 i_1 \ldots i_k}$ die *$k$–ten dividierten Differenzen*. Als Konsequenz erhalten wir die *Newtonsche Darstellung* des Interpolationspolynoms

$$P_{01\ldots n}(x) = f_0 + f_{01}(x - x_0) + \cdots + f_{01\ldots n}(x - x_0) \ldots (x - x_{n-1}).$$

Nachdem das Interpolationspolynom $P_{i_0 i_1 \ldots i_k}$ nur von den Punkten $(x_{i_j}, f_{i_j})$ aber nicht von deren Reihenfolge abhängt, ist auch der Koeffizient $f_{i_0 i_1 \ldots i_k}$ nicht von der Reihenfolge der Indizes abhängig. Die dividierten Differenzen sind also invariant unter Permutation der Indizes. Nachdem man zur Darstellung des Interpolationspolynoms nur die dividierten Differenzen $f_{01\ldots k}$ benötigt, bestimmt man die $f$'s in folgendem Tableau, dem *Differenzenschema*, in dessen erster Zeile am Ende die gesuchten Koeffizienten stehen:

$$\begin{array}{c|cccc} x_0 & \underline{f_0} & & & \\ & & \underline{f_{01}} & & \\ x_1 & f_1 & & \underline{f_{012}} & \\ & & f_{12} & \vdots & \ddots \\ x_2 & f_2 & \vdots & & \\ \vdots & \vdots & & & \end{array}$$

Wie beim Algorithmus von Neville berechnet man auch hier die einzelnen Einträge aus den beiden linken Nachbarn und den dazugehörigen Stützabszissen. Die unterstrichenen Tabelleneinträge sind dabei die gesuchten Koeffizienten des Interpolationspolynoms.

**Beispiel 1.1.4.** *Zu den Stützstellen von Beispiel 1.1.2 berechnen wir das Differenzenschema*

$$\begin{array}{c|cccc} 0 & f_0 = 1 & & & \\ & & f_{01} = 2 & & \\ 1 & f_1 = 3 & & f_{012} = -\frac{5}{6} & \\ & & f_{12} = -\frac{1}{2} & & \\ 3 & f_2 = 2 & & & \end{array}$$

*Das führt zum Interpolationspolynom*

$$P_{012}(x) = 1 + 2x - \tfrac{5}{6}x(x - 1)$$

*und zum Wert*

$$P_{012}(2) = 1 + 4 - \tfrac{5}{6}2 = \tfrac{10}{3}.$$

**Algorithmus 1.1.5.** *Newtonsches Interpolationspolynom*

$a = (f_0, \ldots, f_n)$
**for** $k = 1$ **to** $n$ **do**
  **for** $i = n$ **to** $k$ **step** $-1$ **do**
    $a[i] = (a[i] - a[i-1])/(x_i - x_{i-k})$
  **done**
**done**

Nach Beendigung dieses Algorithmus stehen im Vektor $a$ die Koeffizienten (für die Newtonsche Darstellung) des gesuchten Interpolationspolynoms. Der Aufwand zur Berechnung beträgt $O(n^2)$ elementare Operationen.

Die Auswertung des so gewonnenen Interpolationspolynomes an einer beliebigen Stelle erfolgt praktischer Weise gemäß dem folgenden Algorithmus.

**Algorithmus 1.1.6.** *Verallgemeinertes Horner Schema zur Auswertung des Newtonschen Interpolationspolynomes*

$p = a[n]$
**for** $k = n - 1$ **to** $0$ **step** $-1$ **do**
  $p = a[k] + p(x - x_k)$
**done**

Weitere Informationen zur Newtonschen Interpolation kann man etwa in [**Stoer 1994a**, 2.1.3] oder in [**Schwarz 1986**, 3.4] finden.

**1.1.3. Hermite Interpolation.** Ein etwas verallgemeinertes Interpolationsproblem läßt sich mit einer Verallgemeinerung der dividierten Differenzen behandeln:

> Für $i = 0, \ldots, m$ seien jeweils reelle Zahlenpaare $(x_i, f_i^{(k)})$ mit $k = 0, \ldots, n_i$ gegeben. Das *Hermitesche Interpolationsproblem* besteht dann darin, jenes Polynom $P \in \mathbb{R}^{(n)}[x]$ mit $n + 1 = \sum_{i=0}^{m} n_i$ zu finden, das
>
> $$P^{(k)}(x_i) = f_i^{(k)}$$
>
> für alle $i$ und passenden $k$ erfüllt. Es werden an den Stützstellen also nicht nur Funktionswerte sondern auch (eventuell höhere) Ableitungen vorgegeben.

Wiederum gilt ein Existenz- und Eindeutigkeitssatz

**Theorem 1.1.7.** *Zu beliebigen Zahlen $x_0, \ldots, x_m$, $f_i^{(k)}$ mit $k = 0, \ldots, n_i$ und $i = 0, \ldots, m$ existiert genau ein Polynom $P \in \mathbb{R}^{(n)}[x]$ mit $n + 1 = \sum_{i=0^m} n_i$, das*

$$P^{(k)}(x_i) = f_i^{(k)}$$

*für alle $(i, k)$ erfüllt.*

Beweis. Der Beweis kann analog zum Beweis des Existenzsatzes der normalen Polynominterpolation geführt werden. Es gibt wieder eine Darstellung der Form

$$P(x) = \sum_{i=0}^{m} \sum_{k=0}^{n_i} f_i^{(k)} L_{ik}(x)$$

mit den verallgemeinerten Lagrangepolynomen $L_{ik}$, die rekursiv definiert werden durch

$$\begin{aligned} L_{i,n_i-1}(x) &:= \ell_{i,n_i-1}(x), && i = 0, \ldots, m, \\ L_{ik}(x) &:= \ell_{ik}(x) - \sum_{s=k+1}^{n_i-1} \ell_{ik}^{(s)}(x_i) L_{is}(x), && k = n_i - 2, n_i - 3, \ldots, 0. \end{aligned}$$

In diesen Gleichungen bezeichne $\ell_{jk}(x)$ das Hilfspolynom

$$\ell_{ik}(x) := \frac{(x-x_i)^k}{k!} \prod_{\substack{j=0 \\ j\neq i}}^{m} \left( \frac{x-x_j}{x_i-x_j} \right)^{n_j}, \quad 0 \leq i \leq m, \quad 0 \leq k \leq n_i.$$

Durch Induktion zeigt man leicht, daß für $k = n_i - 1, \ldots, 0$

$$L_{ik}^{(s)}(x_j) = \begin{cases} 1, & i = j \text{ und } k = s, \\ 0, & \text{sonst}, \end{cases}$$

was die Interpolationseigenschaften von $P$ beweist. □

Betrachtet man als Motivation die Beziehung

$$\lim_{x_1 \to x_0} \frac{f(x_1) - f(x_0)}{x_1 - x_0} = f'(x_1),$$

so können wir die verallgemeinerte erste dividierte Differenz

$$f_{01} = f'(x_0)$$

definieren für $x_1 = x_0$. Darum führen wir neue Abszissen $t_j$, $j = 0, \ldots, n$ ein, indem wir die $x_j$ $n_j$–mal wiederholen und der Größe nach ordnen,

$$\underbrace{x_0 = \cdots = x_0}_{n_0} \leq \underbrace{x_1 = \cdots = x_1}_{n_1} \leq \cdots \leq \underbrace{x_m = \cdots = x_m}_{n_m}.$$

Analog verfahren wir mit den $f_j$, was neue Stützordinaten $y_i$ erzeugt. Das verallgemeinerte Newtonsche Interpolationspolynom, das zu den Stützstellen $(t_i, y_i)$ gehört

$$P_{01\ldots n}(x) = y_0 + y_{01}(x - t_0) + \cdots + y_{01\ldots n}(x - t_0)\ldots(x - t_n),$$

löst dann das Hermitesche Interpolationspolynom, falls wir die *verallgemeinerten dividierten Differenzen* $y_{i,i+1,\ldots,i+k}$ gemäß folgender Rekursionsformel bestimmen:

$$y_{i,i+1,\ldots,i+k} = \tfrac{1}{k!} f_j^{(k)},$$

falls $t_i = t_{i+1} = \cdots = t_{i+k} = x_j$ gilt und

$$y_{i,i+1,\ldots,i+k} = \frac{y_{i+1,\ldots,i+k} - y_{i,\ldots,i+k-1}}{x_{i+k} - x_i}$$

sonst.

**Beispiel 1.1.8.** *Seien die folgenden Interpolationsdaten gegeben:*

$$x_0 = 0, \quad f_0^{(0)} = -1, \quad f_0^{(1)} = -2$$
$$x_1 = 1, \quad f_1^{(0)} = 0, \quad f_1^{(1)} = 10, \quad f_1^{(2)} = 40.$$

*Die zugehörigen $t_i$ sind $t_0 = t_1 = 0$, $t_2 = t_3 = t_4 = 1$. Es entsteht das folgende Differenzenschema:*

$$\begin{array}{l|lllll}
t_0 = 0 & -1 = y_0 & & & & \\
 & & -2 = f'(0) = y_{01} & & & \\
t_1 = 0 & -1 = y_1 & & 3 = y_{012} & & \\
 & & 1 = y_{12} & & 6 = y_{0123} & \\
t_2 = 1 & 0 = y_2 & & 9 = y_{123} & & 5 = y_{01234} \\
 & & 10 = f'(1) = y_{23} & & 11 = y_{1234} & \\
t_3 = 1 & 0 = y_3 & & 20 = f''(1) = y_{234} & & \\
 & & 10 = f'(1) = y_{34} & & & \\
t_4 = 1 & 0 = y_4 & & & &
\end{array}$$

*Es führt zum Hermiteschen Interpolationspolynom*

$$P(x) = -1 - 2(x-0) + 3(x-0)^2 + 6(x-0)^2(x-1) + 5(x-0)^2(x-1)^2 = \\ = -1 - 2x + 3x^2 + 6x^2(x-1) + 5x^2(x-1)^2.$$

Für weiterführende Information und Fehlerabschätzungen siehe etwa [**Stoer 1994a**, 2.1.5] oder [**Schwarz 1986**, 3.4].

**1.1.4. Das Restglied der Polynominterpolation.** In diesem Abschnitt wollen wir untersuchen, wie stark ein Interpolationspolynom $P$ von einer Funktion abweichen kann, wenn $P(x_i) = f(x_i)$ gilt für $i = 0, \dots, n$. Auf den ersten Blick erscheint diese Frage unsinnig, kann doch bei geeigneter Wahl von $f$ und $x$ ohne weitere Voraussetzung der Unterschied zwischen $P(x)$ und $f(x)$ beliebig groß gemacht werden. In den Anwendungen ist die Funktion $f$ jedoch meist nicht beliebig und besitzt einige zusätzliche Eigenschaften, die zur Berechnung von Abschätzungen ausgenützt werden können. Das ist besonders dann wichtig, wenn Werte für Funktionen aus Tabellen durch Interpolation geschätzt werden sollen.

**Theorem 1.1.9.** *Wenn $f \in C^{n+1}(\mathbb{R}, \mathbb{R})$ ist, dann existiert für jeden Punkt $x$ ein $\xi$ im kleinsten Intervall, das alle Stützstellen $x_i$ und $x$ enthält mit*

$$f(x) - P_{01\dots n}(x) = \frac{\omega(x) f^{(n+1)}(\xi)}{(n+1)!},$$

*wobei wieder $\omega(x) = \prod_{k=0}^{n}(x - x_k)$.*

Beweis. Die Abschätzung folgt im Prinzip aus dem Mittelwertsatz der Differentialrechnung. □

**Beispiel 1.1.10.** *Sei $f(x) = \sin x$ und seien die 6 Stützstellen $x_i = \frac{i\pi}{10}$ für $i = 0, \dots, 5$ gegeben. Dann gilt die Gleichung*

$$\sin x - P(x) = -\frac{\sin \xi}{720} \prod_{i=0}^{5}(x - x_i)$$

*für ein geeignetes $\xi$, und daher folgt die Abschätzung*

$$|\sin x - P(x)| \le \tfrac{|\omega(x)|}{720}.$$

Außerhalb des durch die Interpolationspunkte überdeckten Intervalls wächst $|\omega(x)|$ sehr stark an. Daher sollte das Interpolationspolynom $P$ auch nicht zur Approximation von $f$ an einer Stelle $\bar{x}$ außerhalb des kleinsten Intervalls, das alle $x_i$ enthält, also zur *Extrapolation* verwendet werden.

Man würde jetzt annehmen, daß durch Hinzufügen weiterer Stützpunkte die Approximationseigenschaften des Interpolationspolynomes verbessert werden. Sei ein fixes Intervall $[a] := [\underline{a}, \overline{a}]$ gegeben. Zu jeder Funktion $f$ und jeder Intervalleinteilung $\Delta = \{\underline{a} = x_0 < x_1 < \dots < x_n = \overline{a}\}$ gibt es ein eindeutiges Interpolationspolynom $P_\Delta^f$ höchstens $n$–ten Grades mit $P_\Delta^f(x_i) = f(x_i)$. Eine Folge $\Delta_m$ von Intervalleinteilungen induziert so eine Folge $P_{\Delta_m}^f$, und man sollte annehmen, daß $P_{\Delta_m}^f$ gegen $f$ konvergiert, wenn die Feinheit der Unterteilung

$$\|\Delta_m\| := \max_i \left| x_{i+1}^{(m)} - x_i^{(m)} \right|$$

für $m \to \infty$ gegen 0 konvergiert. Leider ist diese Vermutung im allgemeinen falsch. Es gilt nämlich folgender Satz

**Theorem 1.1.11** (Faber)**.** *Zu jeder Folge von Intervalleinteilungen $\Delta_m$ von $[a]$ gibt es eine stetige Funktion $f$ auf $[a]$, sodaß die Polynome $P_{\Delta_m}(x)$ für $m \to \infty$ auf $[a]$ nicht gleichmäßig gegen $f(x)$ konvergieren.*

Nur für ganze (auf ganz $\mathbb{C}$ analytische) Funktionen $f$ strebt jede Folge von Interpolationspolynomen (unabhängig von der speziellen Wahl der Zerlegung) gleichmäßig auf $[a]$ gegen $f$. Für alle $f$ gibt es zumindest spezielle Folgen $\Delta_m$ sodaß $P_{\Delta_m}$ gegen $f$ konvergiert.

Äquidistante Intervalleinteilungen $x_i^m = a+i(b-a)/m$ für $i = 0, \dots, m$ garantieren nicht einmal die punktweise Konvergenz für nicht-ganze Funktionen, z.B. nicht für

$$f(x) = \frac{1}{1+x^2}, \quad [a] = [-5, 5],$$
$$f(x) = \sqrt{x}, \quad [a] = [0, 1].$$

## 2. Rationale Interpolation

Speziell in der Nähe von Polstellen versagt die Polynominterpolation völlig. Die Starrheit von Polynomen ist dann zu groß, um die gesuchte Funktion gut genug zu approximieren. Die einfachste Verallgemeinerung der Polynominterpolation, die sich anbietet, ist dann die Interpolation mit einer rationalen Funktion, also einem „Bruch gebildet aus zwei Polynomen".

Nachdem rationale Funktionen selbst Polstellen haben können, können sie auch Stützpunkte in der Nähe von (vermuteten) Polen gut interpolieren. Wir werden allerdings sehen, daß auch rationale Funktionen als einfachste Verallgemeinerung von Polynomen in gewisser Weise „starr" sind; sie können im Gegensatz zu Polynomen nicht alle Interpolationsprobleme lösen. Probleme treten dann auf, wenn einige Punkte bereits auf einer rationalen Funktion niedrigeren Grades liegen, sie also in spezieller Lage sind. Rationale Funktionen „weigern" sich also, Punkte zu treffen, wenn eine Teilmenge bereits die Form einer bestimmten rationalen Funktion festlegt.

**2.1. Grundlagen.** Beginnen wir wieder mit Stützstellen $(x_i, f_i)$ für $i = 0, \dots, n$, und sei wieder $x_j \neq x_k$ für $j \neq k$. Wir versuchen, die Stützstellen mit einer Funktion der Gestalt

$$\Phi^{r,s}(x) = \frac{P^{r,s}(x)}{Q^{r,s}(x)}$$

zu interpolieren, wobei $P^{r,s}(x)$ ein Polynom höchstens vom Grad $r$ und $Q^{r,s}(x)$ ein Polynom höchstens vom Grad $s$ seien:

$$P^{r,s}(x) = \sum_{i=0}^{r} a_i x^i$$
$$Q^{r,s}(x) = \sum_{j=0}^{s} b_j x^j.$$

Die $r+s+2$ Parameter der Funktion $\Phi^{r,s}(x)$ sind jedoch nicht unabhängig voneinander. Multipliziert man den Zähler und den Nenner mit der gleichen Zahl $\lambda$, so verändert das den Bruch nicht; alle Parameter sind also nur bis auf einen gemeinsamen Faktor bestimmt. In Wirklichkeit sind also nur $r+s+1$ Parameter voneinander unabhängig; man kann also höchstens $r+s+1$ Stützpunkte interpolieren. Daher wählen wir $n = r+s$.

Betrachten wir die Interpolationsgleichungen

$$\Phi^{r,s}(x_i) = \frac{P^{r,s}(x_i)}{Q^{r,s}(x_i)} = f_i, \tag{34}$$

so sehen wir, daß wir sie in ein lineares Gleichungssystem in den Koeffizienten $a_j$ und $b_j$ verwandeln können:

$$a_0 + a_1 x_i + \cdots + a_r x_i^r - f_i(b_0 + b_1 x_i + \cdots + b_s x_i^s) = 0,$$

für $i = 0, \dots, r+s$, ein homogenes unterbestimmtes lineares Gleichungssystem mit $r+s+1$ Gleichungen in $r+s+2$ Variablen. Im Rahmen der folgenden Untersuchungen bezeichnen wir dieses System mit $S^{r,s}$.

Nachdem wir nun das Interpolationsproblem in ein lineares Gleichungssystem umgewandelt haben, erscheint es bereits gelöst, etwa mit den Methoden aus den Kapiteln 3 und 4; doch das nächste Beispiel zeigt, warum die Untersuchungen hier erst beginnen.

**Beispiel 2.1.1.** *Sei $r = s = 1$ und versuchen wir die Interpolationsaufgabe für die Punkte $(0,1)$, $(1,2)$ und $(2,2)$ zu lösen. Das zugehörige Gleichungssystem $S^{1,1}$ ist dann*

$$\begin{array}{lll} a_0 & -1 \cdot b_0 & = 0, \\ a_0 + a_1 & -2(b_0 + b_1) & = 0, \\ a_0 + 2a_1 & -2(b_0 + 2b_1) & = 0. \end{array}$$

*Eine der unendlich vielen (für die Lösung des Problemes gleichwertigen) Lösungen ist gegeben durch*

$$a_0 = 0, \quad b_0 = 0, \quad a_1 = 2, \quad b_1 = 1;$$

*die rationale Funktion, die dadurch bestimmt wird, ist*

$$\Phi^{1,1}(x) = \frac{2x}{x} \equiv 2,$$

*wenn man den Definitionsbereich von $\mathbb{R}\backslash\{0\}$ auf ganz $\mathbb{R}$ erweitert. Unglücklicherweise verfehlt die Funktion $\Phi^{1,1}(x)$ die Stützstelle $(x_0, f_0) = (0,1)$. Nachdem jede Lösung des Interpolationsproblems für $r = s = 1$ aber jedenfalls das Gleichungssystem $S^{1,1}$ lösen muß, und $S^{1,1}$ keine anderen Lösungen als die angegebene hat, kann das von uns untersuchte Interpolationsproblem nicht gelöst werden; $(x_0, f_0)$ ist ein* unerreichbarer Punkt.

Das vorangegangene Beispiel zeigt, daß es rationale Interpolationsprobleme gibt, die unlösbar sind. Wir wollen als nächstes versuchen herauszufinden, welche rationale Interpolationsprobleme lösbar sind und welche nicht. Dazu schadet es aber nicht, die mathematischen Begriffe zu klären.

**Definition 2.1.2.** *Ein* rationaler Ausdruck *ist eine Äquivalenzklasse von Paaren von Polynomen $(P, Q)$ mit $Q \not\equiv 0$ bezüglich der Äquivalenzrelation*

$$(P_1, Q_1) \sim (P_2, Q_2) : \Longleftrightarrow \; P_1(x)Q_2(x) \equiv P_2(x)Q_1(x).$$

*Ein Vergleich mit den rationalen Zahlen $\mathbb{Q}$, die mathematisch ganz ähnlich konstruiert werden, und den ganzen Zahlen $\mathbb{Z}$, die algebraisch den Polynomen entsprechen, zeigt, daß zwei rationale Ausdrücke genau dann äquivalent sind, wenn sie durch „Kürzen“ oder „Erweitern“ aus einander hervorgehen.*

*Ein Element der Äquivalenzklasse hat minimalen Zähler– und Nennergrad (dabei unterdrücken wir die Möglichkeit, Zähler und Nenner mit Konstanten zu erweitern); daher sind Zähler und Nenner teilerfremd. Diesen (fast) eindeutig bestimmten Repräsentanten der Äquivalenzklasse $[\Phi]$ werden wir mit $\bar{\Phi}$ bezeichnen.*

Nachdem wir jetzt wissen, worüber wir sprechen, beginnen wir die Untersuchungen. Dabei starten wir bei der Analyse des Systemes $S^{r,s}$ und dessen Lösungen.

**Theorem 2.1.3.** *Sei $S^{r,s}$ wie zuvor gegeben. Dann gelten*

(1) *$S^{r,s}$ hat immer nichttriviale Lösungen, und jede solche gehört zu einem rationalen Ausdruck*

$$\Phi^{r,s}(x) = \frac{P^{r,s}(x)}{Q^{r,s}(x)}.$$

(2) *Wenn $\Phi_1$ und $\Phi_2$ zu Lösungen von $S^{r,s}$ gehören, so gilt $\Phi_1 \sim \Phi_2$; die rationalen Ausdrücke sind also äquivalent.*

(3) *Besitzt $S^{r,s}$ eine teilerfremde Lösung, so ist auch das zu $S^{r,s}$ gehörende Interpolationsproblem lösbar.*

BEWEIS. (1) Aus den Sätzen der linearen Algebra folgt sofort, daß das homogene unterbestimmte lineare Gleichungssystem $S^{r,s}$ unendlich viele Lösungen besitzt. Es bleibt zu zeigen, daß jede nichttriviale Lösung

$$(a_0, \dots, a_r, b_0, \dots, b_s) \neq (0, \dots, 0)$$

einen rationalen Ausdruck bestimmt. Angenommen das ist nicht der Fall. Dann ist das zugehörige $Q^{r,s}$ das Nullpolynom, also sind alle $b_k = 0$. Dann folgt aus den Gleichungen von $S^{r,s}$ für alle $i = 0, \dots, r+s$

$$P(x_i) = 0,$$

und daher hat das Polynom $P$, welches höchstens $r$–ten Grad besitzt, $r+s+1 \geq r+1$ Nullstellen. Aus diesem Grund ist auch $P$ das Nullpolynom, was der Nichttrivialität der Lösung $(a_0, \dots, a_r, b_0, \dots, b_s)$ widerspricht.

(2) Seien $\Phi_1(x) = \frac{P_1(x)}{Q_1(x)}$ und $\Phi_2(x) = \frac{P_2(x)}{Q_2(x)}$ zwei Lösungen des Gleichungssystemes $S^{r,s}$. Betrachten wir das Polynom höchstens $(r+s)$–ten Grades

$$P(x) = P_1(x)Q_2(x) - P_2(x)Q_1(x).$$

Setzen wir die Stützabszissen $x_j$ in dieses Polynom ein, und verwenden wir die Interpolationsbedingungen (34), so erhalten wir

$$P(x_j) = P_1(x_j)Q_2(x_j) - P_2(x_j)Q_1(x_j) = f_1Q_1(x_j)Q_2(x_j) - f_1Q_2(x_j)Q_1(x_j) = 0.$$

Das Polynom $P$ hat also $r+s+1$ Nullstellen, ist also das Nullpolynom. Daher gilt $\Phi_1(x) \sim \Phi_2(x)$.

(3) Für $i = 0, \dots, r+s$ gibt es zwei mögliche Fälle:

(a) $Q^{r,s}(x_i) \neq 0$: In diesem Fall folgt aus der $i$–ten Gleichung von $S^{r,s}$ die $i$–te Interpolationsbedingung

$$\Phi^{r,s}(x_i) = \frac{P^{r,s}(x_i)}{Q^{r,s}(x_i)} = f_i.$$

(b) $Q^{r,s}(x_i) = 0$. Hat das Nennerpolynom an einer Stützstelle eine Nullstelle, so folgt aus der $i$–ten Gleichung von $S^{r,s}$ die Beziehung $P^{r,s}(x_i) = 0$, also hat auch das Zählerpolynom an $x_i$ eine Nullstelle. Man kann daher aus beiden Polynomen $P^{r,s}(x)$ und $Q^{r,s}(x)$ jeweils einen Faktor $(x - x_i)$ herausheben:

$$P^{r,s}(x) = (x - x_i)p(x)$$
$$Q^{r,s}(x) = (x - x_i)q(x),$$

und daher sind $P^{r,s}(x)$ und $Q^{r,s}(x)$ nicht teilerfremd.

□

Diese Resultate beinhalten beinahe schon die gesamten Fakten, die zum Verständnis der rationalen Interpolation nötig sind. Der nächste Satz ist eine Zusammenfassung und Reformulierung dieser Tatsachen.

**Theorem 2.1.4.** (1) *Sei $\Phi^{r,s}$ eine Lösung von $S^{r,s}$, und sei $\bar{\Phi}^{r,s}$ der äquivalente teilerfremde rationale Ausdruck. Die Interpolationsaufgabe ist genau dann lösbar, wenn auch die Parameter von $\bar{\Phi}^{r,s}$ das System $S^{r,s}$ erfüllen.*

(2) *Falls $S^{r,s}$ maximalen Rang hat, so ist die Interpolationsaufgabe genau dann lösbar, wenn die Lösung $\Phi^{r,s}$ von $S^{r,s}$ teilerfremd ist.*

BEWEIS. Der Beweis ist eine Umformulierung der Resultate von Theorem 2.1.3. □

Dieses Theorem ermöglicht es, die Lösbarkeit der Interpolationsaufgabe zu überprüfen. Man bestimmt irgendeine Lösung des Gleichungssystemes $S^{r,s}$ und berechnet (z.B. mit dem Euklidischen Algorithmus) den größten gemeinsamen Teiler von $P^{r,s}$ und $Q^{r,s}$. Hat man diesen gefunden, so kann man sofort durch Abdividieren den rationalen Ausdruck $\bar{\Phi}^{r,s}$ finden. Genau dann, wenn dessen Koeffizienten das System $S^{r,s}$ erfüllen, ist die Interpolationsaufgabe lösbar.

Aber auch über die Fälle, in denen die Interpolationsaufgabe nicht lösbar ist, geben die Untersuchungen Auskunft.

**Theorem 2.1.5.** *Die* **erreichbaren** *Stützpunkte eines* **unlösbaren** *Interpolationsproblemes befinden sich in* spezieller Lage*, das heißt es existiert eine rationale Funktion $\Phi^{\rho,\sigma}(x)$ mit $\rho+\sigma<m$, die die $m+1$ erreichbaren Punkte $(x_{i_0},\dots,x_{i_m})$ interpoliert.*

Beweis. Nehmen wir an, die unerreichbaren Stützpunkte hätten die Indizes $(j_1,\dots,j_\alpha)$. Dann ist nach dem Beweis von Theorem 2.1.3 die Lösung $\Phi^{r,s}(x)$ durch $(x-x_{j_1})\dots(x-x_{j_\alpha})$ kürzbar. Aus dieser Umformung entsteht ein rationaler Ausdruck $\Psi^{\rho,\sigma}(x)$ mit $\rho=r-\alpha$ und $\sigma=s-\alpha$, der das Interpolationsproblem für die $m=r+s+1-\alpha$ erreichbaren Punkte löst. Wegen

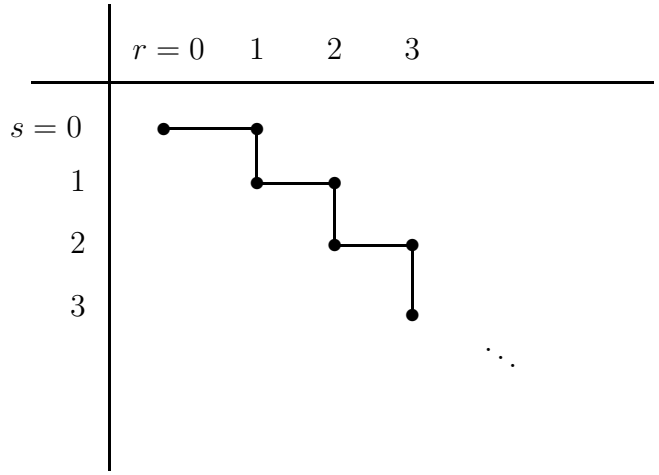
$$\rho+\sigma+1=r+s+1-2\alpha<r+s+1-\alpha$$

befinden sich die erreichbaren Punkte in spezieller Lage. □

Wir erkennen also, daß rationale Interpolationsprobleme nur dann lösbar sind, wenn keine Situationen entstehen, in denen spezielle Lagen von Punkten auftreten. Allerdings zerstören kleine Störungen in den Stützstellen die spezielle Lage und verwandeln ein unlösbares Problem in ein lösbares Problem. Nachdem Punkte in spezieller Lage „viel seltener“ anzutreffen sind als Punkte in allgemeiner Lage, nehmen wir im folgenden an, daß der *generische Fall* vorliegt, die Punkte in allgemeiner Lage sind und das Interpolationsproblem daher lösbar ist.

**2.2. Inverse und Reziproke Differenzen, Der Thielesche Kettenbruch.** Im Spezialfall $s=0$ wird die rationale Interpolation zur Polynominterpolation, und aus diesem Spezialfall können wir ersehen, daß die numerische Lösung des Systemes $S^{r,s}$ einen zu hohen Rechenaufwand benötigt. Darüber hinaus ist die Koeffizientenmatrix von $S^{r,s}$ eine Vandermonde–artige Matrix, die meist schlecht konditioniert ist.

Um Zähler– und Nennergrad ausgewogen zu halten, suchen wir in der Folge nach rationalen interpolierenden Funktionen der Gestalt $\Phi^{n,n}(x)$ oder $\Phi^{n+1,n}(x)$ je nach dem, ob die Anzahl der Stützstellen ungerade oder gerade ist. Dabei gehen wir analog zur Newtonschen Darstellung des Interpolationspolynoms vor, indem wir sukzessive vorgehen, schrittweise Stützstellen dazunehmen und $r$ und $s$ gemäß der folgenden Tabelle wählen:



Untersuchen wir den Fall ungerader Stützstellenzahl; im geraden Fall funktioniert alles analog. Wir versuchen also mit Hilfe einer Funktion $\Phi^{n,n}(x)$ die Stützstellen $(x_i, f_i)$, $i = 0, \ldots, 2n$ zu interpolieren. Nehmen wir an, wir hätten schon eine Lösung gefunden. Dann gilt

$$\begin{aligned}\Phi^{n,n}(x) = \frac{P^n(x)}{Q^n(x)} &= f_0 + \frac{P^n(x)}{Q^n(x)} - \frac{P^n(x_0)}{Q^n(x_0)} = \\ &= f_0 + (x - x_0)\frac{P^{n-1}(x)}{Q^n(x)} = f_0 + \frac{x - x_0}{Q^n(x)/P^{n-1}(x)},\end{aligned}$$

wobei wir die Interpolationsbedingung für $(x_0, f_0)$ verwendet haben. Aus den übrigen Bedingungen folgt dann

$$\frac{Q^n(x_i)}{P^{n-1}(x_i)} = \frac{x_i - x_0}{f_i - f_0} =: \varphi(x_0, x_i).$$

Verwenden wir nun die Interpolationsbedingung für $x_1$, und führen wir eine ähnliche Umformung durch wie zuvor, so erhalten wir

$$\begin{aligned}\frac{Q^n(x)}{P^{n-1}(x)} &= \varphi(x_0, x_1) + \frac{Q^n(x)}{P^{n-1}(x)} - \frac{Q^n(x_1)}{P^{n-1}(x_1)} = \\ &= \varphi(x_0, x_1) + (x - x_1)\frac{Q^{n-1}(x)}{P^{n-1}(x)} = \\ &= \varphi(x_0, x_1) + \frac{x - x_1}{P^{n-1}(x)/Q^{n-1}(x)},\end{aligned}$$

und daher folgt aus den übrigen Interpolationsbedingungen

$$\frac{P^{n-1}(x_i)}{Q^{n-1}(x_i)} = \frac{x_i - x_1}{\varphi(x_0, x_i) - \varphi(x_0, x_1)} =: \varphi(x_0, x_1, x_i).$$

Man erkennt ein Schema, in dem die $\varphi$'s eine gewisse Rolle spielen. Das Bildungsgesetz für die $\varphi$'s kann man auch aus diesen Umformungen ablesen:

$$\begin{aligned}\varphi(x_i) &= f_i \\ \varphi(x_0, \ldots, x_l, x_m, x_n) &= \frac{x_m - x_n}{\varphi(x_0, \ldots, x_l, x_m) - \varphi(x_0, \ldots, x_l, x_n)}.\end{aligned}$$

Nachdem dieses Bildungsgesetz analog zu den Definitionen für die dividierten Differenzen aussieht, nur invertiert, nennt man die $\varphi(x_0, \ldots, x_k)$ *inverse Differenzen*. Manche dieser Differenzen können dabei $\infty$ werden. Treten Ausdrücke auf, die unbestimmt sind, so ist das Interpolationsproblem unlösbar, das haben wir aber oben ausgeschlossen. Mit Hilfe dieser inversen Differenzen kann man die Lösung des Problems gemäß den Umformungen von oben als Kettenbruch darstellen:

$$\begin{aligned}\Phi^{n,n}(x) = \frac{P^n(x)}{Q^n(x)} &= f_0 + \frac{x - x_0}{Q^n(x)/P^{n-1}(x)} = \\ &= f_0 + \frac{x - x_0}{\varphi(x_0, x_1) + \dfrac{x - x_1}{P^{n-1}(x)/Q^{n-1}(x)}} = \cdots = \\ &= f_0 + \cfrac{x - x_0}{\varphi(x_0, x_1) + \cfrac{x - x_1}{\varphi(x_0, x_1, x_2) + \cfrac{}{\ddots \cfrac{}{\cfrac{x - x_{2n-1}}{\varphi(x_0, \ldots, x_{2n})}}}}}.\end{aligned}$$

Der einzige Wermutstropfen in der Ableitung, die die gesuchte rationale Funktion in Form eines Kettenbruches darstellt, der ebenso leicht auszuwerten ist wie ein Interpolationspolynom in Newtonscher Darstellung, ist, daß die inversen Differenzen nicht symmetrisch in ihren Argumenten sind ganz im Gegensatz zu den dividierten Differenzen. Durch eine kleine Änderung in den Definitionen läßt sich dieser Umstand allerdings korrigieren.

**Definition 2.2.1.** *Seien die* reziproken Differenzen $\rho(x_i, \dots, x_{i+k})$ *durch die folgende rekursive Beziehung definiert*

$$\begin{aligned}
\rho(x_i) &:= f_i, \\
\rho(x_i, x_{i+1}) &:= \frac{x_i - x_{i+1}}{f_i - f_{i+1}}, \\
\rho(x_i, x_{i+1}, \dots, x_{i+k}) &:= \frac{x_i - x_{i+k}}{\rho(x_i, \dots, x_{i+k-1}) - \rho(x_{i+1}, \dots, x_{i+k})} - \rho(x_{i+1}, \dots, x_{i+k-1}).
\end{aligned}$$

*Die reziproken Differenzen sind symmetrisch in ihren Argumenten, was man durch Induktion überprüfen kann.*

**Proposition 2.2.2.** *Zwischen den inversen und den reziproken Differenzen gilt die Beziehung*

$$\varphi(x_0, \dots, x_p) = \rho(x_0, \dots, x_p) - \rho(x_0, \dots, x_{p-2})$$

*für $p \geq 2$. Für $p = 0, 1$ stimmen die inversen Differenzen mit den reziproken Differenzen überein.*

Beweis. Vollständige Induktion. Für $p = 0, 1$ gilt die Übereinstimmung nach Definition. Sonst haben wir die Gleichung

$$\begin{aligned}
\varphi(x_0, \dots, x_{p+1}) &= \frac{x_p - x_{p+1}}{\varphi(x_0, \dots, x_p) - \varphi(x_0, \dots, x_{p-1}, x_{p+1})} = \\
&= \frac{x_p - x_{p+1}}{\rho(x_0, \dots, x_p) - \rho(x_0, \dots, x_{p-2}) - (\rho(x_0, \dots, x_{p-1}, x_{p+1}) - \rho(x_0, \dots, x_{p-2}))} = \\
&= \frac{x_p - x_{p+1}}{\rho(x_0, \dots, x_p) - \rho(x_0, \dots, x_{p-1}, x_{p+1})} = \\
&= \rho(x_0, \dots, x_{p+1}) - \rho(x_0, \dots, x_{p-1})
\end{aligned}$$

wegen der Symmetrie von $\rho(x_0, \dots, x_{p+1})$. □

Am einfachsten berechnet man die reziproken Differenzen aus einer Tabelle ähnlich der, mit deren Hilfe man die dividierten Differenzen berechnet.

$$\begin{array}{c|c|ccc}
x_0 & f_0 = \rho(x_0) & & & \\
 & & \underline{\rho(x_0, x_1)} & & \\
x_1 & f_1 = \rho(x_1) & & \underline{\rho(x_0, x_1, x_2)} & \\
 & & \rho(x_1, x_2) & & \underline{\rho(x_0, x_1, x_2, x_3)} \\
x_2 & f_2 = \rho(x_2) & & \rho(x_1, x_2, x_3) & \vdots \\
 & & \rho(x_2, x_3) & \vdots & \\
x_3 & f_3 = \rho(x_3) & \vdots & & \\
\vdots & \vdots & & &
\end{array}$$

Aus den unterstrichenen reziproken Differenzen kann man analog zu vorhin einen Kettenbruch bilden, indem man Proposition 2.2.2 verwendet. Der entstehende Kettenbruch heißt

auch *Thielescher Kettenbruch*:

$$\Phi^{n,n}(x) = \rho(x_0) + \cfrac{x-x_0}{\rho(x_0,x_1)+\cfrac{x-x_1}{\rho(x_0,x_1,x_2)-\rho(x_0)+\cfrac{x-x_2}{\rho(x_0,x_1,x_2,x_3)-\rho(x_0,x_1)+\ddots \cfrac{x-x_{2n-1}}{\rho(x_0,\ldots,x_{2n})-\rho(x_0,\ldots,x_{2n-2})}}}}.$$

Der Fall $\Phi^{n+1,n}$ ist analog.

Die Anwendung dieser Interpolationsmethode wollen wir uns anhand des folgenden Beispieles ansehen.

**Beispiel 2.2.3.** *Wir wollen die Punkte* $(0,0)$, $(1,-1)$, $(2,-\frac{2}{3})$ *und* $(3,9)$ *mit einer rationalen Funktion* $\Phi^{2,1}$ *interpolieren.*

$$\begin{array}{c|c|ccc} 0 & 0=\rho(x_0) & \underline{-1} & & \\ 1 & -1=\rho(x_1) & & \underline{-\frac{1}{2}} & \\ 2 & -\frac{2}{3}=\rho(x_2) & 3 & & \underline{-\frac{1}{2}} \\ 3 & 9=\rho(x_3) & & \frac{59}{6} & \\ & & -\frac{29}{3} & & \end{array}$$

*Aus den unterstrichenen Größen gewinnt man den Thieleschen Kettenbruch*

$$\Phi^{2,1}(x) = 0 + \cfrac{x}{-1+\cfrac{x-1}{-\frac{1}{2}+\cfrac{x-2}{\frac{1}{2}}}} = \frac{4x^2-9x}{-2x+7},$$

*wobei man auf das Ausmultiplizieren in der Anwendung natürlich verzichten würde, da sich der Kettenbruch viel besser auswerten läßt.*

Zusammenfassend kann man folgende beiden Algorithmen zur rationalen Interpolation angeben, die sich von den Algorithmen zur Newtonschen Polynominterpolation nur unwesentlich unterscheiden.

Der Einfachheit halber berechnen wir die Koeffizienten des Kettenbruchs über die inversen Differenzen statt über die reziproken Differenzen.

**Algorithmus 2.2.4.** *Thielescher Kettenbruch (inverse Differenzen)*

**for** $k = 0$ **to** $n$ **do**
  $h = f_i$
  **for** $i = 1$ **to** $i$ **do**
    $h = (x_k - x_{i-1})/(h - a[k-1])$
  **done**
  $a[k] = h$
**done**

Nach Beendigung dieses Algorithmus stehen im Vektor $a$ die inversen Differenzen für den gesuchten Thieleschen Kettenbruch. Der Aufwand zur Berechnung beträgt $O(n^2)$ elementare Operationen.

Die Auswertung des so gewonnenen Kettenbruches an einer beliebigen Stelle erfolgt praktischer Weise gemäß dem folgenden Algorithmus, der die Analogie zur Auswertung des Interpolationspolynomes in der Newtonschen Darstellung, zum verallgemeinerten Horner Schema (Algorithmus 1.1.6) aufzeigt.

**Algorithmus 2.2.5.** *Auswertung des Thieleschen Kettenbruches*

$p = a[n]$
**for** $k = n-1$ **to** $0$ **step** $-1$ **do**
  $p = a[k] + (x - x_k)/p$
**done**

**2.3. Ein Neville–artiger Algorithmus.** Ganz ähnlich wie bei der Polynominterpolation ist auch bei der rationalen Interpolation der Aufwand, den Thieleschen Kettenbruch zu berechnen zu hoch, wenn man ihn nur an einer Stelle auswerten möchte. Für die Polynominterpolation schafft da der Nevillesche Algorithmus Abhilfe. Eine Verallgemeinerung dieses Algorithmus dient nun zur Berechnung des Wertes des Thieleschen Kettenbruches an einer bestimmten Stelle.

Wie beim Nevilleschen Algorithmus gehen wir rekursiv vor, und zu diesem Zweck führen wir die folgende Notation ein:

$$\Phi_t^{r,s}(x) := \frac{P_t^{r,s}(x)}{Q_t^{r,s}(x)}$$

sei die Lösung für das Interpolationsproblem

$$\Phi_t^{r,s}(x_i) = f_i, \quad \text{für } i = t, t+1, \ldots, t+r+s.$$

Den Koeffizienten der höchsten Potenz des Polynomes $P_t^{r,s}(x)$ bezeichnen wir fortan mit $p_t^{r,s}$ und den höchsten Koeffizienten von $Q_t^{r,s}$ mit $q_t^{r,s}$.

**Proposition 2.3.1.** *Mit obiger Notation gelten die folgenden Rekursionsformeln:*

**Startwerte:**

$$P_t^{0,0}(x) = f_t, \quad Q_t^{0,0}(x) = 1$$

**Übergang** $(r-1,s) \to (r,s)$**:**

$$P_t^{r,s}(x) = (x - x_t)q_t^{r-1,s}P_{t+1}^{r-1,s}(x) - (x - x_{r+s+t})q_{t+1}^{r-1,s}P_t^{r-1,s}(x)$$
$$Q_t^{r,s}(x) = (x - x_t)q_t^{r-1,s}Q_{t+1}^{r-1,s}(x) - (x - x_{r+s+t})q_{t+1}^{r-1,s}Q_{t+1}^{r-1,s}(x)$$

**Übergang** $(r,s-1) \to (r,s)$**:**

$$P_t^{r,s}(x) = (x - x_t)p_t^{r,s-1}P_{t+1}^{r,s-1}(x) - (x - x_{r+s+t})p_{t+1}^{r,s-1}P_t^{r,s-1}(x)$$
$$Q_t^{r,s}(x) = (x - x_t)p_t^{r,s-1}Q_{t+1}^{r,s-1}(x) - (x - x_{r+s+t})p_{t+1}^{r,s-1}Q_t^{r,s-1}(x)$$

Beweis. Die Startwerte sind so gewählt, daß sie offensichtlich die Interpolationsbedingungen erfüllen. Im folgenden zeigen wir von den beiden zueinander analogen zweiten Schritten nur den ersten.

Wir nehmen an, daß die Ausdrücke $\Phi_t^{r-1,s}$ und $\Phi_{t+1}^{r-1,s}$ bereits so konstruiert sind, daß sie die zugehörigen Interpolationsaufgaben lösen. Es gilt daher

$$P_\lambda^{r-1,s}(x_i) - f_i Q_\lambda^{r-1,s}(x_i) = 0, \quad \text{mit } i = \lambda, \lambda+1, \ldots, \lambda+r+s-1,$$

für $\lambda = t$ oder $\lambda = t+1$. Definieren wir $P_t^{r,s}$ und $Q_t^{r,s}$ durch die Rekursionsformeln, so ist der Grad von $P_t^{r,s}$ höchstens gleich $r$ und der Grad von $Q_t^{r,s}$ ist höchstens $s$. Darüber hinaus gilt

$$\begin{aligned} P_t^{r,s}(x_i) - f_i Q_t^{r,s}(x_i) = {} & (x_i - x_t)q_t^{r-1,s}(P_{t+1}^{r-1,s}(x_i) - f_i Q_{t+1}^{r-1,s}(x_i)) - \\ & - (x_i - x_{r+s+t})q_{t+1}^{r-1,s}(P_t^{r-1,s}(x_i) - f_i Q_t^{r-1,s}(x_i)) = 0, \end{aligned}$$

wegen obiger Interpolationsbeziehungen. Wir haben also

$$\Phi_t^{r,s}(x_i) = \frac{P_t^{r,s}(x_i)}{Q_t^{r,s}(x_i)} = f_i,$$

weil wir vorausgesetzt haben, daß keine unerreichbaren Punkte auftreten. □

Leider treten in diesen Rekursionsformeln noch die Koeffizienten $p_t^{r,s}$ und $q_t^{r,s}$ auf, deren Berechnung wir unter anderem gerade vermeiden wollten. Glücklicherweise erlauben es die Rekursionsformeln durch Zusammenfassen ebendiese Koeffizienten zu eliminieren.

Daraus kann man dann die folgenden neuen Rekursionsformeln für die $\Phi_t^{r,s}$ herleiten:

**Theorem 2.3.2.** *Für die rationalen Ausdrücke $\Phi_t^{r,s}$ gelten die folgenden Rekursionsformeln:*

**Startwerte:**

$$\Phi_t^{0,0}(x) := f_t, \quad \Phi_t^{r,-1}(x) := \infty, \quad \Phi_t^{-1,s}(x) := 0$$

**Übergang $(r-1,s) \to (r,s)$:**

$$\Phi_t^{r,s}(x) = \Phi_{t+1}^{r-1,s}(x) + \frac{\Phi_{t+1}^{r-1,s}(x) - \Phi_t^{r-1,s}(x)}{\dfrac{x-x_t}{x-x_{r+s+t}}\left(1 - \dfrac{\Phi_{t+1}^{r-1,s}(x) - \Phi_t^{r-1,s}(x)}{\Phi_{t+1}^{r-1,s}(x) - \Phi_{t+1}^{r-1,s-1}(x)}\right) - 1}$$

**Übergang $(r,s-1) \to (r,s)$:**

$$\Phi_t^{r,s}(x) = \Phi_{t+1}^{r,s-1}(x) + \frac{\Phi_{t+1}^{r,s-1}(x) - \Phi_t^{r,s-1}(x)}{\dfrac{x-x_t}{x-x_{r+s+t}}\left(1 - \dfrac{\Phi_{t+1}^{r,s-1}(x) - \Phi_t^{r,s-1}(x)}{\Phi_{t+1}^{r,s-1}(x) - \Phi_{t+1}^{r-1,s-1}(x)}\right) - 1}$$

Beweis. Auch hier beweisen wir nur den Übergang $(r-1,s) \to (r,s)$. Die andere Formel wird wieder analog hergeleitet.

**Schritt 1:** Es gelten die Beziehungen

$$\Phi_t^{r-1,s}(x) - \Phi_{t+1}^{r-1,s-1}(x) = -p_{t+1}^{r-1,s-1} q_t^{r-1,s} \frac{(x-x_{t+1})\dots(x-x_{r+s+t-1})}{Q_t^{r-1,s}(x) Q_{t+1}^{r-1,s-1}(x)}.$$

$$\Phi_{t+1}^{r-1,s}(x) - \Phi_{t+1}^{r-1,s-1}(x) = -p_{t+1}^{r-1,s-1} q_{t+1}^{r-1,s} \frac{(x-x_{t+1})\dots(x-x_{r+s+t-1})}{Q_{t+1}^{r-1,s}(x) Q_{t+1}^{r-1,s-1}(x)}.$$

Wegen

$$\Phi_t^{r-1,s}(x) - \Phi_{t+1}^{r-1,s-1}(x) = \frac{P_t^{r-1,s}(x) Q_{t+1}^{r-1,s-1}(x) - P_{t+1}^{r-1,s-1}(x) Q_t^{r-1,s}(x)}{Q_{t+1}^{r-1,s}(x) Q_{t+1}^{r-1,s-1}(x)}$$

wissen wir, daß das Zählerpolynom höchstens $(r+s-1)$–ten Grades ist, und daß sein höchster Koeffizient laut Konstruktion gleich $-p_{t+1}^{r-1,s-1} q_t^{r-1,s}$ ist. Ferner verschwindet das Zählerpolynom nach Definition von $\Phi_t^{r-1,s}$ und $\Phi_{t+1}^{r-1,s-1}$ an den $r+s-1$ Punkten $x_i$ für $i = t+1, t+2, \dots, t+r+s-1$. Daher hat es die Form

$$-p_{t+1}^{r-1,s-1} q_t^{r-1,s} (x-x_{t+1})\dots(x-x_{r+s+t-1}),$$

und die erste Gleichung ist bewiesen. Die Herleitung der zweiten Beziehung ist analog.

**Schritt 2:** Aus Proposition 2.3.1 wissen wir

$$\Phi_t^{r,s}(x) = \frac{(x-x_t) q_t^{r-1,s} P_{t+1}^{r-1,s}(x) - (x-x_{r+s+t}) q_{t+1}^{r-1,s} P_t^{r-1,s}(x)}{(x-x_t) q_t^{r-1,s} Q_{t+1}^{r-1,s}(x) - (x-x_{r+s+t}) q_{t+1}^{r-1,s} Q_t^{r-1,s}(x)}.$$

Indem wir diesen Bruch mit

$$\frac{-p_{t+1}^{r-1,s-1}(x-x_t)\dots(x-x_{r+s+t-1})}{Q_{t+1}^{r-1,s}(x) Q_t^{r-1,s}(x) Q_{t+1}^{r-1,s-1}(x)}$$

erweitern, erhalten wir nach Schritt 1

$$\Phi_t^{r,s}(x) = \frac{(x - x_t)\Phi_{t+1}^{r-1,s}(x)F_1(x) - (x - x_{r+s+t})\Phi_t^{r-1,s}(x)F_2(x)}{(x - x_t)F_1(x) - (x - x_{r+s+t})F_2(x)},$$

wobei

$$F_1(x) = \Phi_t^{r-1,s}(x) - \Phi_{t+1}^{r-1,s-1}(x),$$
$$F_2(x) = \Phi_{t+1}^{r-1,s}(x) - \Phi_{t+1}^{r-1,s-1}(x).$$

Elementare Umformungen führen diese Beziehung über in die gesuchte Gleichung. □

Im Anwendungsfall verwendet man üblicherweise die Sequenz

$$(0,0) \to (0,1) \to (1,1) \to (1,2) \to \cdots \to (i,i) \to (i,i+1) \to \cdots,$$

und damit kann man die obigen etwas unübersichtlichen Formeln noch vereinfachen. In diesem Fall genügt es, sich anstelle von $(r,s)$ nur $r+s$ zu merken. Ist $y$ der Punkt, an dem wir $\Phi_t^{r,s}$ auswerten wollen, dann setzen wir

$$T_{i,k} := \Phi_t^{r,s}(y), \quad \text{mit } i = r+s+t \text{ und } k = r+s;$$

somit werden die Formeln aus Theorem 2.3.2 zu den rekursiven Beziehungen

$$T_{i,0} := f_i, \tag{35}$$
$$T_{i,-1} := 0, \tag{36}$$
$$T_{i,k} := T_{i,k-1} + \frac{T_{i,k-1} - T_{i-1,k-1}}{\dfrac{y - x_{i-k}}{y - x_i}\left(1 - \dfrac{T_{i,k-1} - T_{i-1,k-1}}{T_{i,k-1} - T_{i-1,k-2}}\right) - 1}. \tag{37}$$

Man kann die Werte $T_{i,k}$ in einem Tableau anordnen und die Berechnungen ähnlich dem Algorithmus von Neville durchführen. Der größte Unterschied besteht darin, daß nicht nur die linken Nachbarn in die Berechnungen eingehen sondern auch das Element direkt links daneben, das zwei Reihen weiter zurückliegt. Bei den Berechnungen in dem Tableau sind also alle vier Ecken eines Rhombus beteiligt.

| $(r,s) =$ | $(0,0)$ | $(0,1)$ | $(1,1)$ | $(1,2)$ | $\cdots$ |
|---|---|---|---|---|---|
| | $f_0 = T_{0,0}$ | | | | |
| $0 = T_{0,-1}$ | | $T_{1,1}$ | | | |
| | $f_1 = T_{1,0}$ | | $T_{2,2}$ | | |
| $0 = T_{1,-1}$ | | $T_{2,1}$ | | $T_{3,3}$ | |
| | $f_2 = T_{2,0}$ | | $T_{3,2}$ | $\vdots$ | $\ddots$ |
| $0 = T_{2,-1}$ | | $T_{3,1}$ | $\vdots$ | | |
| | $f_3 = T_{3,0}$ | $\vdots$ | | | |
| $\vdots$ | $\vdots$ | | | | |

Interpolationsformeln wie diese werden zum Beispiel in Taschenrechnern dazu benützt, tabellierte Funktionen an Stellen auszuwerten, die nicht in den Tabellen enthalten sind. Der Vorteil der rationalen Interpolation gegenüber interpolierenden Polynomen, speziell in der Nähe von Polstellen, sei anhand des aus [**Bulirsch, Rutishauser 1968**] entnommenen folgenden Beispieles belegt.

**Beispiel 2.3.3.** *Für die Funktion $f(x) = \cot(x)$ seien die Werte für ganze Grad in einer Tabelle gegeben. Durch Interpolation soll daraus ein Näherungswert für $\cot 2°30'$ berechnet werden. Polynominterpolation 4. Ordnung nach Neville ergibt das Tableau*

| $x_i$ | $f_i = \cot(x_i)$ | | | | |
|---|---|---|---|---|---|
| 1° | 57.28996163 | | | | |
| | | 14.30939911 | | | |
| 2° | 28.63625328 | | 21.47137102 | | |
| | | 23.85869499 | | 22.36661762 | |
| 3° | 19.08113669 | | 23.26186421 | | 22.63519158 |
| | | 21.47137190 | | 23.08281486 | |
| 4° | 14.30066626 | | 22.18756808 | | |
| | | 18.60658719 | | | |
| 5° | 11.43005230 | | | | |

*Andererseits erhalten wir durch rationale Interpolation mit Graden $(2, 2)$ gemäß den Gleichungen (35)–(37) das folgende Ergebnis:*

| $x_i$ | $f_i = \cot(x_i)$ | | | | |
|---|---|---|---|---|---|
| 1° | 57.28996163 | | | | |
| | | 22.90760673 | | | |
| 2° | 28.63625328 | | 22.90341624 | | |
| | | 22.90201805 | | 22.90369573 | |
| 3° | 19.08113669 | | 22.90411487 | | 22.903765**52** |
| | | 22.91041916 | | 22.90384141 | |
| 4° | 14.30066626 | | 22.90201975 | | |
| | | 22.94418151 | | | |
| 5° | 11.43005230 | | | | |

*Der Vergleich zur exakten Lösung $\cot 2°30' = 22.9037655484\ldots$ zeigt, daß die rationale Interpolation bei vergleichbarem Aufwand viel genauere Werte liefert (ungenaue Stellen sind unterstrichen).*

Ähnlich wie bei der Polynominterpolation ist der Neville–artige Algorithmus der Berechnung des Thieleschen Kettenbruches nur dann vorzuziehen, wenn lediglich der Wert der interpolierenden rationalen Funktion an einer bestimmten Stelle benötigt wird. Sollten mehrere Auswertungen gebraucht werden, dann ist die Verwendung der Algorithmen 2.2.4 und 2.2.5 vorzuziehen.

Zusammenfassend sei der Neville–artige Algorithmus noch einmal explizit aufgeschrieben:

**Algorithmus 2.3.4.** *Neville–artiger Algorithmus zur Auswertung der rationalen Interpolationsfunktion an einer Stelle*

$T_m = (0, \ldots, 0)$
$T = (f_0, \ldots, f_n)$
**for** $k = 0$ **to** $n$ **do**
  **for** $i = k$ **to** $n$ **do**
    $h = 1 - (T[i] - T[i-1])/(T[i] - T_m[i-1])$
    $h = h * (y - x_{i-k})/(y - x_i) - 1$
    $h = (T[i] - T[i-1])/h$
    $T_n[i] = T[i] + h$
  **done**
  $T_m = T$

$T = T_n$

**done**

*Nach Ablauf des Algorithmus enthält $T[n]$ den Wert der interpolierenden rationalen Funktion am Punkt $y$.*

## 3. Trigonometrische Interpolation

Will man periodische Funktionen approximieren, so sind weder rationale Funktionen noch Polynome besonders brauchbar, da keine dieser Funktionenklassen besonders gut dazu geeignet ist, periodische Vorgänge darzustellen. Zwei völlig verschiedene Zugänge führen dabei zum gleichen Problem. O.b.d.A. nehmen wir in diesem Abschnitt an, daß die Periodenlänge der untersuchten Funktion $f$ gleich $2\pi$ ist. Sonst führt man eine lineare Variablentransformation durch.

**3.1. Grundlagen.** Das **erste Problem**, das wir untersuchen, ist wieder ein lineares Interpolationsproblem. Gegeben seien die $N$ Punkte $(x_i, f_i)$ für $i = 0, \ldots, N-1$ mit

$$0 \le x_0 < x_1 < \cdots < x_{N-1} < 2\pi,$$

und wir nehmen an, daß sie Bild–Urbild–Paare einer $2\pi$–periodischen Funktion $f$ sind. Da die elementaren trigonometrischen Funktionen $\sin kx$ und $\cos kx$ für $k \in \mathbb{Z}$ die Periodenlänge $2\pi$ haben, versuchen wir das Interpolationsproblem für die Klasse der Funktionen $\Psi$ der Form

$$\Psi(x) := \frac{A_0}{2} + \sum_{k=1}^{M} (A_k \cos kx + B_k \sin kx) + \frac{A_{M+1}}{2} \cos(M+1)x$$

zu lösen (dabei lassen wir den letzten Term in der Summe weg, falls $N = 2M+1$ ungerade ist). Zur Vereinfachung nehmen wir an, daß das Intervall $[0, 2\pi]$ äquidistant unterteilt ist:

$$x_k := \frac{2k\pi}{N}, \quad \text{für } k = 0, 1, \ldots, N-1.$$

In diesem Fall werden die Formeln und Funktionen bei komplexer Rechnung einfacher und übersichtlicher. Verwenden wir die Moivresche Formel

$$e^{it} = \cos t + i \sin t,$$

so können wir die Funktion $\Psi$ umformen zu

$$p(x) := c_0 + c_1 e^{ix} + c_2 e^{2ix} + \cdots + c_{N-1} e^{(N-1)ix},$$

und dann werden die Interpolationsbedingungen zu

$$p(x_i) = f_i, \quad k = 0, \ldots, N-1.$$

Die Parameter von $\Psi$ und $p$ hängen wie folgt zusammen:

$$A_0 = 2c_0, \quad A_k = c_k + c_{N-k}, \quad B_k = i(c_k - c_{N-k}), \quad A_{M+1} = 2c_{M+1},$$

wobei $k = 1, \ldots, M$. Bis auf diese Gleichungen sind die Interpolationsprobleme für $p$ und $\Psi$ nicht äquivalent. Im allgemeinen stimmen $p$ und $\Psi$ nur an den Stützabszissen $x_i$ überein. Die Beziehungen von oben ermöglichen allerdings, $\Psi$ aus $p$ zu berechnen und umgekehrt. Das trigonometrische Interpolationsproblem führt also auf die Frage, wie eine Funktion $p(x)$, ein trigonometrisches Polynom zu berechnen ist, das die Interpolationsbedingungen erfüllt.

Für die speziell gewählten äquidistanten Stützstellen $x_j$ finden wir, wenn wir die Abkürzungen

$$z := e^{ix}, \quad \omega_k := e^{ix_k} = e^{2k\pi i/N}$$

verwenden, daß das komplexe Polynom

$$P(z) = c_0 + c_1 z + \cdots + c_{N-1} z^{N-1}$$

die Interpolationsbedingungen

$$P(\omega_k) = f_k, \quad k = 0, \dots, N-1$$

erfüllt. Aus der eindeutigen Lösbarkeit der Polynominterpolation folgt daher sofort

**Theorem 3.1.1.** *Zu beliebigen Stützstellen $(2k\pi/N, f_k)$ mit komplexem $f_k$ gibt es genau ein trigonometrisches Polynom*

$$p(x) = c_0 + c_1 e^{ix} + \cdots + c_{N-1} e^{(N-1)ix}$$

*mit $p(2k\pi/N) = f_k$ für $k = 0, 1, \dots, N-1$.*

Die Koeffizienten $c_j$ von $p$ lassen sich auf einfache Weise aus Formeln berechnen. Dazu benötigen wir folgende leicht nachzurechnende Beziehungen

$$\omega_j^k = \omega_k^j$$
$$\omega_k^{-j} = \overline{\omega_k^j}$$
$$\sum_{k=0}^{N-1} \omega_k^j \omega_k^\ell = \begin{cases} N & \text{für } j = \ell, \\ 0 & \text{für } j \neq \ell, \end{cases}$$

das heißt die speziellen Vektoren $w^{(j)} := (\omega_0^j, \omega_1^j, \dots, \omega_{N-1}^j)^\top$ bilden eine Orthogonalbasis von $\mathbb{C}^N$. Aus den Eigenschaften von Orthogonalbasen in euklidischen Vektorräumen erhalten wir also Formeln für die Koeffizienten $c_j$.

**Proposition 3.1.2.** *Für das trigonometrische Polynom $p(x) = \sum_{k=0}^{N-1} c_k e^{ikx}$ gilt*

$$p(x_j) = f_j, \quad j = 0, \dots, N-1$$

*für komplexe Zahlen $f_j$ und $x_j = 2j\pi/N$ genau dann wenn*

$$c_j = \frac{1}{N} \sum_{k=0}^{N-1} f_k \omega_k^{-j} = \frac{1}{N} \sum_{k=0}^{N-1} f_k e^{-2\pi i jk/N}.$$

Die Abbildung $\mathcal{F} : \mathbb{C}^N \to \mathbb{C}^N$ mit $(f_k) \mapsto (c_j)$ heißt *diskrete Fouriertransformation* (DFT). Ihre Umkehrung $(c_j) \mapsto (f_k) = \mathcal{F}^{-1}(c_j)$ entspricht der Auswertung eines trigonometrischen Polynoms und wird *Fouriersynthese* genannt. Der Zusammenhang zwischen $\mathcal{F}$ und $\mathcal{F}^{-1}$ ist

$$f = \mathcal{F}^{-1}(c_j) = N\overline{\mathcal{F}(\overline{c_j})};$$

das heißt man kann die Inverse der diskreten Fouriertransformation wieder durch eine diskrete Fouriertransformation ausdrücken.

Das **zweite Problem** ist das der Approximation einer $2\pi$–periodischen Funktion $f$ durch ein trigonometrisches Polynom $\Psi$ (geformt wie oben), sodaß der Fehler im quadratischen Mittel minimal ist:

$$\int_0^{2\pi} (f(x) - \Psi(x))^2 \, dx = \min.$$

Dieses Problem läßt sich durch folgende Überlegungen auf ein Problem der Funktionalanalysis (das eine leichte Verallgemeinerung einer Fragestellung der linearen Algebra ist) zurückführen. Betrachten wir die stetigen Funktionen $C[0, 2\pi]$ auf dem Intervall $[0, 2\pi]$. Für diese Funktionen können wir ein inneres Produkt definieren als

$$\langle f, g \rangle := \int_0^{2\pi} f(x) g(x) \, dx.$$

Auf diese Weise wird $C[0, 2\pi]$ zu einem Raum mit innerem Produkt, und durch die übliche Definition $\|f\|_2 := \sqrt{\langle f, f \rangle}$ wird er auch zu einem normierten Raum. Unglücklicherweise

konvergiert nicht jede Cauchyfolge $x_k$ in $(C[0,2\pi], \| \quad \|_2)$ gegen ein Element von $C[0,2\pi]$. Ganz ähnlich wie man aus $\mathbb{Q}$ die reellen Zahlen $\mathbb{R}$ konstruiert kann man aus $C[0,2\pi]$ den Raum $L^2[0,2\pi]$ bilden, den Raum der quadratisch Lebesgue–integrierbaren Funktionen auf dem Intervall $[0,2\pi]$, der wie $\mathbb{R}$ vollständig ist. Jede auf $[0,2\pi]$ stetige Funktion und jede Funktion $f$, deren Quadrat $f^2$ Riemann–integrierbar ist, sind in $L^2[0,2\pi]$ enthalten. Der Raum $L^2[0,2\pi]$ zusammen mit dem inneren Produkt $\langle \quad , \quad \rangle$ ist vollständig, also ein *Hilbertraum*. Hilberträume sind euklidischen Vektorräumen sehr ähnlich, mit der Ausnahme, daß sie im allgemeinen unendlich dimensional sind.

Nehmen wir an, wir hätten eine Orthonormalbasis $\{\varphi_0, \varphi_1, \dots\}$ von $L^2[0,2\pi]$, sodaß wir jedes Element $f \in L^2[0,2\pi]$ darstellen können als

$$f(x) = \sum_{i=0}^{\infty} a_i \varphi_i(x). \tag{38}$$

Solche Orthonormalbasen (sind **keine** Basen im Vektorraumsinn) gibt es in $L^2[0,2\pi]$, wie in jedem separablen Hilbertraum. Wie in euklidischen Vektorräumen können wir dann die Koeffizienten $a_k$ der Entwicklung (38) einfach berechnen:

$$\langle \varphi_k, f\rangle = \langle \varphi_k, \sum_{i=0}^{\infty} a_i\varphi_i\rangle = \sum_{i=0}^{\infty} a_i \langle \varphi_k, \varphi_i\rangle = a_k,$$

wobei wir einfach annehmen, daß wir Summe und Integral (das innere Produkt) vertauschen dürfen (man kann beweisen, daß man das immer darf). Brechen wir die Entwicklung (38) beim $N$–Term ab

$$f(x) = \sum_{i=0}^{N} a_i \varphi_i(x),$$

so lösen wir außerdem das Problem, welche Linearkombination der endlich vielen Basiselemente $\{\varphi_0, \dots, \varphi_N\}$ die Lösung des Kleinste–Quadrate–Problem

$$\min \left\| f - \sum_{i=0}^{N} a_i \varphi_i \right\|_2$$

ist. Das ist klar, weil der „Fehlervektor“ $f - \sum_{i=0}^{N} a_i\varphi_i$ orthogonal auf den von $\varphi_0, \dots, \varphi_N$ aufgespannten Teilraum steht.

Alle diese Resultate wären für unser Problem jedoch völlig überflüssig, hätten wir nicht das folgende Resultat:

**Proposition 3.1.3.** *Die trigonometrischen Funktionen* $1, \sin kx, \cos kx$ *für* $k = 1, 2, \dots$ *bilden ein System von paarweise orthogonalen Funktionen in* $L^2[0,2\pi]$. *Es gelten die Beziehungen*

$$\int_0^{2\pi} \cos jx \cos kx\, dx = \begin{cases} 0 & \text{für } j \neq k \\ 2\pi & \text{für } j = k = 0 \\ \pi & \text{für } j = k > 0, \end{cases}$$

$$\int_0^{2\pi} \sin jx \sin kx\, dx = \begin{cases} 0 & \text{für } j \neq k,\ j > 0,\ k > 0 \\ \pi & \text{für } j = k > 0, \end{cases}$$

$$\int_0^{2\pi} \sin jx \cos kx\, dx = 0 \quad \text{für alle } j \geq 0,\ k > 0.$$

*Die Funktionen*

$$\left\{ \frac{1}{\sqrt{2\pi}}, \frac{\cos kx}{\sqrt{\pi}}, \frac{\sin kx}{\sqrt{\pi}} \right\}, \quad k = 1, 2, \dots$$

*bilden eine Orthonormalbasis von $L^2[0, 2\pi]$ im oben besprochenen Sinn.*

Beweis. Der Beweis kann in jedem Buch über Analysis oder Funktionalanalysis gefunden werden, etwa in [**Heuser 1986/2**, XVII]. □

Die Entwicklung in die Orthonormalbasis der trigonometrischen Funktionen kann man zusammenfassend schreiben als

$$f(x) = \frac{A_0}{2} + \sum_{k=1}^{\infty} A_k \cos kx + B_k \sin kx. \tag{39}$$

wobei der Faktor $\frac{1}{2}$ vor $A_0$ ein Normierungsfaktor ist, der sicherstellt, daß sich die Integrale zur Berechnung der Koeffizienten in kompakterer Form darstellen lassen.

Die Koeffizienten der Entwicklung (39) für die Basis der trigonometrischen Funktionen heißen *Fourierkoeffizienten*, die Summe auf der rechten Seite der Entwicklung heißt die *Fourierentwicklung* bzw. die *Fourierreihe* von $f$. Den Übergang von $f$ zu den Fourierkoeffizienten nennt man *Fourieranalyse*. Stellt man umgekehrt $f$ als diese Summe dar und berechnet man Werte von $f$ aus der Fourierreihe (wertet man also die Fourierreihe aus), so bezeichnet man den Vorgang als *Fouriersynthese*, zusammenfassend bezeichnet man diese beiden Übergänge als *Fouriertransformation*.

Zur Berechnung der Fourierkoeffizienten muß man also Integrale der Form

$$A_k = \frac{1}{\pi} \int_0^{2\pi} f(x) \cos kx \, dx \tag{40}$$

$$B_j = \frac{1}{\pi} \int_0^{2\pi} f(x) \sin jx \, dx \tag{41}$$

für $k = 0, 1, 2, \ldots$ und $j = 1, 2, \ldots$ berechnen. Nachdem man aber, wie üblich in der numerischen Mathematik, unendliche Reihen für Berechnungszwecke nicht verwenden kann, ist man darauf angewiesen, die Fourierreihe von $f$ nach endlich vielen Gliedern abzubrechen. Daß das (zumindest für genügend glatte Funktionen) nichts ausmacht, zeigt der folgende Satz

**Theorem 3.1.4.** *Sei $f \in \mathbb{C}^\infty[0, 2\pi]$ eine Funktion mit Periodenlänge $2\pi$. Dann erfüllt die Folge der Fourierkoeffizienten von $f$*

$$\lim_{n\to\infty} n^l A_n = 0$$

*für beliebiges $l$. $(A_n)$ fällt also schneller als jedes Polynom in $n$ wächst; so eine Folge nennt man* schnell fallend. *Ein analoges Resultat gilt für die Folge der $B_k$.*

Setzt man also gemäß dem vorangegangenen Resultat die Fourierkoeffizienten ab einem bestimmten fixen Glied $N$ gleich 0, so ist der entstehende Fehler nicht sehr groß. Ist die transformierte Funktion $f$ nicht unendlich oft differenzierbar, so fallen die Fourierkoeffizienten nicht schnell, doch es gilt immer noch für jedes $f \in L^2[0, 2\pi]$, daß

$$A_0^2 + \sum_{i=1}^{\infty} A_i^2 + B_i^2$$

konvergiert. Selbst für nichtstetige Funktionen bilden also die Fourierkoeffizienten jedenfalls immer eine Nullfolge.

Wollen wir die Fourierkoeffizienten von $f$ berechnen, so müssen wir die Integrale (40) und (41) berechnen. Auf numerischem Weg löst man dieses Problem in diesem Fall am besten mit Hilfe der Trapezregel (siehe Kapitel 6) und äquidistanten Integrationsstützstellen

$$x_j = \frac{2\pi j}{N}, \quad j = 0, \ldots, N.$$

Aus der Integrationsformel erhalten wir dann die Approximation

$$a_k = \frac{2}{N}\sum_{i=1}^{N} f(x_i)\cos kx_i \tag{42}$$

$$b_j = \frac{2}{N}\sum_{i=1}^{N} f(x_i)\sin jx_i, \tag{43}$$

mit $k = 0, 1, \ldots$ und $j = 1, 2, \ldots$ für die Fourierkoeffizienten von $f$.

Nachdem man aus den Summenidentitäten für die trigonometrischen Funktionen

$$\begin{aligned}\cos ky\cos \ell y &= \tfrac{1}{2}(\cos(k+\ell)y + \cos(k-\ell)y)\\ \sin ky\sin \ell y &= \tfrac{1}{2}(\cos(k-\ell)y - \cos(k+\ell)y)\\ \cos ky\sin \ell y &= \tfrac{1}{2}(\sin(k+\ell)y - \sin(k-\ell)y)\end{aligned}$$

kennt, erhält man für die äquidistanten Stützstellen die diskreten Orthogonalitätsrelationen.

**Theorem 3.1.5.** *Die trigonometrischen Funktionen erfüllen für die äquidistanten Stützstellen die diskreten Orthogonalitätsrelationen*

$$\sum_{j=1}^{N}\cos kx_j\cos \ell x_j = \begin{cases} 0, & \text{falls } \frac{k+\ell}{N}\notin\mathbb{Z} \text{ und } \frac{k-\ell}{N}\notin\mathbb{Z}\\ \frac{N}{2}, & \text{falls entweder } \frac{k+\ell}{N}\in\mathbb{Z} \text{ oder } \frac{k-\ell}{N}\in\mathbb{Z}\\ N, & \text{falls } \frac{k+\ell}{N}\in\mathbb{Z} \text{ und } \frac{k-\ell}{N}\in\mathbb{Z}\end{cases}$$

$$\sum_{j=1}^{N}\sin kx_j\sin \ell x_j = \begin{cases} 0, & \text{falls } \frac{k+\ell}{N}\notin\mathbb{Z} \text{ und } \frac{k-\ell}{N}\notin\mathbb{Z} \text{ oder}\\ & \frac{k+\ell}{N}\in\mathbb{Z} \text{ und } \frac{k-\ell}{N}\in\mathbb{Z}\\ -\frac{N}{2}, & \text{falls } \frac{k+\ell}{N}\in\mathbb{Z} \text{ und } \frac{k-\ell}{N}\notin\mathbb{Z}\\ \frac{N}{2}, & \text{falls } \frac{k+\ell}{N}\notin\mathbb{Z} \text{ und } \frac{k-\ell}{N}\in\mathbb{Z}\end{cases}$$

$$\sum_{j=1}^{N}\cos kx_j\sin \ell x_j = 0.$$

Benutzt man diese Aussagen, um das spezielle Fourierpolynom

$$g(x) = \frac{a_0}{2} + \sum_{j=1}^{N-1}(a_j\cos jx + b_j\sin jx) + \frac{a_n}{2}\cos nx$$

zu untersuchen, findet man das (vielleicht etwas überraschende) Resultat

**Theorem 3.1.6.** *Das spezielle Fourierpolynom $g(x)$ zu den äquidistanten Stützstellen $x_j$, dessen Koeffizienten aus den Näherungsformeln (42) und (43) berechnet wurden, ist das eindeutige interpolierende trigonometrische Polynom mit den Stützstellen $\big(x_j, f(x_j)\big)$.*

Nach diesem Satz ist plötzlich das vollkommen verschiedene Problem 2 auf Fall 1 zurückgeführt worden; eine Näherungslösung für das approximierende Fourierpolynom ist also das interpolierende trigonometrische Polynom zu den äquidistanten Stützstellen $x_j$. Man landet bei der Diskretisierung der Fouriertransformation genau bei der diskreten Fouriertransformation, was auch den Namen der ersten Transformation rechtfertigt.

**3.2. Die schnelle Fouriertransformation (FFT).** Die Berechnung der diskreten Fouriertransformation und damit auch die Bestimmung einer Approximation für die Fourierreihe einer Funktion läuft darauf hinaus, Summen der Form

$$c_j = \frac{1}{N}\sum_{k=0}^{N-1} f_k e^{-2\pi ijk/N} \tag{44}$$

zu berechnen. Die direkte Auswertung benötigt $O(N^2)$ Operationen, was für Anwendungen mit sehr großen Zahlen $N$ ungeeignet ist. In der Mitte der Sechzigerjahre haben zuerst [**Cooley, Tukey 1965**] und dann [**Gentleman, Sande 1966**] Verfahren angegeben, die zumindest für spezielle Werte von $N$ nur einen Aufwand von $O(N \log N)$ elementaren Operationen benötigen. Generell wird die Erfindung der Algorithmen, die unter dem Namen „Fast Fourier Transform“ (FFT) bekannt geworden sind, Cooley und Tukey zugeschrieben, doch bereits in [**Good 1958**] wurde ein analoges Verfahren beschrieben.

Alle diese Methoden beruhen darauf, $N$ als Produkt kleinerer Zahlen zu schreiben. Im Idealfall geht man von einer Primfaktorenzerlegung

$$N = p_1^{m_1} p_2^{m_2} \dots p_n^{m_n}$$

aus und führt die Fouriertransformation der Ordnung $N$ schrittweise auf eine Abfolge von Fouriertransformationen niedrigerer Ordnung zurück. Im günstigsten Fall, wenn

$$N = 2^n$$

für eine natürliche Zahl $n$ ist, funktioniert die Methode am besten und bietet die größte Aufwandsreduktion. Ist $N$ eine Primzahl, so ist keine Reduktion der Anzahl der Rechenschritte möglich, und ein Algorithmus, der die Summen (44) auswertet, hat zwangsläufig Ordnung $O(N^2)$. Für Anwendungsfälle ist eine Situation möglichst zu vermeiden, in denen zu große Primfaktoren $p_k$ in der Primfaktorenzerlegung von $N$ auftreten.

Wir wollen uns im folgenden auf den Spezialfall $N = 2^n$ beschränken und dabei die Methode aus [**Gentleman, Sande 1966**] vorstellen, die auch Sande–Tukey Verfahren genannt wird. Sei

$$\rho_m = e^{-\frac{2\pi i}{2^m}}.$$

Mit dieser Notation werden die Summen (44) zu

$$N c_j = \sum_{k=0}^{N-1} f_k \rho_n^{jk}. \tag{45}$$

Die Methode der schnellen Fouriertransformation betrachtet die Summen für gerades und ungerades $j$ getrennt. Faßt man die Terme für $f_k \rho_n^{jk}$ und $f_{k+M} \rho_n^{j(k+M)}$ zusammen (mit $M = N/2$), so wird (45) zu

$$N c_{2\ell} = \sum_{k=0}^{N-1} f_k \rho_n^{2\ell k} = \sum_{k=0}^{M-1} (f_k + f_{k+M}) \rho_{n-1}^{\ell k} =: \sum_{k=0}^{M-1} f_{0,k}^{(n-1)} \rho_{n-1}^{\ell k} \tag{46}$$

$$N c_{2\ell+1} = \sum_{k=0}^{N-1} f_k \rho_n^{(2\ell+1)k} = \sum_{k=0}^{M-1} ((f_k - f_{k+M}) \rho_n^k) \rho_{n-1}^{\ell k} =: \sum_{k=0}^{M-1} f_{1,k}^{(n-1)} \rho_{n-1}^{\ell k}, \tag{47}$$

wobei wir verwendet haben, daß $\rho_n^2 = \rho_{n-1}$ und $\rho_n^M = -1$. Betrachten wir die Gleichungen genau, so sehen wir, daß wir die Summen der Gestalt (44) auf je zwei Summen der halben Länge zurückgeführt haben. Nachdem $N$ eine Zweierpotenz ist, können wir dieses Verfahren rekursiv fortsetzen, was zu den Beziehungen

$$N c_{j 2^{n-m}+r} = \sum_{k=0}^{2 \cdot 2^{m-1}-1} f_{r,k}^{(m)} \rho_m^{jk}, \quad r = 0, 1, \dots, 2^{n-m} - 1, \quad j = 0, 1, \dots, 2 \cdot 2^{m-1} - 1$$

führt, wobei die $f_{r,k}^{(m)}$ rekursiv definiert sind durch

$$\begin{aligned} f_{0,k}^{(n)} &= f_k \\ f_{r,j}^{(m-1)} &= f_{r,j}^{(m)} + f_{r,j+2^{m-1}}^{(m)} \\ f_{r+2^{n-m},j} &= (f_{r,j}^{(m)} - f_{r,j+2^{m-1}})\rho_m^j, \end{aligned}$$

für $k = 0, \ldots, N-1$, $j = 0, \ldots, 2^{m-1}-1$, $r = 0, \ldots, 2^{m-n}-1$ und $m = 1, \ldots, n$. Der Beweis dieser Formeln geschieht durch Induktion nach $m$ und derselben Umformung wie in den Gleichungen (46) und (47). Am Ende erhalten wir für $m = 0$ die Formel

$$c_r = \frac{1}{N} f_{r,0}^{(0)}, \quad r = 0, \ldots, N-1.$$

Die Fouriersummen (44) sind also auf $N$ Fouriertransformationen der Länge 1 zurückgeführt worden, die trivial gelöst werden können. Der Aufwand beträgt in jedem Schritt $3N/2$ elementare Operationen ($N$ Additionen und $N/2$ Multiplikationen). Nachdem $n$ Schritte ausgeführt werden müssen, beträgt der Rechenaufwand $O(Nn) = O(N \log_2 N)$ elementare Operationen.

Versuchen wir nun aus diesen Herleitungen einen Algorithmus zu entwickeln, so stoßen wir zunächst nicht auf Schwierigkeiten, wenn wir zwei Arrays der Länge $N$ verwenden, um die alten Werte $f_{*,*}^{(m)}$ und die neuen Werte $f_{*,*}^{(m-1)}$ auf einmal abspeichern zu können. Ist die Zahl $N$ jedoch sehr groß und versuchen wir daher mit nur einem Array von $N$ Elementen Größe auszukommen, so müssen wir noch einige zusätzliche Überlegungen anstellen.

Es genügt uns dann ein einzelner Array $\tilde{f}$ für die Abspeicherung der $N$ Zahlen $f_{r,k}^{(m)}$ ($r = 0, \ldots, 2^{m-n}-1$, $k = 0, \ldots, 2^m - 1$), wenn wir nach Auswertung von $f_{r,k}^{(m)}$ und $f_{r,k+2^{m-1}}^{(m)}$ mit $f_{r,k}^{(m-1)}$ und $f_{r+2^{n-m},k}^{(m-1)}$ überschreiben. Dies können wir dadurch erreichen, daß wir $f_{r,k}^{(m)}$ im Element $\tilde{f}[\tau(m,r,k)]$ abspeichern, wobei die Indexabbildung $\tau$ die Bedingungen

$$\begin{aligned} \tau(m-1,r,k) &= \tau(m,r,k) \\ \tau(m-1,r+2^{n-m},k) &= \tau(m,r,k+2^{m-1}) \end{aligned}$$

für $m = 1, \ldots, n$, $r = 0, \ldots, 2^{n-m}-1$ und $k = 0, \ldots, 2^m - 1$ erfüllen muß. Als Startwerte verwendet man die natürliche Ordnung der $f_k$, das heißt $\tilde{f}[k] = f_k$ und wir haben

$$\tau(n,0,k) = k.$$

Löst man die Rekursion, so kann man explizite Formeln für $\tau$ angeben. Diese beruhen auf der Binärdarstellung ganzer Zahlen:

$$z = a_0 + a_1 \cdot 2 + \cdots + a_{n-1} \cdot 2^{n-1}, \quad a_k \in \{0,1\}$$

Definiert man die *Bitumkehrabbildung* $u$, eine selbstinverse Abbildung, durch

$$u(z) = a_0 \cdot 2^{n-1} + a_1 \cdot 2^{n-2} + \cdots + a_{n-1},$$

so gilt für die Indexabbildung

$$\tau(m,r,k) = k + u(r)$$

für alle $m = 1, \ldots, n$, $r = 0, \ldots, 2^{n-m}-1$ und $k = 0, \ldots, 2^m - 1$. Der Beweis für diese Behauptung erfolgt übrigens durch Nachrechnen und Induktion. Für $m = n$ stimmt die Behauptung offensichtlich, und der Schritt von $m$ auf $m-1$ läuft folgendermaßen ab:

$$\begin{aligned} k &=: a_0 + a_1 \cdot 2 + \cdots + a_{m-2} \cdot 2^{m-2} + 0 \cdot 2^{m-1}, \\ r &=: a_{n-1} + a_{n-2} \cdot 2 + \cdots + a_m \cdot 2^{n-m-1} + 0 \cdot 2^{n-m}, \end{aligned}$$

und daher gilt

$$\begin{aligned} k + u(r + 2^{n-m}) = \\ &= a_0 + \cdots + a_{m-2} \cdot 2^{m-2} + 1 \cdot 2^{m-1} + a_m \cdot 2^m + \cdots + a_{n-1} \cdot 2^{n-1} = \\ &= k + 2^{m-1} + u(r). \end{aligned}$$

Für $m = 0$ gilt insbesondere

$$\tau(0, r, 0) = u(r), \quad \text{für } r = 0, \ldots, N-1.$$

Zusammenfassend kann man den folgenden Algorithmus für die Fouriertransformation angeben.

**Algorithmus 3.2.1.** *FFT nach Sande und Tukey*

$\tilde{f} = (f_0, \ldots, f_{N-1})$
**for** $m = n$ **to** $1$ **step** $-1$ **do**
  **for** $k = 0$ **to** $2^{m-1} - 1$ **do**
    $e = \rho_m^k$
    **for** $r = 0$ **to** $2^{n-1} - 1$ **step** $2^m$ **do**
      $x = \tilde{f}[r + k]$
      $y = \tilde{f}[r + k + 2^{m-1}]$
      $\tilde{f}[r + k] = x + y$
      $\tilde{f}[r + k + 2^{m-1}] = e(x - y)$
    **done**
  **done**
**done**

Nach Ablauf des Algorithmus berechnet man die gesuchten Fouriertransformierten $c_j$ als

$$c_j = \frac{1}{N} \tilde{f}[u(j)], \quad \text{für } j = 0, \ldots, N-1.$$

Die Verallgemeinerung dieses Verfahrens für allgemeines $N$ sei im folgenden noch kurz skizziert. Nehmen wir an, daß $N = p \cdot m$ als Produkt zweier Zahlen dargestellt ist, wobei $p$ eine Primzahl sei. In diesem Fall fassen wir in (44) die Summen $c_k$ zusammen, deren Indices $k \cong \mu \mod p$ für fixes $0 \leq \mu \leq p-1$ erfüllen. Mit $k = \ell p + \mu$ und $l = 0, \ldots, m-1$ haben wir die Darstellung

$$c_{\ell p + \mu} = \sum_{j=0}^{pm-1} f_j \omega_N^{j(\ell p + \mu)} = \sum_{j=0}^{m-1} \left( \sum_{\nu=0}^{p-1} f_{j+\nu m} \omega_N^{(j+\nu m)(\ell p + \mu)} \right).$$

Verwenden wir

$$\omega_N^{(j+\nu m)(\ell p + \mu)} = \omega_N^{j\ell p} \omega_N^{\nu \ell m p} \omega_N^{(j+\nu m)\mu} = \omega_N^{(j+\nu m)\mu} \left(\omega_N^p\right)^{j\ell},$$

so erhalten wir daraus eine neue Darstellung der Summe

$$c_{\ell p + \mu} = \sum_{j=0}^{m-1} \left( \sum_{\nu=0}^{p-1} f_{j+\nu m} \omega_N^{(j+\nu m)\mu} \right) \omega_m^{j\ell} =: \sum_{j=0}^{m-1} f'_{j+m\mu} \omega_m^{j\ell};$$

das heißt, daß wir die Fouriertransformation der Ordnung $N$ auf $p$ Fouriertransformationen der Ordnung $m$ zurückgeführt haben. Wenn wir nun die Koeffizienten $f'_{j+m\mu}$ genauer unter die Lupe nehmen, so finden wir bei festem $j$ die neue Darstellung

$$f'_{j+m\mu} = \omega_N^{j\mu} \sum_{\nu=0}^{p-1} f_{j+\nu m} \omega_N^{\nu m \mu} = \omega_N^{j\mu} \left( \sum_{\nu=0}^{p-1} f_{j+\nu m} \omega_p^{\mu\nu} \right).$$

Man berechnet die Koeffizienten für die Fouriertransformationen der Ordnung $m$ selbst wieder aus Fouriertransformationen der Ordnung $p$. Folglich ist eine Fouriertransformation der Ordnung $p{\cdot}m$ zurückführbar auf $p$ Fouriertransformationen der Ordnung $m$, deren Koeffizienten aus $m$ Fouriertransformationen der Ordnung $p$ berechnet werden können. Berücksichtigt man, daß für $\mu = 0$ nur eine Summe für $f'_j$ berechnet werden muß, und daß für $\nu = 0$ der Faktor des ersten Summanden gleich 1 ist, so erkennt man, daß der Rechenaufwand für den Reduktionsschritt $3p(p-1)m$ elementare Operationen beträgt, davon $p(p-1)m$ Multiplikationen. Für den Rechenaufwand $Z_N$ einer Fouriertransformation der Ordnung $p \cdot m$ gilt

$$Z_N = Z_{p\cdot m} = p \cdot Z_m + 3p(p-1)m, \quad p > 2.$$

Diese Reduktion kann man jetzt gemäß der Primfaktorenzerlegung von $N$ weiterführen, und man benötigt $m_1 + m_2 + \cdots + m_n$ Reduktionsschritte. Das verringert zwar ebenfalls den Rechenaufwand meist beträchtlich gegenüber $O(N^2)$, doch die Effizienz $O(N \log N)$ wird nur für $N = 2^n$ erreicht.

Zum Abschluß der Untersuchungen über trigonometrische Polynome bleibt noch die Untersuchung, welche Phänomene bei der trigonometrischen Interpolation auftreten. Beginnen wir mit Resultaten über die Fourierreihe einer Funktion.

**Theorem 3.2.2.** *Sei $f \in L^2[0, 2\pi]$ eine $2\pi$–periodische stückweise stetige Funktion mit stückweise stetiger erster Ableitung. Dann konvergiert die zugehörige Fourierreihe*

$$g(x) = \frac{A_0}{2} + \sum_{k=0}^{\infty} (A_k \cos kx + b_k \sin kx)$$

*an $x_0$ gegen*

$$\frac{1}{2}\big(\lim_{y \to x_0 -} f(y) + \lim_{y \to x_0 +} f(y)\big).$$

*Das heißt, falls $f$ an $x_0$ stetig ist, konvergiert $g$ an $x_0$ gegen $f(x_0)$.*

Beweis. Kann in jedem Lehrbuch über Analysis, z.B. in [**Heuser 1986/1**, XVII] nachgelesen werden. □

Wie wir gesehen haben ist das approximierende Fourierpolynom, dessen Koeffizienten durch Trapezregel berechnet werden, genau das interpolierende trigonometrische Polynom. Alle diese Approximationen sind so gestaltet, daß bei Erhöhung von $N$, das heißt Erhöhung des „Polynomgrades“ bei gleichzeitiger Erhöhung der Stützstellenzahl, das interpolierende trigonometrische Polynom gegen die Fourierreihe strebt. Daher leidet die trigonometrische Interpolation nicht an dem selben Mangel wie die Polynominterpolation, daß bei Erhöhung der Stützstellenzahl nicht notwendigerweise eine Verbesserung der Approximationseigenschaften erzielt wird. Trotzdem treten bei der trigonometrischen Interpolation unerwünschte Phänomene auf. Ganz ähnlich wie die Interpolationspolynome zeigen auch die trigonometrischen Polynome Überschwingungsverhalten (siehe Abbildung 5.2), das Gibbs–Phänomen. Allerdings sind diese Schwingungen nur dann wirklich schlimm, wenn die zu approximierende Funktion Unstetigkeitsstellen aufweist.

## 4. Splines

Bislang haben wir Interpolationsprobleme immer in Funktionsklassen gelöst, in denen die einzelnen Funktionen (Polynome, rationale Funktionen, trigonometrische Polynome) über den gesamten interpolierten Bereich durch einen einzigen mathematischen Ausdruck gegeben sind. Alle diese Funktionsklassen haben den Nachteil, daß ihre Elemente auf die eine oder andere Weise „starr“ sind. Polynome lösen zwar jedes Interpolationsproblem, sie sind jedoch nicht in der Lage, die (optische) Form der Punkte in genügender Weise wiederzugeben

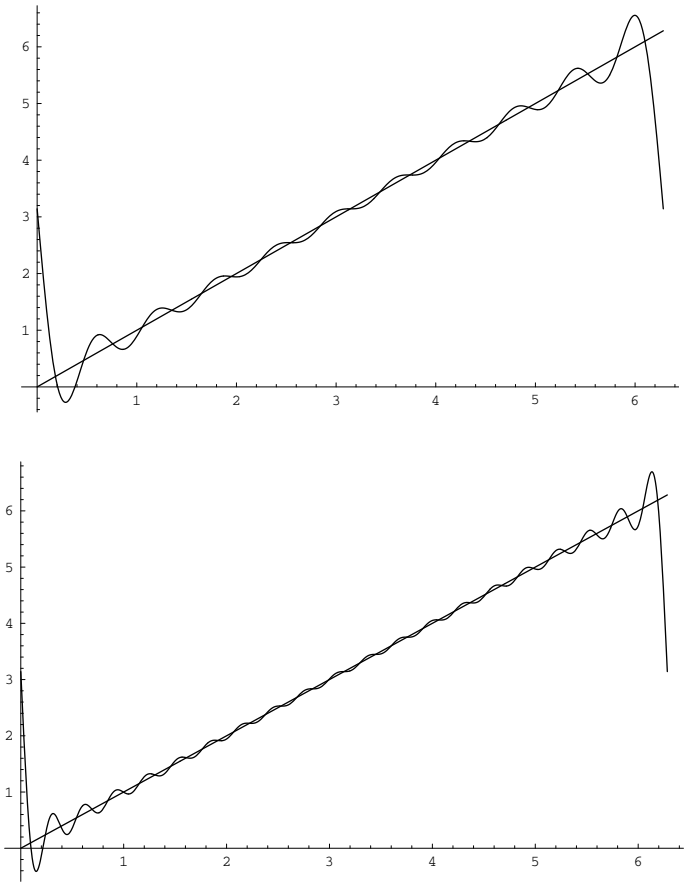



Abbildung 5.2. Gibbs–Phänomen: Oben das Fourierpolynom 10. Grades, unten das 20. Grades

(siehe Abbildung 5.3). Rationale Funktionen auf der anderen Seite sind zwar viel besser als

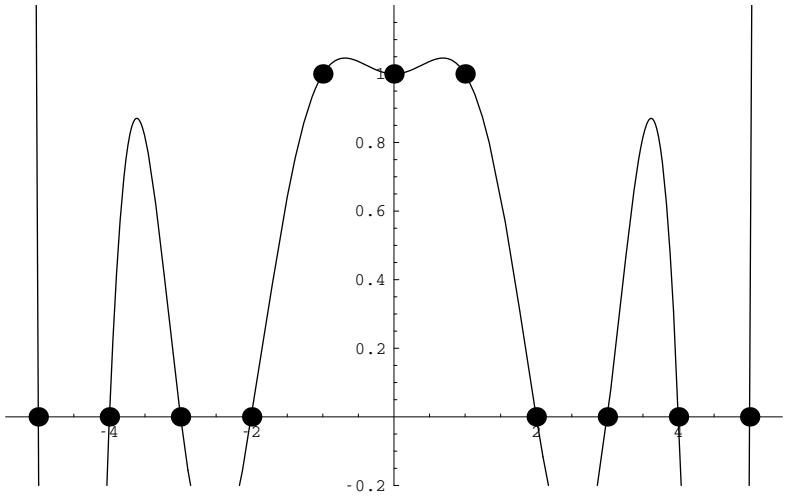



Abbildung 5.3. Oszillationen des Interpolationspolynoms

Polynome, wenn es darum geht, Form wiederzugeben, doch dafür sind sie zu starr, jedes Interpolationsproblem zu lösen, es gibt, wie wir in Abschnitt 2.1 gesehen haben, unerreichbare Punkte (siehe Abbildung 5.4). Darüber hinaus können Polynome und rationale Funktionen

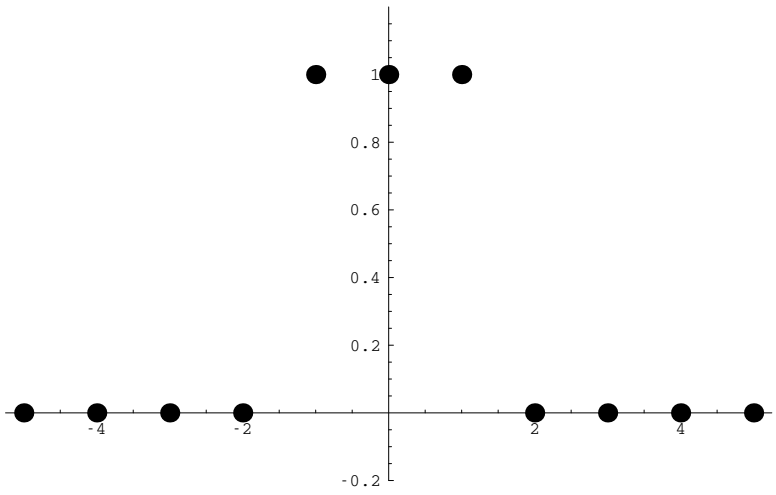



ABBILDUNG 5.4. Unerreichbare Punkte für die rationale Interpolation: $\Phi^{5,5} \equiv 0$!

Eigenschaften wie Periodizität nicht oder nur ungenügend einfangen.

Die letzte Funktionsklasse, die wir in Abschnitt 3 kennengelernt haben, die trigonometrischen Polynome, löst das Problem der Periodizität, doch leidet sie wieder an Oszillationsphänomenen, fast wie die Polynome. Das Gibbs–Phänomen erzeugt Schwingungen wie sie in Abbildung 5.5 zu sehen sind. Sie sind zwar viel kleiner als die Schwingungen der Polynome, doch sie sind größer als notwendig. Außerdem erzwingt die Entwicklung in Grundschwingungen Periodizität. Diese Funktionsklasse ist daher nicht in der Lage nichtperiodische Funktionen gut zu approximieren.

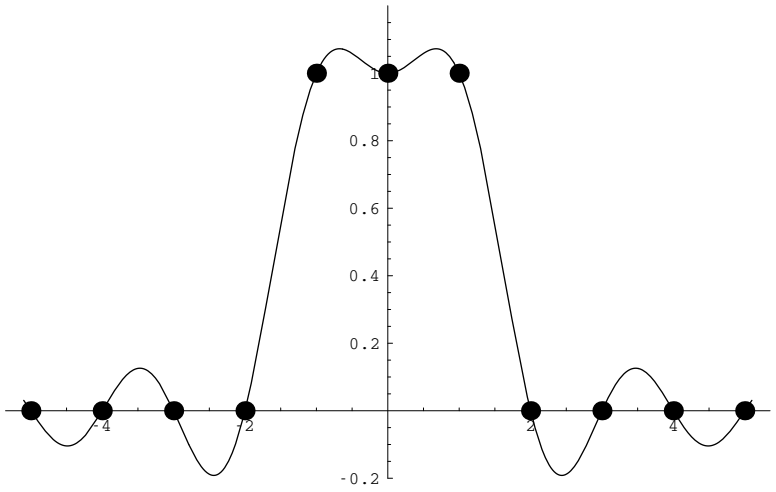



ABBILDUNG 5.5. Trigonometrische Interpolation

**4.1. Grundlagen.** Alle diese Funktionen sind in gewisser Weise starr durch das Festhalten an einem fixen mathematischen Ausdruck, der im gesamten interpolierten Bereich seine Gültigkeit hat, der sogar auf ganz $\mathbb{R}$ definiert ist (mit Ausnahme der Polstellen bei der rationalen Interpolation) und noch dazu leicht auszuwerten ist. Außerdem sind sie in ihrem gesamten Definitionsbereich $C^\infty$, also beliebig oft differenzierbar.

Die nächste Funktionsklasse, die wir untersuchen, versucht das Problem auf grundsätzlich andere Weise zu lösen und vermeidet (oder verringert zumindest) dabei die Probleme der fehlenden Wiedergabe von Form und Periodizität, wie man etwa in Abbildung 5.6 erkennen kann.

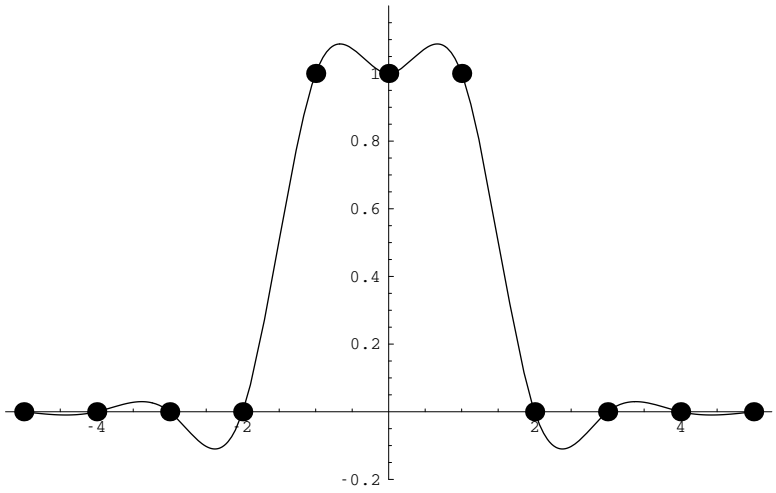



Abbildung 5.6. Kubischer interpolierender Spline

Die Grundidee ist, die Funktionen aus kurzen Stücken zusammenzusetzen, und zwar so, daß die entstehende Abbildung $C^k$ ($k$–mal stetig differenzierbar) ist für ein $k > 0$. Auf solche Weise gebildete Funktionen werden *Splines* genannt. („Spline“ hat in der englischen Sprache mehrere Bedeutungen. Ursprünglich bedeutete Spline dünne Latte, Leiste oder Lamette. Später bezeichnete man eine dünne biegsame Holz- oder Gummileiste, mit der man große Kurven zeichnen konnte — also ein überdimensionales Kurvenlineal — ebenfalls mit Spline.)

**4.2. B–Splines.** Sind die Stücke, aus denen die Splines zusammengebaut sind, Polynome, im übrigen die häufigste Variante, dann spricht man von polynomialen Splines oder schlicht von Splines. Besondere Wichtigkeit haben dabei die kubischen Splines, die wir in Abschnitt 4.3 ausführlich behandeln werden.

Splineinterpolation kann man auch gut auf Kurven (Splinekurven, Bézier Kurven), Flächen (Splinepatches, Bézierpatches) und Hyperflächen verallgemeinern. Mehrdimensionale Interpolation ist Bestandteil von Teil 2, Kapitel 14. Diese Verallgemeinerungen kann man auch in [**Risler 1992**] nachlesen. Ursprünglich wurden die Splines in [**Rutishauser 1960**] eingeführt. Umfassende Darstellungen kann man etwa in [**Böhmer 1974**], [**de Boor 1978**] oder [**Überhuber 1995a**, 9] nachlesen.

Beginnen wir mit den mathematischen Begriffen.

**Definition 4.2.1.** *Sei $\mathcal{P}_k := \mathbb{R}^k[x]$ der $(k+1)$–dimensionale Vektorraum der Polynome höchstens $k$–ten Grades in einer Variablen.*

**Definition 4.2.2.** *Sei $k \geq 0$ eine ganze Zahl, $[a,b] \subset \mathbb{R}$ ein Intervall, und sei $\tau$ eine Folge von $\ell - 1$ Punkten, genannt* Knotenpunkte*, in $[a,b]$ mit*

$$a < \tau_1 < \tau_2 < \cdots < \tau_{\ell-1} < b.$$

*Wir schreiben außerdem $\tau_0 = a$ und $\tau_\ell = b$. Weiters sei eine Folge $r$ von $\ell - 1$ ganzen Zahlen $r_i$ mit $0 \leq r_i \leq k$ gegeben.*

*Der Raum $\mathcal{P}_{k,\tau,r}$ bezeichne dann den Vektorraum der stückweise polynomialen Funktionen vom Grad $\leq k$ auf $[a,b]$, die $C^{r_i-1}$ an $\tau_i$ sind (für $r_i = 0$ sei keine Bedingung an $\tau_i$ gegeben).*

**Lemma 4.2.3.** *Es gilt*

$$\dim \mathcal{P}_{k,\tau,r} = (k+1)\ell - \sum_{i=1}^{l-1} r_i.$$

*Eine Basis für $\mathcal{P}_{k,\tau,r}$ ist gegeben durch die Funktionen*

$$p_{ij}(x) = (x-\tau_i)_+^j,$$

*eingeschränkt auf $[a,b]$ für $j = r_i, \dots, k$ und $i = 0, \dots, \ell-1$, wobei $(x-\tau_i)_+ := \sup(x-\tau_i, 0)$.*

Beweis. Der Raum ist ein Teilraum des Raums $\mathcal{P}_{k,\tau,(0,\dots,0)}$ aller Polynome, die jeweils auf den $l$ Intervallen $[\tau_i, \tau_{i+1}]$ definiert sind und Grad $\leq k$ haben. Eine Basis für diesen Raum ist gegeben durch die Funktionen

$$f_{ij}(x) := \begin{cases} (x-\tau_i)^j & x \in [\tau_i, \tau_{i+1}], \\ 0 & x \notin [\tau_i, \tau_{i+1}]. \end{cases},$$

für $i = 0, \dots, l-1$ und $j = 0, \dots, k$. Sei $f \in \mathcal{P}_{k,\tau,r}$, und sei $P^{(i)} = f|_{[\tau_i,\tau_{i+1}]}$. Dann ist $P^{(i)}$ ein Polynom und

$$P^{(i)}(x) = \sum_{j=0}^{k} a_j^{(i)} (x-\tau_i)^j.$$

Die Bedingung, daß $f$ an $\tau_i$ $C^{r_i}$ ist, übersetzt sich zu den $r_i$ linearen Gleichungen

$$m! a_m^{(i)} = \sum_{j=m}^{k} j(j-1)\dots(j-m) a_j^{(i-1)} \tau_i^{j-m}$$

für $m = 0, \dots, r_i - 1$. Diese Gleichungen sind offensichtlich unabhängig, was die behauptete Aussage bestätigt.

Die Funktionen $(x-\tau_i)_+$ sind $C^0$, und daher ist $(x-\tau_i)_+^j$ an $\tau_i$ $(j-1)$–mal stetig differenzierbar. Nachdem es $(k+1)\ell - \sum_j r_j$ viele solche Funktionen gibt und die $p_{ij}$ offensichtlich linear unabhängig sind, bilden sie eine Basis. □

Die oben genannte Basis ist leider unpraktisch, wenn es darum geht, Interpolationsprobleme zu lösen. Zu diesem Zweck führen wir eine praktischere Familie von Elementen in $\mathcal{P}_{k,\tau,r}$ ein, die B–Splines.

Sei $t = (t_j)$ eine (endliche) Folge reeller Zahlen, *Knoten* genannt, die $t_j \leq t_{j+1}$ erfüllen mögen für alle $j$. Wenn $r$ dieser $t_j$ gleich $\tau$ sind, so nennt man $\tau$ einen *Knoten(punkt) der Ordnung* $r$, bzw. einen *Knoten(punkt) der Multiplizität* $r$.

**Definition 4.2.4.** *Sei $t = (t_0, \dots, t_m)$ eine Folge wie oben. Für $x \in \mathbb{R}$ definieren wir die Funktionen $B_{i,k,t}$ ($i = 0, \dots, m-k-1$) rekursiv wie folgt:*

$$B_{i,0,t}(x) = \begin{cases} 1 & t_i \leq x < t_{i+1} \\ 0 & \textit{sonst}, \end{cases}$$

$$B_{i,k,t}(x) = \omega_{i,k}(x) B_{i,k-1}(x) + (1 - \omega_{i+1,k}(x)) B_{i+1,k-1}(x), \quad \textit{für } k > 0.$$

*Hier ist*

$$\omega_{i,j}(x) = \begin{cases} \frac{x-t_i}{t_{i+j}-t_i} & \textit{wenn } t_i < t_{i+j} \\ 0 & \textit{sonst.} \end{cases}$$

*Diese Funktionen werden B–Splines genannt, und wir werden den Index $t$ immer dann weglassen, wenn klar ist, von welcher Folge $t$ wir sprechen.*

Man kann übrigens auch B–Splines für unendliche Folgen $t$ definieren, nachdem die Definition für jedes $B_{i,k}$ nur endlich viele der $t_j$ verwendet. Der B–Spline $B_{i,k}$ hat Träger $[t_i, t_{i+k+1}]$, und zu seiner Definition werden nur die $t_j$ mit $1 \leq j \leq i+k+1$ verwendet. Aus der Definition kann man darüber hinaus noch folgendes ablesen: Man hat genau dann $B_{i,k} \equiv 0$ wenn für einen Index $i$ gilt, daß $t_i = t_{i+k+1}$.

**Proposition 4.2.5.** *Die B–Splines $B_{i,k}$ haben die folgenden Eigenschaften:*

(1) *$B_{i,k}$ ist ein stückweises Polynom vom Grad $k$.*
(2) *$B_{i,k}(x) = 0$ für $x \notin [t_i, t_{i+k+1}]$, das heißt* $\operatorname{supp} B_{i,k} = [t_i, t_{i+k+1}]$. *Jedes $B_{i,k}$ hat also kompakten Träger.*
(3) *$B_{i,k}(x) > 0$ für $x \in (t_i, t_{i+k+1})$; $B_{i,k}(t_i) = 0$ außer wenn $t_i = t_{i+1} = \cdots = t_{i+k} < t_{i+k+1}$, das heißt wenn $t_i$ ein Knoten der Ordnung $k$ ist, und dann gilt $B_{i,k}(t_i) = 1$.*
(4) *Sei $[a, b]$ ein Intervall mit $t_k \leq a$ und $b \leq t_{m-k}$. Dann gilt*

$$\sum_{i=0}^{m-k-1} B_{i,k}(x) = 1$$

*für alle $x \in [a, b]$. Das heißt die $B_{i,k}$ bilden eine* Partition der Eins *von $[a, b]$.*
(5) *Sei $x \in (t_i, t_{i+k+1})$. Dann gilt $B_{i,k}(x) = 1$ genau dann wenn $x = t_{i+1} = \cdots = t_{i+k}$.*
(6) *$B_{i,k}$ ist rechtsseitig stetig (sogar rechtsseitig unendlich oft differenzierbar) an allen $x \in \mathbb{R}$.*

Beweis. Die Eigenschaften 1, 2, 3, 4 und 6 sind klar für $k = 0$, also folgen 1, 2, 3 und 6 für alle $k$ aus einfachen Induktionsbeweisen. Ferner ist 5 eine Konsequenz aus 3 und 4. Es bleibt also noch 4 zu zeigen. Auch das wollen wir mit vollständiger Induktion angehen. Nachdem der Induktionsanfang bereits erwähnt wurde, fehlt nur noch der Induktionsschritt. Sei $x \in [a, b]$; dann gibt es ein $k \leq j \leq m-k-1$, sodaß $x \in [t_j, t_{j+1})$. Ist $x = t_j$ und gilt $B_{j,k}(x) = 1$, so folgt alles aus 3. In den anderen Fällen gilt

$$\sum_{i=0}^{m-k-1} B_{i,k}(x) = \sum_{i=j-k}^{j} B_{i,k}(x)$$

nach 2 und

$$\sum_{i=j-k}^{j} B_{i,k}(x) = \sum_{i=j-k}^{j} \omega_{i,k}(x) B_{i,k-1}(x) + \sum_{i=j-k}^{j} (1 - \omega_{i+1,k}(x)) B_{i+1,k-1}(x)$$

definitionsgemäß. Faßt man die Terme richtig zusammen, erhält man

$$\sum_{i=j-k}^{j} B_{i,k}(x) = \omega_{j-k,k}(x) B_{j-k,k-1}(x) + \sum_{i=j+1-k}^{j} B_{i,k-1}(x) + (1 - \omega_{j+1,k}(x)) B_{j+1,k-1}(x).$$

Aus 2 folgen $B_{j-k,k-1}(x) = 0$ und $B_{j+1,k-1}(x) = 0$, weil $x \in [t_j, t_{j+1})$. Aus der Induktionsannahme folgt schließlich

$$\sum_{i=j+1-k}^{j} B_{i,k-1}(x) = \sum_{i=0}^{m-(k-1)-1} B_{i,k-1}(x) = 1,$$

was die Formel 4 beweist. □

Falls wir das Intervall $[a, b]$ betrachten und $t_{m-k} = \cdots = t_m = b$ gilt, dann ist die Formel aus Proposition 4.2.5.4 nur richtig für $x \in [a, b)$. Wir definieren dann $B_{m-k-1,k}(b) = 1$ um, was den B–Spline $B_{m-k-1,k}$ linksseitig stetig bei $b$ und die Formel 4.2.5.4 gültig macht. Dies ist allerdings nur dann notwendig, wenn ein Knoten der Ordnung $k+1$ an $b$ existiert.

**Beispiel 4.2.6.** *Die B–Splines in diesem Beispiel basieren auf ganzzahligen Knoten $t_j \in \mathbb{Z}$. Alle Knoten sind einfach, außer wenn explizit anders angegeben.*

*Für $k = 1$ finden wir typische B–Splines in Abbildung 5.7, für $k = 2$ in den Abbildungen 5.8 und 5.9.*

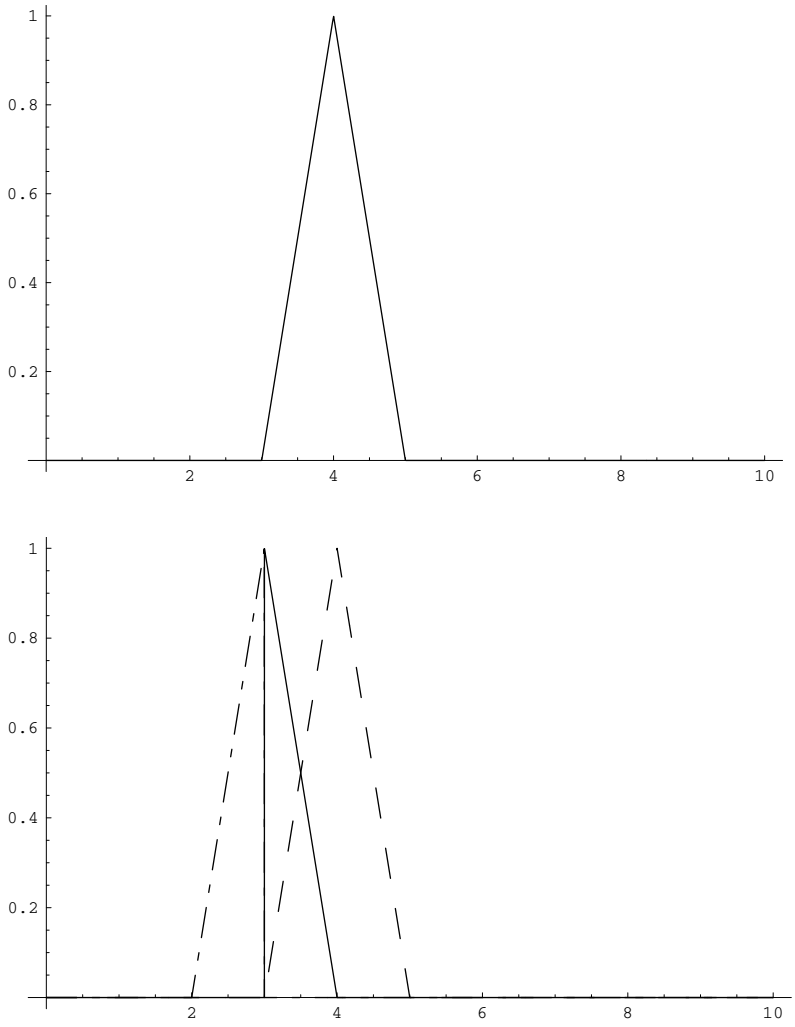



ABBILDUNG 5.7. $B_{i,1}$, oben für einfache Knoten, unten für den doppelten Knoten 3

Kehren wir nun zurück zum Raum $\mathcal{P}_{k,\tau,r}$ für das Intervall $[a, b]$. Sei $t = (t_0, \ldots, t_m)$ eine Folge von Punkten mit $m = k + n = k + \ell(k + 1) - \sum_j r_j$, die die folgenden Bedingungen erfüllt:

(1) $t_i \leq t_{i+1}$,
(2) $t_i \leq a$ für $0 \leq i \leq k$,
(3) $t_i \geq b$ für $n \leq j \leq n + k$,
(4) jeweils $k + 1 - r_i$ der $t_j$ stimmen mit $\tau_i$ überein für $1 \leq i \leq \ell - 1$.

Die Knoten $t_i$ für $i = k+1, \ldots, n-1$ sind eindeutig bestimmt durch die obigen Eigenschaften. Die übrigen $t_i$ müssen nur $\leq a$ bzw. $\geq b$ erfüllen.

**Proposition 4.2.7.** *Ist $t$ wie oben gegeben, dann ist $\mathcal{P}_{k,\tau,r}$ der Raum der stückweisen Polynome vom Grad $\leq k$, die $C^{k-p_j}$ sind an $t_j$, falls $t_j$ ein Knoten der Ordnung $p_j$ ist. Falls $k < p_j$ ist, sei keine Bedingung bei $t_j$ gestellt.*

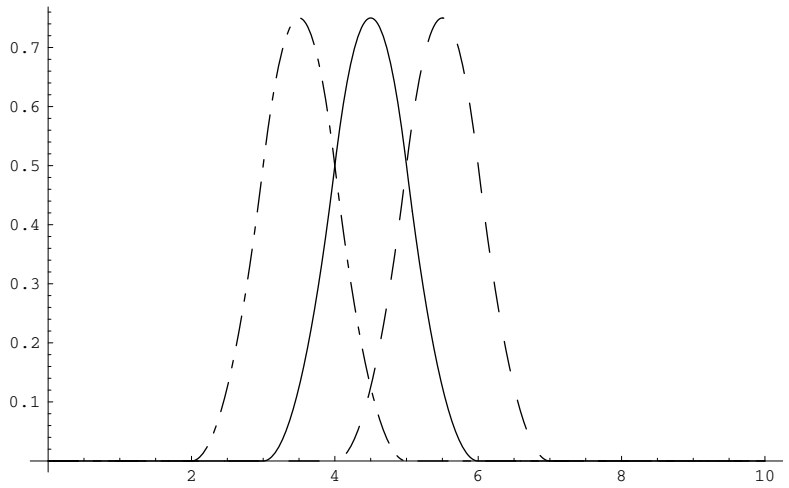



ABBILDUNG 5.8. $B_{i,2}$ für einfache Knoten

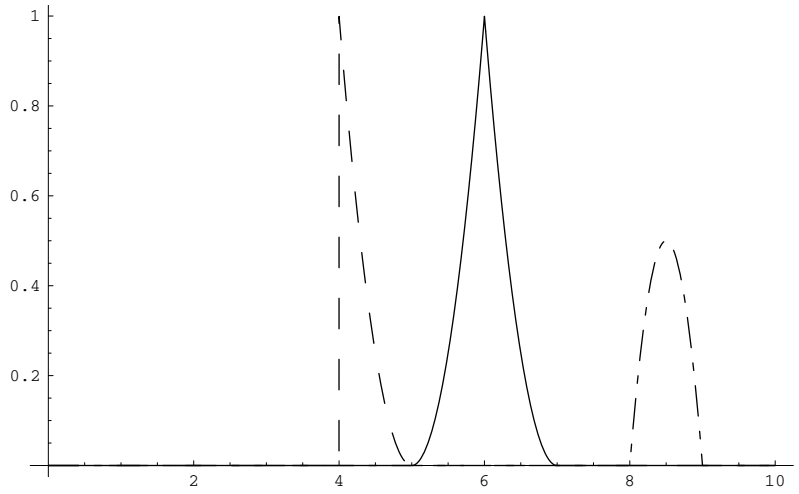



ABBILDUNG 5.9. $B_{i,2}$ für mehrfache Knoten: 4 dreifach, 6, 7 und 9 doppelt.

BEWEIS. Die Behauptung folgt aus den Definitionen von $\mathcal{P}_{k,\tau,r}$ und $t$. □

Aus dem letzten Resultat folgt das für die Verwendbarkeit der B–Splines wichtige

**Theorem 4.2.8.** *Wenn alle Knoten $t_j$ Vielfachheit $\leq k+1$ haben, wobei die $t_i$ wie zuvor definiert seien, dann bilden die B–Splines $B_{i,k,t}$ für $i = 0, \ldots, n-1$ eine Basis für $\mathcal{P}_{k,\tau,r}$. Diese Basis heißt auch die* B–Spline Basis von $\mathcal{P}_{k,\tau,r}$.

BEWEIS. Der Beweis erfolgt in zwei Schritten. Der erste ist unabhängig von der Vielfachheit der Knoten. Der zweite schließlich beweist die Basiseigenschaft.
**Schritt 1:** Ohne Einschränkung der Vielfachheit der Knoten gilt für alle $t \in [a, b]$ die Beziehung

$$(x-t)^k = \sum_{i=0}^{n-1} \psi_{i,k}(t) B_{i,k}(x), \tag{48}$$

wobei wir $\psi_{i,k}(t) = (t_{i+1} - t)(t_{i+2} - t) \ldots (t_{i+k} - t)$ setzen, bzw. $\psi_{i,0}(t) = 1$.

Diese Gleichung beweisen wir mittels Induktion nach $k$. Für $k = 0$ ist sie offensichtlich erfüllt. Die Induktionsannahme ist

$$(x-t)^{k-1} = \sum_{i=0}^{n-1} \psi_{i,k-1}(t) B_{i,k-1}(x) = \sum_{i=1}^{n-1} \psi_{i,k-1}(t) B_{i,k-1}(x),$$

weil $B_{0,k-1}(x) = 0$ für $x \in [a,b]$, da $k+1$ Knoten kleiner oder gleich $a$ sind. Nun verwenden wir die rekursive Definition von $B_{i,k}$:

$$\begin{aligned}\sum_{i=0}^{n-1} \psi_{i,k}(t) B_{i,k}(x) &= \sum_{i=0}^{n-1} \psi_{i,k}(t)\big(\omega_{i,k}(x) B_{i,k-1}(x) + (1-\omega_{i+1,k}(x)) B_{i+1,k-1}(x)\big) = \\ &= \psi_{0,k}(t)\omega_{0,k}(x) B_{0,k-1}(x) + \sum_{i=1}^{n-1} B_{i,k-1}(x)\big(\psi_{i,k}(t)\omega_{i,k}(x) + \\ &\quad + \psi_{i-1,k}(t)(1-\omega_{i,k}(x))\big).\end{aligned}$$

Für $t_i = t_{i+k}$ verschwindet $B_{i,k-1}$ identisch, und für $t_i < t_{i+k}$ erhalten wir

$$\begin{aligned}\psi_{i,k}(t)\omega_{i,k}(x) + \psi_{i-1,k}(t)(1-\omega_{i,k}(x)) &= \psi_{i,k-1}(t)\big((t_{i+k}-t)\omega_{i,k}(x) + (t_i - t)(1-\omega_{i,k}(x))\big) = \\ &= \psi_{i,k-1}(t)\big(\omega_{i,k}(x)(t_{i+k}-t_i) + t_i - t\big) = \\ &= \psi_{i,k-1}(t)(x-t),\end{aligned}$$

wobei wir mehrfach die Definitionen von $\omega_{i,k}$ und $\psi_{i,k}$ verwendet haben. Aus den letzten beiden Beziehungen folgt schließlich unter nochmaliger Berücksichtigung von $B_{0,k-1}(x) = 0$ und der Induktionsannahme

$$\sum_{i=0}^{n-1} \psi_{i,k}(t) B_{i,k}(x) = (x-t) \sum_{i=0}^{n-1} \psi_{i,k-1}(t) B_{i,k-1}(x) = (x-t)(x-t)^{k-1} = (x-t)^k.$$

Aus (48) folgt unmittelbar die weitere Gleichung

$$(x-t_j)_+^k = \sum_{i \geq j} \psi_{i,k}(t_j) B_{i,k}(x), \quad j = 0, \ldots, n-1, \tag{49}$$

da für $x < t_j$ beide Seiten verschwinden; die linke nach Definition und die rechte, da $B_{i,k}(x) = 0$ für alle $i$. Für $x > t_j$ argumentiert man folgendermaßen: Für $i < j$ und $i \geq j-k-1$ ist $\psi_{i,k}(t_j) = 0$, und für $i < j-k-1$ verschwindet $B_{i,k}(x)$. Daher stimmt die rechte Seite mit $\sum_{i=0}^{n-1} \psi_{i,k}(t_j) B_{i,k}(x)$ überein, und die behauptete Gleichung folgt aus (48).

**Schritt 2:** Nachdem wir bereits eine Basis des Raumes $\mathcal{P}_{k,\tau,r}$ kennen, reicht es zu zeigen, daß sich die Basiselemente durch die B–Splines audrücken lassen. Stimmt dann auch noch die Anzahl der B–Splines, oder sind sie linear unabhängig, dann haben wir bewiesen, daß sie eine Basis bilden.

Beginnen wir mit der bekannten Basis

$$\begin{aligned}&(x-a)^s \quad s = 0, \ldots, k \\ &(x-\tau_i)_+^s \quad i = 1, \ldots, \ell-1, \quad s = r_i, \ldots, k.\end{aligned}$$

Aus Schritt 1 folgt, daß $(x-t)^k$ dargestellt werden kann für beliebiges $t$. Differenzieren wir Gleichung (48) nach $t$, so folgern wir auch sofort, daß $(x-t)^s$ für $0 \leq s \leq k$ dargestellt werden kann, daher auch die $(x-a)^s$ für $s = 0, \ldots, k$. Aus Gleichung (49) können wir weiters schließen, daß sich $(x-t_j)_+^k$ ebenfalls durch die $B_{i,k}$ ausdrücken läßt. Für $(x-t_j)_+^s$ für $s < k$

müssen wir noch ein klein wenig arbeiten. Wir haben für $v < k$

$$\frac{1}{(k-v)!}(x-t_j)^{k-v} = \frac{1}{k!}\sum_{i=0}^{n-1}(-1)^v \left.\frac{d^v}{dt^v}\right|_{t_j} \psi_{i,k}(t)B_{i,k}(x).$$

Sind $k+1-r$ Knoten $t_i$ gleich $t_j$, etwa $t_j = t_{j+1} = \cdots = t_{j+k-r}$. Dann hat $\psi_{i,k}$ eine $(k+1-r)$–fache Nullstelle an $t_j$, und daher folgt auch

$$\left.\frac{d^v}{dt^v}\right|_{t_j} \psi_{i,k}(t) = 0, \quad \text{für } v = 0,\dots,k-r \text{ und } j-k \le i \le j-1.$$

Nachdem $\operatorname{supp} B_{i,k} = [t_i, t_{i+k+1}]$ haben wir überdies

$$\sum_{i=0}^{j-1}(-1)^v \frac{d^v}{dt^v}\psi_{i,k}(t_j)B_{i,k}(x) = 0, \quad \text{für } x \ge t_j \text{ und } 0 \le v \le k-r.$$

Fassen wir all das zusammen, so erhalten wir

$$(x-t_j)_+^{k-v} = \frac{(k-v)!}{k!}\sum_{i\ge j} -\frac{d^v}{dt^v}\psi_{i,k}(t_j)B_{i,k}(x),$$

was beweist, daß auch $(x-t_j)_+^s$ für $s = r_j,\dots,k$ darstellbar ist. Daß die $B_{i,k}$ linear unabhängig sind, folgt letztlich aus deren Trägereigenschaften. □

Bevor wir uns um das Interpolationsproblem kümmern stellen, wir zuerst noch einige Algorithmen zusammen, die den Umgang mit B–Splines erleichtern.

**4.2.1. Auswertung.** Abgesehen von der offensichtlichen Idee, in jedem Intervall $[t_i, t_{i+1}]$ die Summe

$$f(x) = \sum_{i=0}^{n-1} a_i B_{i,k}(x)$$

auszurechnen und damit die Polynomdarstellung von $f$ in diesem Intervall zu bestimmen, dann das Horner Schema zu verwenden, kann man die obige Summe auch noch direkt mit Hilfe der rekursiven Definition der $B_{i,k}$ auswerten.

**Proposition 4.2.9.** *Es gilt*

$$f(x) = \sum_i a_i B_{i,k}(x) = \sum_i a_i^{(1)} B_{i,k-1}(x) = \cdots = \sum_i a_i^{(k)} B_{i,0}(x)$$

*mit*

$$a_i^{(j+1)} = \omega_{i,k-j}(x)a_i^{(j)} + (1-\omega_{i,k-j}(x))a_{i-1}^{(j)}.$$

*Dabei verwenden wir für $x < t_k$ die Konvention $B_{i,k} \equiv 0$ für $i \le 0$.*

Beweis. Folgt aus der rekursiven Definition der $B_{i,k}$. □

So kann man übrigens auch die Polynomdarstellung berechnen, indem man $a_i^{(j)}$ explizit von $x$ abhängig macht.

Der Aufwand einer Auswertung benötigt die Berechnung von $\frac{k(k+1)}{2}$ Konvexkombinationen, die jeweils zwei Multiplikationen, eine Division und eine Subtraktion benötigen. Der Gesamtaufwand ist also $2k(k+1)$ elementare Operationen. Dieser Aufwand mag etwas hoch erscheinen, doch er hat die Vorteile, daß er gutartig ist und daß man ihn sehr gut auf Splinekurven verallgemeinern kann.

**4.2.2. Einfügung eines Knoten.** Sei $f(x)$ eine Splinefunktion wie zuvor, definiert mit einer Knotenfolge $t = (t_0, \ldots, t_{n+k})$. Wenn wir einen weiteren Knoten $\hat{t}$ so einfügen, daß $\hat{t} \leq t_{n-1}$, dann können wir die Funktion $f$ nach folgendem Resultat in die $B_{i,k,\tilde{t}}$ für die neu entstandene Folge $\tilde{t}$ entwickeln:

**Proposition 4.2.10.** *Es gilt*

$$\sum_{i=0}^{n-1} a_i B_{i,k,t}(x) = \sum_{i=0}^{n} \hat{a}_i B_{i,k,\tilde{t}}(x)$$

*mit*

$$\hat{a}_i = \begin{cases} a_i & \text{wenn } t_{i+k} \leq \hat{t} \\ \omega_{i,k}(\hat{t})a_i + (1-\omega_{i,k}(\hat{t}))a_{i-1} & \text{wenn } t_i < \hat{t} < t_{i+k} \\ a_{i-1} & \text{wenn } \hat{t} \leq t_i. \end{cases}$$

Beweis. Der Beweis erfolgt durch Induktion nach $k$. Wie beim Auswertungsalgorithmus in 4.2.1 erhalten wir unter Zuhilfenahme der Induktionsannahme

$$\sum_{i=0}^{n-1} a_i B_{i,k,t}(x) = \sum_{i=0}^{n-1} a_i^{(1)} B_{i,k-1,t}(x) = \sum_{i=0}^{n} (a_i^{(1)})\check{} B_{i,k-1,\tilde{t}}(x).$$

Eine erneute Anwendung der Definition der B–Splines führt dann zu

$$\sum_{i=0}^{n} \hat{a}_i B_{i,k,\tilde{t}}(x) = \sum_{i=0}^{n} (\hat{a}_i)^{(1)} B_{i,k-1,\tilde{t}}(x). \tag{50}$$

Vergleicht man die beiden Resultate, so sieht man sofort, daß es genügt, die Gleichung $(a_i^{(1)})\check{} = (\hat{a}_i)^{(1)}$ zu zeigen. Für $k = 0$ ist das klar nach Definition.

Aus 4.2.1 wissen wir, daß

$$a_i^{(1)} = a_i \frac{x - t_i}{t_{i+k} - t_i} + a_{i-1} \frac{t_{i+k} - x}{t_{i+k} - t_i}$$

gilt und außerdem wegen der Induktionsannahme

$$\sum_{i=0}^{n-1} a_i^{(1)} B_{i,k-1,t}(x) = \sum_{i=0}^{n} (a_i^{(1)})\check{} B_{i,k-1,\tilde{t}}(x)$$

$$(a_i^{(1)})\check{} = \begin{cases} a_i^{(1)} & \text{wenn } t_{i+k-1} \leq \hat{t} \\ a_i^{(1)} \frac{\hat{t}-t_i}{t_{i+k-1}-t_i} + a_{i-1}^{(1)} \frac{t_{i+k-1}-\hat{t}}{t_{i+k-1}-t_i} & \text{wenn } t_i < \hat{t} < t_{i+k-1} \\ a_{i-1}^{(1)} & \text{wenn } \hat{t} \leq t_i \end{cases}$$

erfüllt sind. Der komplizierteste Fall liegt vor, wenn $t_i < \hat{t} < t_{i+k-1}$. Dann gilt

$$\begin{aligned}(a_i^{(1)})\check{} = a_i \frac{\hat{t} - t_i}{t_{i+k-1} - t_i} \frac{x - t_i}{t_{i+k} - t_i} + a_{i-1} \frac{\hat{t} - t_i}{t_{i+k-1} - t_i} \frac{t_{i+k} - x}{t_{i+k} - t_i} + \\ + a_{i-1} \frac{t_{i+k-1} - \hat{t}}{t_{i+k-1} - t_i} \frac{x - t_{i-1}}{t_{i+k-1} - t_{i-1}} + a_{i-2} \frac{t_{i+k-1} - \hat{t}}{t_{i+k-1} - t_i} \frac{t_{i+k-1} - x}{t_{i+k-1} - t_{i-1}}.\end{aligned}$$

Wenn wir nun die rechte Seite von (50) unter Verwendung der obigen Formeln umrechnen, so erhalten wir

$$\hat{a}_i = a_i \frac{\hat{t} - t_i}{t_{i+k} - t_i} + a_{i-1} \frac{t_{i+k} - \hat{t}}{t_{i+k} - t_i}$$

und

$$(\hat{a}_i)^{(1)} = \hat{a}_i \frac{x - t_i}{t_{i+k-1} - t_i} + \hat{a}_{i-1} \frac{t_{i+k-1} - x}{t_{i+k-1} - t_i}. \tag{51}$$

Die letzte Formel gilt wegen der Einfügung des neuen Knoten $\hat{t}$ zwischen $t_i$ und $t_{i+k-1}$ in der Folge $\tilde{t}$; aus diesem Grund gilt auch $\tilde{t}_{i+k} = t_{i+k-1}$. Aus den letzten beiden Beziehungen folgt

$$\begin{aligned}(\hat{a}_i)^{(1)} = a_i \frac{x - t_i}{t_{i+k-1} - t_i} \frac{\hat{t} - t_i}{t_{i+k} - t_i} + a_{i-1} \frac{x - t_i}{t_{i+k-1} - t_i} \frac{t_{i+k} - \hat{t}}{t_{i+k} - t_i} + \\ + a_{i-1} \frac{t_{i+k-1} - x}{t_{i+k-1} - t_i} \frac{\hat{t} - t_{i-1}}{t_{i+k-1} - t_{i-1}} + a_{i-2} \frac{t_{i+k-1} - x}{t_{i+k-1} - t_i} \frac{t_{i+k-1} - \hat{t}}{t_{i+k-1} - t_{i-1}},\end{aligned}$$

und ein Vergleich mit (51) zeigt sofort die Gleichung $(a_i^{(1)})\hat{} = (\hat{a}_i)^{(1)}$. Falls $\hat{t} \leq t_i$ oder $\hat{t} \geq t_{i+k-1}$ gilt, folgt die Gleichung offensichtlich aus den Definitionen. □

**4.2.3. Interpolation.** Mit Hilfe der B–Spline Basis können wir nun endlich die Interpolationsaufgabe attackieren. Seien wieder Stützstellen $(x_i, f_i)$ gegeben mit paarweise verschiedenen $x_i$ für $i = 0, \ldots, n-1$, und sei $t = (t_1, \ldots, t_{n+k})$ irgendeine Folge, die B–Splines $B_{i,k,t}$ definiert. Sei $\Phi(x, a_0, \ldots, a_{n-1}) = \sum_{i=0}^{n-1} a_i B_{i,k}(x)$ die Funktionenklasse, die wir zur Lösung der Interpolationsaufgabe heranziehen wollen.

Die Interpolationsbedingungen

$$f(x_i) = \sum_{i=0}^{n-1} a_i B_{i,k}(x_i) = f_i$$

führen auf ein lineares Gleichungssystem der Form

$$\begin{pmatrix} B_{0,k}(x_0) & \cdots & B_{n-1,k}(x_0) \\ \vdots & \ddots & \vdots \\ B_{0,k}(x_{n-1}) & \cdots & B_{n-1,k}(x_{n-1}) \end{pmatrix} \begin{pmatrix} a_0 \\ \vdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} f_0 \\ \vdots \\ f_{n-1} \end{pmatrix}.$$

Im Gegensatz zu den anderen Interpolationsproblemen (Polynome, rationale Funktionen, trigonometrische Funktionen) ist das Gleichungssystem hier die korrekte Methode, die Koeffizienten zu bestimmen. Der Grund dafür liegt in der Trägereigenschaft der B–Splines begründet. Wir wissen aus Proposition 4.2.5.2, daß $\operatorname{supp} B_{i,k} = [t_i, t_{i+k}]$ gilt. Aus diesem Grund sind höchstens $k$ von den $n$ Werten $B_{i,k}(x_j)$ ungleich Null. Die Koeffizientenmatrix des Gleichungssystems, im folgenden $A$ genannt, ist also eine Bandmatrix mit Bandbreite $k$. Die Interpolationsaufgabe ist offenbar genau dann lösbar, wenn die Koeffizientenmatrix $A$ regulär ist. Dafür haben [**Schoenberg, Whitney 1953**] ein Kriterium angegeben:

**Theorem 4.2.11.** *Die Matrix $A = (B_{i,k}(x_j))_{i,j}$ ist genau dann regulär, wenn die Diagonalelemente von $A$ nicht verschwinden, also $B_{i,k}(x_i) \neq 0$ für alle $i$ gilt.*

Außerdem ist die Matrix $A$ total positiv, das heißt alle $r \times r$ Hauptuntermatrizen haben nichtnegative Determinante. Ist $A$ regulär, dann kann man $LR$–Zerlegung ohne Pivotsuche verwenden, um das Gleichungssystem stabil zu lösen. Es entstehen also in $R$ keine zusätzlichen Bänder, und der Aufwand zur Lösung des Systems ist nur $O(k^2 n)$. Nachdem $k$ meist viel kleiner als $n$ und fix gewählt wird, ist der Aufwand zur Lösung des Interpolationsproblemes für Splinefunktionen linear.

Meist wird die Folge $t$ so gewählt, daß alle der $t_i$ mit irgendeinem der $x_j$ übereinstimmen. Dann lassen sich die $B_{i,k}(x_j)$ gut berechnen, und die Matrix $A$ ist regulär sofern die Vielfachheiten der Knoten nicht größer als die Ordnung der B–Splines sind.

**4.3. Kubische Splines.** Spezialfälle der Räume $\mathcal{P}_{k,\tau,r}$ sind die Funktionenklassen $\mathcal{S}_{k,\tau}$, die in der nächsten Definition eingeführt werden. Traditionell werden sie die Räume der Splines vom Grad $k$ mit Knoten $\tau_j$ genannt. Ursprünglich wurde die Klasse der Splines auf genau diese Räume eingeschränkt.

**Definition 4.3.1.** *Sei $\mathcal{S}_{k,\tau}$ der Raum $\mathcal{P}_{k,\tau,r}$ für die spezielle Folge $r = (k, \ldots, k)$. Die Funktionen in $\mathcal{S}_{k,\tau}$ sind definitionsgemäß $C^{k-1}$ und werden üblicherweise mit* Splines vom Grad $k$ *bezeichnet.*

Die am häufigsten verwendeten Splines sind die Splines vom Grad 3, auch *kubische Splines* genannt. Mit ihnen wollen wir uns in diesem Abschnitt beschäftigen. Man könnte sie zwar als B–Splines wie zuvor behandeln, doch ihre spezielle Struktur erlaubt eine etwas direktere, leichter verständliche Vorgehensweise.

Seien wieder Stützstellen $(x_i, f_i)$ für $i = 0, \ldots, n$ gegeben. Die Stützknoten wählt man wieder so, daß $t_i = x_j$ für ein $i$ und daß alle Knoten einfach sind. Man könnte dann wie zuvor vorgehen und die Koeffizienten für die B–Spline Basis wie in 4.2.3 berechnen und den Spline mit Hilfe des Auswertungsalgorithmus aus 4.2.1 auswerten. Die klassische Methode ist jedoch anders. Man kann nämlich in diesem Fall durch direkte Berechnungen mit gleichem Aufwand sofort die Polynomdarstellung des gesuchten Splines in den Teilintervallen bestimmen:

Sei $S(x)$ der gesuchte kubische Spline. In den Intervallen $[x_i, x_{i+1}]$ ist er dann durch einen polynomialen Ausdruck der Form

$$s_i(x) := a_i(x - x_i)^3 + b_i(x - x_i)^2 + c_i(x - x_i) + d_i$$

gegeben. An den Stützstellen ist der Spline nach Definition $C^2$, und daher erhalten wir dort 3 Übergangsbedingungen (für den Funktionswert, die erste und die zweite Ableitung). Um ein günstiges Gleichungssystem zu erhalten, ist es lohnenswert, die Koeffizienten der polynomialen Ausdrücke durch die Funktionswerte und die (noch unbekannten) zweiten Ableitungen von $S$ an den Stützstellen auszudrücken. Sei $f_i'' := S''(x_i)$ und $h_i = x_i - x_{i-1}$. Dann gilt wegen der Übergangsbedingungen für die linke und die rechte Intervallgrenze

$$\begin{array}{lcrcrcrcl}
s_i(x_i) & = & & & & & d_i & = & f_i \\
s_i(x_{i+1}) & = & a_i h_i^3 & + & b_i h_i^2 & + \; c_i h_i & + \; d_i & = & f_{i+1} \\
s_i'(x_i) & = & & & & c_i & & & \\
s_i'(x_{i+1}) & = & 3a_i h_i^2 & + & 2b_i h_i & + \quad c_i & & & \\
s_i''(x_i) & = & & & 2b_i & & & = & f_i'' \\
s_i''(x_{i+1}) & = & 6a_i h_i & + & 2b_i & & & = & f_{i+1}''.
\end{array}$$

Löst man die Gleichungen auf, so kann man alle Koeffizienten aus den folgenden Beziehungen bestimmen:

$$\begin{aligned}
a_i &= \tfrac{1}{6h_i}\big(f_{i+1}'' - f_i''\big), \\
b_i &= \tfrac{1}{2} f_i'', \\
c_i &= \tfrac{1}{h_i}\big(f_{i+1} - f_i\big) - \tfrac{1}{6} h_i \big(f_{i+1}'' + 2f_i''\big), \\
d_i &= f_i.
\end{aligned}$$

Somit genügt es, die sogenannten Momente $f_i''$ zu berechnen. Dies geschieht mit Hilfe der Übereinstimmung der ersten Ableitungen an den Stützstellen. Es gelten

$$\begin{aligned}
s_i'(x_i) &= \tfrac{1}{h_i}\big(f_{i+1} - f_i\big) - \tfrac{1}{6} h_i \big(f_{i+1}'' + 2f_i''\big), \\
s_i'(x_{i+1}) &= \tfrac{1}{h_i}\big(f_{i+1} - f_i\big) + \tfrac{1}{6} h_i \big(2f_{i+1}'' + f_i''\big),
\end{aligned}$$

und wegen $s_i'(x_i) = s_{i-1}'(x_i)$ erhalten wir die Gleichung

$$h_{i-1} f_{i-1}'' + 2(h_{i-1} + h_i) f_i'' + h_i f_{i+1}'' - \tfrac{6}{h_i}(f_{i+1} - f_i) + \tfrac{6}{h_{i-1}}(f_i - f_{i-1}) = 0.$$

Diese Beziehungen für alle $i$ erzeugen ein tridiagonales Gleichungssystem, wobei wir bislang unter den Teppich gekehrt haben, daß bei $x_0$ und $x_{n-1}$ keine Übergangsbedingungen vorgegeben sind. Eine einfache Abzählung von Gleichungen und Unbekannten ergibt einen

Mangel von zwei Bedingungen. Welche Bedingungen man vorschreibt beeinflußt die Art der berechneten Splines. Die folgenden drei Bedingungstypen sind üblich:

**Natürliche Splines:** $S''(x_0) = S''(x_n) = 0$,

**Periodische Splines:** $S'(x_0) = S'(x_n)$ und $S''(x_0) = S''(x_n)$. Weiters muß natürlich $f_0 = f_n$ gelten.

**Splines mit vorgegebenen Ableitungen:** $S'(x_0) = f'_0$ und $S'(x_n) = f'_n$ mit vorgegebenen Werten $f'_0$ und $f'_n$.

Der Großteil der Gleichungen wird von diesen Bedingungen nicht beeinflußt. Im allgemeinen sieht das aufzulösende Gleichungssystem folgendermaßen aus:

$$\begin{pmatrix} 2 & \lambda_0 & & & \xi_0 \\ \mu_1 & 2 & \lambda_1 & & \\ & \mu_2 & \ddots & \ddots & \\ & & \ddots & 2 & \lambda_{k-1} \\ \xi_n & & & \mu_k & 2 \end{pmatrix} \begin{pmatrix} f''_0 \\ f''_1 \\ f''_2 \\ \vdots \\ f''_{k-1} \\ f''_k \end{pmatrix} = \begin{pmatrix} \eta_0 \\ \eta_1 \\ \eta_2 \\ \vdots \\ \eta_{k-1} \\ \eta_k \end{pmatrix}.$$

Nur $k$ und die $\xi_i$ sind von den zusätzlichen Bedingungen abhängig. Nachdem im Fall der periodischen Splines $f_0 = f_n$ gelten muß ist dort $k = n - 1$; sonst gilt $k = n$. In der Koeffizientenmatrix und im Konstantenvektor werden dabei die folgenden Abkürzungen verwendet:

$$\begin{aligned} \lambda_j &= \frac{h_{j+1}}{h_j + h_{j+1}} \\ \mu_j &= \frac{h_j}{h_j + h_{j+1}} \\ \eta_j &= \frac{6}{h_j + h_{j+1}} \left( \frac{f_{j+1} - f_j}{h_{j+1}} - \frac{f_j - f_{j-1}}{h_j} \right) \end{aligned}$$

für $j = 1, \ldots, n-1$. Die übrigen Konstanten sind abhängig von den zusätzlichen Bedingungen:

**Natürliche Splines:**

$$\begin{aligned} \lambda_0 = 0, \quad \eta_0 = 0, \quad \xi_0 = 0, \\ \mu_n = 0, \quad \eta_n = 0, \quad \xi_n = 0. \end{aligned}$$

**Periodische Splines:**

$$\begin{aligned} \lambda_0 = \tfrac{h_n}{h_n + h_{n-1}}, \qquad & \xi_0 = \tfrac{h_{n-1}}{h_n + h_{n-1}}, \\ \xi_{n-1} = \lambda_{n-1}, \quad \eta_0 = \tfrac{6}{h_n + h_{n-1}} & \left( \tfrac{f_1 - f_0}{h_1} - \tfrac{f_0 - f_{n-1}}{h_n} \right). \end{aligned}$$

**vorgegebene Ableitungen:**

$$\begin{aligned} \lambda_0 = 1, \quad \eta_0 = \tfrac{6}{h_1} \left( \tfrac{f_1 - f_0}{h_1} - f'_0 \right), \quad \mu_n = 1, \\ \xi_0 = 0, \quad \eta_n = \tfrac{6}{h_n} \left( f'_n - \tfrac{f_n - f_{n-1}}{h_n} \right), \quad \xi_n = 0. \end{aligned}$$

Außer im periodischen Fall ist das entstehende Gleichungssystem tridiagonal und diagonal dominant. Es läßt sich also durch $LR$–Zerlegung ohne Pivotsuche stabil lösen. Der Aufwand dafür beträgt $\sim 12n$ nach Kapitel 3, ist also wie erwartet linear in der Anzahl der Stützstellen. Auch im periodischen Fall läßt sich das Gleichungssystem leicht lösen, selbst wenn es nicht genau Tridiagonalgestalt hat. Nur zwei Gleichungen weichen von dieser Form ab, und der Aufwand zur Lösung ist wieder linear in $n$.

Eine Zusatzeigenschaft macht kubische Splines besonders interessant. Im Fall der natürlichen Splines gilt nämlich die folgende Minimalitätseigenschaft:

**Theorem 4.3.2.** *Für alle Funktionen $f \in C^2[x_0, x_n]$, die die Interpolationsbedingungen $f(x_i) = f_i$ erfüllen, ist der interpolierende natürliche kubische Spline diejenige Funktion, die die Seminorm*

$$\|f\| := \int_{x_0}^{x_n} |f''(x)|^2 \, dx$$

*minimiert. Der kubische Spline ist also jene interpolierende zweimal stetig differenzierbare Funktion, für die die Gesamtkrümmung minimal ist.*

Eine ähnliche Eigenschaft haben die periodischen Splines (für alle periodischen zweimal stetig differenzierbaren Funktionen) und die Splines mit vorgegebenen Ableitungen am Rand (für alle solchen $C^2$ Funktionen).

**4.4. Andere Splines.** Obwohl die stückweise aus Polynomen zusammengesetzten Splines mit Abstand am häufigsten verwendet werden, gibt es doch für spezielle Interpolationsaufgaben die Möglichkeit, andere Funktionenstücke aneinanderzureihen.

**4.4.1. Exponentialsplines.** Eine Verallgemeinerung der kubischen Splines erhält man, wenn man als Interpolationsfunktion $f$ diejenige $C^2$–Funktion wählt, die die Interpolationsbedingungen

$$f(x_i) = f_i, \quad i = 0, \ldots, n$$

erfüllt, vorgegebene Ableitung am Rand

$$f'(x_j) = f'_j, \quad i = 0, n$$

hat und zu gegebenen $\lambda_i$ das Funktional

$$\sum_{i=0}^{n-1} \int_{x_i}^{x_{i+1}} (f''(x))^2 + \lambda_i (f'(x))^2 \, dx$$

minimiert. Falls alle $\lambda_i = 0$ sind, ist die Interpolationsfunktion genau der kubische Spline.

Andernfalls hat die Funktion $f$ in jedem Teilintervall $[x_i, x_{i+1}]$ die Darstellung

$$a_0^{(i)} + a_1^{(i)}(x - x_i) + a_2^{(i)} \varphi(x - x_i, \lambda_i) + a_3^{(i)} \psi(x - x_i, \lambda_i)$$

mit den Funktionen

$$\varphi(x, \lambda) = \frac{2}{\lambda^2}\big(\cosh(\lambda x) - 1\big) = \frac{e^{\lambda x} + e^{-\lambda x} - 2}{\lambda^2}$$
$$\psi(x, \lambda) = \frac{6}{\lambda^3}\big(\sinh(\lambda x) - \lambda x\big) = \frac{3(e^{\lambda x} - e^{-\lambda x} - 2\lambda x)}{\lambda^3};$$

deshalb der Name *Exponentialsplines*.

**4.4.2. $\nu$–Splines.** In [**Nielson 1974**] wurde eine Klasse von Interpolationsfunktionen vorgestellt, die aus stückweisen kubischen Polynomen entstehen und in manchen Anwendungen günstigere Eigenschaften als kubische Splines und Exponentialsplines besitzen. Leider sind diese Funktionen nicht überall $C^2$. An den Stützstellen springen die zweiten Ableitungen manchmal. Allerdings haben $\nu$–Splines auch eine Minimalitätseigenschaft. Im Funktional, das die Exponentialsplines minimieren, wird der zweite Term diskretisiert. Das entstehende Funktional lautet dann

$$\int_{x_0}^{x_n} \big(f''(x)\big)^2 + \sum_{i=0}^{n} \nu_i \big(f'(x_i)\big)^2.$$

Man kann zeigen, daß die dadurch definierten Funktionen die Gestalt

$$s(x) = a^{(0)} + a^{(1)}x + \sum_{i=0}^{n-1} a_i^{(2)}(x - x_i)_+^2 + \sum_{i=0}^{n-1} a_i^{(3)}(x - x_i)_+^3$$

haben mit geeigneten $a_l^{(k)}$. Im Speziallfall $\nu_j = 0$ für alle $j$ ist $s$ gerade die interpolierende kubische Splinefunktion.

**4.4.3. Formstabile Interpolation.** Eine andere mögliche Verallgemeinerung der stückweisen Polynomfunktionen ist Wahl stückweiser rationaler Funktionen an deren Stelle. In [**Neumaier 1998**] ist eine interpolierende $C^1$–Funktion hergeleitet, die aus rationalen Funktionen zusammengesetzt ist und die optische Form der Stützstellenmenge sehr gut wiederzugeben im Stande ist. Daher auch der Name *formstabile Interpolation*. Die Herleitung dieser Funktionen ist allerdings etwas aufwendiger und sprengt den Rahmen dieses Skriptums.

## 5. Extrapolation

In manchen Anwendungsbereichen (z.B. bei der Integration — siehe Kapitel 6) ist es sinnvoll, eine Interpolationsfunktion auch außerhalb des von den Stützstellen überdeckten Intervalls auszuwerten. Für die Polynominterpolation gibt Theorem 1.1.9 Auskunft über den zu erwartenden Fehler. Speziell der Term $\omega(x)$ steigt außerhalb des Intervalls $[x_0, x_n]$, das von den extremalen Stützabszissen begrenzt wird, stark an. Ähnliches gilt auch für rationale Interpolationsfunktionen. Daher ist es in den meisten Fällen nicht sinnvoll, eine Auswertung außerhalb des Intervalls $[x_0, x_n]$, also eine *Extrapolation*, durchzuführen.

Ein Fall, in dem eine solche Vorgangsweise angebracht ist, tritt jedoch in einigen Anwendungsbereichen auf. Angenommen, man versucht ein Berechnungsproblem durch Diskretisierung zu lösen. Dabei nähert man z.B. kontinuierlich definierte Funktionen durch Treppenfunktionen an oder zerlegt wie bei der Computertomographie (Kapitel 2, Abschnitt 2.5) Gebiete in kleine Quader. Dann führt man die anstehenden Berechnungen für die vereinfachten Funktionen oder Gebiete aus, wobei sich der Aufwand meist stark reduziert. Die Grobheit der Diskretisierung (die Länge der Intervalle, auf denen die Treppenfunktion konstant ist, die Seitenlänge der Quader, ...) wird meist als Parameter (*Diskretisierungsparameter*) mitgeführt. Sei mit $h$ der Diskretisierungsparameter bezeichnet. Dann kann man mit Hilfe des Diskretisierungsverfahrens Näherungswerte $N(h)$ für das zu berechnende Ergebnis bestimmen. Den genauen Wert erhält man, wenn man auf jede Diskretisierung verzichtet, also $N(0)$ bestimmt. Ein *Extrapolationsverfahren* berechnet mehrere Werte $N(h_n)$ für die ersten Glieder einer Nullfolge $(h_n)$, bestimmt eine Interpolationsfunktion durch die Stützstellen $\big(h_i, N(h_i)\big)$ und wertet diese dann an 0 aus, um einen besseren Näherungswert für $N(0)$ zu erhalten als es die $N(h_i)$ sind.

Von den Interpolationsmethoden, die wir bisher kennengelernt haben, sind nur die Polynominterpolation und die rationale Interpolation geeignet, eine Funktion zu konstruieren, die man zur Extrapolation heranziehen kann.

**5.1. Polynomextrapolation.** Seien $(h_i, f_i)$, $i = 0, \ldots, n$, die Stützstellen, mit deren Hilfe das Interpolationspolynom $p$ konstruiert werden soll. Nachdem das Ziel ist, den Wert $p(0)$ zu bestimmen, verwendet man idealerweise den Algorithmus von Neville, siehe 1.1.1, zur Auswertung. Dadurch, daß an $x = 0$ ausgewertet werden soll, vereinfachen sich die Formeln für die Tableaus.

Es geht wieder darum, aus einer Rekursionsformel $p(0)$ zu bestimmen. Analog zu 1.1.1 definieren wir die Werte $P_{j\ldots j+k}$ für $j = 0, \ldots, n$ und $k = 0, \ldots, n - j$

$$\begin{aligned} P_j &:= f_j \\ P_{j\,j+1\ldots j+k} &:= \frac{h_j P_{j+1\ldots j+k} - h_{j+k} P_{j\ldots j+k-1}}{h_{j+k} - h_j}. \end{aligned}$$

Dann gilt $P_{0\dots n} = p(0)$, und dieser Wert kann wieder aus einem Tableau, wie dem folgenden berechnet werden.

| | $k=0$ | 1 | 2 | 3 |
|---|---|---|---|---|
| $h_0$ | $f_0 = P_0$ | | | |
| | | $P_{01}$ | | |
| $h_1$ | $f_1 = P_1$ | | $P_{012}$ | |
| | | $P_{12}$ | | $P_{0123}$ |
| $h_2$ | $f_2 = P_2$ | | $P_{123}$ | |
| | | $P_{23}$ | | |
| $h_3$ | $f_3 = P_3$ | | | |

Ein Beispiel für die Anwendung der Polynomextrapolation ist das Romberg–Verfahren zur numerischen Integration, siehe Kapitel 6 Abschnitt 3.2.

Extrapolation bei 0 ist übrigens der einzige Fall für den eine Auswertung außerhalb des Intervalls, das durch die Stützstellen gegeben ist, sinnvoll erscheint. Betrachten wir nämlich die Fehlerabschätzung aus Theorem 1.1.9, so erkennen wir, daß der Term $\omega(x)$ der Hauptgrund für die Unbrauchbarkeit des Interpolationspolynoms außerhalb des Interpolationsintervalls ist. Sein Betrag steigt dort nämlich sehr stark an. Im für die Polynomextrapolation interessanten Fall ist die Sache anders. Dort gilt $|\omega(0)| = \prod_{k=0}^{n}(h_k) \ll 1$.

**5.2. Rationale Extrapolation.** Statt eines Interpolationspolynoms kann man natürlich auch eine rationale Interpolationsfunktion $\Phi$ verwenden. Wiederum wählt man zur Bestimmung von $\Phi(0)$ einen Neville–artigen Algorithmus (Abschnitt 2.3). Genau wie im Fall der Polynomextrapolation vereinfachen sich die Formeln etwas dadurch, daß die Auswertungsstelle $y = 0$ ist.

Wiederum definiert man rekursiv Werte $T_{i,k}$ mit $k = 0, \dots, n$ und $i = k, \dots, n$ gemäß den folgenden Gleichungen

$$T_{i,0} := f_i, \tag{52}$$

$$T_{i,-1} := 0, \tag{53}$$

$$T_{i,k} := T_{i,k-1} + \frac{T_{i,k-1} - T_{i-1,k-1}}{\dfrac{h_{i-k}}{h_i}\left(1 - \dfrac{T_{i,k-1} - T_{i-1,k-1}}{T_{i,k-1} - T_{i-1,k-2}}\right) - 1}. \tag{54}$$

Man kann die Werte $T_{i,k}$ wieder in einem Tableau anordnen und die Berechnungen ähnlich dem Polynomextrapolationsverfahren durchführen. Der größte Unterschied zur Polynomextrapolation besteht (wie auch schon bei der Interpolation) darin, daß nicht nur die linken Nachbarn in die Berechnungen eingehen sondern auch das Element direkt links daneben, das zwei Spalten weiter zurückliegt. Bei den Berechnungen in dem Tableau sind also alle vier Ecken eines Rhombus beteiligt.

| $(r,s) =$ | $(0,0)$ | $(0,1)$ | $(1,1)$ | $(1,2)$ | $\cdots$ |
|---|---|---|---|---|---|
| | $f_0 = T_{0,0}$ | | | | |
| $0 = T_{0,-1}$ | | $T_{1,1}$ | | | |
| | $f_1 = T_{1,0}$ | | $T_{2,2}$ | | |
| $0 = T_{1,-1}$ | | $T_{2,1}$ | | $T_{3,3}$ | |
| | $f_2 = T_{2,0}$ | | $T_{3,2}$ | $\vdots$ | $\ddots$ |
| $0 = T_{2,-1}$ | | $T_{3,1}$ | $\vdots$ | | |
| | $f_3 = T_{3,0}$ | $\vdots$ | | | |
| $\vdots$ | $\vdots$ | | | | |

Rationale Extrapolation wird z.B. auch bei der numerischen Integration eingesetzt und verbessert meist die Ergebnisse der polynomialen Extrapolationsverfahren. Ein Beispiel dafür sieht man in Kaptiel 6 Abschnitt 3.3.

## 6. Software

Zur Lösung von Interpolationsproblemen stehen viele Unterfunktionen aus den verschiedenen Softwarebibliotheken zur Verfügung. Die Berechnung von Interpolationspolynomen bzw. deren Auswertung kann z.B. mit den Programmen `e01a??` der NAG Bibliothek erfolgen. Diese Softwaresammlung stellt auch Prozeduren zur rationalen Interpolation (`e01raf` und `e01rbf`) zur Verfügung. Auch ISML stellt analoge Funktionen zur Verfügung.

Für die sehr wichtigen Splinefunktionen gibt es in allen Softwarebibliotheken gut verwendbare Funktionen. Die Funktionen `pchsp`, `pchim`, `pchic`,... der Pakete CMLIB, SLATEC und NETLIB berechnen kubische Splinefunktionen, Ableitungen, Auswertungen,... Auch in der `MATH-LIBRARY` der ISML gibt es Funktionen für kubische Splines, allen voran die Funktion `csher`, die für beliebig vorgegebene Funktions- und Ableitungswerte die interpolierende kubische Splinefunktion berechnet und für jedes Teilintervall die Koeffizienten des Teilpolynoms berechnet — zusätzlich gibt es noch andere Funktionen `cs???`, die beinahe alle vorstellbaren Dinge für kubische Splinefunktionen berechnen. In der NAG Bibliothek dienen die Funktionen `e01b??` zur Manipulation von Splinefunktionen.

Nicht nur kubische Splines sondern auch beliebige B–Splines können mit Hilfe der Softwarebibliotheken berechnet werden. `bintk` aus CMLIB und SLATEC bestimmen zu Polynomgrad und Detenpunkten die Koeffizienten in der B–Splinebasis. Auch IMSL und NAG stellen dafür Funktionen zur Verfügung (`bsint` bzw. `e01baf` und `e02baf`). Funktionen zur Auswertung ermöglichen es dann aus den Koeffizienten der B–Spline Basisentwicklung auch Werte der interpolierenden Funktion zu berechnen: NAG/`e02bcf` bzw. ISML/`MATH-LIBRARY/bsval`. Funktionen zur Berechnung von Integralen und Ableitungen, bzw. zur Bestimmung der günstigsten Knotenpunkte komplettieren die bereitgestellten Funktionen für B–Splines.

Auch zur Berechnung von Exponentialsplines existiert Software. Erwähnt seien hier die NETLIB–Programme `A/TENSION` bzw. `TOMS/716`.

Ein eigenes Kapitel ist die FFT. Unmengen von Software existiert zur Lösung dieser Aufgabe. Die bekannteste Sammlung von Funktionen zur FFT ist das FFTPACK aus der NETLIB. Das FFTPACK besteht vor allem aus der Funktion `cfftf` für die komplexe FFT und einigen Funktionen für reelle FFT (`rfftf`) bzw. Sinus– (`sinqf`) und Cosinustransformation (`cosqf`). Die dazugehörigen Rücktransformationen erhält man durch Ersetzen des `f` am Ende der Funktionsnamen durch ein `b`.

Möchte man (besonders auf Vektor– und Parallelrechnern) mehrere Funktionen gleichzeitig transformieren, dann kann man die analogen Funktionen aus dem VFFTPACK, ebenfalls aus der NETLIB Sammlung, verwenden.

Auch in den großen Softwarebibliotheken IMSL und NAG existieren Unterprogramme für die Berechnung schneller Fouriertransformationen. In der `MATH-LIBRARY` der IMSL wurden die Funktionen aus dem FFTPACK übernommen; allerdings wurden zusätzlich Funktionen für die Berechnung mehrdimensionaler schneller Fouriertransformation und deren Umkehrungen hinzugefügt (`fft2d`, `fft2b`, `fft3d`, `fft3b`). Die NAG Sammlung enthält eigenständige Funktionen, die allesamt die Beschränkung aufweisen, daß die Primfaktorenzerlegung von $N$, der Anzahl der Datenpunkte, höchstens 20 Faktoren enthalten darf. Zusätzlich darf kein Faktor selbst größer als 19 sein. Die Funktionen `c06e??` dienen zur eindimensionalen Fouriertransformation. Vektorisierte Versionen (`c06f??`) stehen ebenso zur Verfügung wie

Unterprogramme zur Sinus– und Cosinustransformation. Mit der Funktion `c06fjf` steht außerdem ein Unterprogramm zur Berechnug beliebigdimensionaler Fouriertransformationen zur Verfügung.

In **Matlab** sind die Funktionen `FFT`, `IFFT` zur Berechnung der Fouriertransformation und deren Inverser definiert. `polyfit`, `poly` und `spline` dienen zur Berechnung von Interpolations- und Approximationspolynomen, deren Auswertung und zur Bestimmung interpolierender Splines.

**Mathematica** stellt ebenfalls Funktionen für verschiedene Interpolationsaufgaben zur Verfügung. Besonders `Fourier`, `InterpolatingPolynomial` und `InterpolatingFunction` seien dabei besonders hervorgehoben.

KAPITEL 6

# Differentialrechnung und Integration I: Eindimensionaler Fall

„Differenzieren ist eine Wissenschaft, integrieren ist eine Kunst.“ Daß sich dieser den meisten Mathematikern bekannte Ausspruch in der numerischen Mathematik umkehrt, ist unter anderem Inhalt dieses Kapitels.

## 1. Grundlagen

Beginnen wir mit einer Diskussion des Spruches aus der Einleitung: Mit den bekannten Differentiationsregeln (Linearität, Produktregel, Kettenregel,...), einigen bekannten Umformungstricks und wenigen auswendig gelernten Ableitungsfunktionen kann nach einiger Übung jeder Schüler einer höheren Schule, ob er sich für Mathematik interessiert oder nicht, zu jeder Funktion, die man als Zusammensetzung elementarer Funktionen aufschreiben kann, die Ableitungsfunktion bestimmen.

Bei der Berechnung von Integralen sieht die Sache völlig anders aus. Einige unbestimmte Integrale kann man direkt berechnen, manches andere mit Ansätzen oder speziellen Integrationstricks. Unglücklicherweise ist das die Ausnahme. Bei vielen Funktionen kann man sogar beweisen, daß sich die Stammfunktion nicht als Zusammensetzung elementarer Funktionen schreiben läßt. Es sind auch nicht unbedingt die kompliziert gebauten Funktionen, deren Stammfunktionen sich nicht berechnen lassen, wie man an den folgenden Beispielen erkennen kann:

$$\int x e^{-x^2}\,dx, \quad \text{bzw.} \quad \int e^{-x^2}\,dx.$$

Das zweite Integral ist so wichtig, daß sich die Mathematiker entschlossen haben, das Problem wegzudefinieren.

$$\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{x^2}\,dx$$

Die Eigenschaften der *Fehlerfunktion* wurden genau analysiert, und sie liegt in tabellierter Form vor, sodaß sie ebenso gut verwendet werden kann wie ein Sinus oder eine Exponentialfunktion.

Unglücklicherweise kann man das nicht mit jedem Integral machen, auf das man trifft. Doch das ist im allgemeinen auch nicht nötig. Meist muß man in Anwendungen auch nur *bestimmte Integrale* der Form

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

berechnen. Das Resultat eines solchen Integrals ist eine Zahl, für die die klassische Berechnungsart: „Bestimme die Stammfunktion $F$ von $f$, und weil

$$\int_a^b f(x)\,dx = F(b) - F(a)$$

gilt, werte die Stammfunktion bei $a$ und $b$ aus.“ nur eine Möglichkeit ist. Wie wir schon in Kapitel 1 gesehen haben, ist die klassische Methode, sollte sie einmal funktionieren, nicht

unbedingt numerisch gutartig, besonders wenn die Stammfunktion ein komplexer mathematischer Ausdruck ist.

Wir wollen also in diesem Kapitel versuchen, Möglichkeiten zu finden, den Wert eines bestimmten Integrals auf andere Weise zu approximieren.

Numerische Differentiation ist eine schwierigere Sache. Wie wir wissen, ist analytisch ableiten eine algorithmisch einfache Sache, die etwa von Computerprogrammen problemlos erledigt werden kann. Wollen wir aber den *Wert* der Ableitung an einem bestimmten Punkt bestimmen, so haben wir zuerst einmal die Wahl zwischen zwei Verfahren:

(1) Analytisch differenzieren und Auswerten der Ableitungsfunktion. Dieses Verfahren ist schwierig anzuwenden, denn es benötigt einen Programmteil, der symbolisch rechnen kann, und der für neue Funktionenklassen angepaßt werden muß. Außerdem funktioniert das Verfahren nur für Funktionen, die durch eine explizite Formel gegeben sind, was in Anwendungen oft nicht der Fall ist. Diese Vorgehensweise ist also unpraktikabel, oder zumindest unbefriedigend.

(2) Die Alternative ist, die Definition der Ableitung durch den Differentialquotienten zu verwenden:

$$f'(x) = \lim_{h\to 0} \frac{f(x+h) - f(x)}{h}$$

Da man den Grenzwert nicht direkt bestimmen kann, muß man ihn durch einen Differenzenquotienten approximieren. Doch welches $h$ wählt man für die Approximation. Ist $h$ zu groß, ist der Differenzenquotient zu ungenau, und die Approximation schlecht. Ist $h$ sehr klein, so gilt $f(x+h) \approx f(x)$, da $f$ stetig ist, und daher treten bei der Berechnung des Zählers starke Auslöschungseffekte auf. Daher kann man $h$ nicht beliebig klein wählen, und die Approximationsgüte ist beschränkt. Dies sei auch in der folgenden Tabelle belegt, in der die Ergebnisse des Differentialquotienten für die Funktion $f(x) = x^3$ an der Stelle $x = 1$ und verschiedene $h$ aufgelistet sind. Die Berechnung erfolgte mit doppelter Genauigkeit, also etwa 16 signifikanten Dezimalstellen.

| $h$ | $((1+h)^3 - 1)/h$ |
|---|---|
| $1$ | 7 |
| $10^{-1}$ | 3.310000000000004 |
| $10^{-2}$ | 3.030100000000013 |
| $10^{-3}$ | 3.003000999999728 |
| $10^{-4}$ | 3.000300009998735 |
| $10^{-5}$ | 3.000030000110953 |
| $10^{-6}$ | 3.000002999797857 |
| $10^{-7}$ | 3.000000301511818 |
| $10^{-8}$ | 3.000000003972048 |
| $10^{-9}$ | 3.000000248221113 |
| $10^{-10}$ | 3.000000248221113 |
| $10^{-11}$ | 3.000000248221113 |
| $10^{-12}$ | 3.000266701747023 |
| $10^{-13}$ | 2.997602166487923 |
| $10^{-14}$ | 2.997602166487923 |
| $10^{-15}$ | 3.330669073875469 |
| $10^{-16}$ | 0 |

Man sieht schnell, daß die höchste Genauigkeit für $h \approx 10^{-8}$ etwa 8 signifikante Stellen beträgt. Im allgemeinen findet man, daß die Approximation der Ableitung durch den Differenzenquotienten etwa einen relativen Fehler der Größenordnung

$\sqrt{\texttt{eps}}$ erzielt. Im Fall doppelter Genauigkeit gehen also etwa 8 signifikante Stellen verloren, auch unbefriedigend.

Nachdem sich beide offensichtlichen Verfahren als ungünstig herausgestellt haben, ist einiges an zusätzlichen Überlegungen von Nöten. Wir werden uns in Abschnitt 8 damit beschäftigen.

## 2. Integration - einfache Verfahren

In diesem Abschnitt wollen wir die Grundideen der numerischen Integration, wie sie schon aus der Schule bekannt sind, aufarbeiten. Gleichzeitig dienen sie auch als Ausgangspunkt für die numerisch verwendbareren Verfahren in den späteren Abschnitten.

**2.1. Trapezregel, Mittelpunktregel.** Die einfachste Idee zur Approximation eines bestimmten Integrals stammt aus der geometrischen Interpretation des Integrals. Gemäß Abbildung 6.1 nähert man die Fläche zwischen der Funktion $f$ und der $x$–Achse begrenzt

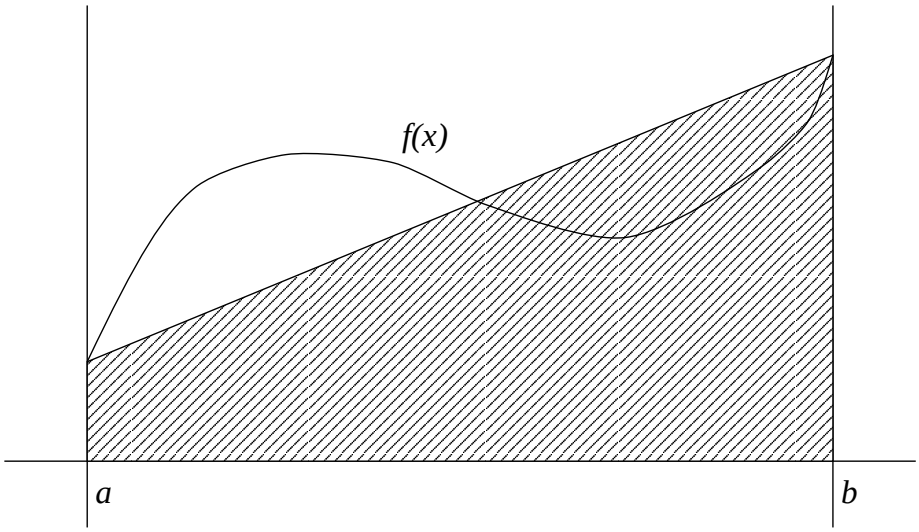



Abbildung 6.1. Trapezregel

durch die Geraden $x = a$ und $x = b$, die ja dem bestimmten Integral entspricht, durch die Fläche des eingezeichneten Trapezes an

$$\int_a^b f(x)\,dx \approx \frac{a-b}{2}(f(a)+f(b)).$$

In Abbildung 6.1 scheint diese Näherung gut genug für eine erste Approximation zu sein, doch Abbildung 6.2 liefert einen Hinweis darauf, was bei dieser Form der Approximation schief gehen kann. Es ist also unbedingt notwendig, eine Fehlerabschätzung zu machen, und das geschieht am besten auf algebraischem Weg. Betrachtet man die Trapezregel aus einem anderen Blickwinkel, kommt man zu dem folgenden Schluß: Um das bestimmte Integral zur Funktion $f$ im Intervall $[a, b]$ zu berechnen, nähert man $f$ durch ein Interpolationspolynom $p$ ersten Grades an. Das Integral über $p$ verwendet man dann als Approximation für das Integral über $f$:

$$p(x) = f(a) + \frac{f(b)-f(a)}{b-a}(x-a)$$

und damit

$$\int_a^b f(x)\,dx \approx \int_a^b p(x)\,dx = \frac{b-a}{2}(f(a)+f(b)) =: T_h^0(f)$$

mit $h = b - a$.

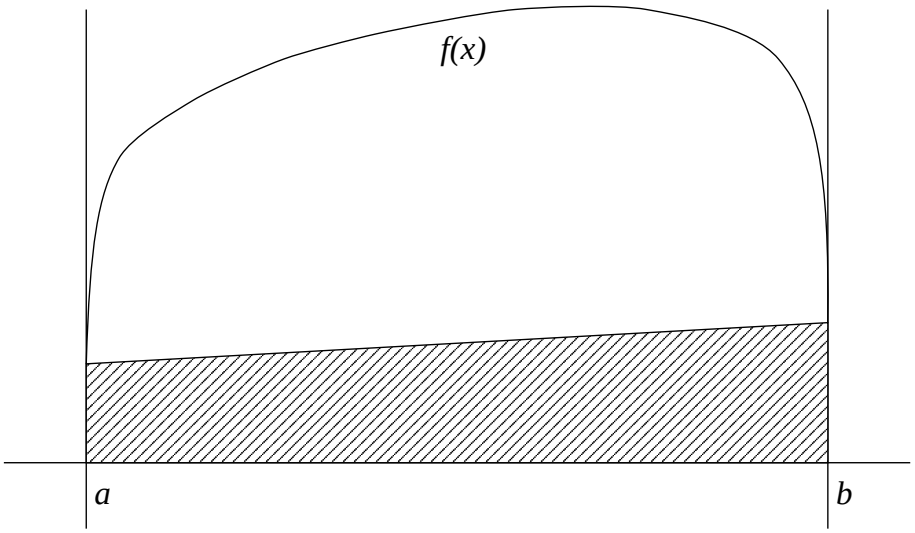



ABBILDUNG 6.2. Eine ungünstige Funktion für die Trapezregel

Der Vorteil dieser Sichtweise ist, daß wir Theorem 1.1.9 aus Kapitel 5 verwenden können, um den Unterschied zwischen $f$ und $p$ abzuschätzen:

$$f(x) - p(x) = (x-a)(x-b)\frac{f''(\xi_x)}{2},$$

für ein $\xi_x \in [a,b]$. Daher gilt

$$\begin{aligned}\int_a^{a+h} f(x)\,dx - T_h^0(f) &= \int_a^{a+h} (f(x)-p(x))\,dx = \frac{1}{2}\int_a^{a+h} f''(\xi_x)(x-a)(x-a-h)\,dx = \\ &= \frac{f''(\eta)}{2}\int_a^{a+h}(x-a)(x-a-h)\,dx = -\frac{f''(\eta)}{12}h^3,\end{aligned}$$

für ein $\eta \in [a,b]$, wobei wir, um in die zweite Zeile umzuformen, den Mittelwertsatz der Integralrechnung verwendet haben. Ist also die Funktion $f$ nicht zu stark gekrümmt auf $[a,b]$ und ist $h$ klein, dann bietet die Trapezregel eine gute Approximation für den Wert des bestimmten Integrals. Ist etwa $|f''(x)| \leq 1$ und $h = 10^{-2}$, dann ist der Fehler kleiner als $10^{-7}$.

Nachdem für allgemeine bestimmte Integrale leider keine dieser Bedingungen zutrifft, muß man sich eine Verbesserung der Methode einfallen lassen, um den möglichen Fehler weiter zu verringern. Wieder ist es eine einfache Idee, die ebendiese Verbesserung ermöglicht. Halbieren wir die Länge des Intervalls, so reduziert sich der Fehler auf ein Achtel. Daher teilen wir $[a,b]$ in $n$ gleich große Teile und approximieren auf jedem der entstehenden Teilintervalle das bestimmte Integral mit Hilfe der Trapezregel. Sei $x_i = a + \frac{i}{n}(b-a)$. Setzen wir jetzt $h = \frac{b-a}{n}$, so erhalten wir die Formel

$$\begin{aligned}\int_a^b f(x)\,dx &= \sum_{i=0}^{n-1}\int_{x_i}^{x_{i+1}} f(x)\,dx = \sum_{i=0}^{n-1}\frac{h}{2}(f(x_i)+f(x_{i+1})) = \\ &= h\left(\frac{f(a)}{2} + \sum_{i=1}^{n-1} f(x_i) + \frac{f(b)}{2}\right) =: T_h(f)\end{aligned}$$

für die zusammengesetzte Trapezregel, die man üblicherweise einfach mit *Trapezregel* bezeichnet. Die Fehlerabschätzung für $T_h(f)$ kann man aus der Fehlerabschätzung für $T_h^0$ herleiten.

Es gilt

$$\int_a^b f(x)\,dx - T_h(f) = -\frac{h^3}{12}\sum_{i=0}^{n-1} f''(\eta_i) = -\frac{h^2(b-a)}{12}\frac{1}{n}\sum_{i=0}^{n-1} f''(\eta_i).$$

Die Summe zusammen mit dem Faktor $1/n$ beschreibt das arithmetische Mittel der Werte $f''(\eta_i)$. Ist $f''$ stetig, dann gibt es wegen des Zwischenwertsatzes ein $\eta \in [a,b]$ mit $f''(\eta) = \sum_{i=0}^{n-1} f''(\eta_i)$, und daher haben wir schließlich die

**Proposition 2.1.1.** *Für die (zusammengesetzte) Trapezregel gilt die Fehlerabschätzung*

$$\int_a^b f(x)\,dx - T_h(f) = -\frac{(b-a)f''(\eta)}{12}h^2, \tag{55}$$

*für ein* $\eta \in [a,b]$.

Diese Abschätzung trifft eine starke Aussage: Wählt man die Unterteilung des Intervalls $[a,b]$ fein genug, dann kann man den Approximationsfehler für das Integral beliebig klein machen. Halbiert man die Feinheit der Unterteilung, dann verkleinert sich der Fehler auf ein Viertel. Halbierung der Unterteilung ist auch die günstigste Wahl, falls man einen bereits berechneten Wert weiter verbessern möchte, da man in diesem Fall alle bereits bestimmten Funktionswerte wiederverwenden kann.

Ein zweites einfaches Verfahren zur Abschätzung eines bestimmten Integrals kann man aus den Riemannsummen konstruieren. Aus der Analysis ist bekannt, daß für jede Riemann–integrierbare Funktion $f$ die Folge der Riemannsummen

$$\sum_{x_i\in\Delta} \frac{1}{x_{i+1}-x_i} f(\xi_i)$$

gegen das bestimmte Integral konvergiert, wenn die Feinheit der Unterteilung $\Delta$ gegen Null geht. Das gilt für jede mögliche Wahl der Zwischenstellen $\xi_i$. Am einfachsten ist, die Mittelpunkte der Teilintervalle als Zwischenstellen zu wählen. Gibt man sich eine äquidistante Unterteilung

$$a = x_0 < x_1 < \cdots < x_{n-1} < x_n = b$$

von $[a,b]$ vor, so erhält man die (zusammengesetzte) *Mittelpunktregel*

$$M_h(f) := \int_a^b f(x)\,dx \approx h\sum_{i=0}^{n-1} f(x_{i+\frac{1}{2}});$$

wieder setzt man $h = \frac{b-a}{n}$ und $x_j = a + \frac{j}{n}(b-a)$. Der Unterschied zwischen den beiden Approximationen ist in Abbildung 6.3 dargestellt.

Fügt man die beiden Methoden zueinander, so erhält man den einfachen Berechnungsalgorithmus 2.1.2. Dabei nimmt man an, daß, wenn sich die beiden Approximationen $T_h(f)$ und $M_h(f)$ nur um $\varepsilon$ unterscheiden, auch der Wert des bestimmten Integrals bis auf Fehler der Größenordnung $\varepsilon$ erreicht ist. Halbiert man in jedem Schritt des Algorithmus die Unterteilung, dann kann man zusätzlich die Eigenschaft

$$T_{\frac{h}{2}}(f) = \tfrac{1}{2}\big(T_h(f) + M_h(f)\big)$$

verwenden, um den Rechenaufwand zu minimieren.

**Algorithmus 2.1.2.** *Gemischte Trapez– und Mittelpunktregel*

$h = b - a$
$n = 1$
$T = \frac{h}{2}(f(a) + f(b))$
**for** $i = 1$ **to** `MAXITER` **do**
  $M = 0$

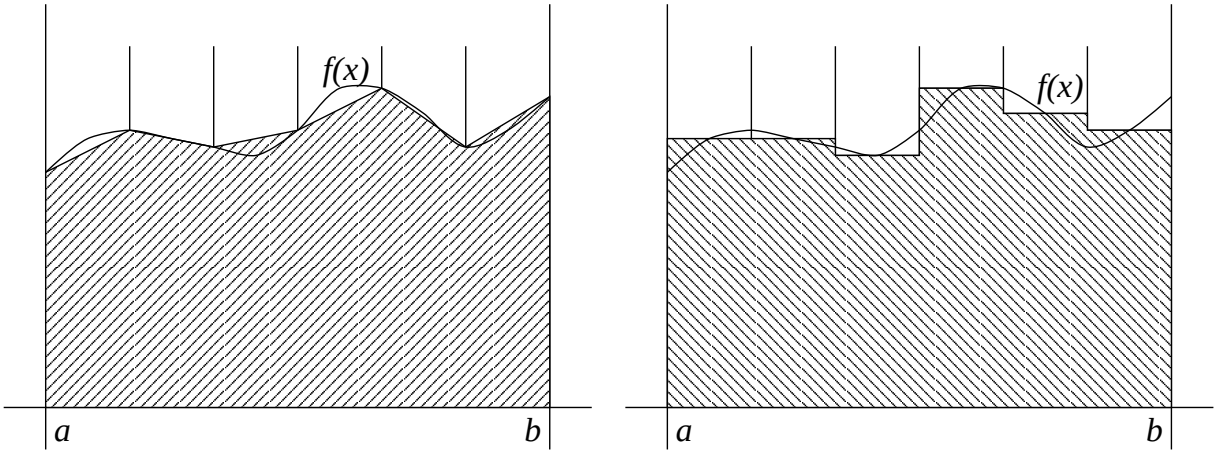



ABBILDUNG 6.3. Trapezregel und Mittelpunktregel

**for** $j = 0$ **to** $n - 1$ **do**
$M = M + f(a + (j + 1/2)h)$
**done**
$M = hM$
$T = (T + M)/2$
$h = h/2$
$n = 2n$
**if** $|T - M| < \varepsilon$ **then**
**return**$(T)$
**endif**
**done**
**warn***("Höchstzahl an Iterationen überschritten,")*
**warn***("erforderliche Genauigkeit nicht erreicht!")*
**return**$(T)$

*wobei eine geeignete Höchstanzahl* `MAXITER` *an Iterationen vom Benutzer vorgegeben wird.*

Wegen Proposition 2.1.1 wissen wir, daß bei genügend kleiner Feinheit $h$ der Unterteilung auch der Fehler der Näherung beliebig klein wird. Genauer ist nach dem $n$–ten Schritt der Approximationsfehler um einen Faktor $1/2^{2n}$ gesunken. Obwohl man den Wert des bestimmten Integrals mit Algorithmus 2.1.2 beliebig genau berechnen kann, ist der Aufwand meist zu groß, da man im $n$–ten Schritt $2^{n-1}$ Funktionswerte berechnen muß; der Rechenaufwand steigt also exponentiell.

Um dieses Problem zu umgehen, muß man neue Methoden der Integralberechnung erfinden. Man kann einerseits versuchen, mittels genauer Fehleranalyse den Wert $T_0(f)$ zu schätzen; das führt zu den Extrapolationsverfahren. Andererseits kann man die Approximation des Integrals in jedem einzelnen Teilintervall verbessern; dieses Vorgehen führt zu den Newton–Cotes Formeln. Genauere Überlegungen ergeben schließlich die adaptiven Verfahren und die Gauß–Quadratur, die heute das wahrscheinlich gebräuchlichste eindimensionale Integrationsverfahren ist.

## 3. Extrapolationsverfahren

Extrapolationsverfahren beruhen auf der Idee, die Näherungswerte $T_{h_n}(f)$ für die ersten Glieder einer Nullfolge $h_n$ zu bestimmen und daraus den Wert für $h = 0$, also den genauen

Wert des Integrals, zu schätzen. Zu diesem Zweck muß man jedoch zuerst den Approximationsfehler genau bestimmen. Der Weg dazu führt über ein Resultat aus der Analysis, die Euler–Maclaurinsche Summenformel.

**3.1. Euler–Maclaurinsche Summenformel.** Die Herleitung der von Euler und später unabhängig von Maclaurin gefundenen Summenformel beginnt mit dem Versuch, eine Summe der Form

$$\sum_{k=1}^{n} f(k),$$

also eine Summe von Flächenstücken der Größe $f(k) \cdot 1$ durch das bestimmte Integral

$$\int_0^n f(x)\,dx$$

zu berechnen. Dazu verwendet man die Sprache des Riemann–Stieltjes Integrales, welches etwa in [**Heuser 1986/1**, XI, 90] entwickelt wird. Es gilt

$$\sum_{k=1}^{n} f(k) = \int_0^n f(x)\,d\lfloor x\rfloor,$$

wobei $\lfloor x\rfloor$ definiert sei als

$$\lfloor x\rfloor := k \quad \text{für } x \in [k, k+1),\ k \in \mathbb{Z}$$

die größte ganze Zahl $\le x$. Ist $f$ stetig differenzierbar, so kann man das Riemann–Stieltjes Integral in ein gewöhnliches Integral umformen:

$$\sum_{k=1}^{n} f(k) = \int_0^n (f(x) + (x - \lfloor x\rfloor)f'(x)\,dx,$$

und etwas symmetrischer:

$$\sum_{k=1}^{n} f(k) = \int_0^n f(x)\,dx + \frac{f(0)+f(n)}{2} + \int_0^n \left(x - \lfloor x\rfloor - \tfrac{1}{2}\right) f'(x)\,dx. \tag{56}$$

Diese Gleichung ist die allgemeinste Form der Eulerschen Summenformel. Man kann auch schon den Zusammenhang mit der Trapezregel sehen. Leider ist sie in dieser Form als Grundlage eines Extrapolationsverfahrens nicht verwendbar. Um sie jedoch in eine brauchbare Form umzuwandeln, müssen wir zunächst noch einige zusätzliche Mathematik entwickeln.

Alles beginnt mit dem Versuch, die Funktion $\frac{x}{e^x-1}$ in eine Potenzreihe zu entwickeln. Setzt man die Reihe an als

$$\frac{x}{e^x-1} = \sum_{j=0}^{\infty} \frac{B_j}{j!} x^j,$$

so kann man durch Multiplikation mit der Reihenentwicklung von $\frac{e^x-1}{x}$, die sich leicht aus der Reihe für $e^x$ berechnen läßt, und durch Koeffizientenvergleich rekursive Beziehungen für die $B_j$ herleiten. Klarerweise sind $B_0 = 1$ und $B_1 = -B_0/2 = -1/2$. Außerdem gilt für jedes $k > 2$

$$\sum_{j=0}^{k-1} \binom{k}{j} B_j = 0.$$

Die Zahlen $B_j$ heißen *Bernoullische Zahlen*, und die ersten ergeben sich zu

$$B_0 = 1, \quad B_1 = -\tfrac{1}{2}, \quad B_2 = \tfrac{1}{6}, \quad B_{2j+1} = 0 \quad \text{für } j \ge 1,$$
$$B_4 = -\tfrac{1}{30}, \quad B_6 = \tfrac{1}{42}, \quad B_8 = -\tfrac{1}{30}, \quad B_{10} = \tfrac{5}{66}, \ldots$$

Alle Zahlen sind rational, und man kann das asymptotische Verhalten

$$B_{2k} \sim 2(2k)!(2\pi)^{-2k}, \quad \text{für } k \to \infty$$

beweisen. Die Bernoullischen Zahlen sind außerordentlich beziehungsreich; sie treten etwa in der Reihenentwicklung des Tangens auf:

$$\tan x = \sum_{k=1}^{\infty} (-1)^{k-1} \frac{2^{2k}(2^{2k}-1)B_{2k}}{(2k)!} x^{2k-1},$$

die man selbst aus der Entwicklung von $x \cot x$ berechnen kann:

$$x \cot x = \sum_{k=0}^{\infty} (-1)^k \frac{2^{2k} B_{2k}}{(2k)!} x^{2k}.$$

Auch die berühmten Summenformeln

$$\sum_{n=1}^{\infty} \frac{1}{n^2} = \frac{\pi^2}{6}, \quad \text{und} \quad \sum_{n=1}^{\infty} \frac{1}{n^4} = \frac{\pi^4}{90}$$

ergeben sich aus einer allgemeineren Beziehung, in der die Bernoullischen Zahlen eine bedeutende Rolle spielen

$$\sum_{n=1}^{\infty} \frac{1}{n^{2p}} = (-1)^{p-1} \frac{B_{2p}(2\pi)^{2p}}{2(2p)!}.$$

Die Herleitung dieser Gleichung kann man etwa in [**Heuser 1986/2**, 148] nachschlagen.

Untersuchen wir als nächstes die Funktion

$$g(x,t) = \frac{xe^{tx}}{e^x - 1};$$

sie ist als Zusammensetzung analytischer Funktionen für alle $t$ und hinreichend kleinen $|x|$ in eine Reihe der Form

$$g(x,t) = \sum_{j=0}^{\infty} \frac{B_j(t)}{j!} x^j \tag{57}$$

entwickelbar. Multipliziert man für die Untersuchung von Gleichung (57) die Potenzreihenentwicklung von $\frac{x}{e^x-1}$ mit derjenigen von $e^{tx}$, so erhält man durch Koeffizientenvergleich eine Gleichung für die Polynome $B_k(t)$

$$B_k(t) = \sum_{j=0}^{k} \binom{k}{j} B_j t^{k-j}, \tag{58}$$

die *Bernoullischen Polynome*. Offensichtlich gilt $B_k(0) = B_k$. Sie hängen über die Gleichungen

$$\frac{d}{dt} B_{k+1}(t) = (k+1) B_k(t), \quad \text{bzw.}$$
$$\int_0^x B_k(t)\, dt = \frac{B_{k+1}(x) - B_{k+1}}{k+1}$$

miteinander zusammen. Weiters folgt aus Gleichung (58) und bekannten Formeln für Summen von Binomialkoeffizienten für alle $k$

$$\int_0^1 B_k(t) = 0.$$

Kehren wir jetzt wieder zur Eulerschen Summenformel (56) zurück. Genauer analysieren wir den Term

$$\int_0^n \left(x - \lfloor x \rfloor - \tfrac{1}{2}\right) f'(x)\,dx.$$

Auf $[0,1)$ stimmt $\left(x - \lfloor x \rfloor - \frac{1}{2}\right)$ mit $B_1(x) = (x - \frac{1}{2})$ überein. Somit kann man die Eigenschaften der Bernoullischen Polynome verwenden, um partiell zu integrieren.

$$\begin{aligned} R_1 &= \int_0^1 \left(x - \lfloor x \rfloor - \tfrac{1}{2}\right) f'(x)\,dx = \int_0^1 B_1(x) f'(x)\,dx = \\ &= \left[f'(x)\int_0^x B_1(t)\,dt\right]_0^1 - \int_0^1 f''(x)\left(\int_0^x B_1(t)\,dt\right) = \\ &= -\int_0^1 f''(x)\frac{B_2(x) - B_2}{2}\,dx = \\ &= \frac{B_2}{2}[f']_0^1 - \frac{1}{2}\int_0^1 B_2(x) f''(x)\,dx. \end{aligned}$$

Ist $f$ genügend oft differenzierbar, dann kann man $R_1$ durch weitere partielle Integrationen als Summe darstellen

$$R_1 = \sum_{k=2}^{m} (-1)^k \frac{B_k}{k!}[f^{(k-1)}]_0^1 + \frac{(-1)^{m+1}}{m!} R_m$$

mit

$$R_m := \int_0^1 B_m(x) f^{(m)}(x)\,dx.$$

In analoger Weise kann man die Integrale $\int_\ell^{\ell+1} \left(x - \lfloor x \rfloor - \frac{1}{2}\right) f'(x)\,dx$ behandeln, indem man die Funktionen $B_k(t)$ von $[0,1)$ periodisch auf ganz $\mathbb{R}$ mit Periodenlänge 1 zu Funktionen $\beta_k(t)$ fortsetzt

$$\beta_k(t) := B_k(t - \lfloor t \rfloor).$$

Auf diese Weise erhält man die *Euler–Maclaurinsche Summenformel*

$$\begin{aligned} \sum_{k=0}^{n} f(k) = \int_0^n f(x)\,dx + \frac{f(0)+f(n)}{2} + \sum_{j=1}^{m} \frac{B_{2m}}{(2m)!}[f^{(2m-1)}]_0^n + \\ + \frac{1}{(2m+1)!}\int_0^n \beta_{2p+1}(x) f^{(2p+1)}(x)\,dx \end{aligned} \tag{59}$$

für eine $(2p+1)$–mal stetig differenzierbare Funktion $f$. Etwas umgewandelt erhält man für ein allgemeines Intervall $[a,b]$, indem man $f(x) := g(a + xh)$ für $x \in [0,n]$ setzt, den Zusammenhang zwischen der Trapezregel und dem Integral

$$\int_a^b g(x)\,dx - T_h(g) = \sum_{j=1}^{m} \frac{B_{2m}}{(2m)!} h^{2m}[g^{(2m-1)}]_a^b + \frac{1}{(2m+1)!} h^{2p+2}\int_a^b \beta_{2p+1}\left(\tfrac{x-a}{h}\right) g^{(2p+1)}(x)\,dx. \tag{60}$$

Diese Fehlerformel ist die Grundlage für die Extrapolationsverfahren zur Berechnung des bestimmten Integrales. Für $h \to 0$ geht der Fehler gemäß der rechten Seite gegen 0. Dabei Unterscheidet sich der Fehler von einem Polynom in $h$ nur aufgrund des Integralterms. Setzt man voraus, daß dieser Term asymptotisch für $h \to 0$ keine Rolle spielt, so kann man annehmen, daß für ein Interpolationspolynom $p$ genügend hohen Grades festgelegt durch einige Werte von $h$ der Wert $p(0)$ gerade den Wert des bestimmten Integrals beschreibt. Auf diese Weise gelangt man zum polynomialen Extrapolationsverfahren aus 3.2. Wenn

man nicht annimmt, daß Polynome zur Beschreibung der Fehlerfunktion ausreichen, kann man stattdessen auch eine rationale Funktion $\Phi$ zur Interpolation der berechneten Werte heranziehen. Auf diese Weise gelangt man zum Verfahren der rationalen Extrapolation, die wir in Abschnitt 3.3 besprechen werden.

**3.2. Polynomextrapolation.** Das Polynomextrapolationsverfahren, auch Romberg Verfahren genannt, bestimmt Näherungswerte $T_{h_n}$ für die Werte $h_i = \frac{1}{2^i}(b-a)$. Das kann etwa mittels Algorithmus 2.1.2 geschehen. Dann führt man gemäß Kapitel 5 Abschnitt 5.1 ein polynomiales Extrapolationsverfahren aus, um den Wert $T_0$ zu schätzen.

**Beispiel 3.2.1.** *Man berechne das Integral*

$$\int_1^2 \frac{1}{x}\,dx = \log(2) = 0.6931471805\ldots$$

*mit Hilfe des Polynomextrapolationsverfahrens. Es entsteht das folgende Tableau*

| $h_n$ | $T_{h_n}$ | | | | |
|---|---|---|---|---|---|
| 1 | 0.75 | | | | |
| | | 0.6944444444 | | | |
| $\frac{1}{2}$ | 0.7083333333 | | 0.6931746032 | | |
| | | 0.6932539683 | | 0.6931474776 | |
| $\frac{1}{4}$ | 0.6970238095 | | 0.6931479015 | | 0.6931471819 |
| | | 0.6931545307 | | 0.6931471831 | |
| $\frac{1}{8}$ | 0.6941218504 | | 0.6931471834 | | |
| | | 0.6931476528 | | | |
| $\frac{1}{16}$ | 0.6933912022 | | | | |

*Man erkennt, daß der Näherungswert $T_{1/16}$ lediglich drei Stellen Genauigkeit besitzt. Der aus den fünf ersten Näherungen geschätzte Wert $T_0$ ist jedoch auf sieben Stellen genau. Die Extrapolation, die auf der Euler–Maclaurinschen Summenformel basiert, hat im vorliegenden Fall die Anzahl richtiger Stellen mehr als verdoppelt.*

**3.3. Rationale Extrapolation.** Um wirklich hohe Genauigkeiten zu erzielen, ist das im vorigen Abschnitt vorgestellte Verfahren etwas zu aufwendig. Das liegt einerseits an der Polynomextrapolation, andererseits am zu hohen Aufwand für die Berechnung der $2^{n+1}-1$ Funktionswerte, falls ein Polynom $n$–ten Grades verwendet werden soll. Ein Ausweg aus dieser Misere ist, anstelle polynomialer Extrapolation rationale Funktionen zu verwenden und die Nullfolge etwas abzuwandeln.

Man definiert die Folge $h_n$ rekursiv etwa wie folgt:

$$\begin{aligned} h_0 &= \tfrac{1}{2}(b-a) \\ h_1 &= \tfrac{1}{3}(b-a) \\ h_n &= \tfrac{h_{n-2}}{2} \end{aligned}$$

und berechnet die Werte $T_{h_n}$ durch eine Abwandlung von Algorithmus 2.1.2. Anschließend bestimmt man mit den Methoden aus Kapitel 5 Abschnitt 5.2 den Näherungswert für $T_0$ durch rationale Extrapolation.

**Beispiel 3.3.1.** *Die Zahl $\pi$ kann durch numerische Berechnung des Integrals*

$$\int_0^1 \frac{1}{1+x^2}\,dx = \arctan(1) = \frac{\pi}{4} = 0.7853981633\ldots$$

*näherungsweise bestimmt werden. Wir berechnen für* $h = \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{6}, \frac{1}{8}, \frac{1}{12}, \frac{1}{16}$ *die Näherungswerte, und davon ausgehend kann das Integral approximiert werden. Der beste Näherungswert, der aus direkten Berechnungen stammt, ist* $T_{\frac{1}{16}} = 0.\underline{785}2354030$ *mit drei korrekten Stellen. Polynomextrapolation gemäß Abschnitt 3.2 liefert* $T_{0,p} = 0.\underline{7853982}585$ *mit sechs korrekten Stellen. Rationale Extrapolation liefert hingegen die Näherung* $T_{0,r} = 0.\underline{7853981}662$ *mit immerhin acht korrekten Stellen.*

## 4. Interpolatorische Quadraturformeln

Der andere Weg zur Verbesserung der Genauigkeit der Approximation bei der Berechnung bestimmter Integrale beruht auf der folgenden Überlegung: Die Trapezregel hat die Eigenschaft, daß sie alle Polynome ersten Grades ohne Approximationsfehler integriert. Man suche also eine neue Quadraturformel[1] der Art

$$\int_a^b f(x)\,dx \approx \sum_{i=0}^{n} A_i f(x_i)$$

mit paarweise verschiedenen Stützstellen $x_i \in [a,b]$, die Polynome $n$–ten Grades exakt integriert. Wieder liegt die Grundidee der Bestimmung der $A_i$ in der Berechnung eines Interpolationspolynomes.

Um das Interpolationspolynom zu bestimmen, kann man die Lagrangesche Darstellung heranziehen (siehe Kapitel 5 Abschnitt 1.1). Seien die Punkte $(x_i, f(x_i))$, $i = 0, \ldots, n$, gegeben. Dann hat das Interpolationspolynom $n$–ten Grades $P_n$ zu obigen Punkten die Gestalt

$$P_n(x) = \sum_{i=0}^{n} f(x_i) L_i(x).$$

Die Koeffizienten $A_i$ der Quadraturformel können aus dieser Darstellung leicht bestimmt werden:

$$A_i = \int_a^b L_i(x)\,dx. \tag{61}$$

Für das Interpolationspolynom gilt dann nämlich

$$\int_a^b P_n(x)\,dx = \sum_{i=0}^{n} f(x_i) \int_a^b L_i(x)\,dx = \sum_{i=0}^{n} A_i f(x_i).$$

Diese Beziehung gilt für alle polynomialen Quadraturformeln.

Allgemeinere interpolatorische Quadraturformeln verwenden nicht Interpolationspolynome sondern andere Interpolationsfunktionen, wie etwa kubische Splines. In diesem Fall läuft die Berechnung des Integrals auf die Lösung eines linearen Gleichungssystemes hinaus.

**4.1. Newton–Cotes Formeln.** Werden die Stützstellen für eine polynomiale Quadraturformel innerhalb des Intervalls $[a,b]$ äquidistant gewählt, so erhalten wir, je nach Wahl des Polynomgrades $n$, eine der Newton–Cotes Formeln. Viele dieser auf mathematisch ähnliche Weise entstehenden Quadraturformeln haben historisch bedingt eigene Namen erhalten (Keplersche Faßregel, Simpsonsche Regel, 3/8–Regel,...), und alle Newton–Cotes Formeln können ebenso wie deren einfachste Variante, die Trapezregel, aufgeteilt auf Teilintervalle angewendet werden, um die Genauigkeit weiter zu verbessern.

[1]Früher war die Aufgabe zur Flächenberechnung folgendermaßen formuliert: „Man finde ein Quadrat gleichen Flächeninhalts zu der Fläche, die von den Kurven ... begrenzt wird." Daher das Wort *Quadratur*.

Aus Gleichung (61) kann man die Newton–Cotes Formeln berechnen. Im allgemeinen kann man sie durch Substitution unter dem Integral auf Standardform transformieren. Die rationalen Zahlen $A_i$ kann man außerdem auf gemeinsamen Nenner $s$ bringen und tabellieren:

$$\int_a^b f(x)\,dx = (b-a)\int_0^1 f(a+x(b-a))\,dx \approx \frac{b-a}{ns}\sum_{i=0}^{n}\sigma_i f_i := N_1^{(n)}(f),$$

wobei $\sigma_i = sA_i$ und $f_i = f(a+ih)$ mit $h = \frac{b-a}{n}$, weil die Stützstellen äquidistant gewählt sind.

Aus Kapitel 5 Theorem 1.1.9 können wir auch wieder eine Fehlerabschätzung herleiten:

$$\int_a^b f(x)\,dx - N_1^{(n)}(f) = -h^{p+1}Kf^{(p)}(\xi);$$

dabei hängen $K$ und $p$ von $n$ aber nicht von $f$ ab. Wir sagen, daß die Quadraturformel die Ordnung $p$ besitzt, wenn $p$ die größte ganze Zahl ist, sodaß alle Polynome kleineren Grades exakt integriert werden.

Für $n = 1, 2, \ldots, 6$ erhalten wir die folgenden Newton–Cotes Formeln:

| $n$ | $\sigma_i$ | | | | | | | $ns$ | Fehler | Name |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | | | | | | 2 | $-h^3\frac{1}{12}f^{(2)}(\xi)$ | Trapezregel |
| 2 | 1 | 4 | 1 | | | | | 6 | $-h^5\frac{1}{90}f^{(4)}(\xi)$ | Simpson–Regel |
| 3 | 1 | 3 | 3 | 1 | | | | 8 | $-h^5\frac{3}{80}f^{(4)}(\xi)$ | 3/8–Regel |
| 4 | 7 | 32 | 12 | 32 | 7 | | | 90 | $-h^7\frac{8}{945}f^{(6)}(\xi)$ | Milne–Regel |
| 5 | 19 | 75 | 50 | 50 | 75 | 19 | | 288 | $-h^7\frac{275}{12096}f^{(6)}(\xi)$ | |
| 6 | 41 | 216 | 27 | 272 | 27 | 216 | 41 | 840 | $-h^9\frac{9}{1400}f^{(8)}(\xi)$ | Weddle–Regel |

Für größere Werte von $n$ treten negative Gewichte $\sigma_i$ auf, und die Formeln werden instabil, weil sich kleine Änderungen von $f$ auf die Summe stärker als auf das Integral auswirken und Auslöschung auftritt.

**4.2. Andere Polynom-basierte Quadraturformeln.** Verwendet man anstelle der Newtonschen Interpolationspolynome Hermitsche Interpolation, dann kann man unter Verwendung von Ableitungen des Integranden $f$ die Ordnung der Quadraturformeln weiter erhöhen. Im einfachsten Fall, wenn man nur die Randpunkte und die Ableitungen am Rand vorschreibt, erhält man die Integrationsregel

$$\int_a^b f(x)\,dx \approx M_1(f) = \tfrac{h}{2}(f(a)+f(b)) + \tfrac{h}{12}(f'(a)-f'(b)),$$

eine Quadraturformel vierter Ordnung wie die Fehlerabschätzung

$$\int_a^b f(x)\,dx - M_1(f) = -\frac{h^5}{720}f^{(4)}(\xi)$$

zeigt. Man kann natürlich auch nicht-äquidistante Stützstellen $x_i$ verwenden; die beste Wahl führt in diesem Fall zu den Integrationsregeln von Gauß, die in Abschnitt 5 besprochen werden.

Andere Integrationsformeln verwenden Stützstellen, die entweder außerhalb des Integrationsintervalles liegen, oder die ein Intervall überdecken, das kleiner ist als das Integrationsintervall. Viele dieser Regeln wurden in der Vergangenheit untersucht, doch heute sind alle diese Formeln ungebräuchlich und werden kaum mehr verwendet.

**4.3. Spline-basierte Quadraturformeln.** Anstelle von Interpolationspolynomen kann man den interpolierenden Spline zu den Stützstellen $(x_i, f(x_i))$, $i = 0, \dots, n$, bestimmen und integrieren. Am günstigsten wählt man dabei die beiden Zusatzbedingungen $s'(a) = f'(a)$ und $s'(b) = f'(b)$ und die dazugehörigen kubischen Splines. Aus der Darstellung von $s$ mit Hilfe der Momente $f_i''$ kann man die Quadraturformel schreiben als

$$\int_a^b f(x)\,dx \approx \frac{h}{2}\left(f(a) + 2\sum_{i=1}^{n-1} f(x_i) + f(b)\right) - \frac{h^3}{24}\left(f_0'' + 2\sum_{i=1}^{n-1} f_i'' + f_n''\right).$$

Der erste Term ist gleich der zusammengesetzten Trapezregel. Den Korrekturterm berechnet man aus dem Gleichungssystem für die $f_i''$ aus Kapitel 5 Abschnitt 4.3.

## 5. Gauß–Quadratur

Kommen wir nun zu der Grundlage für die heute gebräuchlichsten Integrationsmethoden. Die Idee basiert auf folgender Überlegung: Bei den Newton–Cotes Formeln kann man durch die Wahl der $n+1$ Koeffizienten $A_i$ eine Quadraturformel definieren, die Polynome bis zum Grad $n$ exakt integriert. Man hat jedoch mehr als $n+1$ freie Parameter. Fügt man zur Liste der Parameter nicht nur die Koeffizienten $A_i$ sondern auch die Stützabszissen $x_i$ hinzu, so sollte man eine Quadraturformel konstruieren können, die Polynome bis zum Grad $2n+1$ exakt integriert. Der Weg zu dieser Konstruktion führt über orthogonale Polynome.

Gleichzeitig ermöglicht dieser Zugang, das zu lösende Problem etwas zu verallgemeinern. Sei $w : \mathbb{R} \to \mathbb{R}$ eine Gewichtsfunktion, die auf $(a, b)$ positiv ist. Dann definiert

$$\langle f, g\rangle := \int_a^b f(x)g(x)w(x)\,dx$$

ein inneres Produkt auf $C[a, b]$. Nachdem Integralausdrücke wie oben im Verlauf dieses Kapitels öfters auftreten, führen wir noch die folgende Notation ein:

$$\int_a^b a(x)w(x)\,dx =: \int a.$$

Die Aufgabe in diesem Abschnitt sei, den Ausdruck $\int f$ für Funktionen $f$ möglichst gut durch einen Ausdruck der Form

$$\int f \approx \sum_{i=0}^{n} A_i f(x_i)$$

zu approximieren.

**5.1. Orthogonale Polynome.** Nachdem der oben definierte Ausdruck $\langle f, g\rangle$ ein inneres Produkt auf $C[a, b]$ definiert, induziert er auch ein inneres Produkt auf dem Raum $\mathbb{R}[x]$ der Polynome. Ausgehend von der Basis $\{1, x, x^2, x^3, \dots\}$ kann man mit Hilfe des Gram–Schmidtschen Orthogonalisierungsverfahrens eine Orthogonalbasis $\{p_0, p_1, \dots\}$ des Raumes $\mathbb{R}[x]$ bezüglich $\langle\ ,\ \rangle$ konstruieren. Die Basiselemente $p_i$ seien o.B.d.A. monisch ($p(x) = x^n + a_{n-1}x^{n-1} + \cdots + a_1x + a_0$); sie heißen *orthogonale Polynome*.

**Proposition 5.1.1.** *Die Orthogonalen Polynome $\{p_0, p_1, \dots\}$ bezüglich $\langle\ ,\ \rangle$ haben folgende Eigenschaften.*

(1) *Der Grad von $p_k$ ist $k$,*
(2) *$p_{n+1}$ steht orthogonal auf alle Polynome vom Grad $\le n$,*
(3) *$p_0(x) \equiv 1$,*
(4) *$p_1(x) = x - \alpha_1$ mit*

$$\alpha_1 = \frac{\int x}{\int 1}.$$

(5) *Für $n \geq 1$ gilt die Drei–Terme–Beziehung*

$$p_{n+1}(x) = xp_n(x) - \alpha_{n+1}p_n(x) - \beta_{n+1}p_{n-1}(x)$$

*mit*

$$\alpha_{n+1} = \frac{\int xp_n^2}{\int p_n^2}, \quad \beta_{n+1} = \frac{\int xp_np_{n-1}}{\int p_{n-1}^2}.$$

(6) *Für $n \geq 1$ sind die Nullstellen von $p_{n+1}$ allesamt einfach und reell. Außerdem liegen alle im Intervall $[a, b]$.*

BEWEIS. (1) Nach dem Gram–Schmidt–Verfahren gilt die Beziehung

$$p_n = x^n - \sum_{i=0}^{n-1} \langle p_i, x^n \rangle p_i.$$

Der Rest des Beweises folgt mittels vollständiger Induktion. $p_0 \equiv 1$, und daher ist $\operatorname{grad} p_0 = 0$. Sei die Behauptung für $k$ bewiesen. Dann folgt aus obiger Formel, daß $\operatorname{grad} p_{k+1} = \operatorname{grad} x^{k+1} = k+1$, weil von $x^{k+1}$ nur Terme kleineren Grades abgezogen werden.

(2) Sei $q$ ein Polynom vom Grad $\leq n$. Dann läßt sich $q$ entwickeln in der Basis der orthogonalen Polynome

$$q(x) = \sum_{i=0}^{k} a_i p_i(x),$$

wobei $k = \operatorname{grad} q$ gilt, wegen 1. Aus dieser Beziehung folgt

$$\langle q, p_{n+1} \rangle = \sum_{i=0}^{k} a_i \langle p_i, p_{n+1} \rangle = 0,$$

wegen der Orthogonalitätseigenschaft der Polynome $p_i$.

(3) ist offensichtlich.

(4) $p_1(x) = x - \alpha_1$, und aus der Orthogonalitätsbeziehung

$$0 = \langle 1, p_1 \rangle = \int p_1 = \int x - \alpha_1 \int 1$$

folgt die Behauptung.

(5) Nachdem $p_{n+1}$ als monisch vorausgesetzt wird, hat das Polynom $p_{n+1} - xp_n$ den Grad $n$ und läßt sich somit in die Basiselemente $\{p_0, p_1, \ldots, p_n\}$ entwickeln. Setzen wir die Entwicklung an als

$$p_{n+1} = xp_n - \alpha_{n+1}p_n - \beta_{n+1}p_{n-1} - \sum_{i=0}^{n-2} \gamma_{i,n+1}p_i,$$

so können wir wieder die Orthogonalitätsbeziehungen verwenden (es gelte $j \leq n-2$):

$$\begin{aligned}
0 = \langle p_{n+1}, p_n\rangle &= \int xp_n^2 - \alpha_{n+1}\int p_n^2 - \beta_{n+1}\langle p_{n-1}, p_n\rangle - \sum_{i=0}^{n-2}\gamma_{i,n+1}\langle p_i, p_n\rangle = \\
&= \int xp_n^2 - \alpha_{n+1}\int p_n^2, \\
0 = \langle p_{n+1}, p_{n-1}\rangle &= \int xp_np_{n-1} - \alpha_{n+1}\langle p_n, p_{n-1}\rangle - \beta_{n+1}\int p_{n-1}^2 - \\
&\quad - \sum_{i=0}^{n-2}\gamma_{i,n+1}\langle p_i, p_{n-1}\rangle = \\
&= \int xp_np_{n-1} - \beta_{n+1}\int p_{n-1}^2, \\
0 = \langle p_{n+1}, p_j\rangle &= \int xp_np_j - \alpha_{n+1}\langle p_n, p_j\rangle - \beta_{n+1}\langle p_{n-1}, p_j\rangle - \sum_{i=0}^{n-2}\gamma_{i,n+1}\langle p_i, p_j\rangle = \\
&= \int xp_jp_n - \gamma_{j,n+1}\int p_j^2.
\end{aligned}$$

Aus den ersten beiden Gleichungen folgen die behaupteten Beziehungen für $\alpha_{n+1}$ und $\beta_{n+1}$. Die dritte Gleichung impliziert für $j \leq n-2$

$$\gamma_{j,n+1} = \frac{\langle xp_j, p_n\rangle}{\int p_j^2} = 0,$$

weil $xp_j$, ein Polynom vom Grad $j+1 \leq n$, nach 2 orthogonal auf $p_n$ steht.

(6) Seien $x_0, x_1, \ldots, x_k$ die Nullstellen ungerader Vielfachheit von $p_{n+1}$, die in $[a,b]$ liegen. Können wir $k = n$ zeigen, so sind wir fertig, da die $x_i$ dann die $n+1$ Nullstellen von $p_{n+1}$ sind, damit ist ihre Vielfachheit eins. Nehmen wir also an, daß $k < n$ gilt. Betrachte

$$q(x) = (x-x_0)(x-x_1)\ldots(x-x_k).$$

Dann gilt $\operatorname{grad} q = k < n$, und daher folgt aus 2

$$\langle p_{n+1}, q\rangle = \int p_{n+1}q = 0. \tag{62}$$

Nachdem jede Nullstelle von $p_{n+1}$, die nicht gleich einer der Zahlen $\{x_0, \ldots, x_k\}$ ist, entweder gerade ist oder außerhalb von $[a,b]$ liegt, wechselt $p_{n+1}$ das Vorzeichen in $[a,b]$ genau an den Punkten $\{x_0, \ldots, x_k\}$. Da auch das Polynom $q$ nach Konstruktion an genau diesen Punkten das Vorzeichen wechselt, hat $qp_{n+1}$ auf ganz $[a,b]$ dasselbe Vorzeichen. Weil $w$ positiv ist, gilt

$$\int p_{n+1}q = \int_a^b p_{n+1}(x)q(x)w(x)\,dx \neq 0,$$

ein Widerspruch zu (62). Daher ist $k = n$, und die Behauptung ist bewiesen.

□

Für verschiedene Gewichtsfunktionen $w$ und Intervalle $[a,b]$ sind die orthogonalen Polynome tabelliert und haben sogar Namen erhalten:

| $w(x)$ | $[a,b]$ | Name |
|---|---|---|
| $1$ | $[-1,1]$ | Legendre |
| $1/\sqrt{1-x^2}$ | $[-1,1]$ | Tschebyscheff, erster Art |
| $\sqrt{1-x^2}$ | $[-1,1]$ | Tschebyscheff, zweiter Art |
| $(1-x)^\alpha(1+x)^\beta$, $\alpha,\beta>-1$ | $[-1,1]$ | Jacobi |
| $e^{-x}$ | $[0,\infty)$ | Laguerre |
| $x^\alpha e^{-x}$, $\alpha>-1$ | $[0,\infty)$ | Laguerre, verallgemeinert |
| $e^{-x^2}$ | $(-\infty,\infty)$ | Hermite |

Von diesen sind die Legendre–, die Laguerre–, die Tschebyscheff– und die Hermitepolynome wahrscheinlich die bekannteren.

**5.2. Gauß–Quadratur.** Die Eigenschaften der orthogonalen Polynome ermöglichen jetzt, eine interpolatorische Quadraturformel

$$\int f \approx \sum_{i=0}^{n} A_i f(x_i)$$

zu konstruieren, die Polynome vom Grad $\leq 2n+1$ exakt integriert.

**Theorem 5.2.1.** *Seien $\{x_0,\dots,x_n\}$ die Nullstellen des orthogonalen Polynoms $p_{n+1}$, und sei*

$$A_i = \int L_i, \quad i=0,1,\dots,n,$$

*wobei $L_i$ das $i$–te Lagrange Polynom über $x_0,x_1,\dots,x_n$ ist. Für eine beliebige Funktion $f$ sei*

$$G_n f := \sum_{i=0}^{n} A_i f(x_i).$$

*Dann gilt für alle Polynome $p$ vom Grad $\leq 2n+1$*

$$\int p = G_n p.$$

Beweis. Nachdem $G_n$ eine interpolatorische Quadraturformel mit $n+1$ Abszissen ist, integriert $G_n$ alle Polynome vom Grad $\leq n$ exakt. Sei also $p$ ein Polynom mit Grad $\leq 2n+1$. Dann kann man $p$ durch $p_{n+1}$ dividieren und erhält eine Gleichung

$$p = p_{n+1}q + r$$

mit Polynomen $q$ und $r$ vom Grad $\leq n$. Daher folgt

$$\begin{aligned}
G_n p &= \sum_{i=0}^{n} A_i p(x_i) \\
&= \sum_{i=0}^{n} A_i (p_{n+1}(x_i) q(x_i) + r(x_i)) \quad \text{nach Konstruktion} \\
&= \sum_{i=0}^{n} A_i r(x_i) \quad \text{weil } p_{n+1}(x_i) = 0 \\
&= G_n r \\
&= \int r \quad \text{weil } G_n \text{ Polynome vom Grad} \leq n \text{ exakt integriert} \\
&= \int (p_{n+1} q + r) \quad \text{wegen Proposition 5.1.1.2} \\
&= \int p.
\end{aligned}$$

□

Eine wichtige Folgerung aus diesem Resultat, das die Stabilität der Gauß–Formeln impliziert, ist die Positivität der Koeffizienten $A_i$. Es folgt nämlich

$$L_i(x_j) = L_i^2(x_j) = \begin{cases} 0 & \text{wenn } i \neq j, \\ 1 & \text{wenn } i = j. \end{cases}$$

Wegen $L_i^2(x) \geq 0$ und $\operatorname{grad}(L_i^2) = 2n < 2n+1$ gilt

$$0 < \int L_i^2 = G_n L_i^2 = \sum_{j=0}^{n} A_j L_i^2(x_j) = A_i.$$

Nachdem $A_0 + \cdots + A_n = \int 1$, ist kein Koeffizient größer als $\int 1$. Das hat zur Konsequenz, daß keine einzelnen Werte $f(x_i)$ zu stark gewichtet werden, woraus die Gutartigkeit der Gaußschen Integrationsmethode folgt. Speziell gilt

**Theorem 5.2.2.** *Der Fehlerterm der Gaußschen Quadraturformeln beträgt*

$$\int f - G_n f = \frac{f^{(2n+2)}(\xi)}{(2n+2)!} \int p_{n+1}^2,$$

*sofern $f \in C^{2n+2}[a,b]$, wobei $\xi \in [a,b]$. Ferner folgt aus der Positivität der Koeffizienten $A_i$ und dem Satz von Weierstraß für stetige Funktionen $f$*

$$\lim_{n\to\infty} G_n f = \int f.$$

**5.3. Gauß–Kronrod Formeln.** Wenn man die verschiedenen anderen Integrationsmethoden, wie Extrapolationsverfahren und Newton–Cotes Formeln, mit der Gaußschen Integration vergleicht, so findet man aus den Fehlerformeln, daß die Gauß–Quadratur die höchste Ordnung besitzt und daher wahrscheinlich auch die genauesten Resultate liefert. Möchte man jedoch ein gegebenes Integral mit der Genauigkeit $\varepsilon$ approximieren, so findet man schnell heraus, wie schwierig es ist, das korrekte $n$ herauszufinden, für das $\int f - G_n f < \varepsilon$ gilt. Das liegt vor allem an der Tatsache, daß sich $f^{(2n+2)}$ nur schwer über $[a,b]$ abschätzen läßt.

Geht man aufgrund dieser Schwierigkeiten dazu über, der Reihe nach mehrere $G_k f$ zu berechnen, so stößt man schnell auf das Problem, daß (eventuell bis auf den Intervallmittelpunkt) keine zwei Gaußformeln dieselben Abszissen $x_j$ besitzen. Man kann also die bereits berechneten Funktionswerte nicht wiederverwenden. Diese Eigenschaft der Gaußschen Formeln bedingt, daß in der praktischen Anwendung die theoretischen Vorteile dieser Methode sehr schnell verloren gehen.

Eine Lösung dafür schlug [**Kronrod 1965**] vor: Man gibt im nächsten Schritt die $n$ bereits berechneten Abszissen der Formel $G_n$ fest vor und konstruiert eine $(2n+1)$–Punktformel, die die dann größtmögliche Ordnung $3n+1$ besitzt für $n$ gerade, bzw. $3n+2$ für $n$ ungerade. Die neuen Abszissen liegen in den Intervallen

$$(a, x_0), (x_0, x_1), \ldots, (x_{n-1}, x_n), (x_n, b).$$

Im nächsten Schritt erhöht man $n$ wieder bei Beibehaltung der bereits berechneten Abszissen. Es gibt Softwareprogramme und Tabellen, die die Abszissen und die Koeffizienten $A_i$ berechnen. Gauß–Kronrod Integration ist in den meisten Mathematikpaketen (Mathematica, MAPLE, ...) das bevorzugte Verfahren zur eindimensionalen numerischen Integration.

## 6. Adaptive Verfahren

Eine Möglichkeit die Genauigkeit von Quadraturverfahren zu steigern ist, das Intervall in Teilstücke aufzuteilen, wie wir schon bei der zusammengesetzten Trapezregel gesehen haben. Ein ähnliches Verfahren ist auch bei Verwendung anderer Newton–Cotes Formeln erfolgreich. Leider erhöht eine feinere Unterteilung auch den Rechenaufwand beträchtlich. Die Beobachtung, daß die feine Unterteilung nicht in jedem Teilabschnitt des Intervalls $[a, b]$ notwendig wäre, führt zu den *adaptiven Integrationsverfahren.*

Ein adaptives Verfahren unterteilt das Integrationsintervall rekursiv solange bis in jedem Teilintervall die erforderliche Genauigkeit erreicht ist. In jedem Teil wird zur Berechnung des Integrals eine einfache Newton–Cotes Formel verwendet. Das führt dazu, daß die am Ende gewählte Unterteilung dort fein wird wo der Integrand stark variiert und dort grob bleibt wo der Integrand sehr gut durch das Interpolationspolynom approximiert wird.

Als Abbruchkriterium für die weitere Unterteilung verwendet man zwei verschiedene Näherungswerte $J_1$ und $J_2$ für das Integral über dasselbe Teilstück. Man kann für $J_1$ etwa die Trapezregel und für $J_2$ den Wert der Simpsonregel verwenden. Dann stoppt man die weitere Halbierung, wenn

$$J_1 + \tilde{I} = J_2 + \tilde{I}$$

ist für einen Schätzwert $\tilde{I}$, der die korrekte Größenordnung des Integrales richtig wiedergibt. In [**Gander 1985**] kann man einen Algorithmus finden, der mittels Definition einer rekursiven Prozedur die fortgesetzte Unterteilung durchführt. Möchte man auf rekursive Strukturen verzichten, so kann man ein adaptives Verfahren etwa folgendermaßen formulieren. Dieses Verfahren verwendet die Trapezregel und die Simpsonregel.

**Algorithmus 6.0.1.** *Adaptives Quadraturverfahren*

$a_0 = a$; $a_1 = b$
$f_0 = f(a)$; $f_1 = f(b)$
$\tilde{I} = (b-a)(f_0 + f_1)/2$
$I = 0$
$j = 0$; $k = 1$; $p = 1$; $s = 1$
$i_1 = 1$; `Ein Vektor von Indices, welche Intervalle betrachtet werden`
**for** $i = 0$ **to** `MAXITER` **do**
    $h = a_k - a_j$
    $m = (a_k + a_j)/2$

```
    g = f(m)
    J_1 = h(f_j + f_k)/2
    J_2 = (J_1 + 2hg)/3
    if Ĩ + J_1 = Ĩ + J_2 then
      p = p + 1; s = s + 1
      a_p = m; f_p = g
      k = p; i_s = p
    else
      I = I + J_2
      j = i_s
      s = s − 1
      k = i_s
    endif
    if s = 0 then
      return(I)
    endif
done
```

Die Subunterteilungen, die ein adaptives Integrationsverfahren in etwa erzeugt, sind in Abbildung 6.4 dargestellt.

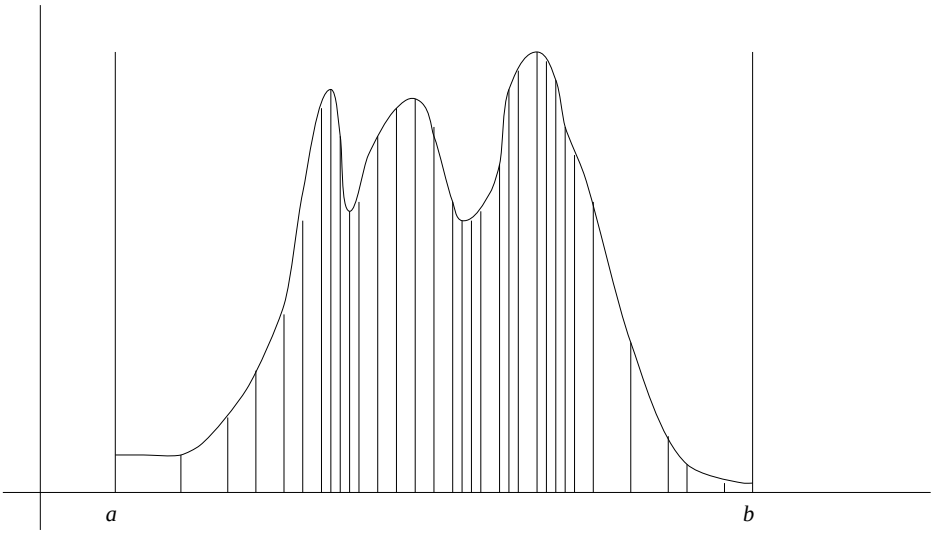



ABBILDUNG 6.4. Adaptives Quadraturverfahren

## 7. Transformationen

Auf die Wichtigkeit, ein Integral vor Anwendung eines der beschriebenen Integrationsverfahren auf eine vernünftige Gestalt zu transformieren, sei explizit hingewiesen. Besonders, wenn der Integrand im Integrationsintervall Singularitäten aufweist, ist Vorsicht geboten.

Die am häufigsten angewendete Transformation ist eine algebraische Substitution, wie im folgenden Beispiel dargestellt:

$$\int_0^1 x^{\frac{p}{q}} f(x)\,dx = q\int_0^1 t^{p+q-1} f(t^q)\,dt.$$

Das erste Integral enthält eine Singularität an 0, falls $p \in \mathbb{Z}$, $q \in \mathbb{N}$ und $p > -q$; $f$ sei analytisch in $[0,1]$. Andererseits hat für $p+q-1 \geq 0$ das zweite Integral keine Singularität.

Mit einer Substitution der Form $x = \tanh t$ kann man unbekannte Randsingularitäten über dem Intervall $[0, 1]$ dadurch auflösen, daß das Integrationsintervall auf $(-\infty, \infty)$ übergeführt wird und die Singularitäten dabei verloren gehen.

Befindet man sich bereits auf einem unbeschränkten Integrationsintervall, fällt aber der Integrand so langsam ab, daß keine Integrationsmethode (Trapezregel, Gaußquadratur) ein vernünftiges Resultat liefert, so kann man mit Hilfe der Transformation $x = \sinh t$ die Fallgeschwindigkeit der Funktion so weit steigern, daß schließlich exponentielles Abklingen eintritt. Es gibt kaum Integranden, für die endlich viele Schritte dieser Substitution nicht ausreichen, um exponentielles Abklingen zu erreichen.

Das sind nur wenige Beispiele von Substitutionen, die man übrigens auch numerisch statt algebraisch durchführen kann. Beim Auftreten von Singularitäten oder stark variierenden Integranden lohnt es sich jedoch immer, zuerst einige Transformationen durchzuführen bevor man ein numerisches Integrationsverfahren anwendet, auch wenn es einige analytische Arbeit benötigt. In diesem Fall wird auch numerisch Integrieren wieder eine Kunst.

## 8. Numerische Differentiation

Wie schon angedeutet, dreht sich numerisch der eingangs zitierte Spruch um: „Numerisch Integrieren ist eine Wissenschaft, Numerisch Differenzieren ist eine Kunst."

In Abschnitt 1 haben wir die beiden Methoden des Differenzierens untersucht, die sich vom mathematischen Standpunkt her am meisten aufdrängen, und wir haben beide als unbefriedigend erkannt. Andererseits gibt es nicht viel Wahl, und so wollen wir versuchen, die beiden Ansätze zu verbessern und zugänglicher bzw. numerisch verwendbarer zu machen.

**8.1. Automatische Differentiation — Vorwärtsmethode.** Wie wir in der Einleitung festgestellt haben, ist ein numerischer Zugang zur Differentiation nicht einfach, und im allgemeinen ist die Genauigkeit, mit der Ableitungen numerisch approximiert werden können, begrenzt. Daher wäre ein Verfahren praktisch, mit dem man die Stärke der symbolischen Differentiation erhalten könnte, ohne tatsächlich ein Programm verwenden zu müssen, das symbolisch rechnen kann.

Der Weg zu diesem Verfahren führt über die Rechenregeln der Differentiation, speziell über die Kettenregel. Die meisten modernen Programmiersprachen erlauben es dem Anwender, die arithmetischen Operatoren umzudefinieren (*operator overloading*) und für neue Zahlentypen zu verwenden.

Dieses Faktum kann man ausnutzen, indem man eine neue Algebra einführt, deren Elemente aus Paaren $(f, f')$ reeller Zahlen bestehen, für die die Rechenoperationen $\circ$ in geeigneter Weise definiert werden. Diese Algebra der *Differentialzahlen* werden wir im folgenden mathematisch exakt definieren.

**Definition 8.1.1.** *Sei $D\mathbb{R}$ die Menge $\mathbb{R} \times \mathbb{R}$ von Paaren reeller Zahlen, im folgenden mit*

$$df = (f, f')$$

*bezeichnet zusammen mit den folgenden arithmetischen Operationen:*

$$\begin{aligned}
df \pm dg &:= (f \pm g, f' \pm g') \\
\lambda df &:= (\lambda f, \lambda f') \\
df * dg &:= (f * g, f' * g + f * g') \\
df/dg &:= (h, (f' - h * g')/h) \quad \textit{mit } h = f/g \textit{ für } g \neq 0 \\
df^n &:= \begin{cases} (h * f, n * h * f') & \textit{mit } h = f^{n-1} \quad \textit{für } 1 \leq n \in \mathbb{R} \\ (h, n * h * f'/f) & \textit{mit } h = f^n \qquad \textit{für } 1 > n \in \mathbb{R},\ f > 0 \end{cases}
\end{aligned}$$

$$df^{dg} := (h, h * (k * g' + g * f'/f)) \quad \textit{mit } k = \log(f),\ h = \exp(g * k) \textit{ für } g > 0$$

*Die Funktionen $\varphi : \mathbb{R} \to \mathbb{R}$ werden auf $D\mathbb{R}$ gemäß folgender Regel fortgesetzt:*

$$\varphi(df) := (\varphi(f), \varphi'(f) * f')$$

*wo $\varphi(f)$ und $\varphi' f$ definiert sind. Zum Beispiel gelten*

$$\begin{aligned}
\operatorname{abs}(df) &:= (\operatorname{abs}(f), \operatorname{sgn}(f) * f') \quad \textit{für } f \neq 0,\\
\sqrt{df} &:= (h, f'/(2h)) \quad \textit{mit } h = \sqrt{f},\\
\exp(df) &:= (h, h * f') \quad \textit{mit } h = \exp(f),\\
\log(df) &:= (\log(f), f'/f) \quad \textit{für } f > 0,\\
\sin(df) &:= \begin{cases} (h, \sqrt{1-h^2} * f') & \textit{mit } h = \sin(f) \quad \textit{für } \sin(f) \geq \sqrt{2}/2 \\ (\sqrt{1-h^2}, h * f') & \textit{mit } h = \cos(f) \quad \textit{für } \sin(f) < \sqrt{2}/2. \end{cases}
\end{aligned}$$

$D\mathbb{R}$ bildet zusammen mit den Operationen $+, -, *$ eine kommutative Algebra mit Einselement $(1, 0)$ (und Nullelement $(0, 0)$). Die Algebra ist nicht nullteilerfrei, da etwa $(0, 1) * (0, 1) = (0, 0)$ gilt. Die Differentialzahlen der Form $a = (a, 0)$ bilde eine Teilalgebra von $D\mathbb{R}$, die isomorph zu $\mathbb{R}$ ist und daher mit $\mathbb{R}$ identifiziert wird. Differentialzahlen dieser Form heißen auch Konstanten.

Um die Differentialzahlen zur Berechnung von Ableitungen beliebiger Funktionen verwenden zu können ist noch die folgende Beobachtung, die in einfacher Weise aus den Definitionen folgt, von Nöten.

**Proposition 8.1.2.** *Seien $f$, $g$ reelle Funktionen $\mathbb{R} \to \mathbb{R}$, die an $x_0 \in \mathbb{R}$ differenzierbar sind, und seien*

$$df = (f(x_0), f'(x_0)), \qquad dg = (g(x_0), g'(x_0))$$

*Differentialzahlen.*

(1) *Sei $\circ$ irgendeine der oben definierten arithmetischen Operationen, und sei die Funktion $p$ definiert als*

$$p(x) := f(x) \circ g(x).$$

*$p$ ist dann offensichtlich bei $x_0$ differenzierbar, und es gilt*

$$(p(x_0), p'(x_0)) = df \circ dg.$$

(2) *Ist $\varphi : \mathbb{R} \to \mathbb{R}$ eine an $x_0$ differenzierbare Funktion, und sei $q$ gegeben durch*

$$q(x) := f(\varphi(x)).$$

*Dann ist auch $q$ bei $x_0$ differenzierbar, und es gilt*

$$(q(x_0), q'(x_0)) = \varphi(df).$$

Aus dieser Proposition folgt als einfaches Korollar folgender zentraler Satz über Differentialzahlen.

**Theorem 8.1.3.** *Sei $f$ ein Ausdruck in einer Variable $x$, zusammengesetzt aus arithmetischen Operationen und elementaren Funktionen, und sei die Differentialzahl $f(dx)$ mit $dx = (x_0, 1)$ definiert. Dann ist die durch $f$ beschriebene Funktion an $x_0$ differenzierbar, und es gilt*

$$(f(x_0), f'(x_0)) = f(dx).$$

Man kann also Differentialzahlen zur Bestimmung der Ableitung einer beliebigen durch einen Ausdruck gegebenen Funktion $f$ verwenden **ohne einen Ausdruck für $f'$ zu kennen**.

Das folgende Beispiel für die Anwendung von Differentialzahlen ist aus [**Neumaier 2000**] genommen.

**Beispiel 8.1.4.** *Angenommen, wir wollen die Ableitung der Funktion*

$$f(x) = \frac{(x-1)(x+3)}{x+2} = x - \frac{3}{x+2}$$

*an der Stelle $x_0 = 3$ bestimmen. Um mit den Ergebnissen der symblischen Differentiation zu vergleichen, bestimmen wir die Ableitungsfunktion*

$$f'(x) = 1 + \frac{3}{(x+2)^2}$$

*Funktionsauswertungen liefern sofort*

$$f(3) = \frac{12}{5} = 2.4, \quad f'(3) = \frac{28}{25} = 1.12.$$

*Verwenden wir nun Differentialzahlen zur Bestimmung von $f'(3)$, so müssen wir $f(dx)$ für $dx = (3,1)$ berechnen. Um die Unabhängigkeit vom arithmetischen Ausdruck zu belegen, berechnen wir für beide oben angegebenen arithmetischen Ausdrücke.*

$$\begin{aligned}(f(3), f'(3)) &= \frac{((3,1)-1) * ((3,1)+3)}{(3,1)+2} = \\ &= \frac{(2,1)*(6,1)}{(5,1)} = \frac{(12,8)}{(5,1)} = \\ &= \left(\frac{12}{5}, \frac{8 - \frac{12}{5} * 1}{5}\right) = \left(\tfrac{12}{5}, \tfrac{28}{25}\right).\end{aligned}$$

*bzw.*

$$\begin{aligned}(f(3), f'(3)) &= (3,1) - \frac{3}{(3,1)+2} = \\ &= (3,1) - \frac{3}{(5,1)} = (3,1) - \left(\frac{3}{5}, \frac{0 - \frac{3}{5} * 1}{5}\right) = \\ &= \left(\tfrac{12}{5}, \tfrac{28}{25}\right).\end{aligned}$$

Die Methode, mittels Differentialzahlen Ableitungen von Funktionen zu berechnen nennt man auch *Vorwärtsmethode der automatischen Differentiation*. Man kann sie sowohl auf höhere Ableitungen als auch auf die Bestimmung partieller Ableitungen (des Gradienten) von Funktionen mehrerer Veränderlicher verallgemeinern.

Hierbei ist die Berechnung höherer Ableitungen $f''(x_0), \ldots, f^{(n)}(x_0)$ mit relativ geringem Aufwand von $n^2 N$ Operationen möglich, falls $N$ Operationen zur Auswertung von $f$ nötig sind.

Die Bestimmung des Gradienten einer mehrdmensionalen Funktion hat dahingehend höheren Aufwand, und in Teil 2 in Kapitel 16 werden wir die Rückwärtsmethode der automatischen Differentiation kennenlernen, die hilft den Aufwand auch für die Bestimmung partieller Ableitungen klein zu halten.

**8.2. Differenzenquotienten — Extrapolation.** Die Differentiationsmethode aus Abschnitt 8.1 funktioniert natürlich nur für Funktionen, die durch einen Ausdruck gegeben sind. In der Praxis trifft man allerdings oftmals auf Funktionen, für die nur die Bestimmung von Funktionswerten möglich ist, oftmals durch einen aufwendigen Algorithmus (etwa das Lösen einer Differentialgleichung).

In der Einleitung haben wir schon den einfachsten Fall zur numerischen Annäherung der Ableitung $f'(x_0)$ zu einer gegebenen Funktion $f$ betrachtet, den *Vorwärtsdifferenzenquotienten.*

$$p(h) := f[x_0, x_0 + h] = \frac{f(x_0 + h) - f(x_0)}{h}$$

Den Approximationsfehler findet man sofort durch Taylorentwicklung

$$p(h) = f'(x_0) + \sum_{i=1}^{k} \frac{f^{(i+1)(x_0)}}{(i+1)!} h^i + O(h^{k+1}), \tag{63}$$

wenn $f$ an $x_0$ mindestens $(k+2)$–mal differenzierbar ist. Wenn die Auswertung von $f$ sehr aufwendig ist, approximiert man mit einem fix gewählten $p(h)$ und handelt sich einen Fehler der Größenordnung $O(h)$ ein. Die damit erzielbare Genauigkeit ist dann begrenzt wegen der in der Einleitung besprochenen Auslöschungseffekte.

Mit einem geeigneten Trick kann man die Approximationseigenschaften verbessern ohne die Anzahl der Funktionsauswertungen von $f$ zu erhöhen. Verwenden wir nämlich den *zentralen Differenzenquotienten*

$$\begin{aligned} p(h^2) = f[x_0 + h, x_0 - h] &= \frac{f(x_0 + h) - f(x_0 - h)}{2h} = \\ &= f'(x_0) + \sum_{i=1}^{k} \frac{f^{(2i+1)}(x_0)}{(2i+1)!} h^{2i} + O(h^{2k+2}), \end{aligned} \tag{64}$$

so können wir die Ableitung mit dem kleineren Fehler $O(h^2)$ approximieren, falls $f$ wenigstens zweimal differenzierbar ist.

**Beispiel 8.2.1.** *Wir verwenden wieder die Funktion $f(x) = x^3$ wie in der Einleitung, um die Ergebnisse vergleichen zu können.*

| $h$ | $((1+h)^3 - (1-h)^3)/(2h)$ |
|---|---|
| 1 | 4 |
| $10^{-1}$ | 3.010000000000002 |
| $10^{-2}$ | 3.000100000000006 |
| $10^{-3}$ | 3.000000999999863 |
| $10^{-4}$ | 3.000000009999448 |
| $10^{-5}$ | 3.000000000097369 |
| $10^{-6}$ | 2.999999999919734 |
| $10^{-7}$ | 3.000000000086267 |
| $10^{-8}$ | 2.999999992869817 |
| $10^{-9}$ | 3.000000081687659 |
| $10^{-10}$ | 3.000000248221113 |
| $10^{-11}$ | 3.000000248221113 |
| $10^{-12}$ | 3.000100168293329 |
| $10^{-13}$ | 2.99926750102486 |
| $10^{-14}$ | 2.997602166487923 |
| $10^{-15}$ | 3.164135620181696 |
| $10^{-16}$ | 1.665334536937735 |

In dem Beispiel erkennen wir, daß die Genauigkeit besser wird, bei $h = 10^{-6}$ beträgt die größte Genauigkeit 10 signifikante Stellen. Wir haben also zwei signifikante Stellen gewonnen. Es treten aber wieder genau die gleichen Auslöschungseffekte auf wie für den Vorwärtsdifferenzenquotienten.

Will man dieses Problem umgehen, ist es notwendig, die Funktion öfter als zweimal auszuwerten. Kann man sich das leisten, so bietet sich folgende Betrachtungsweise an. Die Taylorentwicklungen in den Gleichungen (63) und (64) deuten darauf hin, daß sich die Funktion $p(h)$ wie ein Polynom bei $h = 0$ verhält. Es ist also wie bei der Idee des Rombergverfahrens (siehe Abschnitt 3.2) einen Versuch wert, den $p(0)$ durch Polynomextrapolation zu schätzen und zu hoffen, daß die Approximation besser ist als die Approximationen für die einzelnen $h$.

Dies ist im allgemeinen auch der Fall, wie wir aus folgenden einfachen Rechnungen sehen können:

**Beispiel 8.2.2.** *Wir berechnen ein letztes Mal die Ableitung von $f(x) = x^3$ an $x = 1$. Diesmal verwenden wir die Werte für $h = 2^{-1}, 2^{-2}, 2^{-3}, 2^{-4}$ und den Neville-artigen Algorithmus zur Polynomextrapolation. Mit den Vorwärtsdifferenzenquotienten erhalten wir den Schätzwert $f'(1) = 3$, mit 15 richtigen signifikanten Stellen, errechnet aus 4 Funktionswerten von $f$, von denen keiner als Approximation getaugt hätte ($p(2^{-4}) = 3.19140625$). Das ist natürlich auch durch die Einfachheit der Funktion $f$ bedingt, doch im allgemeinen ist derselbe Effekt bemerkbar.*

Die zentralen Differenzenquotienten liefern noch schneller noch genauere Approximationen, doch man muß für eine Extrapolation mit $n$ Differenzenquotienten immerhin $2n$ Funktionswerte berechnen statt $n + 1$ Werte für die Vorwärtsdifferenzenquotienten.

**Bemerkung 8.2.3.** *Eine ausführliche Fehleranalyse, die man etwa in* [**Neumaier 2000**, 3.2] *nachschlagen kann liefert für die bezüglich Genauigkeit optimale Schrittweite $h$ den Schätzwert*

$$h_{opt} = \left| \frac{f(x)}{f[x + h_0, x - h_0]} \right| \mathsf{eps}^{1/2}$$

*für den Vorwärtsdifferenzenquotienten, und die maximal erreichbare Rechengenauigkeit ist $O(\sqrt{\mathsf{eps}})$. Für die zentralen Differenzenquotienten wählt man*

$$h_{opt} = \left| \frac{f(x)}{f[x + h_0, x - h_0]} \right| \mathsf{eps}^{1/3}$$

*und erhält eine Genauigkeit von $O(\mathsf{eps}^{2/3})$. Verwendet man Extrapolation mit 5 Werten von $p(h)$ mit $h = h_0 2^{-i}$, $i = 0, \ldots, 4$, dann erhält man eine Maximalgenauigkeit von $O(\mathsf{eps}^{5/6})$ bei der optimalen Wahl von $h_0 = O(\mathsf{eps}^{1/5})$. Für Extrapolation von $p(h^2)$ steigt die Maximalgenauigkeit auf den beinahe perfekten Wert $O(\mathsf{eps}^{11/10})$, wobei man am günstigsten $h = O(\mathsf{eps}^{1/10})$ wählt.*

Für die Approximation höherer Ableitungen wählt man höhere dividierte Differenzen, und wieder ist Extrapolation die beste Methode zur Schätzung der Ableitung. Für zweite Ableitungen verwendet man den höheren zentralen Differenzenquotienten

$$p(h^2) := 2f[x - h, x, x + h] = \frac{f(x + h) - 2f(x) + f(x - h)}{h^2},$$

und bei $h = O(\mathsf{eps}^{1/4})$ ist der Approximationsfehler $O(\mathsf{eps}^{1/2})$. Mit quadratischer Approximation (3 Werte) verbessert sich das zu $h = O(\mathsf{eps}^{1/8})$ mit einer Fehlergröße von $O(\mathsf{eps}^{3/4})$.

Für dritte Ableitungen erhalten wir

$$p(h^2) := 6f[x-2h, x-h, x+h, x+2h] = \frac{\frac{1}{2}(f(x + 2h) - f(x - 2h)) - (f(x + h) - f(x - h))}{h^3},$$

und der Fehler ist von Größenordnung $O(\mathsf{eps}^{2/5})$ bei optimaler Wahl von $h = O(\mathsf{eps}^{1/5})$. Wieder verbessert Extrapolation diese Werte.

Man bemerke, daß $h$ für höhere Ableitungen immer größer gewählt werden muß, da die Auslöschungseffekte sich viel stärker auswirken. Gleichzeitig werden die Fehlerordnungen größer und die Notwendigkeit zur Extrapolation stärker.

Wie beim Integrieren liefert darüber hinaus rationale Extrapolation noch bessere Ergebnisse als Polynomextrapolation, doch sei hier keine genaue Fehleranalyse angegeben, da das den Rahmen der Vorlesung sprengen würde.

## 9. Software

Wie bei den Interpolationsproblemen gibt es große Mengen an Standardsoftware zusätzlich zu den in Mathematica bzw. MAPLE enthaltenen Gauß–Kronrod Verfahren.

Sowohl NAG als auch IMSL, SLATEC und CMLIB beinhalten eine große Sammlung spezieller Integrationsprogramme. Außerdem steht mit `QUADPACK` eine eigene Bibliothek zur Verfügung, die so auf Integration spezialisiert ist, wie es `LAPACK` auf lineare Algebra ist.

Bei den Software–Integrationsprogrammen ist es jedoch besonders wichtig, die richtige Funktion für den jeweiligen Integranden zu wählen.

Es gibt in allen Sammlungen Funktionen für global adaptive Unterteilungsstrategien (`QUADPACK/qag`, `NAG/d01aue`, `IMSL/MATH-LIBRARY/qdag`), für Integranden mit Singularitäten unbekannten Typs (`QUADPACK/qags`, `NAG/d01aje`, `IMSL/MATH-LIBRARY/qdags`), für Singularitäten mit bekannter Lage (`QUADPACK/qagp`, `NAG/d01ale`, `IMSL/MATH-LIBRARY/qdagp`), für algebraisch–logarithmische Endpunktsingularitäten (`IMSL/MATH-LIBRARY/qdaws`, `NAG/d01ape`, `QUADPACK/qaws`), für unendliche Integrationsbereiche (`IMSL/MATH-LIBRARY/qdagi`, `NAG/d01ame`, `QUADPACK/qagi`) und für einige weitere Quadraturaufgaben wie verschiedene Gewichtsfunktionen und Cauchysche Hauptwerte. Beinahe alle diese Programme verwenden Gauß–Kronrod Formeln oder Abwandlungen bzw. spezielle Methoden zur Behandlung der Singularitäten.

KAPITEL 7

# Nichtlineare Gleichungssysteme I: Eindimensionaler Fall, Nullstellen

## 1. Grundlagen

Die Lösung von Gleichungssystemen ist eine der wichtigsten Teilaufgaben der numerischen Mathematik. Den einfachsten Fall bilden die linearen Gleichungssysteme, die wir mit den Methoden der numerischen linearen Algebra in den Kapiteln 3 und 4 algorithmisch gelöst haben.

Nichtlineare Gleichungssysteme sind eine Stufe schwieriger, da die Bandbreite der Probleme viel größer ist. Lineare Gleichungssysteme ähneln einander sehr stark, und man kann den Koeffizienten der Variablen eindeutig ansehen, ob das Problem Lösungen hat, und ob die Lösung eindeutig ist. Die einzigen Schwierigkeiten des linearen Falls sind Rundungsfehler.

Nichtlineare Gleichungen präsentieren sich ganz anders. Es genügt nicht *ein* Problem zu untersuchen, um Informationen über alle anderen zu erhalten. Es ist algorithmisch möglich, wie wir in diesem Kapitel sehen werden, einzelne Lösungen der Gleichung zu finden, doch bereits die Untersuchung, ob Lösungen existieren und wie viele Lösungen existieren ist ein extrem schwieriges Problem. In Teil 2 Kapitel 13, wenn wir den mehrdimensionalen Fall behandeln, werden wir dazu kurze Untersuchungen anstellen, doch es wird erst in Teil 2 Kapitel 18 möglich sein, Algorithmen zu entwickeln, die sicherstellen können, alle Lösungen eines nichtlinearen Gleichungssystems zu finden.

In diesem Abschnitt wollen wir jedoch mit dem einfachsten Fall beginnen und uns auf nichtlineare Gleichungen in einer Variablen beschränken. Ferner wollen wir uns damit begnügen Approximationen einzelner Lösungen zu finden ohne Wert auf die Vollständigkeit der Lösungsmenge zu legen.

**1.1. Problemstellung.** Sei $f : \mathbb{R} \to \mathbb{R}$ eine Funktion. Wir suchen zu einem $\eta \in \mathbb{R}$ *ein* $\xi \in \mathbb{R}$ mit $f(\xi) \approx \eta$. Doch sogar dieser einfache Fall ist für eine numerische Behandlung noch zu allgemein. Wir werden für die Funktion $f$ mindestens noch *stetig* voraussetzen. Außerdem können wir die Gleichung $f(\xi) = \eta$ umformen zu $f(\xi) - \eta = 0$, und auf diese Weise kommen wir zu der folgenden Problemstellung

> Sei $f : \mathbb{R} \to \mathbb{R}$ stetig. Wir suchen ein $\xi \in \mathbb{R}$ mit $f(\xi) = 0$.

Wegen der Umformung auf „ $= 0$“ bedeutet die Lösung einer nichtlinearen Gleichung also das Auffinden einer Nullstelle der Funktion $f$, es wir daher auch (eindimensionales, univariates) *Nullstellenproblem* genannt.

Trotz seiner Eingeschränktheit tritt bereits das eindimensionale Nullstellenproblem in der numerischen Mathematik an vielen Stellen als wichtiger Teil anderer Verfahren auf. So ist etwa

(1) Ein Minimum oder Maximum einer Funktion $f$ über einem Intervall $[a, b]$ entweder einer der Randpunkte oder eine Nullstelle der Ableitungsfunktion $f'(x)$, die in $]a, b[$ liegt.

(2) Die Singularitäten einer Funktion $f : \mathbb{R} \to \mathbb{R}$, also die Punkte $\xi \in \mathbb{R}$ mit $\lim_{x\to\xi} f(x) = \pm\infty$, lassen sich als Nullstellen der reziproken Funktion

$$g(x) = \begin{cases} \frac{1}{f(x)} & x \neq \xi \\ 0 & x = \xi \end{cases}$$

finden.

(3) Die Eigenwerte einer Matrix $A$ sind die Nullstellen ihres charakteristischen Polynoms $p(\lambda) = \det(A - \lambda\mathbb{I})$. Dieser Fall wird aber üblicherweise nicht direkt als Nullstellenproblem behandelt sondern mit Hilfe anderer Methoden bewältigt, wie in Kapitel 9 beschrieben.

(4) Wie wir in Teil 2 Kapitel 19 sehen werden, kann man Randwertprobleme der Form

$$\begin{aligned} y''(x) = f(x, y, y'), &\quad y : [a, b] \to \mathbb{R} \\ y(a) = y_a, &\quad y(b) = y_b, \end{aligned} \tag{65}$$

wenn sie lösbar sind, durch Betrachtung der entsprechenden Anfangswertprobleme

$$\begin{aligned} y_t''(x) = f(x, y_t, y_t'), &\quad y_t : [a, b] \to \mathbb{R} \\ y_t(a) = y_a, &\quad y_t'(a) = t, \end{aligned}$$

für verschiedene Parameter $t$ behandeln. In diesem Fall ist offenbar $y_t$ eine Lösung von 65, wenn $t$ eine Nullstelle der Funktion

$$h(t) = y_t(b) - y_b$$

ist.

Wie wir sehen, können Funktionsauswertungen von $f$ einfach oder sehr aufwendig sein. Es kann eine explizite Formel von $f$ bekannt sein, oder es ist wie in (4) nur ein Algorithmus zur Auswertung von $f$ gegeben. Ist kein expliziter arithmetischer Ausdruck für $f$ bekannt, so ist es außerdem meist nicht möglich, die Ableitung von $f$ zu bestimmen.

Unser Ziel wird es also sein, Algorithmen zur Nullstellensuche zu finden, die möglichst wenige Funktionswerte verwenden und möglichst keine Ableitungsinformation benutzen.

## 2. Bisektionsverfahren

**2.1. Grundlagen.** Der einfachste Algorithmus zur Nullstellensuche basiert auf einer wichtigen, aus der Analysis bekannten Eigenschaft der reellen Zahlen, dem Intervallschachtelungsprinzip, und einer Eigenschaft reeller, stetiger Funktionen, dem Zwischenwertsatz.

Sei $f$ wieder die zu untersuchende stetige reelle Funktion. Nehmen wir an, wir kennen zwei Punkte $a$ und $b$, an denen $f$ unterschiedliches Vorzeichen hat, d.h. für die $f(a)f(b) < 0$ gilt. In einem solchen Fall folgt aus dem Zwischenwertsatz, daß $f$, das ja im Intervall $[a, b]$ das Vorzeichen wechselt, wenigstens eine Nullstelle in $]a, b[$ besitzt. Das Paar $(a, b)$ wird in diesem Fall auch als *Klammer* (*Einschließung*) bezeichnet.

Wiederholte Bisektion an geschickt gewählten Teilungspunkten erzeugt eine Folge immer kleiner werdender Intervalle, die nach dem Intervallschachtelungsprinzip am Ende eine Nullstelle von $f$ enthalten wird.

**2.2. Mittelpunktbisektion.** Im einfachsten Fall, wenn man außer Funktionswerten keine weitere Information über $f$ besitzt, wählt man als Teilungspunkt in jedem Schritt den Mittelpunkt des Einschließungsintervall. Das führt zum äußerst stabilen (aber auch langsamen) Algorithmus der **Mittelpunktbisektion**.

**Algorithmus 2.2.1.** *Mittelpunktbisektion*

*f=f(a)*
*fh=f(b)*
**if** $f * fh > 0$ **then**
  *error "Kein Vorzeichenwechsel in* $[a, b]$*"*
**endif**
**if** $f > 0$ **then**
  *x=a*
  *a=b*
  *b=x* ; `jetzt gilt` $f(a) \leq 0 \leq f(b)$
**endif**
*i=2*
**if** $f == 0$ **then**
  *x=a*
**else**
  *x=(a+b)/2*
  *i=i+1*
  *f=f(x)*
  **while** $|a - b| \geq \varepsilon$ **do**
    **if** $f \geq 0$ **then**
      *b=x*
    **endif**
    **if** $f \leq 0$ **then**
      *a=x*
    **endif**
    *x=(a+b)/2*
    *i=i+1*
    *f=f(x)*
  **done**
**endif**
**stop** *"f hat eine Nullstelle* $\xi$ *mit* $|\xi - x| \leq \frac{1}{2}\varepsilon$*".*

**Beispiel 2.2.2.** *Wir verwenden die Funktion* $f(x) = x^2 - 5$*, um ausgehend von den Startwerten* $a = 1$ *und* $b = 3$ *die Nullstelle* $\xi = \sqrt{5} = 2.2360679774997\ldots$ *zu approximieren. Algorithmus 2.2.1 liefert die folgenden Resultate:*

| $i$ | $x_i$ | $i$ | $x_i$ |
|---|---|---|---|
| *3* | 2 | *10* | 2.2421875 |
| *4* | 2.5 | *11* | 2.23828125 |
| *5* | 2.25 | *12* | 2.236328125 |
| *6* | 2.125 | *13* | 2.2353515625 |
| *7* | 2.1875 | *14* | 2.23583984375 |
| *8* | 2.21875 | *15* | 2.236083984375 |
| *9* | 2.234375 | *16* | 1.2359619140625 |

*Nach den ersten zehn Iterationsschritten ist das Einschließungsintervall um einen Faktor* $2^{10} = 1024$ *verkleinert worden, die Genauigkeit ist also um 3 Dezimalstellen verbessert worden. Damit das Verfahren die ansprechende Genauigkeit von etwa 10 Dezimalstellen erreicht, sind zusätzlich zu den angegebenen 14 Schritten weitere 20 Iterationen nötig.*

**Proposition 2.2.3.** *Bezeichnen wir mit* $a_i$*,* $b_i$ *und* $x_i$ *die Einschließungen und berechneten Mittelpunkte im* $i$*–ten Iterationsschritt, dann erfüllt der maximale Fehler in* $x_i$ *die folgende Beziehung*

$$|x_i - \xi| \leq \tfrac{1}{2}|b_i - a_i| = 2^{-i+2}|b - a|$$

*für $i > 2$.*

BEWEIS. Einsetzen in die Definition des Algorithmus. □

**Proposition 2.2.4.** *Algorithmus 2.2.1 terminiert nach*

$$t = 2 + \lceil \log_2 \frac{|b-a|}{\varepsilon} \rceil$$

*Funktionsauswertungen.*

BEWEIS. Folgt sofort aus der Beziehung in Proposition 2.2.3 und der Abbruchbedingung des Algorithmus 2.2.1. □

Wegen Proposition 2.2.3 konvergiert die Mittelpunktbisektion global und (R–)linear mit Faktor $q = \frac{1}{2}$.

Achtet man nicht auf korrekt gerichtete Rundung, so kann die Fehlerabschätzung in Proposition 2.2.3 ungültig werden, doch man kann immer noch sagen, daß das letzte berechnete $x_i$ eine gute Approximation für $\xi$ ist, in dem Sinne, daß die durch Gleitkommaarithmetik berechnete Funktion $f$ nahe bei $x_i$ das Vorzeichen wechselt.

## 3. Sekantenverfahren

**3.1. Grundlagen.** Will man das langsame Konvergenzverhalten der Bisektionsmethode aus Abschnitt 2 verbessern, verwendet man, daß glatte Funktionen in der Nähe einfacher Nullstellen gut linearisiert also durch Geraden approximiert werden können.

Davon ausgehend argumentiert man folgendermaßen: Kennt man zwei Approximationen $x_1$ und $x_2$ einer Nullstelle der Funktion $f$, so ist die Nullstelle selbst im allgemeinen nahe dem Schnittpunkt der Geraden durch $(x_1, f(x_1))$ und $(x_2, f(x_2))$, der *Sekante* von $f$ bei $x_1$ und $x_2$, mit der $x$–Achse.

Mathematisch drückt sich das folgendermaßen aus. Die Sekante ist das lineare Interpolationspolynom (siehe Kapitel 5) zu den Stützstellen $(x_1, f(x_1))$ und $(x_2, f(x_2))$, daher gegeben durch

$$P_{01}(x) = f(x_1) + f[x_1, x_2](x - x_1).$$

Die Nullstelle $\xi$ dieser Funktion ist leicht berechenbar. Sie existiert, falls für die Steigung $f[x_1, x_2] \neq 0$ gilt, und in diesem Fall erhalten wir

$$\xi = x_1 - \frac{f(x_1)}{f[x_1, x_2]}. \tag{66}$$

Nachdem wir erwarten, daß die Nullstelle $\xi$ der Sekante eine bessere Approximation für die Nullstelle der Funktion $f$ ist als die ursprünglichen Näherungswerte $x_1$ und $x_2$, können wir Gleichung (66) zur Definition eines Iterationsverfahrens heranziehen. Für eine graphische Darstellung siehe Abbildung 7.1

Man beginnt mit zwei Näherungswerten $x_0$ und $x_1$ für die Nullstelle, deren Funktionswerte verschiedene Funktionswerte aufweist. Dann berechnet man weitere Approximationen $x_i$,

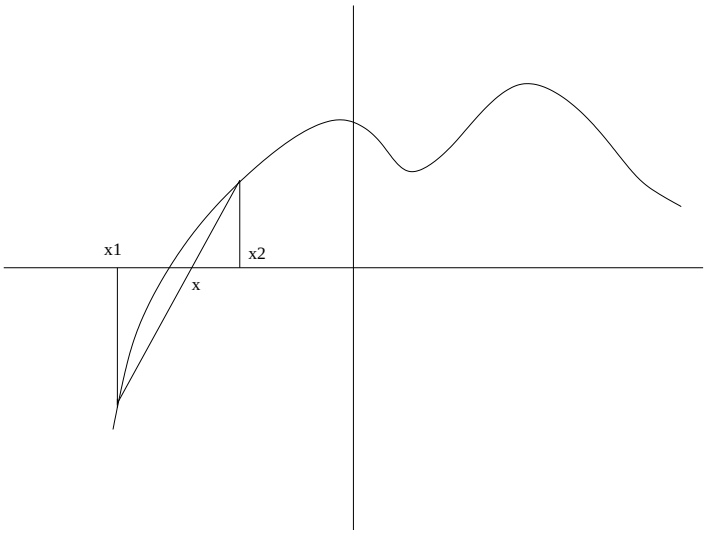



ABBILDUNG 7.1. Ein Schritt des Sekantenverfahrens

$i \geq 2$ gemäß folgender Iterationsvorschrift

$$\begin{aligned} x_{i+1} &= x_i - \frac{f(x_i)}{f[x_i, x_{i-1}]} = \\ &= x_i - \frac{(x_i - x_{i-1})f(x_i)}{f(x_i) - f(x_{i-1})} = \\ &= x_i - \frac{x_i - x_{i-1}}{1 - f(x_{i-1})/f(x_i)} = \\ &= \frac{x_{i-1}f(x_i) - x_i f(x_{i-1})}{f(x_i) - f(x_{i-1})}. \end{aligned}$$

wobei die letzte der vier äquivalenten Darstellungen meist numerisch instabiler ist als die anderen drei.

**Algorithmus 3.1.1.** *Sekantenverfahren*

$x_1 = a$
$x_2 = b$
*i=2*
**while** $|f(x_i)| \geq \varepsilon$ **do**
$x_{i+1} = x_i - \frac{x_i - x_{i-1}}{1 - f(x_{i-1})/f(x_i)}$
*i=i+1*
**done**
**stop** *"f hat eine Nullstelle $\xi$ nahe $x_i$".*

**Beispiel 3.1.2.** *Betrachten wir die Funktion $f(x) = x^2 - 5$ mit den Startwerten $x_0 = 1$ und $x_1 = 3$. Das Iterationsverfahren 3.1.1 liefert die Ergebnisse*

| $i$ | $x_i$ | $f(x_i)$ |
|---|---|---|
| *0* | 1 | $-4$ |
| *1* | 3 | 4 |
| *2* | 2 | $-1$ |
| *3* | 2.2 | $-0.159999999999999$ |
| *4* | 2.23809523809524 | 0.0090702947845811 |
| *5* | 2.23605150214592 | $-7.36797509617304 \cdot 10^{-5}$ |
| *6* | 2.23606797003472 | $-3.33848246825141 \cdot 10^{-8}$ |
| *7* | 2.23606797749982 | $1.23456800338317 \cdot 10^{-13}$ |
| *8* | 2.23606797749979 | $8.88178419700125 \cdot 10^{-16}$ |

**3.2. Konvergenzgeschwindigkeit.** Das Sekantenverfahren konvergiert lokal superlinear gegen $\xi$, wenn $\xi$ einfache Nullstelle der Funktion $f$ ist.

**Proposition 3.2.1.** *Sei $f$ in der Nähe der Nullstelle $\xi$ zweimal stetig differenzierbar, und es gelte $f'(\xi) \neq 0$. Sei $c := \frac{1}{2}f''(\xi)/f'(\xi)$. Dann ist die durch das Sekantenverfahren definierte Folge $x_i$ konvergent gegen $\xi$, für alle Startwerte $x_0$ und $x_1$, die genügend nahe bei $\xi$ sind, und es gilt die Abschätzung*

$$x_{i+1} - x_i = c_i(x_i - \xi)(x_{i-1} - \xi)$$

*mit*

$$\lim_{i\to\infty} c_i = c.$$

*Speziell gilt, daß das Sekantenverfahren lokal überlinear konvergiert.*

Beweis. Wegen der Newtonschen Interpolationsformel gilt

$$0 = f(\xi) = f(x_i) + f[x_i, x_{i-1}](\xi - x_i) + f[x_i, x_{i-1}, \xi](\xi - x_i)(\xi - x_{i-1}).$$

In einer genügend kleinen Umgebung von $\xi$ ist die Steigung $f[x_i, x_{i-1}]$ ungleich Null, da $f'(\xi) \neq 0$ vorausgesetzt ist. Damit kann man umformen zu

$$\begin{aligned}\xi &= x_i - \frac{f(x_i)}{f[x_i, x_{i-1}]} - \frac{f[x_i, x_{i-1}, \xi]}{f[x_i, x_{i-1}]}(\xi - x_i)(\xi - x_{i-1}) = \\ &= x_{i+1} - c_i(\xi - x_i)(\xi - x_{i-1}),\end{aligned}$$

wobei wir

$$c_i := \frac{f[x_i, x_{i-1}, \xi]}{f[x_i, x_{i-1}]}$$

gesetzt haben.

Liegen nun $\eta_1$ und $\eta_2$ in einer genügen kleinen abgeschlossenen Kugel $B_r(\xi)$ um $\xi$, dann bleibt der Ausdruck

$$c_i(\eta_1, \eta_2) := \frac{f[\eta_1, \eta_2, \xi]}{f[\eta_1, \eta_2]}$$

beschränkt. Sei $\bar{c}$ eine obere Schranke, und definieren wir $r_0 := \min(r, 1/2\bar{c})$. Dann gilt für $x_i, x_{i-1} \in B_{r_0}(\xi)$

$$|x_{i+1} - \xi| \leq \tfrac{1}{2}|x_i - \xi| \leq r_0,$$

sodaß für Startwerte $x_0, x_1 \in B_{r_0}(\xi)$ die gesamte Folge der $x_i$ in der $r_0$–Kugel um $\xi$ bleibt. Mit Hilfe eines einfachen Induktionsargumentes folgt $|x_i - \xi| \leq 2^{2-i}r_0$, und daher konvergiert die Folge der $x_i$ gegen $\xi$.

Aus dieser Tatsache folgt darüber hinaus noch, daß

$$\lim_{i\to\infty} c_i = \frac{f[\xi, \xi, \xi]}{f[\xi, \xi]} = \frac{1}{2}\frac{f''(\xi)}{f'(\xi)} = c,$$

und mit $q_i := |c_i||x_{i-1} - \xi|$ folgt weiters die Beziehung

$$|x_{i+1} - \xi| = q_i|x_i - \xi|.$$

Weil $\lim q_i = 0$ gilt, ist das Sekantenverfahren lokal überlinear konvergent. □

Leider gilt das Resultat nicht für alle Startwerte $x_0$, $x_1$. Das Verfahren ist also nicht global konvergent. Ein Beispiel für das Versagen der Sekantenmethode kann man in Abbildung 7.2 finden.

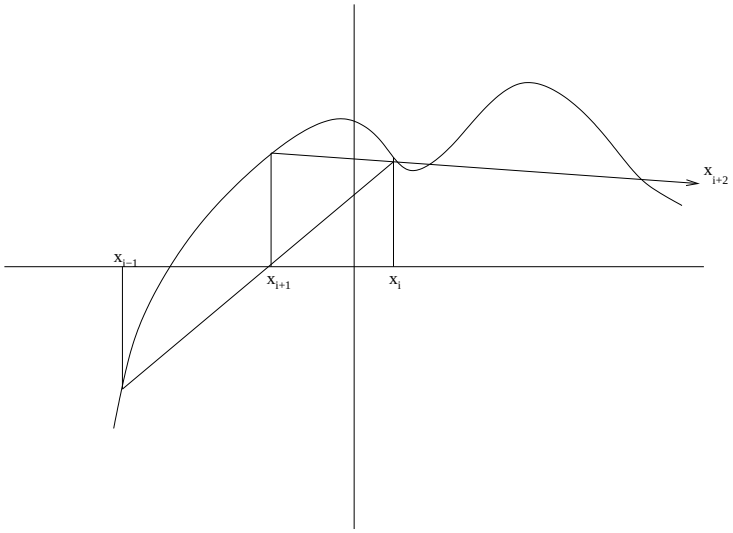



ABBILDUNG 7.2. Nichtkonvergenz des Sekantenverfahrens

**3.3. Mehrfache Nullstellen — Nullstellencluster.** Wie wir schon in früheren Kapiteln gesehen haben, hat $f$ genau dann eine $m$–fache Nullstelle an $\xi$, wenn die Funktion in einer Umgebung von $\xi$ geschrieben werden kann als

$$f(x) = (x - \xi)^m g(x)$$

mit stetiger Funktion $g$, die $g(\xi) \neq 0$ erfüllt. Für mehrfache Nullstellen ($m > 1$) ist das Sekantenverfahren nur *linear konvergent*.

**Beispiel 3.3.1.** *Sei $f(x) = x^2$, und wir verwenden das Sekantenverfahren, um die einzige Nullstelle $\xi = 0$ dieser Funktion mit den Starwerten $x_1 = 1$ und $x_2 = \frac{1}{2}$ zu finden. Der Algorithmus 3.1.1 erzeugt die Folge*

$$x_i = \frac{1}{3}, \frac{1}{5}, \frac{1}{8}, \frac{1}{13}, \dots \quad x_{34} = 1.08 \cdot 10^{-7}.$$

*Untersuchen wir das Verhalten etwas genauer, und setzen wir $1/x_i = n_i$, so sehen wir*

$$x_{i+1} = \frac{x_{i-1}x_i^2 - x_i x_{i-1}^2}{x_i^2 - x_{i-1}^2} = \frac{(x_i - x_{i-1})x_i x_{i-1}}{(x_i - x_{i-1})(x_i + x_{i-1})} = \frac{x_i x_{i-1}}{x_i + x_{i-1}}$$

$$n_{i+1} = \frac{x_i + x_{i-1}}{x_i x_{i-1}} = \frac{1}{x_i} + \frac{1}{x_{i-1}} = n_i + n_{i-1}.$$

*Die Nenner der erzeugten Folge sind also die Fibonacci–Zahlen, und daher ist die Konvergenz (Q-)linear mit Konvergenzfaktor $q = (\sqrt{5} - 1)/2$.*

Verändert man eine Funktion mit einer $m$–fachen Nullstelle $\xi$ ein wenig, das passiert etwa durch Rundungsfehler bei der Auswertung, dann hat die gestörte Funktion $\tilde{f}(x)$ bis zu $m$ verschiedene Nullstellen in der unmittelbaren Umgebung von $\xi$, einen sogenannten *Nullstellencluster*. Ist die Störung klein genug, so entstehen Nullstellen, die numerisch von

der mehrfachen Nullstelle der ungestörten Funktion ununterscheidbar sind. In diesem Fall konvergieren die meisten Nullstellenverfahren genauso langsam gegen einfache Nullstellen innerhalb des Nullstellenclusters wie sie gegen die mehrfache Nullstelle konvergieren würden. Im Groben kann man sagen, daß $m$–fache Nullstellencluster in einem Bereich von $\sqrt[m]{\mathsf{eps}}$ numerisch nicht von einer $m$–fachen Nullstelle unterscheidbar sind.

Ist eine gerade Anzahl einfacher Nullstellen $\xi_1, \ldots, \xi_{2k}$ in einem Nullstellencluster in einer Umgebung eines Punktes $\xi^*$ gegeben, so ist es schwierig, einen Vorzeichenwechsel der Funktion $f$ zu finden, da das Produkt der Linearfaktoren $x - \xi_i$ positiv ist für alle Werte $x$ außerhalb des von den $\xi_i$ erzeugten Intervalls. Um einen Vorzeichenwechsel zu finden, muß man also einen Wert innerhalb des Clusters finden, was im Prinzip ebenso schwierig ist wie eine Nullstelle selbst zu finden.

Nachdem in der Praxis Nullstellen von höherer als zweiter Ordnung nur sehr selten auftreten, betrachten wir in der Folge nur den Fall einer doppelten Nullstelle. In diesem Zusammenhang bemerken wir, daß wir damit auch den Fall zweier eng benachbarter Nullstellen beschrieben haben.

Gilt $f(x) = (x-\xi)^2 g(x)$, so kann man diese Nullstelle nicht durch einen Vorzeichenwechsel von $f$ erkennen. Allerdings ist $\xi$ in diesem Fall eine einfache Nullstelle der Funktion

$$h(x) = (x - \xi)\sqrt{|g(x)|} = \operatorname{sgn}(x - \xi)\sqrt{|f(x)|},$$

und wenn wir auf $h$ einen Schritt des Sekantenverfahrens anwenden, erhalten wir die neue Iterationsvorschrift

$$x_{i+1} = x_i - \frac{x_i - x_{i-1}}{1 - h(x_{i-1})/h(x_i)} = x_i - \frac{x_i - x_{i-1}}{1 \pm \sqrt{f(x_{i-1})/f(x_i)}}.$$

Das Vorzeichen ist positiv, falls $\xi$ im von $x_{i-1}$ und $x_i$ begrenzten Intervall liegt, und sonst negativ. Nachdem wir $\xi$ nicht kennen, wählen wir einfach das negative Vorzeichen und nennen das durch die Vorschrift

$$x_{i+1} = x_i - \frac{x_i - x_{i-1}}{1 - \sqrt{f(x_{i-1})/f(x_i)}}$$

definierte Iterationsverfahren, das *Wurzelsekantenverfahren*.

**Algorithmus 3.3.2.** *Wurzelsekantenverfahren*

$x_1 = a$
$x_2 = b$
*i=2*
**while** $|f(x_i)| \geq \varepsilon$ **do**
  $x_{i+1} = x_i - \frac{x_i - x_{i-1}}{1-\sqrt{f(x_{i-1})/f(x_i)}}$
  **if** $|f(x_{i+1})| > |f(x_i)|$**then**
    $h = x_{i+1}$
    $x_{i+1} = x_i$
    $x_i = h$
  **endif**
  *i=i+1*
**done**
**stop** *"f hat eine Nullstelle $\xi$ nahe $x_i$".*

Algorithmus 3.3.2 konvergiert lokal superlinear im Fall einer doppelten Nullstelle $\xi$.

**Beispiel 3.3.3.** *Die Funktion $f(x) = x^4 - 2x^3 + 2x^2 - 2x + 1$ hat die doppelte Nullstelle $\xi = 1$. Beginnt man Algorithmus 3.3.2 mit den Startwerten $x_1 = 0$, $x_2 = \frac{6}{5}$, so erhält man die folgenden Resultate.*

| $i$ | $x_i$ | $x_{i+1}$ | $f(x_{i+1})$ |
|---|---|---|---|
| 1 | 0 | 1.2 | 0.0976 |
| 2 | 1.74522604659078 | 1.2 | 0.0976 |
| 3 | 1.2 | 1.05644633886449 | 0.00674222757113 |
| 4 | 1.05644633886449 | 1.00526353445128 | 0.00005570200768 |
| 5 | 1.00526353445128 | 1.00014620797161 | 0.00000004275979 |
| 6 | 1.00014620797161 | 1.00000038426329 | 0.0000000000003 |
| 7 | 1.00000038426329 | 1.00000000002636 | 0 |

*Man sieht leicht die superlineare Konvergenz, und bemerkt weiters die geringe Grenzgenauigkeit der Nullstelle, obwohl der Funktionswert von $f$ bereits* 0 *erreicht hat. Der Grund dafür ist die hohe Rundungsfehleranfälligkeit mehrfacher Nullstellen (in diesem Fall $\sqrt{\mathsf{eps}}$).*

Für ein allgemeines $f$ kann natürlich während des Algorithmus 3.3.2 ein Vorzeichenwechsel auftreten, obwohl die beiden Startwerte dasselbe Vorzeichen hatten. In diesem Fall betrachten wir das als günstige Fügung, da wir dann ein global konvergentes Verfahren zur Nullstellensuche, eine Einschließungsmethode, anwenden können. Die Konstruktion eines solchen Verfahrens, das die günstigen Eigenschaften der Algorithmen 2.2.1, 3.1.1 und 3.3.2 verbindet, werden wir im nächsten Abschnitt in Angriff nehmen.

## 4. Global superlinear konvergente Verfahren

In diesem Abschnitt wollen wir versuchen, ein global konvergentes Einschließungsverfahren zu konstruieren, das schneller konvergiert als das Mittelpunktbisektionsverfahren.

**4.1. Die naive Sekanten-Bisektions-Methode.** Ein erster Schritt in die richtige Richtung ist der Versuch, indem man den Vorzeichenwechsel im Sekantenverfahren erzwingt. Das könnte zu globaler Konvergenz führen, da man auf diesem Weg ein Einschließungsverfahren konstruiert.

Man definiert das Iterationsverfahren um zu

**Algorithmus 4.1.1.** *Iterationsschritt der naiven Sekanten-Bisektion*

$x_{i+1} = x_i - f(x_i)/f[x_i, x_{i-1}]$
**if** $f(x_{i-1})f(x_{i+1}) < 0$ **then**
  $x_i = x_{i-1}$
**endif**

Das erzwingt in jedem Schritt einen Vorzeichenwechsel, doch leider konvergiert das Verfahren oft nicht. Jetzt ist nicht schuld, daß sich die Punkte von der Nullstelle entfernen können, sondern der Umstand, daß die Länge der Intervalle $[]\{x_i, x_{i-1}\}$ nicht gegen 0 konvergiert. Der Grund ist, daß die $x_i$ meist nur von einer Seite gegen $\xi$ konvergieren. Darüber hinaus ist die Konvergenz dieses Verfahren im allgemeinen auch nur linear.

**4.2. Sekanten-Bisektions-Verfahren mit Vorzeichenwechselsuche.** Bemüht man sich ein bißchen mehr und programmiert man sorgfältig, dann kann man die Eigenschaften von Bisektions- und Sekantenverfahren so kombinieren, daß ein robuster Algorithmus entsteht, der global superlinear konvergiert. Man wird allerdings sehen, daß die Komplexität des Verfahrens schon sehr groß ist. Es wurde aus [**Neumaier 2000**] entnommen.

**Algorithmus 4.2.1.** *Sekanten-Bisektions-Verfahren mit Vorzeichenwechselsuche*

*f=f(x)*
*nf=1*
**if** $f == 0$ **then**
  **return(**(x, "Nullstelle erraten")**)**
**endif**

```
f2=f(x2)
nf=2
if f2 == 0 then
  return((x2, "Nullstelle erraten"))
endif

; finde einen Vorzeichenwechsel
while f*f2 > 0 do
      ; setze den besten Punkt an x2.
      if |f| < |f2| then
        x1=x2
        f1=f2
        x2=x
        f2=f
      else
        x1=x
        f1=f
      endif
      ; Wurzelsekantenmethode
      x = x2 − (x2 − x1)/(1 − max(√(f1/f2), 2))
      if x == x2 || nf == nfmax then
        return((x2, "Doppelte Nullstelle oder kein Vorzeichenwechsel"))
      endif
      f=f(x)
      nf=nf+1
      if f == 0 then
        return((x, "Nullstelle erraten"))
      endif

      ; wir haben einen Vorzeichenwechsel gefunden
      slow=0
      while nf < nfmax do
            ; Berechnung eines neuen Punktes xx und der Genauigkeit
            if slow==0 then
              ; Sekantenschritt
              if |f| < |f2| then
                xx = x − (x − x2) ∗ f/(f − f2)
                acc = |xx − x|
              else
                xx = x2 − (x2 − x) ∗ f2/(f2 − f)
                acc = |xx − x2|
              endif
            elseif slow==1 then
              ; Sekanten-Extrapolationsschritt
              if f1 ∗ f2 > 0 then
                q = max(f1/f2, 2 ∗ (x − x1)/(x − x2))
                xx = x2 − (x2 − x1)/(1 − q)
                acc = |xx − x|
              else
```

```
    q = max(f1/f, 2 * (x2 - x1)/(x2 - x))
    xx = x - (x - x1)/(1 - q)
    acc = |xx - x|
  endif
else
  ; Schritt mit geometrischem Mittel
  if x * x2 > 0 then
    xx = x * sqrt(x2/x1)
  elseif x == 0 then
    xx = 0.1 * x2
  elseif x2 == 0 then
    xx = 0.1 * x
  else
    xx = 0
  endif
  acc = max(abs(xx - [x, x2]))
endif

; Abbruchtests
if acc ≤ 0 then
  return((xx, "Approximative einfache Nullstelle"))
endif
ff=f(xx)
if ff == 0 then
  return((xx, "Einfache Nullstelle"))
endif

; Berechne Verkleinerungsfaktor und neue Einschließung
if f2 * ff < 0 then
  rf = (x2 - xx)/(x2 - x)
  x1=x
  f1=f
  x=xx
  f=ff
else
  rf = (x - xx)/(x - x2)
  x1=x2
  f1=f2
  x2=xx
  f2=ff
endif

; erzwinge zwei aufeinanderfolgende Mittelungsschritte,
; im nichtlokalen Fall (slow=2)
if rf > 0.7 ||| slow==2 then
  slow=slow+1
else
  slow=0
endif
```

**done**
**done**

Der Algorithmus beginnt mit zwei Startwerten $x$ und $x2$. Liefern diese keinen Vorzeichenwechsel, so wird die Wurzelsekantenmethode verwendet, in der Hoffnung, daß sie eine doppelte Nullstelle oder einen Vorzeichenwechsel liefert. Tritt keiner dieser Fälle ein, so wird nach *nfmax* Schritten erfolglos abgebrochen.

Die Variable *slow* zählt die Anzahl der aufeinanderfolgenden Schritte, die nicht mindestens zu einer Reduktion der Länge des Einschließungsintervalls um den Faktor $1/\sqrt{2}$ geführt haben. Ist das mindestens zweimal hintereinander passiert, dann werden zwei aufeinanderfolgende Bisektionsschritte gemacht (hier wird anstelle des arithmetischen das geometrische Mittel verwendet, da dieses für sehr große Intervalle günstigere Unterteilungspunkte liefert).

War nur ein langsamer Schritt dabei, dann nimmt man an, daß das durch einen Sekantenschritt passiert ist, der auf die falsche Seite der Nullstelle geführt hat. Daher werden der neueste Punkt $x$ und der letzte Punkt, an dem $f$ dasselbe Vorzeichen hatte, zu einem linearen Extrapolationsschritt herangezogen. Dabei wird darauf geachtet, die Schrittweite auf vernünftige Weise einzuschränken.

Algorithmus 4.2.1 ist ein robustes, effizientes Verfahren zur Nullstellensuche, das lokal superlinear gegen einfache und doppelte Nullstellen konvergiert. Wird ein Vorzeichenwechsel gefunden, dann konvergiert die Methode sogar global, doch in ungünstigen Fällen kann die Konvergenz linear mit halber Konvergenzgeschwindigkeit im Vergleich zur Mittelpunktsbisektion sein.

Wird kein Vorzeichenwechsel gefunden, kann das Verfahren auch nicht konvergent sein, doch das ist nicht überraschend. Schließlich existieren Funktionen ohne Nullstelle.

Üblicherweise reichen jedoch etwa 10–15 Funktionswerte aus, um eine Nullstelle mit voller erreichbarer Genauigkeit von ca. 16 Dezimalstellen (einfache Nullstelle) zu finden. Daher ist eine Wahl von $nfmax == 20$ gut, um überflüssige Funktionsauswertungen im Fall der Nicht-Konvergenz zu vermeiden.

Für unstetige Funktionen muß man aufpassen, da in diesem Fall ein Vorzeichenwechsel nicht automatisch die Existenz einer Nullstelle garantiert. In diesem Fall muß man zusätzlich eine Dämpfungsmethode verwenden, die genauer im mehrdimensionalen Fall (Teil 2, Abschnitt 13) beschrieben wird.

## 5. Genauere Konvergenzuntersuchung

Möchte man die genauere Konvergenzordnung der Iterationsverfahren untersuchen, dann muß man noch ein paar Untersuchungen anstellen.

### 5.1. Grundlagen.

**Lemma 5.1.1.** *Seien $p_0, p_1, \ldots, p_s$ nichtnegative Zahlen mit*

$$p := -1 + \sum_{i=0}^{s} p_i > 0,$$

*und sei $\kappa$ eine positive Lösung der Gleichung*

$$\kappa^{s+1} = \sum_{i=0}^{s} p_i \kappa^{s-i}. \tag{67}$$

*Ist $e_i$ eine gegen $+\infty$ divergierende Folge mit*

$$e_{i+1} \geq \alpha + \sum_{j=0}^{s} p_j e_{i-j}$$

*für eine Zahl $\alpha \in \mathbb{R}$, dann existieren $\beta > 0$ und $\gamma \in \mathbb{R}$ mit*

$$e_i \geq \beta\kappa^i + \gamma$$

*für alle $i \geq 1$.*

Beweis. Wir wählen $i_0 \geq s$ so groß, daß $e_i + \alpha/p > 0$ für alle $i \geq i_0 - s$ und setzen

$$\beta := \min_{i=i_0-s,\ldots,i_0} \{\kappa^{-i}(e_i + \alpha/p)\}.$$

Dann ist $\beta > 0$, und wir haben

$$e_i \geq \beta\kappa^i - \alpha/p$$

für alle $i \geq i_0 - s$. Nach Konstruktion gilt die Gleichung bereits für $i = i_0 - s, \ldots, i_0$, und nach Induktion gilt es für alle $i \geq i_0 - s$ wegen

$$\begin{aligned} e_{i+1} &\geq \alpha + \sum_{j=0}^{s} p_j e_{i-j} \geq \\ &\geq \alpha + \sum_{j=0}^{s} p_i(\beta\kappa^{i-j} - \alpha/p) = \\ &= \beta\kappa^{i-s} \sum_{j=0}^{s} p_j \kappa^{s-j} - (-p + \sum_{j=0}^{s} p_j)\alpha/p = \\ &= \beta\kappa^{i+1} - \alpha/p. \end{aligned}$$

Wählt man nun

$$\gamma := \min\{-\alpha/p, e_1 - \beta\kappa, e_2 - \beta\kappa^2, \ldots, e_{i_0-s-1} - \beta\kappa^{i_0-s-1}\},$$

dann erhält man

$$e_i \geq \beta\kappa^i + \gamma$$

für alle $i \geq 1$. □

Es ist möglich zu beweisen, daß Gleichung (**??**) genau eine positive reelle Lösung $\kappa$ hat, und daß diese Lösung

$$1 < \kappa < 1 + \max\{p_0, \ldots, p_s\}$$

erfüllt.

**Proposition 5.1.2.** *Für einfache Nullstellen hat das Sekantenverfahren die Konvergenzordnung $\kappa = (1 + \sqrt{5})/2 \approx 1.618\ldots$.*

Beweis. Nach Proposition 3.2.1 gilt für das Sekantenverfahren

$$|x_{i+1} - \xi| \ldots \bar{c}|x_i - \xi||x_{i-1} - \xi|$$

mit $\bar{c} = \sup c_i$. Zieht man den Logarithmus aus dieser Beziehung erhält man

$$e_{i+1} \geq e_i + e_{i-1} + \alpha$$

mit $\alpha = \log_{10} \bar{c}$. Lemma 5.1.1 impliziert dann

$$e_i \geq \beta\kappa^i + \gamma$$

mit $\beta > 0$, $\gamma \in \mathbb{R}$ und $\kappa$ die positive Lösung der Gleichung

$$\kappa^2 = \kappa + 1.$$

□

**5.2. Die Methode von Opitz.** Man kann die Konvergenzordnung der Sekantenmethode basierend auf den Ergebnissen des vorigen Abschnittes verbessern, indem man nicht nur Informationen aus dem letzten Schritt sondern auch weiter zurück liegende Punkte mitverwendet.

Ein interessantes Verfahren, die Methode von Opitz ([**Opitz 1958**]) basiert darauf, die Pole der Funktion $h = 1/f$ zu approximieren, indem man den Spezialfall

$$h(x) = \frac{a}{\xi - x}$$

betrachtet. Für diese Funktion sind die dividierten Differenzen

$$h[x_{i-s}, \dots, x_i] = \frac{a}{(\xi - x_{i-s}) \dots (\xi - x_i)}$$

einfach zu berechnen. Bildet man Quotienten aufeinanderfolgender Terme, erhält man

$$\xi = x_i + \frac{h[x_{i-s}, \dots, x_{i-1}]}{h[x_{i-s}, \dots, x_i]}.$$

Für allgemeine Funktionen verwendet man diese Beziehung, um Approximationen für $\xi$ aus gegebenen $x_1, \dots, x_{s+1}$ iterativ zu bestimmen. In der Praxis erzeugt man sich dabei $x_2, \dots, x_{s+1}$ für $s > 1$ aus den Optiz–Formeln niedrigerer Ordnung.

**Theorem 5.2.1.** *Für Funktionen der Form*

$$h(x) = \frac{p_{s-1}(x)}{x - \xi},$$

*mit $p_{s-1} \in \mathbb{R}^{s-1}[x]$ gibt die Opitz–Formel den exakten Pol $\xi$ in einem Schritt.*

Beweis. Folgt aus Polynomdivision wegen

$$h(x) = \frac{a}{\xi - x} + p_{s-2}(x)$$

mit $a = -p_{s-1}(\xi)$, und das Polynom $p_{s-2}$ fällt in den in der Formel auftretenden dividierten Differenzen weg. □

**Theorem 5.2.2.** *Sei $\xi$ ein einfacher Pol der Funktion $h$ (eine einfache Nullstelle von $f$), und sei $h(x)(x-\xi)$ wenigstens $C^s$ in einer Umgebung von $\xi$. Dann konvergiert die Folge $x_i$ die durch die Opitz–Iteration erzeugt wird, für alle Anfangswerte, die genügend nahe bei $\xi$ liegen, und*

$$(\xi - x_{i+1}) = c_i^{(s)} \prod_{j=0}^{s} (\xi - x_{i-j})$$

*für $i > s$, wobei $c_i^{(s)}$ gegen eine Konstante konvergiert für $i \to \infty$. Das impliziert, daß das Verfahren von Opitz lokal superlinear konvergiert Konvergenzordnung $\kappa_s = \kappa$, gegeben als positive reelle Lösung der Gleichung*

$$\kappa^{s+1} = \kappa^s + \kappa^{s-1} + \dots + 1.$$

Beweis. [**Opitz 1958**] oder [**Neumaier 2000**, 5.4.4]. □

Wählt man $s = 1$, dann stimmt die Methode von Opitz mit dem Sekantenverfahren überein. Das Verfahren von Opitz ist für $s > 1$ dem Sekantenverfahren überlegen, und die Konvergenzordnung wächst monoton mit $s$. Es muß jedoch zur Erreichung globaler Konvergenz ebenfalls auf eine Kombination mit einem Einschließungsverfahren, also einer Bisektionsmehode zurückgegriffen werden. Weil das Verfahren aber von $s$ Pukten abhängt,

kann die „Divergenzerkennung“ erst nach $s$ Schritten beginnen (in einem Algorithmus analog zu 4.2.1 erst bei *slow* $\geq s$), und somit ist im ungünstigen Fall der linearen Konvergenz die Konvergenzgeschwindigkeit lediglich $1/s$ der des Bisektionsverfahrens.

Andererseits kann man mit der Methode von Opitz Konvergenzraten beliebig nahe bei 2 finden, und man benötigt gar nicht einmal sehr hohe $s$.

| $s$ | Ordnung | $s$ | Ordnung |
|---|---|---|---|
| 1 | 1.61803 | 6 | 1.99196 |
| 2 | 1.83929 | 7 | 1.99603 |
| 3 | 1.92756 | 8 | 1.99803 |
| 4 | 1.96595 | 9 | 1.99902 |
| 5 | 1.98358 | 10 | 1.99951 |

## 6. Fehleranalyse

**6.1. Grenzgenauigkeit.** Nehmen wir an, daß die Gleitkommaapproximation $\mathrm{gl}(f(x))$ für $f(x)$ in einer Umgebung von $\xi$ die Abschätzung

$$|\,\mathrm{gl}(f(x)) - f(x)| \leq \delta$$

erfüllt. Wir suchen mit unseren Methoden Punkte, die Nullstellen $\tilde{\xi}$ von $\mathrm{gl}(f)$ sind. Der Funktionswert der wahren Funktion erfüllt also nur $|f(\tilde{\xi})| \leq \delta$.

Ist $\xi$ eine $m$–fache Nullstelle von $f$, dann ist $f(x) = (x-\xi)^m g(x)$ mit $g(\xi) \neq 0$, und es folgt

$$|\tilde{\xi} - \xi| = \sqrt[m]{\left|\frac{f(\tilde{\xi})}{g(\tilde{\xi})}\right|} \leq \sqrt[m]{\frac{\delta}{|g(\tilde{\xi})|}} = O(\sqrt[m]{\delta}).$$

Im Spezialfall einfacher Nullstellen gilt

$$g(\tilde{\xi}) = \frac{f(\tilde{\xi}) - f(\xi)}{\tilde{\xi} - \xi} = f[\tilde{\xi}, \xi] \approx f'(\xi),$$

also gilt für den absoluten Fehler in erster Näherung

$$|\tilde{\xi} - \xi| \doteq \frac{\delta}{|f'(\xi)|}.$$

Man kann also die folgenden Schlüsse ziehen:

(1) Für sehr kleine $|f'(\xi)|$, also für sehr flache Funktionen wird der Fehler in $\tilde{\xi}$ sehr stark verstärkt. In diesem Fall ist $\xi$ schlecht konditioniert.
(2) Speziell für mehrfache Nullstellen ist die Kondition schlecht, da die Anzahl der korrekten Stellen nur $1/m$ der Mantissenlänge beträgt.
(3) Eine doppelte Nullstelle kann von zwei einfachen Nullstellen nicht unterschieden werden, wenn diese weniger als $O(\sqrt{\delta})$ auseinander liegen.

**6.2. Deflation.** Um einige oder alle Nullstellen einer Funktion $f$ zu berechnen, kommt eine Standardmethode, *Deflation* genannt, zum Einsatz. Es funktioniert analog zur Polynomdivision folgendermaßen.

Sind bereits Nullstellen $\xi_1, \ldots, \xi_s$ ($s \geq 1$) bekannt, mit den zugehörigen Vielfachheiten $m_1, \ldots, m_s$, so kann man die übrigen Nullstellen von $f$ finden, indem man die Funktion

$$g(x) := \frac{f(x)}{\prod_{j=1}^{s}(x-\xi_j)^{m_j}}$$

auf Nullstellen untersucht.

Obwohl die numerisch berechneten Werte der $\xi_k$ mit Fehlern behaftet sind, ist diese *implizite Deflation* stabil von einem numerischen Standpunkt. Denn selbst wenn die einzelnen $\xi_k$ ungenau sind, sind die übrigen Nullstellen von $g$ immer noch exakte Nullstellen von $f$. Einzig im Fall von Nullstellenclustern, können numerische Schwierigkeiten auftreten, doch diese sind ohnehin nur schwer unterscheidbar von mehrfachen Nullstellen und schlecht konditioniert.

Warnung. Für Polynome ist es numerisch sehr instabil, *explizite Deflation*, also das Abdividieren von Faktoren, durchzuführen. Der Grund dafür ist, daß die Nullstellen von Polynomen zwar stabil sind unter kleinen relativen Änderungen der Koeffizienten aber instabil unter kleinen absoluten Änderungen, die aber beim expliziten Abdividieren auftreten. Daher ist auch bei Polynomen die implizite Deflation vorzuziehen.

## 7. Newton-Verfahren

Zum Abschluß noch eine Bemerkung zum bekanntesten Nullstellensuchverfahren, der *Newton–Iteration*, die durch die Vorschrift

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

definiert ist. Wir werden sie im mehrdimensionalen (Teil 2, Kapitel 13) genauer behandeln, da sie sich dort als Methode der Wahl herausstellen wird. Im eindimensionalen leidet sie an einem wesentlichen Nachteil

> Sie benötigt die Ableitung von $f$, die oft approximiert werden muß und relativ viel Zusatzaufwand zur Berechnung von $f$ bedeutet.

Im eindimensionalen Fall bedeutet das, daß die quadratische Konvergenz nicht ausreicht, um den Nachteil des höheren Aufwandes in einem Schritt auszugleichen. Im mehrdimensionalen kehrt sich dies um.

KAPITEL 8

# Datenanalyse und Funktionsapproximation

## 1. Grundlagen

**1.1. Approximationsprobleme, siehe Neumaier, Bd II, S. 18f.**

**1.2. Klassifikationsprobleme.**

**1.3. Orthogonale Polynome.**

## 2. Auswertung von Reihen, Konvergenzbeschleunigung

**2.1. Padé-Approximation.**

**2.2. Eta– und Epsilonalgorithmus.**

## 3. Algorithmen zur Klassifikation

.

KAPITEL 9

# Numerische Lineare Algebra III: Eigenwertprobleme

## 1. Grundlagen

Die Berechnung der Eigenwerte einer Matrix spielt in vielen Anwendungsgebieten eine große Rolle. Die Modelle **??** und **??** aus Kapitel 2 sind gute Beispiele dafür.

**1.1. Normalformen.** Eine Eigenwertzerlegung einer quadratischen Matrix $A \in \mathbb{K}^{n\times n}$ ist eine Faktorisierung

$$A = X\Lambda X^{-1},$$

wobei $X$ eine reguläre Matrix, gebildet aus einer Basis von Eigenvektoren zu $A$, und $\Lambda = \operatorname{diag}(\lambda_1, \ldots, \lambda_n)$ die aus den Eigenwerten zusammengesetzte Diagonalmatrix sind. Aus der linearen Algebra ist bekannt, daß nicht jede Matrix eine Eigenwertzerlegung besitzt.

Die Frage welche Matrizen diagonalisierbar sind (eine Eigenwertzerlegung besitzen) führt über den folgenden algebraischen Ausdruck:

**Definition 1.1.1.** *Das* charakteristische Polynom *der Matrix $A$ ist definiert durch den Ausdruck*

$$p_A(x) := (-1)^n \det(A - x\mathbb{I}).$$

*Das Vorzeichen wurde dabei so gewählt, daß $p$ monisch ist. Die Nullstellen des charakteristischen Polynoms sind genau die Eigenwerte von $A$. Die Vielfachheit der Nullstelle heißt die* algebraische Vielfachheit *des Eigenwertes.*

*Die Menge aller Eigenvektoren zum fix gewählten Eigenwert $\lambda$ bildet zusammen mit dem Nullvektor einen Vektorraum $E_\lambda$, den* Eigenraum *zum Eigenwert $\lambda$. Die Dimension von $E_\lambda$, also die maximale Anzahl linear unabhängiger Eigenvektoren zu $\lambda$, heißt die* geometrische Vielfachheit *von $\lambda$.*

Stimmen für jeden Eigenwert geometrische und algebraische Vielfachheit überein, so existiert eine Basis aus Eigenvektoren, und daher ist $A$ diagonalisierbar.

**Definition 1.1.2.** *Für jede invertierbare Matrix $T$ ist die Abbildung*

$$A \mapsto TAT^{-1}$$

*eine* Ähnlichkeitstransformation. *Zwei Matrizen $A$ und $B$ heißen* ähnlich, *wenn es eine Ähnlichkeitstransformation gibt, die $A$ und $B$ verbindet; d.h. es gibt eine reguläre Matrix $T$ mit $B = TAT^{-1}$.*

Das folgende Resultat ist auch bestens bekannt aus der linearen Algebra:

**Proposition 1.1.3.** *Zwei ähnliche Matrizen $A$ und $B$ haben dasselbe charakteristische Polynom, dieselben Eigenwerte mit denselben algebraischen und geometrischen Vielfachheiten.*

Beweis. Nachdem $A$ und $B$ ähnlich sind, existiert eine Matrix $T$ mit $B = TAT^{-1}$. Dann ist aber

$$\begin{aligned} p_{TAT^{-1}}(x) &= \det(x\mathbb{I} - TAT^{-1}) = \det(T(x\mathbb{I} - A)T^{-1}) = \\ &= \det(T)\det(x\mathbb{I} - A)\det(T)^{-1} = \det(x\mathbb{I} - A) = p_A(x). \end{aligned}$$

Daher stimmen die Eigenwerte und deren algebraische Vielfachheiten überein. Ist $E_\lambda$ der Eigenraum der Matrix $A$. Dann ist $V_\lambda := TE_\lambda$ der Eigenraum zum Eigenwert $\lambda$ der Matrix $B = TAT^{-1}$. Weil $T$ invertierbar ist, stimmen die Dimensionen von $E_\lambda$ und $V_\lambda$ überein. □

Es existieren jedoch Matrizen, die nicht diagonalisierbar sind. Für diese Matrizen existiert eine Verallgemeinerung der Eigenwertzerlegung, die *Jordan–Zerlegung*:

$$A = TJT^{-1},$$

wobei $J$ eine Blockdiagonalmatrix ist aufgebaut aus sogenannten Jordanblöcken. Ein solcher Jordanblock hat die Gestalt

$$\begin{pmatrix} \lambda & 1 & 0 & \cdots & 0 \\ 0 & \lambda & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & \lambda & 1 \\ 0 & \cdots & \cdots & 0 & \lambda \end{pmatrix}.$$

Es gibt jedoch noch andere Normalformen, die die Eigenwerte einer Matrix verraten. Die wichtigste von diesen ist die Schur–Zerlegung. Sie ist eine Verallgemeinerung des Satzes über symmetrische (hermitesche) Matrizen, der besagt, daß jede hermitesche Matrix $A$ unitär diagonalisierbar ist, d.h. es gibt eine unitäre Matrix $U$ mit

$$A = U\Lambda U^*.$$

Die Verallgemeinerung führt zur Schur–Zerlegung:

**Theorem 1.1.4.** *Jede quadratische Matrix $A \in \mathbb{K}^{n\times n}$ besitzt eine Faktorisierung der Form*

$$A = URU^*,$$

*die* Schur–Zerlegung*, mit einer unitären Matrix $U$ und einer oberen Dreiecksmatrix $R$.*

Beweis. Der Beweis erfolgt mittels Induktion. Für $n = 1$ ist die Behauptung trivial. Nehmen wir also an, daß $n \geq 2$ ist. Sei $x$ irgendein normierter Eigenvektor von $A$ und $\lambda$ der dazugehörige Eigenwert. Sei $U_1$ eine unitäre Matrix, deren erste Spalte mit $x$ übereinstimmt. Dann hat $U_1^*AU_1$ die Gestalt

$$U_1^*AU_1 = \begin{pmatrix} \lambda & y \\ 0 & C \end{pmatrix}$$

mit einem Vektor $y$ und einer $(n-1)\times(n-1)$–Matrix $C$. Nach der Induktionsannahme existiert eine Matrix $U_2$, die $C$ Schur–faktorisiert $C = U_2R_2U_2^*$. Setzen wir

$$U = U_1 \begin{pmatrix} 1 & 0 \\ 0 & U_2 \end{pmatrix},$$

so gilt

$$U^*AU = \begin{pmatrix} \lambda & yU_2 \\ 0 & R_2 \end{pmatrix} =: R.$$

□

Ist $A$ hermitesch, so stimmt die Schur–Zerlegung mit der Eigenwertzerlegung überein. Nachdem die Determinante einer oberen Dreiecksmatrix $R$ genau das Produkt der Hauptdiagonalelemente ist, sind ebendiese Hauptdiagonalelemente (wegen der Gestalt des charakteristischen Polynoms) auch die Eigenwerte von $R$. Aus Proposition 1.1.3 folgt, daß man mit Hilfe einer Schur–Zerlegung die Eigenwerte einer beliebigen Matrix $A$ bestimmen kann.

**1.2. Allgemeines über Algorithmen zur Eigenwertberechnung.** Obwohl die obigen Resultate einen großen theoretischen Wert haben, sind sie zur numerischen Bestimmung der Eigenwerte nur bedingt verwendbar. Besonders das Standardverfahren, das man in der linearen Algebra lernt, ist unbrauchbar. Dort berechnet man die Eigenwerte von $A$, indem man die Nullstellen des charakteristischen Polynomes $p_A(x)$ sucht. Die Gründe für die Unverwendbarkeit des Standardzuganges sind die folgenden: Erstens ist die Berechnung des charakteristischen Polynoms sehr aufwendig ($O(n!)$), und zweitens ist die Nullstellenberechnung von Polynomen numerisch sehr instabil (Nullstellen von Polynomen sind schlecht konditioniert).

Das charakteristische Polynom ermöglicht es jedoch einen wesentlichen mathematischen Aspekt der Eigenwertberechnung zu beleuchten. Sei

$$p(x) = x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$$

ein beliebiges monisches Polynom. Es gilt

$$p(x) = (-1)^n \det \begin{pmatrix} -x & & & & & -a_0 \\ 1 & -x & & & & -a_1 \\ & 1 & -x & & & -a_2 \\ & & \ddots & \ddots & & \vdots \\ & & & 1 & -x & -a_{n-2} \\ & & & & 1 & (-x - a_{n-1}) \end{pmatrix}$$

$$= \det(x\mathbb{I} - \begin{pmatrix} 0 & & & & & -a_0 \\ 1 & 0 & & & & -a_1 \\ & 1 & 0 & & & -a_2 \\ & & \ddots & \ddots & & \vdots \\ & & & 1 & 0 & -a_{n-2} \\ & & & & 1 & -a_{n-1} \end{pmatrix});$$

$p$ ist also das charakteristische Polynom obiger Matrix. Ist $\lambda$ eine Nullstelle von $p$, so ist $\lambda$ auch ein Eigenwert von $A$ und ein Eigenvektor zu $\lambda$ ist $(1, \lambda, \lambda^2, \ldots, \lambda^{n-1})^*$. $A$ heißt *Begleitmatrix* zu $\lambda$. Die Existenz dieser Begleitmatrix zeigt, daß das Bestimmen von Polynomnullstellen und die Berechnung von Eigenwerten äquivalente mathematische Probleme sind, und an dieser Stelle kommt ein algebraisches Resultat ins Spiel, das auf Arbeiten von Abel und Galois im neunzehnten Jahrhundert aufbaut und von Abel im Jahr 1824 bewiesen worden ist.

**Theorem 1.2.1** (Abel 1824)**.** *Für jedes $n \geq 5$ existiert ein Polynom $p$ mit rationalen Koeffizienten vom Grad $n$, das eine reelle Nullstelle $r$ besitzt mit der Eigenschaft, daß $r$ nicht geschrieben werden kann als algebraischer Ausdruck, der rationale Zahlen, Additionen, Subtraktionen, Multiplikationen, Divisionen und $k$–te Wurzeln enthält. Anders ausgedrückt existiert keine Formel und damit kein endlicher algebraischer Algorithmus, der aus den Koeffizienten eines Polynoms vom Grad $n \geq 5$ die Nullstellen berechnet.*

Der oben zitierte Satz impliziert über die Äquivalenz zwischen Eigenwerten und Nullstellen von Polynomen auch die Unmöglichkeit der Bestimmung der Eigenwerte allgemeiner Matrizen $A$ mit Hilfe endlicher Algorithmen. Das hat die folgende Konsequenz: In den Kapiteln 3 und 4 haben wir Verfahren zur Lösung elementarer linear algebraischer Probleme kennengelernt, die könnten wir sie in exakter Arithmetik implementieren die behandelten Probleme exakt lösen würdeen. *Für Eigenwertberechnung ist eine solche Vorgangsweise unmöglich!* Daher muß jeder Algorithmus zur Lösung des Eigenwertproblemes approximativ sein. Fast alle Eigenwert–Algorithmen funktionieren *iterativ*.

Die meisten Verfahren zur Eigenwertberechnung versuchen eine Schur–Zerlegung von $A$ zu berechnen, indem sie eine Folge von unitären Ähnlichkeitstranformationen mit Matrizen $Q_i$ ausführen, sodaß das Produkt

$$\underbrace{Q_j^* \dots Q_2^* Q_1^*}_{Q^*} A \underbrace{Q_1 Q_2 \dots Q_j}_{Q}$$

gegen eine obere Dreiecksmatrix $R$ konvergiert für $j \to \infty$.

Um den Aufwand zu minimieren geschieht das in zwei Phasen. Zuerst wird die Matrix $A$ mittels unitären Ähnlichkeitstransformationen in Hessenbergform (siehe Abschnitt 2) transformiert und dann wird die Hessenbergmatrix $H$ iterativ Schur–faktorisiert (siehe Abschnitt 3).

## 2. Transformationsverfahren

Abhängig von der Gestalt der Matrix $A$ transformiert man sie mittels einem direkten Verfahren entweder in *Hessenberggestalt*, eine *Hessenbergmatrix* ist eine Matrix der Form

$$H = \begin{pmatrix} * & * & \cdots & \cdots & * \\ * & * & \ddots & & \vdots \\ 0 & * & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & * & * \\ 0 & \cdots & 0 & * & * \end{pmatrix},$$

wenn $A$ allgemeine Gestalt hat, oder in Tridiagonalform, wenn $A$ hermitesch ist.

**2.1. Transformation auf Hessenberggestalt.** Die Transformation einer allgemeinen Matrix in Hessenbergform funktioniert analog zur $QR$–Zerlegung, die in Kapitel 4 vorgestellt worden ist. Man verwendet Householder–Transformationen $Q_j$ um sukzessive die Nullen spaltenweise zu erzeugen. Anhand einer $(5 \times 5)$–Matrix sei die Methode beschrieben (Hervorgehoben sind diejenigen Elemente, die sich im jeweiligen Schritt ändern.):

$$\underset{A}{\begin{pmatrix} * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \end{pmatrix}} \xrightarrow{Q_1^* \cdot} \underset{Q_1^* A}{\begin{pmatrix} * & * & * & * & * \\ \mathbf{*} & \mathbf{*} & \mathbf{*} & \mathbf{*} & \mathbf{*} \\ \mathbf{0} & \mathbf{*} & \mathbf{*} & \mathbf{*} & \mathbf{*} \\ \mathbf{0} & \mathbf{*} & \mathbf{*} & \mathbf{*} & \mathbf{*} \\ \mathbf{0} & \mathbf{*} & \mathbf{*} & \mathbf{*} & \mathbf{*} \end{pmatrix}} \xrightarrow{\cdot Q_1} \underset{Q_1^* A Q_1}{\begin{pmatrix} * & \mathbf{*} & \mathbf{*} & \mathbf{*} & \mathbf{*} \\ * & \mathbf{*} & \mathbf{*} & \mathbf{*} & \mathbf{*} \\ & \mathbf{*} & \mathbf{*} & \mathbf{*} & \mathbf{*} \\ & \mathbf{*} & \mathbf{*} & \mathbf{*} & \mathbf{*} \\ & \mathbf{*} & \mathbf{*} & \mathbf{*} & \mathbf{*} \end{pmatrix}}$$

Man führt also zuerst einen Schritt wie bei einer $QR$–Zerlegung aus mit der einzigen Änderung, daß man bei der Wahl der Householdermatrix die erste Zeile von $A$ ignoriert. Danach multipliziert man von rechts mit der hermitesch konjugierten der Householdermatrix. Dies läßt die erste Spalte unverändert, und daher bleiben die bereits erzeugten Nullen unberührt. Die weiteren Schritte sehen dann etwa folgendermaßen aus:

$$\underset{A}{\begin{pmatrix} * & * & * & * & * \\ * & * & * & * & * \\ & * & * & * & * \\ & * & * & * & * \\ & * & * & * & * \end{pmatrix}} \xrightarrow{Q_2^* \cdot} \underset{Q_2^* Q_1^* A Q_1}{\begin{pmatrix} * & * & * & * & * \\ * & * & * & * & * \\ & \mathbf{*} & \mathbf{*} & \mathbf{*} & \mathbf{*} \\ & \mathbf{0} & \mathbf{*} & \mathbf{*} & \mathbf{*} \\ & \mathbf{0} & \mathbf{*} & \mathbf{*} & \mathbf{*} \end{pmatrix}} \xrightarrow{\cdot Q_2} \underset{Q_2^* Q_1^* A Q_1 Q_2}{\begin{pmatrix} * & * & \mathbf{*} & \mathbf{*} & \mathbf{*} \\ * & * & \mathbf{*} & \mathbf{*} & \mathbf{*} \\ & & \mathbf{*} & \mathbf{*} & \mathbf{*} \\ & & \mathbf{*} & \mathbf{*} & \mathbf{*} \\ & & \mathbf{*} & \mathbf{*} & \mathbf{*} \end{pmatrix}}$$

Wiederholt man $n-2$ solche Householderschritte, so erhält man eine unitäre Ähnlichkeitstransformation auf Hessenberggestalt

$$\underbrace{Q_{n-2}^* \dots Q_2^* Q_1^*}_{Q^*} A \underbrace{Q_1 Q_2 \dots Q_{n-2}}_{Q} =: H.$$

Der untenstehende Algorithmus ist eine Abwandlung von Algorithmus 2.2.7 aus Kapitel 4.

**Algorithmus 2.1.1.** *Hessenberg Transformation*

**for** $k = 1$ **to** $n-2$ **do**
$x = A_{k+1:n,k}$
$v_k = e^{i \arg(x_1)} \|x\|_2 e_1 + x$
$v_k = v_k / \|v_k\|_2$
$A_{k+1:n,k:n} = A_{k+1:n,k:n} - 2v_k(v_k^* A_{k+1:n,k:n})$
$A_{1:n,k+1:n} = A_{1:n,k+1:n} - 2(A_{1:n,k+1:n} v_k)v_k^*$
**done**

Der Algorithmus unterscheidet sich nur durch die letzte Zeile innerhalb der Schleife vom Householder–Schmidt Verfahren zur Berechnung der $QR$–Zerlegung. Wieder wird das Produkt $Q = \prod_{i=1}^{n-2} Q_i$ nicht explizit berechnet. Sollten Matrix–Vektor–Produkte benötigt werden, können sie mittels der Algorithmen 2.2.8 und 2.2.9 aus Kapitel 4 iterativ bestimmt werden.

Der Aufwand zur Bestimmung der Hessenbergtransformierten ist in etwa doppelt so groß wie für eine $QR$–Zerlegung. Asymptotisch beträgt er

$$\sim \frac{10}{3} n^3.$$

**2.2. Transformation auf Tridiagonalgestalt.** Im hermiteschen Fall führen die unitären Ähnlichkeitstransformationen $A$ wieder in eine hermitesche Matrix über. Wendet man also den Algorithmus zur Hessenbergtransformation auf eine hermitesche Matrix $A$ an, so erhält man als Resultat eine hermitesche obere Hessenbergmatrix $H$, also eine hermitesche Tridiagonalmatrix $T$. In diesem Fall kann man die meisten Rechenschritte zur Berechnung der Produkte mit den Householdermatrizen von rechts einsparen, da man das Ergebnis bereits kennt, und man kann die Symmetrie von Ausgangs– und Resultatmatrizen verwenden, um den Aufwand auf

$$\sim \frac{4}{3} n^3$$

elementare Rechenoperationen zu reduzieren.

**2.3. Gutartigkeit der Transformationen.** Wie aus der Konstruktion zu erwarten ist, ist die Hessenbergtransformation ein gutartiger Algorithmus. Erstens wird mit unitären Matrizen gearbeitet und zweitens ist er sehr ähnlich zum Householder–Schmidt–Verfahren zur Berechnung einer $QR$–Zerlegung. In der Tat gilt der folgende Satz:

**Theorem 2.3.1.** *Sei die Hessenbergtransformation $A = QHQ^*$ der Matrix $A$ aus Algorithmus 2.1.1 berechnet und seien die rundungsfehlerbehafteten Ergebnisse mit $\tilde{Q}$ und $\tilde{H}$ bezeichnet. In dieser Situation gilt*

$$\tilde{Q}\tilde{H}\tilde{Q}^* = A + \Delta A, \quad \frac{\|\Delta A\|}{\|A\|} = O(\mathsf{eps})$$

*für eine Fehlermatrix* $\Delta A$.

## 3. QR–Verfahren

In diesem Abschnitt wollen wir das verbreitetste Verfahren zur Berechnung der Eigenwerte einer oberen Hessenbergmatrix vorstellen. Man kann den Algorithmus auch für allgemeine Matrizen $A$ anwenden, doch der Aufwand ist in diesem Fall im allgemeinen $O(n^4)$ oder mehr.

Im folgenden wird der Algorithmus aus Motivationsgründen nur für reelle symmetrische Matrizen, also für Tridiagonalmatrizen, entwickelt; er läßt sich jedoch beinahe ohne Änderung auf Hessenbergmatrizen verallgemeinern. Das einzige, das wirklich geändert werden muß, ist die Wahl der richtigen Shiftstrategie, doch darauf wird in Abschnitt 3.6 genauer eingegangen.

**3.1. Der Rayleigh–Quotient.** Um Schätzwerte für Eigenwerte zu berechnen, ist der folgende Ausdruck wesentlich:

**Definition 3.1.1.** *Der* Rayleigh–Quotient *eines Vektors* $x \in \mathbb{K}^n$ *bezüglich der Matrix* $A$ *ist der Skalar*

$$r(x) = \frac{x^* Ax}{\langle x, x\rangle}.$$

Ist $y$ Eigenvektor zum Eigenwert $\lambda$, so gilt $r(y) = \lambda$. Die Formel läßt sich aus der Lösung eines Kleinste–Quadrate–Problems motivieren. Man sucht die Zahl $r(x)$, die $\|Ax - r(x)x\|_2$ minimiert. Stellt man für dieses Problem die Normalengleichungen auf ($x$ ist die Matrix, $Ax$ ist der Vektor $b$, die rechte Seite), so erhält man $r(x)x^*x = x^*Ax$, und eine Umformung dieser Beziehung führt zum Rayleigh–Quotienten.

Betrachtet man $r(x)$ als Abbildung $\mathbb{K}^n \to \mathbb{R}$, so kann man den Gradienten berechnen. Eine einfache Anwendung der Quotientenregel liefert

$$\nabla r(x) = \frac{\nabla(x^*Ax)\langle x,x\rangle - x^*Ax\nabla(\langle x,x\rangle)}{\langle x,x\rangle^2} = \frac{2Ax}{\langle x,x\rangle} - \frac{2x(x^*Ax)}{\langle x,x\rangle^2} = \frac{2}{\langle x,x\rangle}\big(Ax - r(x)x\big).$$

Werten wir $\nabla r(x)$ wieder an einem Eigenvektor $y$ zum Eigenwert $\lambda$ aus, so ist das Resultat

$$\nabla r(y) = \frac{2}{\langle y,y\rangle}\big(\lambda y - \lambda y\big) = 0;$$

die Eigenvektoren sind also gerade die stationären Punkte der Funktion $r$. Aus dieser Tatsache und der Taylorentwicklung der Funktion $r$ erhalten wir schließlich noch das Ergebnis

**Proposition 3.1.2.** *Sei* $q$ *einer der Eigenvektoren von* $A$. *Dann gilt*

$$r(x) - r(q) = O(\|x - q\|_2^2), \quad \textit{für } x \to q.$$

*Der Rayleigh–Quotient ist also ein Schätzwert für den Eigenwert zum Eigenvektor* $q$, *der in einer Umgebung von* $q$ quadratisch genau *ist.*

**3.2. Potenziteration und inverse Iteration.** Wie man der Abbildung 9.1 entnehmen kann, kann man erwarten, daß die Folge $\frac{v}{\|v\|}, \frac{Av}{\|Av\|}, \frac{A^2v}{\|A^2v\|}, \frac{A^3v}{\|A^3v\|}, \ldots$ mit einem (beinahe) beliebigen Startvektor $v$ gegen einen Eigenvektor zum größten Eigenwert $\lambda_1$ konvergiert. Nehmen wir an, daß der Startvektor $v$ der Folge nicht orthogonal auf $q_1$, den Eigenvektor zu $\lambda_1$ steht. Dann gilt eine Gleichung der Gestalt

$$\frac{v}{\|v\|} = a_1q_1 + a_2q_2 + \cdots + a_nq_n,$$

wobei die $q_i$ die Eigenvektoren von $A$ bezeichne. Die einzelnen Folgenglieder erfüllen dann die Beziehung

$$\frac{A^kv}{\|A^kv\|} = c_k(\lambda_1^k a_1q_1 + \lambda_2^k a_2q_2 + \cdots + \lambda_n^k a_nq_n).$$

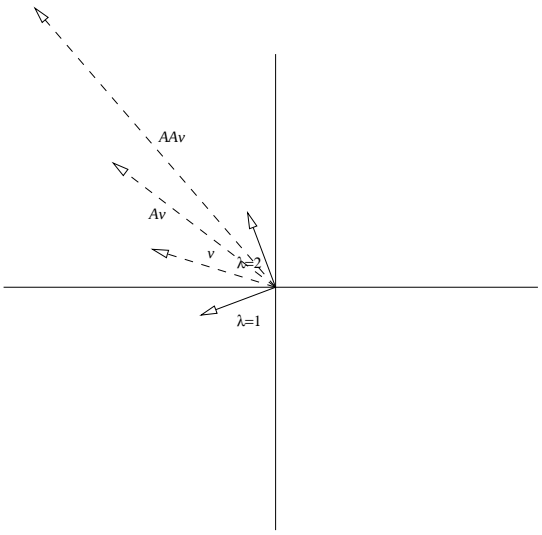



ABBILDUNG 9.1. Potenziteration

für irgendeine Konstante $c_k$. Die Gleichung kann noch etwas umgeformt werden zu

$$\frac{A^k v}{\|A^k v\|} = c_k \lambda_1^k \left( a_1 q_1 + a_2 \left(\tfrac{\lambda_2}{\lambda_1}\right)^k q_2 + \cdots + a_n \left(\tfrac{\lambda_n}{\lambda_1}\right)^k q_n \right).$$

Weil die Konstanten $c_k$ so gewählt sind, daß die Norm der Folgenglieder konstant gleich eins bleibt, erhalten wir den folgenden Satz.

**Theorem 3.2.1.** *Unter den Voraussetzungen* $|\lambda_1| > |\lambda_2| \geq \cdots \geq |\lambda_n| \geq 0$ *und* $\langle q_1, v\rangle \neq 0$ *konvergiert die Folge* $(v^{(k)} := A^k v/\|A^k v\|)$ *und die Folgenglieder erfüllen*

$$\|v^{(k)} - (\pm q_1)\| = O\left(\left|\frac{\lambda_2}{\lambda_1}\right|^k\right), \quad |\lambda^{(k)} - \lambda_1| = O\left(\left|\frac{\lambda_2}{\lambda_1}\right|^{2k}\right)$$

*für* $k \to \infty$, *wobei* $\lambda^{(k)} := r(v^{(k)})$ *den zu* $v^{(k)}$ *gehörigen Rayleigh–Quotienten bezeichne. Das* $\pm$ *bedeutet, daß in jedem Schritt entweder die eine oder die andere Wahl des Vorzeichens die Abschätzung richtig macht.*

Den Algorithmus der Potenziteration kann man folgendermaßen formulieren:

**Algorithmus 3.2.2.** *Potenziteration*
$v^{(0)}$ *ist beliebig mit* $\|v^{(0)}\| = 1$.
**for** $k = 1$ **to** ? **do**
$w = Av^{(k-1)}$
$v^{(k)} = w/\|w\|$
$\lambda^{(k)} = (v^{(k)})^\top A v^{(k)}$ `; Rayleigh--Quotient`
**done**

Das Abbruchkriterium ist in diesem Fall nicht angegeben, da gerade an diesem Punkt sich Top–Qualitäts–Software auszeichnet. Der mathematische Hintergrund dazu ist meist nicht so ausgeprägt; es zählen eher Erfahrungstatsachen und statistische Untersuchungen.

Leider ist die Potenziteration nur von beschränktem Nutzen, da sie erstens nur den größten Eigenwert und einen dazupassenden Eigenvektor finden kann und zweitens die Konvergenz nur linear mit dem Faktor $|\lambda_2/\lambda_1|$ ist, der für viele Matrizen nahe bei 1 liegt; die Konvergenz ist also meist sehr langsam. Glücklicherweise lassen sich diese beiden Fakten durch einen einfachen Trick ändern.

Für jedes $\mu \in \mathbb{R}$, das nicht Eigenwert von $A$ ist, stimmen die Eigenvektoren von $A$ und $(A-\mu\mathbb{I})^{-1}$ überein. Ist nämlich $Ax = \lambda x$, so folgt

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

Nehmen wir nun an, daß $\mu$ sehr nahe an einem bestimmten Eigenwert $\lambda_j$ ist, so ist $(\lambda_j-\mu)^{-1}$ sehr viel größer als $(\lambda_i-\mu)^{-1}$ für $i \neq j$. Wenden wir also die Potenziteration statt auf die Matrix $A$ auf die Matrix $(A-\mu\mathbb{I})^{-1}$ an, so wird die erzeugte Folge relativ rasch gegen den zu $\lambda_j$ gehörenden Eigenvektor $q_j$ konvergieren. Das entstehende Verfahren heißt *inverse Iteration* und ist ein Standardverfahren zur Bestimmung eines Eigenvektors bei gegebenem Schätzwert für den Eigenwert. Aus Theorem 3.2.1 kann man sofort das folgende Theorem ablesen:

**Theorem 3.2.3.** *Sei $\lambda_j$ der Eigenwert, der $\mu$ am nächsten liegt, und sei $\lambda_k$ der zweitnächste. Dann gilt $|\mu-\lambda_j| < |\mu-\lambda_k| \leq |\mu-\lambda_i|$ für $i \neq j$. Sei weiters $\langle q_j, v\rangle \neq 0$; dann konvergiert die Folge $((A-\mu\mathbb{I})^{-k}/\|(A-\mu\mathbb{I})^{-k}\|)$ und ihre Glieder $v^{(k)}$ erfüllen*

$$\|v^{(k)} - (\pm q_j)\| = O\left(\left|\frac{\mu-\lambda_j}{\mu-\lambda_k}\right|^k\right), \quad |\lambda^{(k)}-\lambda_j| = O\left(\left|\frac{\mu-\lambda_j}{\mu-\lambda_k}\right|^{2k}\right)$$

*für $k \to \infty$, wobei $\lambda^{(k)}$ wieder die Rayleigh–Quotienten zu $v^{(k)}$ bezeichnet.*

Diese Konvergenzbeschleunigung ist ein außerordentlich wichtiges Werkzeug bei der numerischen Eigenwert/Eigenvektor–Suche. Der folgende Algorithmus faßt sie zusammen:

**Algorithmus 3.2.4.** *Inverse Iteration*

$v^{(0)}$ *beliebig mit* $\|v^{(0)}\| = 1$.
**for** $k = 1$ **to** ? **do**
  *Löse* $(A-\mu\mathbb{I})w = v^{(k-1)}$ *nach* $w$.
  $v^{(k)} = w/\|w\|$
  $\lambda^{(k)} = (v^{(k)})^\top A v^{(k)}$ `; Rayleigh--Quotient`
**done**

**3.3. Rayleigh–Quotient Iteration.** Wir kennen also jetzt eine Möglichkeit, einen Eigenwert zu schätzen, wenn wir einen Schätzwert für einen Eigenvektor kennen (den Rayleigh–Quotienten); und wir kennen eine Möglichkeit, einen Vektor zu berechnen, der näher an einem Eigenvektor liegt, wenn wir einen guten Schätzwert für einen Eigenwert kennen (die inverse Iteration). Eine naheliegende Idee ist also, die beiden Methoden miteinander zu kombinieren. Daraus resultiert die *Rayleigh–Quotient Iteration.*

**Algorithmus 3.3.1.** *Rayleigh–Quotient Iteration*

$v^{(0)}$ *beliebig mit* $\|v^{(0)}\| = 1$.
$\lambda^{(0)} = (v^{(0)})^\top A v^{(0)}$
**for** $k = 1$ **to** ? **do**
  *Löse* $(A-\lambda^{(k-1)}\mathbb{I})w = v^{(k-1)}$ *nach* $w$.
  $v^{(k)} = w/\|w\|$
  $\lambda^{(k)} = (v^{(k)})^\top A v^{(k)}$ `; Rayleigh--Quotient`
**done**

Die kleine Änderung im Algorithmus, das Anpassen des Shiftwertes $\mu$ in jedem Schritt, hat dramatische Auswirkungen auf die Geschwindigkeit des Algorithmus. Jetzt kombinieren sich nämlich die lineare Konvergenz der inversen Iteration und die quadratische Genauigkeit des Rayleigh–Quotienten. Es ist also nicht weiter verwunderlich, daß das folgende Resultat gilt.

**Theorem 3.3.2.** *Die Rayleigh–Quotient Iteration konvergiert für fast alle (alle bis auf eine Menge vom Maß Null) Startwerte $v^{(0)} \in \mathbb{R}^n$ gegen ein Eigenwert/Eigenvektor–Paar. Wenn sie konvergiert, so konvergiert sie schließlich mit* kubischer *Geschwindigkeit. Ist also $v^{(0)}$ genügend nahe am Grenzvektor $q_j$, so gelten*

$$\|v^{(k+1)} - (\pm q_j)\| = O(\|v^{(k)} - (\pm q_j)\|^3)$$

*und*

$$|\lambda^{(k+1)} - \lambda_j| = O(|\lambda^{(k)} - \lambda_j|^3)$$

*für $k \to \infty$. Die Vorzeichen in der ersten Gleichung müssen auf beiden Seiten geeignet gewählt werden.*

Um zu unterstreichen wie schnell kubische Konvergenz ist (sie verdreifacht in jedem Iterationsschritt die Anzahl der korrekten Stellen), sei das folgende Beispiel zitiert:

**Beispiel 3.3.3.** *Sei die Matrix*

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

*gegeben. Mit dem Startwert $v^{(0)} = 1/\sqrt{3}(1,1,1)^\top$ sind die ersten Glieder der Folge der $\lambda^{(k)}$:*

$$\lambda^{(0)} = \underline{5}, \quad \lambda^{(1)} = \underline{5.21}31\ldots, \quad \lambda^{(2)} = \underline{5.214319743}184\ldots$$

*Der wahre Wert des Eigenwertes zu demjenigen Eigenvektor, der $v^{(0)}$ am nächsten liegt, ist $\lambda = 5.214319743377$. Nach drei Iterationen ist also bereits zehnstellige Genauigkeit erreicht. Drei weitere Iterationsschritte würden bei genügend genauer Rechnung die Zahl der korrekten Stellen auf etwa 270 erhöhen.*

Jeder Schritt der Rayleigh–Quotienten Iteration benötigt die Lösung eines linearen Gleichungssystems. Weil die Matrix in jedem Schritt verändert wird, kann man auch nicht im Vorhinein $LR$–zerlegen, und damit ist der Aufwand für einen Iterationsschritt $O(n^3)$.

Ist allerdings $A$ eine obere Hessenbergmatrix, so senkt sich der Aufwand durch die große Anzahl von Nullen auf $O(n^2)$, und für Tridiagonalmatrizen gar auf $O(n)$, was die Wichtigkeit der ersten Phase, der Hessenbergtransformation, noch einmal unterstreicht.

**3.4. QR–Verfahren ohne Shifts.** Obwohl die Rayleigh–Quotienten Iteration eine hervorragende Möglichkeit bietet, einzelne Eigenwert/Eigenvektor–Paare effizient zu berechnen, ist das Verfahren doch zu aufwendig, wenn es darum geht, alle Eigenwerte einer Matrix zu berechnen. Der folgende absurd einfach wirkende Algorithmus ist der Start zur Lösung dieses Problems.

**Algorithmus 3.4.1.** *$QR$–Verfahren ohne Shifts*

$A^{(0)} = A$
**for** $k = 1$ **to** ? **do**
  $Q^{(k)}R^{(k)} = A^{(k-1)}$ ; $QR$`--Zerlegung von` $A^{(k-1)}$
  $A^{(k)} = R^{(k)}Q^{(k)}$ ; `Ausmultiplizieren in umgekehrter Reihenfolge`
**done**

Auf den ersten Blick erscheint es unklar, warum dieser Algorithmus überhaupt etwas vernünftiges berechnen soll. Trotzdem steht die Behauptung, daß die Folge $(A^{(k)})$ gegen eine obere Dreiecksmatrix konvergiert (bzw. gegen eine Diagonalmartix im hermiteschen Fall). Das einzige, das man leicht überprüfen kann ist, daß wirklich unitäre Ähnlichkeitstransformationen ausgeführt werden:

$$A^{(k)} = {Q^{(k)}}^* A^{(k-1)} Q^{(k)}.$$

Der Rest hängt stark mit einem weiteren Verfahren zusammen, der gleichzeitigen Iteration, welche wiederum auf der Blockpotenziteration aufbaut.

**3.4.1. Blockpotenziteration.** Was mag passieren, wenn man bei der Potenziteration anstelle eines einzelnen Vektors eine Menge $\{v_0^{(0)}, \dots, v_m^{(0)}\}$ von linear unabhängigen Vektoren gleichzeitig verwendet? Nachdem $A$ regulär ist, sind die Mengen $\{A^k v_0^{(0)}, \dots, A^k v_m^{(0)}\}$ jeweils wieder linear unabhängig. In Matrixnotation können wir das etwa folgendermaßen schreiben:

$$V^{(0)} := (v_0^{(0)} | \cdots | v_m^{(0)})$$

sei die Startmatrix. Definiere $V^{(k)}$ als

$$V^{(k)} := A^k V^{(0)} = (v_0^{(k)} | \cdots | v_m^{(k)}).$$

Nachdem unser Hauptinteresse dem Raum aufgespannt von den Spalten von $V^{(k)}$ gilt, bestimmen wir eine Orthonormalbasis dafür mit Hilfe einer reduzierten $QR$–Zerlegung

$$\hat{Q}^{(k)} \hat{R}^{(k)} = V^{(k)}.$$

Nachdem der von $V^{(k)}$ aufgespannte Raum aus Gründen, die analog sind zu den Gründen für die Konvergenz der Potenziteration, gegen den Raum konvergiert, der von den Eigenvektoren zu den $m$ betragsgrößten Eigenwerten konvergiert, konvergiert die Matrix $\hat{Q}^{(k)}$ gegen $Q = (q_1 | \cdots | q_m)$, die Matrix gebildet aus ebendiesen Eigenvektoren.

Der Beweis dafür ist ebenfalls analog zum Beweis für die Konvergenz der Potenziteration. Man entwickelt alle $v_j^{(0)}$ in die Eigenvektoren von $A$ und verwendet danach die Formeln für die Gram–Schmidt Orthonormalisierung. Es resultiert der folgende Satz:

**Theorem 3.4.2.** *Seien die $m$ betragsgrößten Eigenwerte vom Betrag her verschieden*

$$|\lambda_1| > \cdots > |\lambda_m| \geq |\lambda_{m+1}| \geq \cdots \geq 0,$$

*und sind die $m$ führenden Hauptminoren der Matrix $\hat{Q}^\top V^{(0)}$ nichtverschwindend, wobei $\hat{Q} = (q_1 | \cdots | q_m)$ ist. Dann konvergieren die Spalten der Matrizen $Q^{(k)}$ mit linear gegen die Eigenvektoren zu den $m$ betragsgrößten Eigenwerten von $A$:*

$$\|q_j^{(k)} - (\pm q_j)\| = O(C^k),$$

*für $1 \leq j \leq m$, wobei*

$$C = \max_{1 \leq k \leq m} \frac{|\lambda_{k+1}|}{|\lambda_k|} < 1$$

*ist. Wieder muß man das Vorzeichen $\pm$ in jedem Schritt und für jedes $j$ geeignet wählen.*

Beweis. Sei $\hat{Q}$ ergänzt zu einer orthogonalen Matrix $Q$ aus Eigenvektoren von $A$. Dann gilt $A = Q \Lambda Q^\top$, und sei $\hat{\Lambda}$ der führende $(m \times m)$–Teil der Matrix $\Lambda$. Dann gilt

$$V^{(k)} = A^k V^{(0)} = Q \Lambda^k Q^\top V^{(0)} = \hat{Q} \hat{\Lambda}^k \hat{Q}^{(k)} V^{(0)} + O(|\lambda_{m+1}|^k)$$

für $k \to \infty$. Aus der Hauptminorenbedingung folgt, daß $\hat{Q}^\top V^{(0)}$ regulär ist. Daher kann man die obige Gleichung umformen zu

$$V^{(k)} = \big(\hat{Q} \hat{\Lambda}^k + O(|\lambda_{m+1}|^k)\big) \hat{Q}^\top V^{(0)}.$$

Nachdem $\hat{Q}^\top V^{(0)}$ regulär ist, ist der Spaltenraum von $V^{(k)}$ derselbe wie der Spaltenraum von

$$\hat{Q} \hat{\Lambda}^k + O(|\lambda_{m+1}|^k).$$

Klarerweise konvergiert dieser Raum linear gegen den Spaltenraum von $\hat{Q}$. Nachdem das Argument auch für alle führenden Teile der Matrix $V^{(k)}$ gilt, weil jeder der ersten $m$ Hauptminoren ungleich Null ist, konvergieren auch alle führenden Teilmengen der Spalten von $V^{(k)}$

gegen die entsprechende führende Spaltenmenge von $\hat{Q}$. Aus der Definition der $QR$–Zerlegung folgt der Rest der Behauptungen. □

**3.4.2. Gleichzeitige Iteration.** Für numerische Zwecke ist das Verfahren der Blockpotenziteration denkbar ungünstig, da jeder einzelne der Vektoren $v_j^{(0)}$ gegen ein Vielfaches des Eigenvektors zum betragsgrößten Eigenwert konvergiert. Die Kondition der Basis $\{v_0^{(k)}, \ldots, v_m^{(k)}\}$ wird also mit zunehmendem $k$ immer schlechter. Die Rundungsfehler würden also mit der Zeit die Informationen zerstören. Die Lösung des Problemes ist jedoch simpel. Man orthonormalisiert die Basis in jedem Schritt der Iteration. Das verändert den aufgespannten Raum nicht, verhindert aber die Verschlechterung der Kondition. Dieses Vorgehen führt zum Algorithmus der *gleichzeitigen Iteration*.

**Algorithmus 3.4.3.** *Gleichzeitige Iteration*
*Wähle $Q^{(0)}$ mit orthogonalen Spalten*
**for** $k = 1$ **to** ? **do**
$Z = A\hat{Q}^{(k-1)}$
$\hat{Q}^{(k)}\hat{R}^{(k)} = Z$ ; `reduzierte` $QR$`--Zerlegung` **done**

**Theorem 3.4.4.** *Aus der Form dieses Algorithmus folgt, daß er unter den Voraussetzungen von Theorem 3.4.2 auch analoge Ergebnisse wie in Theorem 3.4.2 gelten.*

**3.4.3. Zusammenhang zwischen gleichzeitiger Iteration und $QR$–Verfahren.** Das Ziel dieses Abschnittes ist es zu zeigen, daß das $QR$–Verfahren äquivalent ist zur gleichzeitigen Iteration angewendet auf die Startmatrix $Q^{(0)} = \mathbb{I}$. Da die Startmatrix und damit auch alle iterierten Matrizen $Q^{(k)}$ quadratisch sind, muß man keine reduzierten $QR$–Zerlegungen berechnen sondern kann volle $QR$–Faktorisierungen verwenden. Im folgenden seien die erzeugten Matrizen statt mit $\hat{Q}^{(k)}$ mit $\underline{Q}^k$ und statt $\hat{R}^{(k)}$ mit $R^{(k)}$ bezeichnet. Verfolgt man mit diesen Bezeichnungen den Algorithmus 3.4.3, so kann man die folgenden Beziehungen zwischen den einzelnen Matrizen ablesen. Die Gleichungen (68), (69) und (70) folgen direkt aus dem Algorithmus. (71) ist eine Definition für die Matrizen $A^{(k)}$:

$$\underline{Q}^{(0)} = \mathbb{I} \tag{68}$$
$$Z^{(k)} = A\underline{Q}^{(k-1)} \tag{69}$$
$$Z^{(k)} = \underline{Q}^{(k)}R^{(k)} \tag{70}$$
$$A^{(k)} = \left(\underline{Q}^{(k)}\right)^\top A\underline{Q}^{(k)} \tag{71}$$

Auf ähnliche Weise kann man den $QR$–Algorithmus in Gleichungen zusammenfassen. Beziehung 75 dient wieder als Definition der Matrix $\underline{Q}^{(k)}$:

$$A^{(0)} = A \tag{72}$$
$$A^{(k-1)} = Q^{(k)}R^{(k)} \tag{73}$$
$$A^{(k)} = R^{(k)}Q^{(k)} \tag{74}$$
$$\underline{Q}^{(k)} = Q^{(1)}Q^{(2)}\ldots Q^{(k)} \tag{75}$$

Die Übereinstimmung in den Bezeichnungen $A^{(j)}$ und $\underline{Q}^{(j)}$ in den Gleichungen für beide Algorithmen wurden nicht von ungefähr gewählt. Der nächste Satz wird zeigen, daß es sich dabei um jeweils dieselben Matrizen handelt. Für beide Algorithmen sei zusätzlich noch die Bezeichnung

$$\underline{R}^{(k)} = R^{(k)}R^{(k-1)}\ldots R^{(1)} \tag{76}$$

eingeführt. Mit Hilfe dieser Notation kann man den folgenden Satz beweisen

**Theorem 3.4.5.** *Die Algorithmen 3.4.1 und 3.4.3 erzeugen identische Folgen von Matrizen* $\underline{R}^{(k)}$, $\underline{Q}^{(k)}$ *und* $A^{(k)}$, *die definiert sind durch die* $QR$*–Faktorisierung der* $k$*–ten Potenzen von* $A$

$$A^k = \underline{Q}^{(k)}\underline{R}^{(k)}.$$

$A^{(k)}$ *ist die Projektion*

$$A^{(k)} = \left(\underline{Q}^{(k)}\right)^\top A\underline{Q}^{(k)}.$$

Beweis. Der Beweis erfolgt mittels Induktion nach $k$. Für $k = 0$ ist die Übereinstimmung offensichtlich. Für beide Algorithmen gilt nämlich $A^0 = \underline{Q}^{(0)} = \underline{R}^{(0)} = \mathbb{I}$ und $A^{(0)} = A$, wie man leicht aus den Gleichungen (68), (71), (72) und (76) ablesen kann. Für $k \geq 1$ folgt für die gleichzeitige Iteration

$$A^k = A\underline{Q}^{(k-1)}\underline{R}^{(k-1)} = Z^{(k)}\underline{R}^{(k-1)} = \underline{Q}^{(k)}R^{(k)}\underline{R}^{(k-1)} = \underline{Q}^{(k)}\underline{R}^{(k)},$$

wobei man in dieser Reihenfolge die Induktionsvoraussetzung und die Gleichungen (69), (70), (76) verwendet.

Andererseits beweist man für das $QR$–Verfahren

$$\begin{aligned} A^k &= A\underline{Q}^{(k-1)}\underline{R}^{(k-1)} = A^{(0)}Q^{(1)}Q^{(2)}\dots Q^{(k-1)}\underline{R}^{(k-1)} = \\ &= Q^{(1)}A^{(1)}Q^{(2)}\dots Q^{(k-1)}\underline{R}^{(k-1)} = \dots = \\ &= Q^{(1)}Q^{(2)}\dots Q^{(k-1)}A^{(k-1)}\underline{R}^{(k-1)} = \underline{Q}^{(k-1)}Q^{(k)}R^{(k)}\underline{R}^{(k-1)} = \underline{Q}^{(k)}\underline{R}^{(k)}, \end{aligned}$$

wobei man in dieser Reihenfolge die Induktionsvoraussetzung, dann die Gleichungen (75), $k-1$ Mal die Gleichungen (73) und (74), wieder (74) und zuletzt (76) verwendet.

Damit ist die Äquivalenz der Algorithmen gezeigt und die erste Behauptung. Die zweite Behauptung folgt zum Beispiel aus (71). □

Aus der Äquivalenz von $QR$–Verfahren und gleichzeitiger Iteration können wir auch Resultate über die Konvergenz des $QR$–Verfahrens herleiten. Aus (71) folgt, daß die Diagonalelemente der Matrizen $A^{(j)}$ Rayleigh–Quotienten von $A$ zu den Spalten von $\underline{Q}^{(k)}$ sind. Wenn diese Spalten gegen Eigenvektoren von $A$ konvergieren, konvergieren die Hauptdiagonalelemente gegen die zugehörigen Eigenwerte. Die Elemente abseits der Hauptdiagonale werden aus inneren Produkten zweier verschiedener Spalten von $\underline{Q}^{(k)}$ gebildet, und da diese Spalten orthogonal werden, verschwinden sie im Grenzwert. Also konvergiert $A^{(k)}$ gegen eine Diagonalmatrix. Ferner folgt aus Theorem 3.4.4 das Resultat über die Konvergenz des $QR$–Verfahrens

**Theorem 3.4.6.** *Sei der Algorithmus 3.4.1 angewendet auf die reelle symmetrische Matrix* $A$, *deren Eigenwerte*

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

*erfüllen und deren entsprechende Eigenvektormatrix* $Q$ *nichtverschwindende Hauptminoren hat. Dann gilt für* $k \to \infty$, *daß* $A^{(k)}$ *linear mit Konstante* $\max_k |\lambda_{k+1}|/|\lambda_k|$ *gegen* $\operatorname{diag}(\lambda_1, \dots, \lambda_n)$ *konvergiert. Ferner konvergiert* $\underline{Q}^{(k)}$ *mit derselben Geschwindigkeit gegen* $Q$.

**3.5. QR–Verfahren mit Shifts.** Niemand würde sich für das $QR$–Verfahren interessieren, wenn die erreichbare Konvergenzordnung nur linear ist. Doch ein Trick, der dem Übergang von der Potenziteration zur Rayleigh–Quotienten Iteration entspricht, verleiht dem Algorithmus Flügel. Es ist dies die Einführung von Shifts $A \to A - \mu\mathbb{I}$. Eine weitere Verbesserung, die *Deflation* zerteilt die Matrix $A^{(k)}$ in Untermatrizen. Der praktisch verwendete $QR$–Algorithmus ist im folgenden zusammengefaßt.

**Algorithmus 3.5.1.** *$QR$–Verfahren mit Shifts*

*Wähle eine Toleranz $\varepsilon$.*
$\left(Q^{(0)}\right)^\top A^{(0)}Q^{(0)} = A$ ; `tridiagonalisieren von` $A$
**for** $k = 1$ **to** ? **do**
  *Wähle einen Shift* $\mu^{(k)}$
  $Q^{(k)}R^{(k)} = A^{(k-1)} - \mu^{(k)}\mathbb{I}$ ; $QR$`--Zerlegung`
  $A^{(k)} = R^{(k)}Q^{(k)} + \mu^{(k)}\mathbb{I}$
  **if** $A_{j,j+1} < \varepsilon$ **then**
    $A_{j,j+1} = 0$
    $A_{j+1,j} = 0$
    *Zerlege* $A^{(k)}$ *in zwei Teilmatrizen* $\begin{pmatrix} A_1 & 0 \\ 0 & A_2 \end{pmatrix}$
    *Wende das $QR$–Verfahren ab nun auf $A_1$ und $A_2$ getrennt an.*
  **endif**
**done**

Dieser Algorithmus ist das Standardverfahren zur Bestimmung der Eigenwerte und Eigenvektoren einer Matrix $A$ seit Anfang der Sechzigerjahre. Erst zu Beginn der Neunzigerjahre ist ein neues Verfahren entwickelt worden, das beginnt das $QR$–Verfahren zu verdrängen, die Divide-and-conquer Algorithmen, die in Abschnitt 4.3 kurz beschrieben werden.

Definiert man $\underline{Q}^{(k)}$ und $\underline{R}^{(k)}$ analog zu Abschnitt 3.4.3, so kann man leicht herleiten, daß $\underline{Q}^{(k)}$ und $\underline{R}^{(k)}$ eine $QR$–Zerlegung einer geshifteten Potenz von $A$ bilden:

$$(A - \mu^{(k)}\mathbb{I})(A - \mu^{(k-1)}\mathbb{I}) \dots (A - \mu^{(1)}\mathbb{I}) = \underline{Q}^{(k)}\underline{R}^{(k)}.$$

Die erste Spalte von $\underline{Q}^{(k)}$ ist das Resultat einer wiederholt geshifteten Potenziteration mit dem Startvektor $e_1$. Die letzte Spalte ist das Resultat einer wiederholt geshifteten inversen Iteration angewendet auf den Startvektor $e_n$. Sind die Shifts gute Schätzwerte für einen Eigenwert, so konvergiert diese letzte Spalte sehr schnell gegen einen Eigenvektor.

Wie man $\mu^{(k)}$ am besten wählt, ist Untersuchungsgegenstand des nächsten Abschnittes.

**3.6. Shift–Strategien.** Aus den Resultaten über die Rayleigh–Quotienten Iteration kann man herauslesen was sehr gute Eigenwertschätzwerte $\mu^{(k)}$ sind. Man wählt am besten den Rayleigh–Quotienten

$$\mu^{(k)} = \frac{\left(q_n^{(k)}\right)^\top A q_n^{(k)}}{q_n^{(k)}} = \left(q_n^{(k)}\right)^\top A q_n^{(k)}.$$

Wie wir schon zuvor festgestellt haben, benötigt man keinen weiteren Rechenaufwand zur Bestimmung von $\mu^{(k)}$. Es gilt nämlich

$$\left(q_n^{(k)}\right)^\top A q_n^{(k)} = e_n^\top \underline{Q}^{(k)\top} A \underline{Q}^{(k)} e_n = e_n^\top A^{(k)} e_n = A_{nn}^{(k)},$$

man kann also den benötigten Rayleigh–Quotienten von der Hauptdiagonale der Matrix $A^{(k)}$ ablesen. Die Wahl $\mu^{(k)} = A_{nn}^{(k)}$ heißt wegen dieser Zusammenhänge auch *Rayleigh–Quotienten Shift*. Aus den Verbindungen zwischen $QR$–Verfahren mit Shifts und der Rayleigh–Quotienten Iteration folgt, daß das $QR$–Verfahren mit Rayleigh–Quotienten Shift schließlich kubisch konvergiert. Trotz der phantastischen Konvergenzgeschwindigkeit ist dieser Shift nicht erste Wahl.

Der Grund dafür ist, daß die Konvergenz des Verfahrens nicht garantiert werden kann. Betrachtet man nämlich die Matrix

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

so sieht man daß das $QR$–Verfahren ohne Shift nicht konvergiert:

$$A = Q^{(1)}R^{(1)} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$
$$A^{(1)} = R^{(1)}Q^{(1)} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} = A.$$

Der Rayleigh–Quotienten Shift $\mu = A_{nn}$ hat keinen Effekt, da $A_{nn} = 0$, also versagt auch das $QR$–Verfahren mit Rayleigh–Quotienten Shift. Das Problem ist, daß die Eigenwerte der Matrix gleich 1 und $-1$ sind. Soll also die Schätzung 0 für die Eigenwerte verbessert werden, so ist wegen der Symmetrie unklar welcher Eigenwert favorisiert werden soll, und der Schätzwert wird gar nicht verbessert. In so einem Fall versagt der Rayleigh–Quotienten Shift, und das $QR$–Verfahren konvergiert mit dieser Shift–Strategie gar nicht.

Um Symmetrien dieser Art zu brechen untersuchen wir die rechte untere $2 \times 2$ Teilmatrix von $A^{(k)}$:

$$B = \begin{pmatrix} A_{n-1,n-1} & A_{n-1,n} \\ A_{n,n-1} & A_{nn} \end{pmatrix}.$$

Der *Wilkinson–Shift* ist definiert als derjenige Eigenwert von $B$, der näher an $A_{nn}$ liegt. Im Fall eines Unentschiedens wählt man irgendeinen. Eine numerisch stabile Formel für den Wilkinson–Shift ist

$$\mu = A_{nn} - \frac{\operatorname{sgn}(\delta) A_{n,n-1}^2}{|\delta| + \sqrt{\delta^2 + A_{n-1,n-1}^2}},$$

wobei $\delta = \frac{1}{2}(A_{n-1,n-1} - A_{nn})$ ist. Sollte $\delta = 0$ gelten, so setzt man $\operatorname{sgn}(\delta) = 1$. Im generischen Fall erzielt der Wilkinson–Shift ebenso wie der Rayleigh–Quotienten Shift kubische Konvergenz. Außerdem gilt, daß im schlimmsten Fall wenigstens noch quadratische Konvergenz erreicht wird, das $QR$–Verfahren zusammen mit dem Wilkinson–Shift konvergiert also im Gegensatz zum Rayleigh–Quotienten Shift immer.

Im obigen Beispiel ist der Wilkinson–Shift gleich 1, und die Konvergenz erfolgt in einem Schritt.

Es gibt noch andere Shift–Strategien, besonders solche, die die explizite Subtraktion $A^{(k-1)} - \mu^{(k)}\mathbb{I}$ vermeiden, um Auslöschungseffekte zu minimieren. Solche *implizite Shift–Strategien* verbreiten sich in der letzten Zeit immer mehr. Sie sind aber vor allem dann von Vorteil, wenn unsymmetrische Hessenbergmatrizen Schur–faktorisiert werden sollen. Im unsymmetrischen Fall muß auch das Auftreten komplexer Eigenwerte in Betracht gezogen werden. Ein $QR$–Doppelschritt ist dann die Lösung. Genauere Beschreibungen dieser Verfahren können etwa in [**Golub, Van Loan 1996**, chapters 7 and 8] nachgeschlagen werden.

Zum Abschluß der Abhandlungen über das $QR$–Verfahren muß noch etwas über die Gutartigkeit des Algorithmus ausgesagt werden.

**Theorem 3.6.1.** *Sei eine reelle symmetrische Matrix $A \in \mathbb{K}^{n\times n}$ diagonalisiert mittels Algorithmus 3.5.1, wobei $\mu^{(k)}$ der Wilkinson–Shift sei, und es mögen $\tilde{\Lambda}$ und $\tilde{Q}$ die berechneten Faktoren bezeichen. Dann gilt*

$$\tilde{Q}\tilde{\Lambda}\tilde{Q}^* = A + \Delta A, \quad \frac{\|\Delta A\|}{\|A\|} = O(\mathsf{eps})$$

*für eine geeignete Fehlermatrix $\Delta A$. Daher erfüllen die bestimmten Eigenwerte $\tilde{\lambda}_j$ die Abschätzung*

$$\frac{|\tilde{\lambda}_j - \lambda_j}{\|A\|} = O(\mathsf{eps}).$$

Der Algorithmus ist also gutartig. Der Aufwand von $\sim \frac{4}{3}n^3$ ist umso bemerkenswerter als er etwa ein Drittel des Aufwandes zur standardmäßigen Berechnung eines Matrixproduktes zweier $n \times n$ Matrizen ist.

## 4. Andere Verfahren

Im Laufe der Zeit sind neben dem $QR$–Verfahren noch andere Algorithmen zur Eigenwertberechnung interessant gewesen oder geworden.

**4.1. Jacobi Verfahren.** Der älteste Algorithmus wurde 1845 von Jacobi konstruiert. Er basiert auf der Tatsache, daß zwar Matrizen der Größe $5 \times 5$ oder mehr nicht explizit diagonalisiert werden können, kleinere Matrizen abar schon.

Im Jacobi Verfahren diagonalisiert man immer wieder $2 \times 2$ Teilmatrizen in der Form

$$J^\top \begin{pmatrix} a & b \\ b & c \end{pmatrix} J = \begin{pmatrix} \neq 0 & 0 \\ 0 & \neq 0 \end{pmatrix}$$

und hofft, daß dabei die gesamte Matrix gegen eine Diagonalmatrix konvergiert. Das Standardverfahren wählt für $J$ Rotationsmatrizen der Form

$$J = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}$$

mit

$$\tan(2\theta) = \frac{2b}{c-a};$$

solch eine Matrix wird in der Literatur mit *Jacobirotation* bezeichnet.

Als zu diagonalisierende Untermatrix wählt man üblicherweise diejenige $2 \times 2$ Teilmatrix, die das betragsgrößte Nebendiagonalelement enthält. Man kann zeigen, daß bei dieser Wahl die Matrix mit quadratischer Geschwindigkeit gegen eine Diagonalmatrix konvergiert. Die Konvergenzordnung ist also 2 im Gegensatz zu 3 beim $QR$–Verfahren.

Der Hauptvorteil des Jacobi–Verfahrens ist die gute Parallelisierbarkeit. Nachdem in jedem Schritt immer nur zwei Zeilen und Spalten betroffen sind, kann man gleichzeitig mehrere Jacobirotationen beinahe unabhängig voneinander anwenden. Ist die Zahl der zur Verfügung stehenden CPUs sehr groß, so übertrifft das Jacobi–Verfahren das nur schlecht parallelisierbare $QR$–Verfahren. Seit dem Aufkommen von Divide–and–conquer Algorithmen, die ebenfalls gut parallelisierbar sind aber viel schneller konvergieren, ist die Bedeutung des Jacobi–Verfahrens stark zurückgegangen.

**4.2. Bisektion.** Das Bisektionsverfahren dient vor allem der Bestimmung eines kleinen Teils der Eigenwerte. Mit Hilfe der *Bisektion* kann man etwa die größten 5 Eigenwerte, die kleinsten 10 oder alle im Intervall $[a, b]$ finden. Sind die gesuchten Eigenwerte bestimmt, so kann man die zugehörigen Eigenvektoren durch einen inversen Iterationsschritt berechnen.

Die Idee zur Bestimmung der Eigenwerte liegt darin, die Nullstellen des charakteristischen Polynoms $p_A(x) = \det(x\mathbb{I} - A)$ zu suchen. Ja, obwohl a priori die Nullstellenbestimmung bei Polynomen instabil ist! Man versucht jedoch nicht, die Nullstellen des Polynoms aus den Koeffizienten zu berechnen (das ist instabil), sondern verwendet eine etwas subtilere Variante.

Beginnen wir mit einer irreduziblen symmetrischen Tridiagonalmatrix $A$ (irreduzibel bedeutet, daß kein Nebendiagonalelement verschwindet). Andernfalls könnte man $A$ auf Tridiagonalgestalt transformieren und in Teilmatrizen zerlegen. Mit $A^{(1)}, A^{(2)}, \ldots$ seien die Hauptuntermatrizen von $A$ bezeichnet. Das wesentliche mathematische Resultat ist jetzt, daß die Eigenwerte von $A^{(k)}$ und die Eigenwerte von $A^{(k+1)}$ folgende Gleichung erfüllen:

$$\lambda_j^{(k+1)} < \lambda_j^{(k)} < \lambda_{j+1}^{(k+1)};$$

die Eigenwerte von $A^{(k)}$ liegen also zwischen denen von $A^{(k+1)}$. Diese Eigenschaft ist es, die einem ermöglicht, die Anzahl der Eigenwerte in einem gegebenen Intervall zu zählen.

Es gilt nämlich, daß die Anzahl der negativen Eigenwerte der Tridiagonalmatrix $A \in \mathbb{K}^{n\times n}$ gleich der Anzahl der Vorzeichenwechsel in der Folge

$$1, \det(A^{(1)}), \det(A^{(2)}), \ldots, \det(A^{(n)})$$

ist; diese Folge ist auch unter dem Namen *Sturmsche Kette* bekannt. Tritt 0 in der Folge auf, so definiert man als Vorzeichenwechsel auch einen Übergang von 0 auf $-$ oder $+$ aber nicht einen Übergang von $-$ oder $+$ auf 0. Indem man $A$ um ein Vielfaches der Identität verschiebt, kann man mit dieser Methode auch die Anzahl der Eigenwerte im Intervall $(-\infty, b)$ oder im Intervall $(a, \infty)$ zählen. Schneidet man diese Intervalle, so ist es auch möglich, die Anzahl der Eigenwerte in Intervallen der Form $[a, b)$ zu bestimmen.

Eine weitere wesentliche Beobachtung ist, daß die Determinanten von $A^{(k)}$, $A^{(k-1)}$ und $A^{(k-2)}$ für eine Tridiagonalmatrix durch eine Drei–Terme–Rekurrenz zusammenhängen:

$$\det(A^{(k)}) = A_{kk} \det(A^{(k-1)}) - A^2_{k-1,k} \det(A^{(k-2)}),$$

und damit kann man auch das charakteristische Polynom mit relativ geringem Aufwand berechnen:

$$p^{(k)}(x) = (A_{kk} - x)p^{(k-1)}(x) - A^2_{k-1,k}p^{(k-2)}(x)$$

ist eine Rekurrenz, die die charakteristischen Polynome für die einzelnen Teilmatrizen $A^{(k)}$ zueinander in Beziehung setzt. Als Anfangswerte setzt man $p^{(-1)}(x) = 0$ und $p^{(0)}(x) = 1$. Wertet man sukzessive die Polynome $p^{(k)}(x)$ an verschiedenen Stellen $x$ aus und zählt man Vorzeichenwechsel, so kann man beliebig kleine Intervalleinschließungen für alle Eigenwerte von $A$ berechnen.

**4.3. Divide–and–conquer.** Die modernsten Verfahren zur Eigenwertberechnung sind die sogenannten Divide–and–conquer Algorithmen. Sie stellen die bedeutendste Innovation auf dem Gebiet der Eigenwertberechnung dar seit Erfindung des $QR$–Verfahrens in den frühen Sechzigerjahren. Von Cuppen 1981 eingeführt dauerte es etwa zehn Jahre bis sich der Algorithmus durchsetzte. Obwohl er etwa zweimal so schnell wie das $QR$–Verfahren ist und er sich viel besser parallelisieren läßt, haben Stabilitätsprobleme die Verbreitung des Verfahrens behindert.

Im folgenden wird auf die Probleme, die mit der Stabilisierung des Algorithmus zusammenhängen, nicht näher eingegangen. Es wird im Gegenteil nur die Grundidee präsentiert.

Sei $T$ eine symmetrische irreduzible Tridiagonalmatrix. Dann kann $T$ in zwei Untermatrizen zerlegt werden:

$$T = \begin{array}{|cc|} \hline T_1 & \beta \\ \beta & T_2 \\ \hline \end{array} = \begin{array}{|cc|} \hline \hat{T}_1 & \\ & \hat{T}_2 \\ \hline \end{array} + \begin{array}{|cc|} \hline \beta & \beta \\ \beta & \beta \\ \hline \end{array}.$$

Hier entsteht die Matrix $\hat{T}_1$ durch Subtrahieren von $\beta$ vom rechten unteren Element von $T_1$. Analog entsteht $\hat{T}_2$ durch Subtraktion von $\beta$ vom linken oberen Element von $T_2$. Zusammenfassend kann man also feststellen, daß *sich jede Tridiagonalmatrix schreiben läßt als die Summe einer* $2 \times 2$ *Blockdiagonalmatrix mit tridiagonalen Blöcken und einer Rang–1 Korrektur.*

Der *Divide–and–conquer Algorithmus* funktioniert durch fortgesetzte Unterteilung der Matrix $T$ gemäß dem obigen Verfahren. Oft stoppt man bevor man $2 \times 2$ Matrizen erreicht

hat, deren Eigenwerte man explizit berechnen kann, und bestimmt schon für $10 \times 10$ Untermatrizen die Eigenwertzerlegung mit dem $QR$–Verfahren. Danach berechnet man jeweils aus den Eigenwerten von den Untermatrizen $\hat{T}_1$ und $\hat{T}_2$ und der Rang–1 Korrektur die Eigenwerte von $T$. Dies geschieht folgendermaßen: Angenommen wir haben Diagonalisierungen

$$\hat{T}_1 = Q_1 D_1 Q_1^*, \quad \hat{T}_2 = Q_2 D_2 Q_2^*$$

bereits berechnet. Dann folgt aus der Zerlegung von $T$ die Beziehung

$$T = \begin{pmatrix} Q_1 & \\ & Q_2 \end{pmatrix} \left( \begin{pmatrix} D_1 & \\ & D_2 \end{pmatrix} + \beta z z^\top \right) \begin{pmatrix} Q_1 & \\ & Q_2 \end{pmatrix}^\top$$

mit dem Vektor $z^\top = (q_1^\top, q_2^\top)$, wobei $q_1$ die letzte Spalte von $Q_1$ und $q_2$ die erste Spalte von $Q_2$ bezeichnet.

Wir haben das Problem also darauf reduziert, die Eigenwerte einer Matrix der Gestalt

$$D + ww^\top$$

mit einer Diagonalmatrix $D$ zu berechnen, also einer Diagonalmatrix plus einer Rang–1 Korrektur. Wir können annehmen, daß $w_j \neq 0$ für alle $j$, da sonst das Problem reduzibel wäre. Ist nun $q$ ein Eigenvektor von $D + ww^\top$ zum Eigenwert $\lambda$, so gilt

$$(D + ww^\top)q = \lambda q \implies (D - \lambda \mathbb{I})q + w(w^\top q) = 0.$$

Weil $\lambda$ kein Eigenwert von $D$ sein kann, ist $(D - \lambda \mathbb{I})$ invertierbar und es folgt

$$q + (D - \lambda \mathbb{I})^{-1} w(w^\top q) = 0 \implies w^\top q + w^\top (D - \lambda \mathbb{I})^{-1} w(w^\top q) = 0,$$

was man umschreiben kann als

$$f(\lambda)(w^\top q) = 0$$

mit der Funktion

$$f(x) = 1 + \sum_{j=1}^{n} \frac{w_j^2}{d_j - x}.$$

Nachdem $w^\top q \neq 0$ ist — andererseits wäre $q$ Eigenvektor von $D$, hätte also genau eine nichtverschwindende Komponente $q_j$; dann wäre aber $w_j = 0$, ein Widerspruch — gilt

$$f(\lambda) = 0$$

genau dann, wenn $\lambda$ ein Eigenwert von $D + ww^\top$ ist. Diese Gleichung heißt auch *Sekulargleichung*. Die Form der rationalen Funktion $f(x)$ impliziert, daß sie genau $n$ Nullstellen besitzt, die mit einem Newton–Verfahren (siehe Kapitel 7) schnell bestimmt werden können. Zur Bestimmung jeder einzelnen Nullstelle benötigt man $O(1)$ Aufwand, gesamt also $O(n)$ für alle $n$ Nullstellen. Der Gesamtaufwand für Unterteilung und Rekombination bei Lösung der Sekulargleichung beträgt $O(n^2)$.

Speziell bei der Berechnung von Eigenwerten und Eigenvektoren ist der Vorteil des Divide–and–conquer gegenüber dem $QR$–Verfahren offenbar. Die Reduktion der symmetrischen Matrix auf Tridiagonalgestalt (Phase 1) ist bei beiden Algorithmen gleich $\sim \frac{8}{3}n^3$, doch der Iterationsprozeß verbraucht beim $QR$–Verfahren $\sim 6n^3$ Flops, während er beim Divide–and–conquer Algorithmus $\sim \frac{4}{3}n^3$ elementare Rechenschritte verbraucht, was in Summe einer Verbesserung von $\sim 9n^3$ auf $\sim 4n^3$ Flops entspricht. Schwerer wiegt noch die Möglichkeit zur Parallelisierung, die den Siegeszug dieses Verfahrens in den nächsten Jahren unvermeidlich machen wird.

## 5. Berechnung der Singulärwertzerlegung

Endlich, nach Analyse der Eigenwerte berechnenden Algorithmus kann man auch ein Verfahren zur Bestimmung der Singulärwertzerlegung einer Matrix $A$ konstruieren. Wieder wird das Verfahren in zwei Phasen aufgeteilt. In Phase 1 wird die $n \times m$ Matrix $A$ in eine quadratische Bidiagonalmatrix $B$ der Größe $\min(m,n) \times \min(m,n)$ transformiert. Danach werden die singulären Werte von $B$ in Phase 2 iterativ approximiert.

**5.1. Transformation auf Bidiagonalgestalt.** Es gibt zweieinhalb verschiedene Verfahren, eine rechteckige Matrix $A$ auf *Bidiagonalform* zu bringen, also $A = UBV^*$ zu zerlegen mit

$$B = \begin{pmatrix} * & * & 0 & \cdots & 0 \\ 0 & * & * & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & * & * \\ 0 & \cdots & \cdots & 0 & * \end{pmatrix}.$$

Die einfachere ist die Methode der *Golub–Kahan Bidiagonalisierung*, im folgenden mit GK–Bidiagonalisierung bezeichnet. Mit Hilfe von Householder–Spiegelungen erzeugt man abwechselnd Nullspalten und Nullzeilen wie anhand der untenstehenden Matrix angedeutet:

$$\underset{A}{\begin{pmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{pmatrix}} \xrightarrow{U_1^*\cdot} \underset{U_1^*A}{\begin{pmatrix} * & * & * & * \\ \mathbf{0} & * & * & * \\ \mathbf{0} & * & * & * \\ \mathbf{0} & * & * & * \\ \mathbf{0} & * & * & * \\ \mathbf{0} & * & * & * \end{pmatrix}} \xrightarrow{\cdot V_1} \underset{U_1^*AV_1}{\begin{pmatrix} * & * & \mathbf{0} & \mathbf{0} \\ & * & * & * \\ & * & * & * \\ & * & * & * \\ & * & * & * \\ & * & * & * \end{pmatrix}}$$

$$\xrightarrow{U_2^*\cdot} \underset{U_2^*U_1^*AV_1}{\begin{pmatrix} * & * & & \\ & * & * & * \\ & \mathbf{0} & * & * \\ & \mathbf{0} & * & * \\ & \mathbf{0} & * & * \\ & \mathbf{0} & * & * \end{pmatrix}} \xrightarrow{\cdot V_2} \underset{U_2^*U_1^*AV_1V_2}{\begin{pmatrix} * & * & & \\ & * & * & \mathbf{0} \\ & & * & * \\ & & * & * \\ & & * & * \\ & & * & * \end{pmatrix}}$$

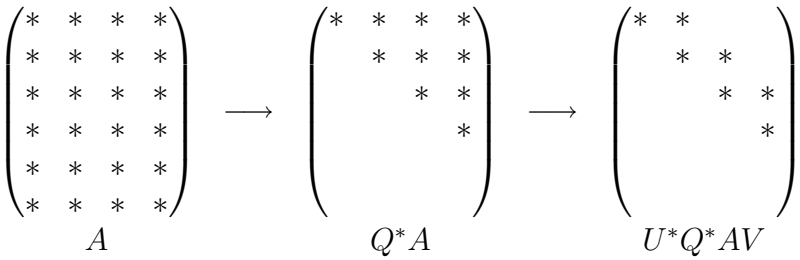
$$\xrightarrow{U_3^*\cdot} \underset{U_3^*\ldots U_1^*AV_1V_2}{\begin{pmatrix} * & * & & \\ & * & * & \\ & & * & * \\ & & \mathbf{0} & * \\ & & \mathbf{0} & * \\ & & \mathbf{0} & * \end{pmatrix}} \xrightarrow{U_4^*\cdot} \underset{U_4^*\ldots U_1^*AV_1V_2}{\begin{pmatrix} * & * & & \\ & * & * & \\ & & * & * \\ & & & * \\ & & & \mathbf{0} \\ & & & \mathbf{0} \end{pmatrix}}$$

Dieses Verfahren entspricht also zwei gleichzeitig durchgeführten Householder $QR$–Faktorisierungen, eine von rechts die andere von links. Der Aufwand für die GK–Bidiagonalisierung beträgt

$$\sim 4nm^2 - \tfrac{4}{3}m^3$$

Flops.

Falls $n \gg m$ ist, so erweist sich dieses Verfahren als zu aufwendig, da dann ein Großteil der zu erzeugenden Nullen unterhalb der Hauptdiagonale sitzt. Das von Lawson, Hanson und Chan vorgeschlagene Verfahren, die LHC–Bidiagonalisierung verfolgt daher die folgende Idee:



$A$ $\qquad$ $Q^*A$ $\qquad$ $U^*Q^*AV$

Man beginnt mit der Berechnung einer $QR$–Zerlegung von $A = QR$. Das erzeugt alle unterhalb der Hauptdiagonale liegenden Nullen in einem Schritt. Für die übrigbleibende $m \times m$ obere Dreiecksmatrix $R$ bestimmt man die Bidiagonalisierung mittels GK–Bidiagonalisierung. Dieser Trick reduziert den Aufwand auf

$$\sim 2nm^2 + 2m^3$$

Flops. Das Verfahren ist billiger als die GK–Bidiagonalisierung, wenn $n > \frac{5}{3}m$.

Das zweieinhalbte Verfahren ist eine leichte Verallgemeinerung des LHC–Verfahrens und wird Drei–Schritt–Bidiagonalisierung genannt. Der Trick ist, die $QR$–Zerlegung nicht zu Beginn durchzuführen sondern irgendwann in der Mitte des Algorithmus. Im GK–Prozeß wird eine Matrix nämlich in jedem Schritt ein klein wenig schmäler. Nach $k$ Schritten hat die noch zu bearbeitende Matrix die Dimensionen $(n-k) \times (m-k)$, und wenn das Verhältnis genügend groß ist, führt man die $QR$–Zerlegung durch und reduziert das Problem wie im LHC–Verfahren auf eine quadratische Dreiecksmatrix. Die maximale Einsparung erzielt man, wenn $(n-k)/(m-k) = 2$ erreicht ist. Diese Wahl liefert die Aufwandsabschätzung

$$\sim 4nm^2 - \tfrac{4}{3}m^3 - \tfrac{2}{3}(n-m)^3,$$

eine moderate Verbesserung gegenüber den anderen beiden Verfahren.

**5.2. Berechnung der Singulärwerte.** Nachdem man das Problem darauf reduziert hat, die Singulärwerte einer quadratischen Bidiagonalmatrix zu errechnen, muß man nur das Problem nur noch mit einem geeigneten Eigenwertproblem in Verbindung bringen:

Sei die Singulärwertzerlegung $A = U\Sigma V^*$ bekannt. Dann gelten die beiden Gleichungen

$$A^*A = V\Sigma^*\Sigma V^* = V\Sigma^2 V^*$$

und

$$\begin{pmatrix} 0 & A^* \\ A & 0 \end{pmatrix} = \begin{pmatrix} V & V \\ U & -U \end{pmatrix} \begin{pmatrix} \Sigma & 0 \\ 0 & -\Sigma \end{pmatrix} \begin{pmatrix} V & V \\ U & -U \end{pmatrix}^*,$$

die jeweils die Singulärwerte und singulären Vektoren in eine Eigenwertzerlegung umschreiben. Es hat jedoch jede der beiden Formulierungen seine Nachteile. Die erste Form, die die Singulärwerte von $A$ mit der *Kovarianzmatrix* $A^*A$ in Verbindung bringt, leidet an der starken Erhöhung der Konditionszahl (es gilt ja $\kappa_2(A^*A) = \kappa_2(A)^2$). Ist die Matrix $A$ schlecht konditioniert, so birgt diese Tatsache eine mögliche Instabilität. Auf der anderen Seite verdoppelt die zweite Formulierung die Dimension der zu behandelnden Matrix und erhöht damit die im $QR$–Verfahren zu leistende Arbeit um einen Faktor 4.

Der gebräuchlichste Algorithmus ist eine leichte Abwandlung des $QR$–Verfahrens, aufbauend auf der Matrix $A^*A$ und der ersten Formulierung. Das Instabilitätsproblem wird dadurch umgangen, daß die Matrix $A^*A$ nie explizit gebildet sondern nur implizit $QR$–zerlegt wird. Die genaue Vorgehensweise, sei im folgenden illustriert:

Sei

$$B = \begin{pmatrix} d_1 & f_1 & 0 & \cdots & 0 \\ 0 & d_2 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & \ddots & f_{n-1} \\ 0 & \cdots & \cdots & 0 & d_n \end{pmatrix}$$

die zu zerlegende Bidiagonalmatrix. Dann wählen wir eine Art Wilkinson–Shift, indem wir die Eigenwerte der rechten unteren $2\times 2$–Teilmatrix $T$ der Tridiagonalmatrix $B^*B$ bestimmen (diese Teilmatrix kann fast ohne zusätzlichen Rechenaufwand bestimmt werden)

$$T = \begin{pmatrix} d_{n-1}^2 + f_{n-2}^2 & d_{n-1}f_{n-1} \\ d_{n-1}f_{n-1} & d_n^2 + f_{n-1}^2 \end{pmatrix}$$

und denjenigen auswählen ($\lambda$), der näher bei $d_n^2 + f_{n-1}^2$ liegt. Bestimmen wir dann eine Givens–Rotation

$$\begin{pmatrix} c_1 & s_1 \\ -s_1 & c_1 \end{pmatrix}^\top \begin{pmatrix} d_1^2 - \lambda \\ d_1 f_1 \end{pmatrix} = \begin{pmatrix} * \\ 0 \end{pmatrix},$$

$c_1 = \cos\theta_1$ und $s_1 = \sin\theta_1$, und setzen wir $G_1 = G(1, 2, \theta_1)$.

Danach wenden wir $G_1$ auf $B$ an. Am folgenden Beispiel mit $n = 6$ seien die weiteren Schritte illustriert:

$$BG_1 = \begin{pmatrix} * & * & & & & \\ \boldsymbol{*} & * & * & & & \\ & & * & * & & \\ & & & * & * & \\ & & & & * & * \\ & & & & & * \end{pmatrix}$$

Der eine Eintrag unterhalb der Hauptdiagonale stört, und daher konstruieren wir sukzessive weitere Givensrotationen $U_1$, $V_2$, $U_2$,..., $V_{n-1}$ und $U_{n-1}$ mit deren Hilfe wir das Element wieder Richtung rechts unten „aus der Matrix schieben“. Die ersten drei Schritte seien im folgenden angegeben:

$$U_1^* BG_1 = \begin{pmatrix} * & * & \boldsymbol{*} & & & \\ & * & * & & & \\ & & * & * & & \\ & & & * & * & \\ & & & & * & * \\ & & & & & * \end{pmatrix}$$

$$U_1^* BG_1V_2 = \begin{pmatrix} * & * & & & & \\ & * & * & & & \\ & \boldsymbol{*} & * & * & & \\ & & & * & * & \\ & & & & * & * \\ & & & & & * \end{pmatrix}$$

$$U_2^* U_1^* BG_1V_2 = \begin{pmatrix} * & * & & & & \\ & * & * & \boldsymbol{*} & & \\ & & * & * & & \\ & & & * & * & \\ & & & & * & * \\ & & & & & * \end{pmatrix}$$

und so weiter. Der Vorgang endet mit einer neuen Bidiagonalmatrix $\bar{B}$:

$$\bar{B} = (U_1 \dots U_{n-1})^* B (G_1 V_2 \dots V_{n-1}) = \bar{U}^* B \bar{V}.$$

Implizit hat man damit einen geshifteten $QR$–Schritt für die symmetrische Tridiagonalmatrix $B^*B$ ausgeführt. Das so konstruierte Verfahren konvergiert mit kubischer Geschwindigkeit gegen eine Diagonalmatrix $B$ und damit gegen eine Singulärwertzerlegung. Jedoch ist es nötig, zwei verschiedene Deflationsschritte zur Verkleinerung des Problems zusätzlich einzubauen. Erstens, wenn sich $B$ zerlegen läßt

$$B = \begin{pmatrix} B_1 & 0 \\ 0 & B_2 \end{pmatrix}$$

kann man das Problem direkt in zwei kleinere aufsplitten, doch auch wenn $d_k = 0$ gilt für ein $k$ kann man das Problem zerteilen. Wieder verwendet man geeignete Givensrotationen, um das entsprechende Nebendiagonalelement $f_k$ auch auf Null zu transformieren. Das Verfahren sei ebenfalls anhand eines $6 \times 6$ Beispieles illustriert.

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

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

$$H_2^* H_1^* B = \begin{pmatrix} * & * & & & & \\ & * & * & & & \\ & & 0 & & & * \\ & & & * & * & \\ & & & & * & * \\ & & & & & * \end{pmatrix}$$

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

und das Problem zerfällt. Setzt man alle diese Algorithmenteile zusammen erhält man das implizite $QR$–Verfahren zur stabilen Berechnung der Singulärwertzerlegung einer Bidiagonalmatrix $B$.

Der Gesamtaufwand zur Bestimmung der Singulärwertzerlegung wird völlig vom Bidiagonalisierungsverfahren dominiert. Jeder Schritt der Iteration benötigt nämlich lediglich einen Aufwand von $O(n)$, was einen Gesamtaufwand von $O(n^2)$ für die gesamte Phase 2 ergibt. Verwendet man Drei–Schritt–Bidiagonalisierung, ist der asymptotische Aufwand zur Bestimmung einer Singulärwertzerlegung

$$\sim 4nm^2 - \tfrac{4}{3}m^3 - \tfrac{2}{3}(n-m)^3.$$

## 6. Software

An Software seien vor allem wieder die Funktionen der LAPACK Sammlung hervorgehoben. Sie enthält Unterprogramme zur Berechnung von Eigenwerten und Eigenvektoren für symmetrische Matrizen (`syev`, `syevd`, `syevx`), tridiagonale Matrizen (`steqr` mit $QR$–Verfahren, `stedc` mit Divide–and–conquer), für unsymmetrische Matrizen (`geev`, `geevx`) und Hessenbergmatrizen (`hseqr`, `hsein`) und viele mehr. Auch Programme zur Bestimmung von Tridiagonalisierungen, Bidiagonalisierungen und Hessenbergtransformationen sind enthalten. Weiters existieren natürlich auch Programme zur Berechnung einer Singulärwertzerlegung (`gesvd`,...).

Die Programmpakete wie MAPLE, Mathematica und MATLAB enthalten einfache Funktionen zur Berechnung der Eigenwerte und Eigenvektoren. Dort sind meist $QR$–Verfahren implementiert.

KAPITEL 10

# Numerische Lineare Algebra IV: Iterationsverfahren

## 1. Grundlagen

In den vergangenen Kapiteln haben wir eine Handvoll Methoden kennengelernt, die verschiedenen Aufgaben der linearen Algebra auf numerischem Wege zu lösen. In Kapitel 9, wo es darum ging, Eigenwerte und Eigenvektoren zu bestimmen, sind zum ersten Mal Verfahren aufgetaucht, die selbst bei rundungsfehlerfreier Rechnung keine exakten Resultate mehr liefern. In allen anderen Fällen sind die Algorithmen so konstruiert, daß sie in endlich vielen Rechenschritten im Prinzip das mathematische Problem exakt lösen können; solche Verfahren nennt man *direkte Verfahren*. Auch Eigenwerte berechnende Algorithmen bestehen im herkömmlichen Fall aus einer direkten Phase (z.B. Hessenbergtransformation) und einer iterativen Phase, die aber nur auf stark vereinfachte Probleme angewendet wird. Eine Gemeinsamkeit aller dieser Algorithmen ist die Größenordnung $O(n^3)$ des Rechenaufwandes.

Für wirklich große Probleme (etwa Computertomographie — siehe Modell 2.5 aus Kapitel 2) ist dieser Aufwand jedoch viel zu hoch. Er ist auch in dem Sinne unverständlich hoch als in jedes Matrixproblem nur $O(n^2)$ (bei dünnbesetzten Matrizen nur $O(n)$) Zahlen eingehen. Warum sollte dann der Aufwand zur Berechnung immer $O(n^3)$ sein?

In den vergangenen vier Jahrzehnten ist die maximale Dimension der gerade noch lösbaren Matrixberechnungen etwa alle fünfzehn Jahre um eine Zehnerpotenz gestiegen. Von $n = 20$ in den Fünfzigerjahren auf etwa $n = 20000$ Mitte der Neunzigerjahre. Im gleichen Zeitraum ist die maximale Leistungsfähigkeit der Computersysteme hingegen um einen Faktor $10^9$ gestiegen. Man kann also den $O(n^3)$ Flaschenhals deutlich erkennen. Trotz einer gewaltigen Verbesserung der Hardware ist die maximale Dimension der Matrixberechnungen „nur“ um einen Faktor $10^3$ gestiegen.

Nun, ganz korrekt sind die oben stehenden Aussagen nicht. Es existieren Algorithmen für die meisten Probleme der linearen Algebra, die einen geringeren Aufwand als $O(n^3)$ haben. Für die Lösung eines linearen Gleichungssystems haben Coppersmith und Winograd 1986 ein Verfahren gefunden, das Ordnung $O(n^{2.376})$ hat. Allerdings ist der Koeffizient des höchsten Terms $n^{2.376}$ so groß, daß der Gesamtaufwand für heutzutage berechenbare Probleme trotz des niedrigen Exponenten höher ist als bei der $LR$–Zerlegung mit Spaltenpivotsuche.

Die einzige Idee, die noch hilft, große Systeme mit Dimensionen jenseits von $10^6$ zu lösen, ist Lösungen nicht mehr direkt zu bestimmen sondern approximativ durch Verwendung eines Iterationsverfahrens. Im Gegensatz zu den Algorithmen aus Kapitel 9 versucht man jedoch, den Aufwand eines einzelnen Schrittes so stark wie möglich zu reduzieren. Mehr als $O(n)$ Rechenoperationen kann man für einen einzelnen Iterationsschritt in einem Algorithmus für Probleme riesiger Dimension nicht verwenden. Gelingt es dann auch noch, das Verfahren so zu gestalten, daß lediglich $O(1)$ Iterationsschritte nötig sind, so hätte man das ideale Verfahren gefunden, doch meist ist die höchste erreichbare Verbesserung von $O(n^3)$ auf $O(n^2)$. Heutzutage Ende der Neunzigerjahre übertreffen Iterationsverfahren wegen der hohen Konstanten vor den führenden Termen $n^2$ die direkten Algorithmen in praktischen höchst dimensionalen Anwendungsproblemen um einen Faktor zehn. Doch durch die steigende Leistungsfähigkeit der Computersysteme, die in der Zukunft die Lösung noch größerer Probleme ermöglichen

werden, wird der Siegeszug der Iterationsverfahren nicht mehr aufzuhalten sein, da spätestens dann der kleinere Exponent zu wachsenden Geschwindigkeitsunterschieden führen wird.

Iterationsverfahren können $O(n^3)$ jedoch nicht immer übertreffen. Für Zufallsmatrizen ist das sehr unwahrscheinlich. Der Weg zur Geschwindigkeitssteigerung führt über die Ausnützung der Matrixstruktur, die in den meisten Anwendungsproblemen alles andere als zufällig ist. Hochdimensionale Probleme führen meist auf dünnbesetzte Matrizen (oft gar auf Matrizen mit Bandstruktur), die in jeder Zeile nur $\nu$ Einträge haben, die von Null verschieden sind. Für solche Matrizen kann man Matrix–Vektor Produkte anstatt in $O(n^2)$ Rechenschritten mit $O(\nu n)$ elementaren Operationen berechnen, und Matrix–Vektor Produkte sind die einzigen komplexen Rechenoperationen, die in einen Iterationsschritt eingehen.

**1.1. Krylov–Räume.** Nachdem wir schon festgestellt haben, daß wir uns beim Entwurf der Iterationsalgorithmen auf Matrix–Vektor Produkte einschränken wollen, ist zu erwarten, daß Räume, die von Vektoren $b$ und deren iterierte Produkte mit einer Matrix $A$ aufgespannt werden, zentrale Bedeutung erlangen werden:

**Definition 1.1.1.** *Gegeben eine Matrix $A \in \mathbb{K}^{n\times n}$ und einen Vektor $b \in \mathbb{K}^n$, dann ist die dazu assoziierte* Krylovfolge *gegeben durch die iterierten Produkte von $A$ mit $b$:*

$$b, Ab, A^2b, A^3b, \dots$$

*Der von den ersten $n$ Vektoren der Krylov–Folge aufgespannte Vektorraum $\mathcal{K}_n$ heißt der $n$–te* Krylov–Raum *zu $A$ und $b$:*

$$\mathcal{K}_n := \langle b, Ab, A^2b, \dots, A^{n-1}b\rangle.$$

*Die Matrix $K_n$, die als Spalten die ersten $n$ Glieder der Krylov–Folge enthält, heißt die $n$–te Krylov–Matrix zu $A$ und $b$:*

$$K_n := \left(b\middle|Ab\middle|A^2b\middle|\cdots\middle|A^{n-1}b\right).$$

Die meisten Iterationsalgorithmen funktionieren, indem sie das zu bearbeitende Problem in Krylovräume wachsender Ordnung projizieren und die Lösungen der niedriger dimensionalen Problems zur Approximation der Lösung der hochdimensionalen Aufgabe verwendet. Dabei werden implizit $QR$–Zerlegungen und Hessenbergreduktionen von Krylov–Matrizen bestimmt.

Die im folgenden vorgestellten Verfahren können entsprechend der folgenden Tabelle angeordnet werden. Dabei steht die Abkürzung CG für das *konjugierte Gradientenverfahren* (englisch: conjugate gradients) und die Abkürzung GMRES für Verallgemeinerte–Minimale–Residuen (englisch: generalized minimal residuals)

| | $Ax = \lambda x$ | $Ax = b$ |
|---|---|---|
| $A \neq A^*$ | Arnoldi Iteration | GMRES Verfahren |
| $A = A^*$ | Lanczos Iteration | CG Verfahren |

## 2. Arnoldi Iteration

Das zentrale Verfahren, auf dem die meisten anderen Iterationsverfahren aufbauen ist die Arnoldi Iteration. In Kapitel 4 haben wir zwei grundverschiedene Methoden kennengelernt, eine $QR$–Faktorisierung einer Matrix zu berechnen, das Gram–Schmidt Verfahren und das Householder Verfahren. Dort haben wir aus Stabilitätsgründen die Householder–Reflexionen dem Gram–Schmidt Verfahren vorgezogen. Doch das Gram–Schmidt Verfahren hat einen wichtigen Vorteil. Man kann es nach $n$ Schritten unterbrechen und erhält eine unvollständige $QR$–Zerlegung, eine reduzierte $QR$–Faktorisierung der ersten $n$ Spalten von $A$.

Zur Berechnung einer Hessenbergreduktion existieren ebenfalls zwei Standardverfahren — Householder Reflexionen (Kapitel 9) und die Arnoldi Iteration, die also ein Analogon des Gram–Schmidt Verfahrens für Hessenbergreduktion ist.

**2.1. Das Iterationsverfahren.** Nehmen wir an, wir hätten bereits eine Hessenbergreduktion von $A$ bestimmt:

$$A = QHQ^*$$

mit untärer Matrix $Q$ und oberer Hessenbergmatrix $H$, oder umgeformt $AQ = QH$. Sei $Q_m$ die $m \times n$ Teilmatrix von $Q$, die aus den ersten $m$ Spalten von $Q$ besteht:

$$Q_m = \left( q_1 \middle| q_2 \middle| \cdots \middle| q_m \right).$$

Mit $\hat{H}_m$ sei die $(m+1) \times m$ linke obere Teilmatrix von $H$ bezeichnet, selbst wieder eine obere Hessenbergmatrix:

$$\hat{H}_m = \begin{pmatrix} h_{11} & & \cdots & h_{1m} \\ h_{21} & h_{22} & & \vdots \\ & \ddots & \ddots & \vdots \\ & & h_{m,m-1} & h_{mm} \\ & & & h_{m+1,m} \end{pmatrix}.$$

Mit diesen Bezeichnungen gilt die Gleichung

$$AQ_m = Q_{m+1}\hat{H}_m.$$

Schreibt man diese Gleichung spaltenweise auf und extrahiert daraus die Beziehung für die letzte, die $m$–te, Spalte, so erhält man

$$Aq_m = h_{1m}q_1 + \cdots + h_{mm}q_m + h_{m+1,m}q_{m+1}$$

oder umgeformt

$$h_{m+1,m}q_{m+1} = Aq_m - h_{mm}q_m - \cdots - h_{1m}q_1. \tag{77}$$

Betrachtet man diese Gleichung in den Spezialfällen $m = 1$ und $m = 2$:

$$\begin{aligned} h_{21}q_2 &= Aq_1 - h_{11}q_1 \\ h_{32}q_3 &= Aq_2 - h_{22}q_2 - h_{12}q_1 = \\ &= \tfrac{1}{h_{21}}A^2q_1 - \tfrac{h_{11}+h_{22}}{h_{21}}Aq_1 + \big(\tfrac{h_{11}h_{22}}{h_{21}} - h_{12}\big)q_1, \end{aligned}$$

so erkennt man, daß für alle $i$

$$q_i \in \mathcal{K}_i,$$

dem Krylov–Raum zu $A$ und $q_1$, gilt. Der $i$–te Vektor erfüllt nach Gleichung (77) eine $i+1$–Terme Rekurrenzrelation, die nur sich und die anderen Krylov–Vektoren enthält. Mit Hilfe dieser Gleichung formuliert man den Algorithmus der Arnoldi Iteration

**Algorithmus 2.1.1.** *Arnoldi Iteration*

*b beliebig*
$q_1 = b/\|b\|$
**for** $m = 1$ **to** ? **do**
  $v = Aq_m$
  **for** $j = 1$ **to** $m$ **do**
    $h_{jm} = q_j^* v$
    $v = v - h_{jm}q_j$
  **done**
  $h_{m+1,m} = \|v\|$

**if** $h_{m+1,m} = 0$ **then**
**stop**
**endif**
$q_{m+1} = v/h_{m+1,m}$
**done**

Das Arnoldi Verfahren erzeugt Vektoren $q_j$, die orthonormal sind und die Krylov–Räume $\mathcal{K}_k$ aufspannen. Das ist äquivalent dazu, daß es eine $QR$–Zerlegung der zugehörigen Krylov–Matrix $K_n$

$$K_n = Q_n R_n$$

berechnet. Der Vorteil ist, daß weder $K_n$ noch $R_n$ explizit berechnet werden, denn beide Matrizen sind ausgesprochen schlecht konditioniert, weil alle Spalten von $K_n$ dazu tendieren, denselben dominanten Eigenvektor von $A$ zu approximieren (siehe Abschnitt 3.2 in Kapitel 9, die Potenziteration). In Analogie zur Blockpotenziteration kann man annehmen, daß die Matrix $Q_m$ die ersten $m$ dominanten Eigenwerte von $A$ approximiert.

Führen wir eine weitere Notation ein: $H_m$ sei die Matrix $\hat{H}_m$ ohne die letzte Zeile. Dann gilt die Beziehung $H_m = Q_m^* Q_{m+1} \hat{H}_m$, weil die Matrizen $Q_m$ und $Q_{m+1}$ die ersten $m$ bzw. $m+1$ Spalten der unitären Matrix $Q$ enthalten. Aus der Zerlegung $A = QHQ^*$ enthalten wir folglich die Beziehung

$$H_m = Q_m^* A Q_m.$$

Diese Matrix repräsentiert die orthogonale Projektion von $A$ auf den Krylov–Raum $\mathcal{K}_m$ dargestellt in der Basis $\{q_1, \ldots, q_m\}$. (Bezüglich der Standardbasis des $\mathbb{K}^m$ ist der orthogonale Projektor gegeben durch die Formel $Q_m Q_m^* A$.) Diese Abbildung nimmt einen Vektor $v \in \mathcal{K}_m$, wendet $A$ darauf an und projiziert das Ergebnis $Av$ wieder auf $\mathcal{K}_m$ zurück. Diese Projektion trägt den Namen *Rayleigh–Ritz Verfahren*, nicht zuletzt, weil die Einträge von $H_m$ gerade die Rayleigh Quotienten bezüglich $A$ der Vektoren $q_j$ sind.

Weil $H_m$ eine Projektion der Matrix $A$ ist, kann man annehmen, daß die Eigenwerte von $H_m$ einen starken Zusammenhang mit den Eigenwerten von $A$ besitzen. Die $m$ Eigenwerte von $H_m$, im folgenden mit $\theta_j^{(m)}$ bezeichnet, werden auch *Arnoldi Eigenwertschätzer* oder *Ritz–Zahlen* genannt.

**Theorem 2.1.2.** *Die Matrizen $Q_m$, die die Arnoldi Iteration erzeugt, sind die unitären Faktoren von reduzierten $QR$–Zerlegungen der Krylov–Matrizen*

$$K_m = Q_m R_m.$$

*Die Hessenbergmatrizen $H_m$ sind die zugehörigen Projektionen*

$$H_m = Q_m^* A Q_m$$

*auf die Krylov–Räume $\mathcal{K}_m$, und die Iterationsschritte hängen über die Gleichung*

$$A Q_m = Q_{m+1} \hat{H}_m$$

*zusammen.*

**2.2. Polynomiale Approximation.** Haben wir einen Vektor $x \in \mathcal{K}_m$ gegeben, so kann man ihn in die ursprüngliche Basis $\{b, Ab, \ldots, A^{m-1}b\}$ entwickeln:

$$x = a_0 b + a_1 Ab + \cdots + a_{m-1} A^{m-1} b = q(A)b,$$

wobei $q$ das Polynom

$$q(z) = a_0 + a_1 z + \cdots + a_{m-1} z^{m-1}$$

bezeichnet. Aus diesem Grund kann man Krylov–Räume immer mit Hilfe von Matrixpolynomen analysieren. Sei für die folgende Analyse $\mathbb{K}^k_{\text{mon}}[z]$ die Bezeichnung der Menge der

monischen Polynome höchstens $k$–ten Grades in einer Variablen. Dann löst das Arnoldi Verfahren das folgende Approximationsproblem:

Finde das Polynom $p^m \in \mathbb{K}^m_{\text{mon}}[z]$, das die Norm

$$\|p^n(A)b\|_2$$

minimiert.

Dieses Approximationsproblem hat genau eine Lösung, sofern $\dim \mathcal{K}_m = m$ ist, nämlich das charakteristische Polynom von $H_m$. Die Nullstellen dieses charakteristischen Polynoms sind genau die Ritz–Zahlen $\theta_j^{(m)}$, welche über die Arnoldi Iteration bestimmt werden können.

Die Polynominterpretation ermöglicht es auch, die Invarianzeigenschaften der Arnoldi Iteration zu bestimmen:

**Theorem 2.2.1.** *Wird die Arnoldi Iteration angewendet auf die Matrix $A \in \mathbb{K}^{n\times n}$ dann gilt*

**Translationsinvarianz:** *Wird $A$ um $\sigma\mathbb{I}$ verschoben ($A \to A + \sigma\mathbb{I}$), so ändern sich die Ritz–Zahlen zu $\theta_j^{(m)} + \sigma$.*

**Skalierungsinvarianz:** *Wird $A$ durch $\sigma A$ ersetzt, so werden auch die Ritz–Zahlen skaliert: $\theta_j^{(m)} \to \sigma\theta_j^{(m)}$.*

**Unitäre Invarianz:** *Ersetzt man $A$ durch eine unitär ähnliche Matrix $UAU^*$, und verändert man $b$ zu $Ub$, so bleiben die Ritz–Zahlen gleich.*

*In allen drei Fällen verändern sich die Ritz–Vektoren $Q_m y_j$, die den Eigenvektoren $y_j$ von $H_m$ entsprechen, nicht.*

**2.3. Arnoldi Iteration und Eigenwerte.** Die letzten beiden Resultate zeigen auch schon auf, wie die Arnoldi Iteration Eigenwerte von $A$ auffindet. Betrachten wir dazu eine Matrix, die nur $m \ll n$ verschiedene Eigenwerte besitzt. Dann besitzt $A$ ein Minimalpolynom $p$ vom Grad $m$, dessen Nullstellen genau die Eigenwerte von $A$ sind, und es gilt $p(A) = 0$. Dieses Polynom ist also dann jenes Polynom $p^m$, das von der Arnoldi Iteration berechnet wird (jedenfalls, wenn der Startvektor $b$ Komponenten in Richtung jedes Eigenraumes von $A$ besitzt).

Im generischen Fall versucht man dieses Minimalpolynom zu approximieren, weil es zu hohen Grad hat. Deswegen wählt man das Polynom $m$–ten Grades, das $\|p^m(A)\|_2$ minimiert und erhält anstelle eines Minimalpolynoms ein „Pseudo–Minimalpolynom“. Die Ritz–Werte, die Nullstellen dieses Pseuso–Minimalpolynoms, sind demnach gute Approximationen für die Nullstellen des Minimalpolynoms, die Eigenwerte von $A$.

Mit zunehmendem Polynomgrad approximieren die Nullstellen von $p^m$ die Eigenwerte von $A$ immer besser, und dieser Fortschritt kann graphisch mit Hilfe der *Arnoldi–Lemniskaten* dargestellt werden. Eine Lemniskate ist eine Familie von Kurven

$$\{z \in \mathbb{C} : |q(z)| = C\},$$

wo $q$ ein Polynom ist und $C \in \mathbb{R}$ eine reelle Zahl. Wählt man speziell $q = p^m$ und

$$C = \frac{\|p^m(A)b\|}{\|b\|},$$

so erhält man die $m$–te Arnoldi–Lemniskate. Diese Arnoldi–Lemniskaten approximieren das Spektrum von $A$ dahingehend, daß sie mit steigender Iterationszahl die geometrische Form des Spektrums immer besser approximieren. Abbildung 10.1 stellt diese Lemniskaten für verschiedene Iterationsschritte dar. In der Abbildung erkennt man auch, daß die Arnoldi–Lemniskaten zuerst kleine Teile abspalten, die die extremen Eigenwerte einhüllen und sich mit geometrischer Geschwindigkeit zusammenziehen. Die „großen Wolken“ werden immer

ABBILDUNG 10.1. Arnoldi–Lemniskaten zu den Iterationsschritten $m = 1, 2, 4, 8, 12, 16$ für eine $100 \times 100$ Matrix $A$. Die kleinen Punkte sind die Eigenwerte von $A$, währen die großen Punkte die Ritz–Zahlen repräsentieren.

besser eingeschlossen je höher der Polynomgrad wird, und die Ritz–Zahlen verteilen sich immer mehr innerhalb dieser Wolken.

Die Konvergenz der Arnoldi Iteration genauer zu analysieren ist eine äußerst schwierige Aufgabe, und sie wird bis heute noch nicht vollständig verstanden.

## 3. Das GMRES Verfahren

Das GMRES Verfahren ist eine Abwandlung der Arnoldi Iteration, die dazu verwendet wird, lineare Gleichungssysteme $Ax = b$ zu lösen. Die Idee ist, den Lösungsvektor $x_* := A^{-1}b$ durch denjenigen Vektor $x_m \in \mathcal{K}_m$ zu approximieren, der die Norm des Residuums $r_m = b - Ax_m$ minimiert. Man löst also in anderen Worten ein Kleinste–Quadrate Problem, um $x_*$ zu bestimmen.

Die Formulierung des Kleinste–Quadrate Problems ist einfach. Sei $K_m$ die $m$–te Krylov–Matrix zu $A$ und $b$. Dann ist unser Problem, denjenigen Vektor $v$ zu finden, der

$$\|AK_m v - b\|_2$$

minimiert. Ist dieser Vektor erst gefunden, erhält man $x_n = K_n v$. Das Kleinste–Quadrate Problem könnte man z.B. dadurch lösen, daß man eine $QR$–Zerlegung der Matrix $AK_m$ berechnet.

Dieses Verfahren wäre jedoch numerisch instabil (aus den schon bekannten Gründen: die katastrophale Kondition von $K_m$). Stattdessen berechnet man mit Hilfe der Arnoldi Iteration eine Folge von Matrizen $Q_m$, die Orthonormalbasen für die aufeinander folgenden Krylov–Räume $\mathcal{K}_m$ bilden. Wir schreiben also $x_m = Q_m y$ statt $x_m = K_m c$. Daher lautet unser neues Kleinste–Quadrate Problem: Finde einen Vektor $w \in \mathbb{K}^m$ mit

$$\|AQ_m w - b\|_2 \text{ ist minimal.}$$

Mit dieser Formulierung und Gleichung (77) kann man die Dimension des Problems reduzieren und das Kleinste–Quadrate Problem umformulieren zu

$$\|Q_{m+1}\hat{H}_m w - b\|_2$$

ist ein Minimum. Nun sind beide Vektoren innerhalb der Norm im Spaltenraum von $Q_{m+1}$, und man kann ohne die Norm zu verändern mit $Q_{m+1}^*$ von links multiplizieren. Verwendet

man überdies noch die Gleichung $Q_{m+1}^* b = \|b\|_2 e_1$, die aus der Konstruktion der $Q_i$ folgt, so erhält man schließlich die GMRES Formulierung des Kleinste–Quadrate Problems

$$\|\hat{H}_m w - \|b\|_2 e_1\|_2 = \min .$$

Im $n$–ten Schritt des GMRES Verfahrens lösen wir dieses Problem nach $w$ und setzen $x_m = Q_m w$. Dies läßt sich als Algorithmus formulieren:

**Algorithmus 3.0.1.** *GMRES Verfahren*

$q_1 = b/\|b\|_2$
**for** $m = 1$ **to** ? **do**
*Führe Schritt n der Arnoldi Iteration (Alg. 2.1.1) aus*
*Suche w, das* $\|\hat{H}_m w - \|b\|_2 e_1\|_2$ *minimiert*
$x_m = Q_m w$
**done**

Der Schritt „Suche $w$,. . . “ ist ein $(m+1) \times m$ Kleinste–Quadrate Problem, das mit Hilfe eines Update Prozesses gelöst werden kann. Hat man die $QR$–Zerlegung der Matrix $\hat{H}_m$ berechnet, so kann man mit Hilfe einer einzelnen Givens–Rotation eine $QR$–Faktorisierung von $\hat{H}_{m+1}$ berechnen, und somit reduziert sich der Aufwand zur Lösung dieses Kleinste–Quadrate–Problems auf $O(m)$.

GMRES hängt ebenso wie das Arnoldi Verfahren mit Polynomapproximation zusammen. Man sucht wieder ein Polynom $p_m$, das

$$\|p_m(A)b\|_2$$

minimiert. Diesmal wählt man das Polynom jedoch nicht im Raum $\mathbb{K}^m_{\text{mon}}[z]$ sondern in $\mathbb{K}^m_1[z]$, der Menge der Polynome $q$ höchstens $m$–ten Grades, die $q(0) = 1$ erfüllen. GMRES löst genau dieses Optimierungsproblem. Wie die Arnoldi Iteration hat auch das GMRES Verfahren diverse Invarianzeigenschaften:

**Theorem 3.0.2.** *Sei das GMRES Verfahren auf die Matrix A angewendet. Dann gilt*

**Skalierungsinvarianz:** *Wenn man A durch $\sigma A$ ersetzt, für eine komplexe Zahl $\sigma$, und wenn gleichzeitig b gegen $\sigma b$ ausgetauscht wird, so verändern sich die Residuen $r_m$ zu $\sigma r_m$.*

**Unitäre Invarianz:** *Der Übergang $A \to UAU^*$ und $b \to Ub$ für eine unitäre Matrix U führt zu einer Veränderung der Residuen von $r_m$ auf $U^* r_m$.*

Das GMRES Verfahren ist nicht translationsinvariant wie die Arnoldi Iteration, da die Menge der Polynome mit $q(0) = 1$ im Gegensatz zu den monischen Polynomen nicht translationsinvariant ist.

Die Konvergenzgeschwindigkeit des GMRES Algorithmus ist (wie bei fast allen modernen Iterationsverfahren) eine schwierig zu untersuchende Angelegenheit. Lediglich zwei Tatsachen sind offensichtlich.

- GMRES konvergiert monoton, d.h.

  $$\|r_{k+1}\|_2 \leq \|r_k\|_2.$$

  Das folgt, weil offensichtlich $\mathcal{K}_k \subset \mathcal{K}_{k+1}$.
- Nach höchstens $n$ Schritten gilt $\|r_n = 0\|$, das Problem ist also gelöst; wenigstens bei rundungsfehlerfreier Rechnung.

Diese beiden Aussagen sind im numerischen Zusammenhang leider völlig unbrauchbar. Damit der GMRES Algorithmus brauchbar ist, muß Konvergenz (oder beinahe Konvergenz) nach $m \ll n$ Schritten eintreten.

Was wir noch wissen, ist, daß

$$\|r_m\|_2 = \|p_m(A)b\|_2 \leq \|p_m(A)\|_2\|b\|_2$$

minimal ist. Für generische $A$ und $b$ bestimmt demnach die Zahl

$$\frac{\|r_m\|_2}{\|b\|_2} \leq \inf_{p_m \in \mathbb{K}_1^m[z]} \|p_m(A)\|_2$$

die Konvergenzrate von GMRES. Das führt auf die mathematisch sehr anspruchsvolle Frage, wie klein $\|p_m(A)\|_2$ werden kann wenn man eine Matrix $A$ und eine natürliche Zahl $m$ gegeben hat. Für diagonalisierbare Matrizen kann man diese Zahl abschätzen:

**Theorem 3.0.3.** *Sei $A = V\Lambda V^{-1}$ eine Eigenwertzerlegung von $A$. Dann gilt nach dem $m$–ten Schritt der GMRES Iteration für das Residuum $r_m$ die Abschätzung*

$$\frac{\|r_m\|_2}{\|b\|_2} \leq \inf_{p_m \in \mathbb{K}_1^m[z]} \|p_m(A)\|_2 \leq \kappa_2(V) \inf_{p_m \in \mathbb{K}_1^m[z]} \|p_m\|_{\Lambda(A)},$$

*wobei wir für eine beliebige Menge $M$*

$$\|p\|_M := \sup_{z \in M} |p(z)|$$

*setzen und $\Lambda(A)$ das Spektrum von $A$ (die Menge der Eigenwerte von $A$) bezeichnet.*

Ist $A$ also nicht zu weit davon entfernt normal zu sein, ist also $\kappa_2(V) \not\gg 1$, und können Polnome gefunden werden, deren Größe auf dem Spektrum von $A$ rasch mit wachsendem Polynomrad fällt, so konvergiert das GMRES Verfahren schnell. Ansonsten kann die Konvergenz kriechend sein. Durch geeignete Vorkonditionierung (siehe Abschnitt 7) kann die Matrix meist so verändert werden, daß eine hohe Konvergenzgeschwindigkeit erzielt werden kann.

## 4. Lanczos Iteration

Die Lanczos Iteration ist die Spezialisierung der Arnoldi Iteration auf hermitesche Matrizen $A$. In diesem Fall sind die Matrizen $H_m$ symmetrisch und tridiagonal. Daher sind ihre Eigenwerte, die Ritz–Zahlen oder Lanczos Schätzungen, reell. Die $(n+1)$–Terme Rekurrenz (77) wird zu einer Drei–Terme Rekurrenz, und mit der Notation

$$H_m = \begin{pmatrix} a_1 & b_1 & & & \\ b_1 & a_2 & b_2 & & \\ & b_2 & a_3 & \ddots & \\ & & \ddots & \ddots & b_{m-1} \\ & & & b_{m-1} & a_m \end{pmatrix}$$

kann man Algorithmus 2.1.1 umformulieren:

**Algorithmus 4.0.4.** *Lanczos Iteration*

*$b$ beliebig*
$q_1 = b/\|b\|_2$
$b_0 = 0$
$q_0 = 0$
**for** $m = 1$ **to** ? **do**
  $v = Aq_m$ ; $Aq_m - b_{m-1}q_{m-1}$ `für größere Stabilität`
  $a_m = q_m^\top v$
  $v = v - b_{m-1}q_{m-1} - a_m q_m$
  $b_m = \|v\|_2$
  **if** $b_m = 0$ **then**

**stop**
**endif**
$q_{m+1} = v/b_m$
**done**

Dieser Algorithmus benötigt in jedem Iterationsschritt eine Matrix–Vektor Multiplikation, ein inneres Produkt und einige wenige Vektoroperationen. Für dünnbesetzte Matrizen bedeutet das Aufwand $O(n)$ mit einer sehr kleinen Konstante für jeden Schritt. Der nächste Satz faßt einige Eigenschaften der Lanczos Iteration zusammen:

**Theorem 4.0.5.** *Die Matrizen $Q_m$ gebildet aus den berechneten Vektoren $q_j$ sind reduzierte QR–Faktoren der Krylov–Matrizen $K_m$:*

$$K_m = Q_m R_m.$$

*Die tridiagonalen Matrizen $H_m$*

$$H_m = Q_m^* A Q_m$$

*sind die entsprechenden Projektionen auf die Krylov–Räume, und aufeinanderfolgende Ergebnisse der Iteration hängen folgendermaßen zusammen:*

$$AQ_m = Q_{m+1}\hat{H}_m.$$

*Diese Gleichung kann man auch in Form einer Drei–Terme Rekurrenz umschreiben:*

$$Aq_m = b_{m-1}q_{m-1} + a_m q_m + b_m q_{m+1}.$$

*Solange die Lanczos Iteration nicht abbricht, also der Krylov–Raum $\mathcal{K}_m$ Dimension $m$ besitzt, ist das charakteristische Polynom von $H_m$ das eindeutige Polynom $p^m \in \mathbb{K}^m_{\mathrm{mon}}[z]$, das das Approximationsproblem*

$$\|p^m(A)b\| = \min$$

*erfüllt.*

Üblicherweise findet auch die Lanczos Iteration die extremen Eigenwerte der Matrix $A$ zuerst. Üblicherweise konvergieren einige Lanczos–Schätzwerte linear gegen diese extremen Eigenwerte. Genauer tendiert die Lanczos Iteration dazu, gegen Eigenwerte in den Regionen zu konvergieren, die zu „dünn mit Eigenwerten besetzt sind“. Gibt es Regionen $[a, b]$, in denen die Eigenwerte „beinahe gleichverteilt“ sind, so konvergiert die Lanczos Iteration gegen Tschebyscheff Punkte über diesem Intervall — Die $m$ Tschebyscheff Punkte im Intervall $[-1, 1]$ sind definiert als

$$x_j = \cos\left(\frac{(j-\frac{1}{2})\pi}{m}\right), \quad 1 \le j \le m.$$

Rundungsfehler haben einen sehr großen Einfluß auf die Lanczos Iteration, und es ist wichtig, diese im Auge zu behalten. Der Grund dafür ist, daß im Gegensatz zur Arnoldi Iteration die Vektoren $q_1, q_2, \ldots$ durch die $(m+1)$–Terme Rekurrenz explizit dazu gezwungen werden, aufeinander orthogonal zu stehen. Das Ausnützen der Symmetrie in der Lanczos Iteration bedingt, daß eine mathematische Identität Grund für die Orthogonalität wird, diese quasi aus dem Nichts durch die Daten entsteht. Dieser Prozeß wird durch die Rundungsfehler mitunter empfindlich gestört.

Daß die Lanczos Iteration in der Praxis trotzdem außerordentlich wertvoll ist, ist ebenfalls ein Phänomen, das noch nicht vollständig analysiert ist. Was üblicherweise passiert, ist das Auftreten von „Geistereigenwerten“. Besonders exponierte Eigenwerte werden im Verlauf des Verfahrens bei späteren Iterationsschritten plötzlich mehrfach getroffen, wobei die Vielfachheiten, die der Algorithmus produziert, mit den aktuellen Vielfachheiten der Eigenwerte nichts zu tun haben. Eine vernünftig klingende Erklärung für das Auftreten dieser Geistereigenwerte ist die folgende: Konvergenz eines Ritz–Wertes gegen einen Eigenwert annihiliert

die entsprechende Eigenvektorkomponente im Vektor $b$. Durch Rundungsfehler wird diese Komponente jedoch nicht vollständig entfernt; es bleibt ein winzig kleiner Rest übrig. Dieser wird jedoch in den folgenden Iterationsschritten wieder so weit verstärkt, daß im weiteren Verlauf wieder eine neue Ritz–Zahl konstruiert werden muß, die diese Pseudokomponente erneut annihiliert wird, usw.

Ein Detail am Rande: Die Lanczos Iteration hängt stark mit orthogonalen Polynomen zusammen, und die Drei–Terme Beziehung der Lanczos Iteration entspricht der Drei–Terme Beziehung bei der Konstruktion von orthogonalen Polynomen. Genauere Zusammenhänge kann man etwa in [**Trefethen, Bau 1997**, 37] nachlesen.

## 5. Konjugierte Gradienten

Obwohl in der heutigen Zeit das Verfahren der konjugierten Gradienten als Spezialfall einer Anwendung der Arnoldi Iteration auftritt, war dieses Verfahren 1952 der eigentliche Beginn der Krylov–Raum Iterationsverfahren. Hestenes und Stiefel führten es ein, um symmetrische positiv definite lineare Gleichungssysteme schnell zu lösen, wenn die Eigenwerte gut verteilt sind.

Die Aufgabe, ein Gleichungssystem $Ax = b$ zu lösen für eine hermitesche Matrix $A$ könnte man mit dem üblichen GMRES Algorithmus lösen, doch dieser benötigt mehr Aufwand als nötig. Durch die Symmetrie kann man wieder die $(m + 1)$–Terme Rekurrenz durch eine Drei–Terme Rekurrenz ersetzen. Diese Verfahren sind auch unter den Namen *Konjugierte Residuen* oder *MINRES* (englisch: minimal residuals) bekannt.

Hier sei im folgenden der einfachste Fall beschrieben — die Lösung eines linearen Gleichungssystemes mit symmetrischer positiv definiter Koeffizientenmatrix $A$.

Das Verfahren der *konjugierten Gradienten*, im folgenden CG genannt, ist ein System von Rekurrenzgleichungen, das die eindeutige Folge von Punkten $x_m \in \mathcal{K}_m$ generiert, das die Eigenschaft hat, in jedem Schritt die $A$–Norm des Fehlers

$$\|e_n\|_A := \|x_* - x_n\|_A$$

zu minimieren ($x_* = A^{-1}b$ sei wieder die exakte Lösung). Dabei ist die $A$–Norm eines Vektors $y$ gegeben durch

$$\|y\|_A = \sqrt{y^\top Ay}.$$

Der Originalalgorithmus von Hestenes und Stiefel läuft folgendermaßen ab:

**Algorithmus 5.0.6.** *Konjugierte Gradienten Iteration*

$x_0 = 0$
$r_0 = b$
$p_0 = r_0$
**for** $m = 1$ **to** ? **do**
  $\alpha_m = (r_{m-1}^\top r_{m-1})/(p_{m-1}^\top A p_{m-1})$ `; Schrittweite`
  $x_m = x_{m-1} + \alpha_m p_{m-1}$ `; Näherungslösung`
  $r_m = r_{m-1} - \alpha_m A p_{m-1}$ `; Residuum`
  $\beta_m = (r_m^\top r_m)/(r_{m-1}^\top r_{m-1})$ `; Verbesserung in diesem Schritt`
  $p_m = r_m + \beta_m p_{m-1}$ `; neue Suchrichtung`
**done**

Der Algorithmus ist außerordentlich einfach und sehr unaufwendig. Die einzige Schwierigkeit in der Implementation ist die Wahl eines Konvergenzkriteriums; dies ist jedoch eine Sache für Spezialisten und nicht Gegenstand dieser Vorlesung. In jedem Schritt tritt neben einigen Vektoroperationen nur ein Matrix–Vektor Produkt auf. D.h. für dünnbesetzte Matrizen ist der Aufwand für einen Iterationsschritt $O(n)$.

Warum der Algorithmus funktioniert, sei im folgenden kurz erläutert. Alles beginnt mit dem folgenden Satz

**Theorem 5.0.7.** *Sei $A$ symmetrisch und positiv definit und $b$ beliebig. Solange der Iterationsprozeß nicht konvergiert hat ($r_{m-1} \neq 0$) treten im Algorithmus keine Divisionen durch Null auf, und es gelten die folgenden Identitäten:*

$$\begin{aligned}\mathcal{K}_m &= \langle x_1, x_2, \dots x_m\rangle = \langle p_0, p_1, \dots p_{m-1}\rangle = \\ &= \langle r_0, r_1, \dots r_{m-1}\rangle = \langle b, Ab, \dots A^{m-1}b\rangle.\end{aligned}$$

*Außerdem sind die Residuen orthogonal*

$$r_m^\top r_j = 0, \quad \text{für } j < m$$

*und die Suchrichtungen $A$–konjugiert*

$$p_m^\top A p_j = 0, \quad \text{für } j < m.$$

Beweis. Der Beweis ist ein Induktionsbeweis ohne wesentliche technische Schwierigkeiten. □

Diese Orthogonalitätseigenschaften enthalten bereits die wesentlichen Aussagen über das Verfahren. Der folgende Satz ist eigentlich nur eine Folgerung daraus.

**Theorem 5.0.8.** *Hat das Verfahren noch nicht konvergiert, dann ist $x_m$ der eindeutige Punkt in $\mathcal{K}_m$, der $\|e_m\|_A$ minimiert. Die Konvergenz ist monoton*

$$\|e_m\|_A \leq \|e_{m-1}\|_A,$$

*und $e_m = 0$ wird für ein $m \leq n$ erreicht.*

In Gleitkommaarithmetik ist die Garantie für Konvergenz nach höchstens $n$ Schritten leider nicht mehr haltbar. In der Praxis ist der erforderliche Genauigkeit bereits nach $m \ll n$ Schritten erreicht, was die theoretische exakte Konvergenz für $m = n$ unwichtig erscheinen läßt. Worauf man jedoch besonders zu achten hat ist eine gute Vorkonditionierung (siehe Abschnitt 7), um eine kriechende Konvergenz zu vermeiden.

Der bekannteste Satz über die Konvergenz des CG Verfahrens ist der folgende:

**Theorem 5.0.9.** *Sei das Konjugierte–Gradienten Verfahren angewendet auf die symmetrische positiv definite Matrix $A$ mit $\kappa_2(A) = \kappa$ und den Vektor $b$. Sei $x_0$ der Startvektor des Verfahrens, und sei*

$$e_j = x_j - x_*,$$

*wobei $x_* = A^{-1}b$ für die exakte Lösung steht. Dann erfüllen die $A$–Normen der Fehler*

$$\frac{\|e_m\|_A}{\|e_0\|_A} \leq 2 \Big/ \left( \left(\frac{\sqrt{\kappa}+1}{\sqrt{\kappa}-1}\right)^m + \left(\frac{\sqrt{\kappa}+1}{\sqrt{\kappa}-1}\right)^{-m} \right) \leq 2\left(\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}\right)^m.$$

*Für $k \to \infty$ verhält sich der Fehler asymptotisch wie*

$$\sim \left(1 - \frac{2}{\sqrt{\kappa}}\right)^k.$$

*Man kann also für moderate Konditionszahlen $\kappa$ erwarten, daß der Algorithmus in $O(\sqrt{\kappa})$ Schritten konvergiert.*

Eine der wesentlichen Bedeutungen des CG Verfahrens liegt übrigens darin, daß man es als Algorithmus zur lokalen Optimierung einer reellwertigen Funktion verwenden kann. Diese Anwendung ist Gegenstand von Teil **??**, Kapitel 17.

## 6. Überblick über andere Iterationsverfahren

Der Nachteil der Krylov–Iterationen für nichtsymmetrische Probleme ist der in jedem Schritt steigende Aufwand durch Verwendung einer $(m+1)$–Terme Rekurrenz. Durch Einführung eines zweiten Krylov–Raumes (für $A^*$) kann man auch für unsymmetrische Probleme Drei–Terme Rekurrenzen herleiten. Die Verfahren, die so aufgebaut sind heißen *Biorthogonalisierungsverfahren*. Die beiden bekanntesten Vertreter sind das BCG (biconjugate gradients — bikonjugierte Gradienten) und das CGN (auch CGNR) (conjugate gradients for normal equations — konjugierte Gradienten für die Normalengleichungen) Verfahren.

**6.1. CGN Verfahren.** Das CGN Verfahren ist einfach das CG Verfahren angewendet auf die Normalengleichungen

$$A^*Ax = A^*b.$$

$A^*A$ ist hermitesch und positiv definit, und daher ist das CG Verfahren anwendbar. Die entsprechenden Krylov–Räume werden erzeugt durch die Vektoren

$$\{A^*b, (A^*A)A^*b, \ldots, (A^*A)^{m-1}A^*b\}$$

Die Matrix $(A^*A)$ wird dabei nicht explizit gebildet sondern durch zwei Multiplikationen ersetzt. Das CG Verfahren minimiert die $(A^*A)$–Norm von $x_m - x_*$, und es gilt

$$\|e_m\|_{A^*A} = \sqrt{e_m^* A^* A e_m} = \|Ae_m\|_2 = \|r_m\|_2.$$

Das CGN Verfahren minimiert also die 2–Norm des Residuums in jedem Schritt; es ist also eine Minimale–Residuen Methode wie GMRES. Da sich jedoch die Krylov–Räume unterscheiden, sind die beiden Methoden ebenfalls grundverschieden. Die Konvergenzrate für das CGN Verfahren kann aus derjenigen für das CG Verfahren hergeleitet werden: es gilt

$$\frac{\|r_m\|_2}{\|r_0\|_2} \leq 2\left(\frac{\kappa - 1}{\kappa + 1}\right),$$

weil $\kappa_2(A^*A) = \kappa_2(A)^2$. Für große $\kappa$ impliziert das, daß man mit $O(\kappa)$ Iterationen rechnen muß bis zur Konvergenz auf vorgegebene Genauigkeit, ein Wert, der für praktische Anwendungen viel zu hoch ist; speziell wenn die Matrizen nicht sehr gut konditioniert sind.

Das CGN Verfahren ist jedoch dann im Vorteil gegenüber dem CMRES Verfahren, wenn die Matrix brave Singulärwerte besitzt, sich die Eigenwerte aber in einem Ring rund um den Ursprung anordnen. Dann benötigt das CMRES Verfahren $m \approx n$ Iterationsschritte während CGN relativ rasch konvergiert.

**6.2. BCG Verfahren.** Das BCG Verfahren basiert auf der Idee, eine Drei–Terme Rekurrenz dadurch zu erzwingen, daß es eine Faktorisierung der Form

$$A = VTV^{-1}$$

bestimmt mit tridiagonaler Matrix $T$. Verwendet man auch noch die adjungierte Gleichung

$$A^* = V^{-*}T^*(V^{-*})^{-1},$$

so kann man beide Beziehungen verwenden, um die Faktorisierung zu berechnen. Dabei hilft, daß zwar $V$ nicht mehr unitär ist, aber immer noch die beiden Matrizen $V$ und $V^{-*}$ „orthogonal zueinander“ sind:

$$(V^{-*})^*V = \mathbb{I};$$

daher auch der Ausdruck Biorthogonalisierungsverfahren.

Setzen wir $W = V^{-*}$, so stehen die Spalten $v_i$ bzw. $w_j$ biorthogonal aufeinander im folgenden Sinn:

$$w_i^* v_j = \delta_{ij}.$$

Fassen wir wieder die ersten $m$ Vektoren in die Matrix $V_m$ bzw. $W_m$ zusammen, so kann man die Biorthogonalitätsbeziehungen schreiben als

$$W_m^* V_m = V_m^* W_m = \mathbb{I}.$$

Daraus folgen schließlich die zur Arnoldi Iteration analogen Gleichungen

$$\begin{aligned} AV_m &= V_{m+1}\hat{T}_m \\ A^*W_m &= W_{m+1}\hat{S}_m \\ T_m = S_m^* &= W_m^* A V_m, \end{aligned}$$

mit Tridiagonalmatrizen $T_m$ und $S_m$. $T_m$ ($S_m$) erhält man wieder aus $\hat{T}_m$ ($\hat{S}_m$) durch Löschung der letzten Zeile (Spalte). Diese Gleichungen entsprechen den beiden Drei–Terme Rekurrenzen

$$\begin{aligned} Av_m &= \gamma_{m-1}v_{m-1} + \alpha_m v_m + \beta_m v_{m+1}, \\ A^*w_m &= \bar{\beta}_{m-1}w_{m-1} + \bar{\alpha}_m w_m + \bar{\gamma}_m w_{m+1}, \end{aligned}$$

wobei die $\alpha_j$ die Hauptdiagonale, die $\beta_j$ die untere Nebendiagonale und die $\gamma_j$ die obere Nebendiagonale von $T$ besetzen.

Die Vektoren $v_j$ liegen im Krylov–Raum zu $A$ und $v_1$, während die $w_j$ im Krylov–Raum zu $A^*$ und $w_1$ liegen. Der klassische Algorithmus, der diese Beziehungen ausnützt, ist der folgende:

**Algorithmus 6.2.1.** *Bikonjugierte Gradienten*

$x_0 = 0$
$p_0 = r_0 = b$
$q_0 = s_0 =$*beliebig*
**for** $m = 1$ **to** ? **do**
  $\alpha_m = (s_{m-1}^* r_{m-1})/(q_{m-1}^* A p_{m-1})$
  $x_m = x_{m-1} + \alpha_m p_{m-1}$
  $r_m = r_{m-1} - \alpha_m A p_{m-1}$
  $s_m = s_{m-1} - \alpha_m A q_{m-1}$
  $\beta_m = (s_m^* r_m)/(s_{m-1}^* r_{m-1})$
  $p_m = r_m + \beta_m p_{m-1}$
  $q_m = s_m + \beta_m q_{m-1}$
**done**

Es gilt $s_m^* r_j = 0$ und $q_m^* A p_j = 0$, woraus wieder die Konvergenz folgt.

Es existieren noch einige weitere Biorthogonalisierungsverfahren, wie CGS (cojugate gradient squared), QMR (quasi minimal residuals), TFQMR (transpose–free QMR), ... alle haben ihre Vor– und Nachteile. Bislang konnte sich jedoch noch keiner dieser Algorithmen durchsetzen. Die Weiterentwicklung der Verfahren stellt im Moment einen großen Forschungsschwerpunkt im Bereich der numerischen linearen Algebra dar, und die Zukunft wird zeigen, welches der Verfahren schließlich die Oberhand gewinnen wird.

## 7. Vorkonditionierung

Wie schon mehrfach in diesem Kapitel erwähnt, ist *Vorkonditionierung* ein wesentlicher Schritt vor der Lösung eines linearen Gleichungssystemes. Der mathematische Hintergrund ist der folgende:

Für jede invertierbare Matrix $M$ haben

$$Ax = b \quad \text{und} \quad M^{-1}Ax = M^{-1}b$$

dieselben Lösungen. Wenn man allerdings das Gleichungssystem mit einem Iterationsverfahren löst, so hängt die Konvergenzgeschwindigkeit im einen System von den Eigenschaften von $A$, im anderen System jedoch von den Eigenschaften von $M^{-1}A$ ab. Wählt man den *Vorkonditionierer* $M$ gut, so kann das zweite Gleichungssystem mitunter um ein Vielfaches schneller gelöst werden als das erste.

Nun kann man aber $M^{-1}A$ nicht explizit berechnen. Daher muß man den Vorkonditionierer so wählen, daß man Multiplikationen $M^{-1}Ax$ effizient berechnen kann. Außerdem soll das Produkt $M^{-1}A$ schöne Eigenschaften haben, etwa daß $\|M^{-1}A - \mathbb{I}\|$ klein ist, oder daß $M^{-1}A$ wenigstens fast normal ist und die Eigenwerte in einem Cluster liegen.

Es gibt eine Menge an Verfahren, die solche Vorkonditionierer erzeugen, wie etwa *unvollständige Cholesky–Zerlegungen*, *Diagonalskalierungen* oder *Block–Vorkonditionierer*. Alle oder auch nur einige dieser Vorkonditionierer zu dikutieren würde aber den Rahmen dieser Vorlesung sprengen, und daher sei hier nur noch einmal deutlich auf die Notwendigkeit verwiesen und auf die Literatur zu diesem Thema ([**Golub, Van Loan 1996**] wäre eventuell ein guter Punkt, um eine Beschäftigung mit dieser Problematik zu beginnen).

# Literaturverzeichnis


[Rump 1988] Rump, S.M., *Algorithm for verified inclusions—theory and practice*, in "Reliability in Computing" (Moore, R.E., ed.), Academic Press, San Diego, 1988.

[Hansen 1992] Hansen, E., *Global Optimization using Interval Analysis*, Monographs in Pure and Applied Mathematics, Marcel Dekker, New York, 1992.

[Überhuber 1995a] Überhuber, C., *Computernumerik 1*, Springer Verlag, Berlin Heidelberg New York, 1995.

[Überhuber 1995b] Überhuber, C., *Computernumerik 2*, Springer Verlag, Berlin Heidelberg New York, 1995.

[Goldberg 1991] Goldberg, D., *What every Computer Scientist should Know About Floating-Point Arithmetic*, ACM Computing Surveys **23** (1991), 17–27.

[Reichel, Zöchling 1990] Reichel, H.-C., Zöchling, J., *Tausend Gleichungen - und was nun? - Computertomographie als Einstieg in ein aktuelles Thema des Mathematikunterrichtes*, Didaktik der Mathematik **4** (1990), 245–270.

[Stoer 1994a] Stoer, J., *Numerische Mathematik 1*, Springer Verlag, Berlin Heidelberg New York, 1994.

[Stoer 1994b] Stoer, J., *Numerische Mathematik 2*, Springer Verlag, Berlin Heidelberg New York, 1994.

[Ören 1979] Ören, T.I., *Concepts for Advanced Computer Assisted Modelling*, ACM Computing Surveys(1979),

[Skeel 1980] Skeel, R.D., *Iterative Refinement Implies Numerical Stability for Gaussian Elimination*, Math. Comp. **35** (1980), 817–832.

[Higham 1996] Higham, N.J., *Accuracy and Stability of Numerical Algorithms*, SIAM Publications, Philadelphia, 1996.

[Trefethen, Bau 1997] Trefethen, L.N., Bau, III D., *Numerical Linear Algebra*, SIAM, Philadelphia, 1997.

[Golub, Van Loan 1996] Golub, G.H., Van Loan, C.F., *Matrix Computations*, 3rd edition, The Johns Hopkins University Press, Baltimore, 1996.

[Schwarz 1986] Schwarz, H.R., *Numerische Mathematik*, 2. Auflage, B.G. Teubner, Stuttgart, 1986.

[Heuser 1986/1] Heuser, H., *Lehrbuch der Analysis, Teil 1*, B.G. Teubner, Stuttgart, 1986.

[Heuser 1986/2] Heuser, H., *Lehrbuch der Analysis, Teil 2*, B.G. Teubner, Stuttgart, 1986.

[Bulirsch, Rutishauser 1968] Bulirsch, R., Rutishauser, H., *Interpolation und genäherte Quadratur*, in "Mathematische Hilfsmittel des Ingeneurs, Part III" (Sauer, R., Szabo, I., eds.), Springer Verlag, Berlin New York Heidelberg, 1968.

[Cooley, Tukey 1965] Cooley, J.W., Tukey, J.W., *An algorithm for the machine calculation of complex Fourier series*, Math. Comput. **19** (1965), 297–301.

[Gentleman, Sande 1966] *Fast Fourier transforms — For fun and profit*, in "Proc. AFIPS 1966 Fall Joint Computer Conference",**29**, Spartan Books, Washington D.C., 1966.

[Good 1958] Good, I.J., *The interaction algorithm and practical Fourier series*, J. Roy. Statist. Soc. Ser. B **20** (1958), 361–372.; Addendum: **22**(1960), 372–375

[Risler 1992] Risler, J.–J., *Mathematical Methods for CAD*, Cambridge University Press, Cambridge, 1992.

[Rutishauser 1960] Rutishauser, H., *Bemerkungen zur glatten Interpolation*, ZaMP **11** (1960), 508–513.

[Böhmer 1974] Böhmer, K., *Spline–Funktionen*, B.G. Teubner, Stuttgart, 1974.

[de Boor 1978] Boor, C. de, *A Practical Guide to Splines*, Springer, Berlin New York Heidelberg, 1978.

[Schoenberg, Whitney 1953] Schoenberg, I.J., Whitney, A., *On Polya frequency functions, III: The positivity of translation determinants with an application to the interpolation problem by spline curves*, Trans. Amer. Math. Soc. **74** (1953), 246–259.

[Nielson 1974] Nielson, G. M., *Some Piecewise Polynomial Alternatives to Splines Under Tension*, in "Computer Aided Geometric Design" (Barnhill, R. E., Riesenfeld, R. F., eds.), Academic Press, New York San Francisco London, 1974.

[Kronrod 1965] Kronrod, A. S., *Nodes and Weights of Quadrature Formulas*, Consultants Bureau, New York(1965),

[Gander 1985] Gander, W., *Computermathematik*, Birkhäuser, Basel, 1985.

[Broyden et al. 1970] Broyden, C. G., Dennis, J. E., Moré, J. J., *On the local and superlinear convergence of quasi–Newton methods*, J. Inst. Math. Appl. **12** (1970), 223-245.

[Gill et al. 1974] Gill, P. E., Golub, G. H., Murray, W., Saunders, M. A., *Methods for modifying matrix factorizations*, Math. Comput. **28** (1974), 505–535.

[Oren, Luenberger 1974] Oren, S. S., Luenberger, D. G., *Self–scaling variable metric (SSVM) algorithms. I. Criteria and sufficient conditions for scaling a class of algorithms*, Manage. Sci. **20** 845–862.

[Knuth 1969] Knuth, D. E., *The Art of Computer Programming*, Addison–Wesley, Reading, Massachusetts, 1969.

[Hermeline 1982] Hermeline, F., *Triangulation automatique d'un polyèdre en dimension N*, RAIRO, Analyse numérique **16, 3** (1982), 211–242.

[Bronstein, Semendjajev 1989] Bronstein, I. N., Semendjajev, K. A., *Taschenbuch der Mathematik*, Verlag Harri Deutsch, Thun Frankfurt/Main, 1989.

[Hart 1968] Hart, J. F., et al., *Computer Approximations*, John Wiley, New York, 1968.

[Metropolis, Ulam, 1949] Metropolis, N., Ulam, S. M., *The Monte Carlo method*, J. Amer. Statist. Assoc. **44** (1949), 335–341.

[Niederreiter 1992] Niederreiter, H., *Random Number Generation and Quasi–Monte Carlo Methods*, SIAM, Philadelphia, 1992.

[Kuipers, Niederreiter 1974] Kuipers, L., Niederreiter, H., *Uniform Distribution of Sequences*, John Wiley, New York, 1974.

[Koksma 1942/43] Koksma, J. F., *Een algemeene stelling uit de theorie der gelijkmatige verdeeling modulo 1*, Mathematica B (Zutphen) **11** (1942/43), 7–11.

[Hlawka 1961] Hlawka, E., *Funktionen von beschränkter Variation in der Theorie der Gleichverteilung*, Ann. Mat. Pura Appl. **54** (1961), 325–333.

[Schmidt 1972] Schmidt, W. M., *Irregularities of distribution. VII*, Acta Arith. **21** (1972), 45–50.

[IEEE 754–1985] *Standard for Binary Floating Point Arithmetic*, ANSI/IEEE Standard **754** (1985),

[Neumaier 2000] Neumaier, A., *Introduction to Numerical Analysis*, Cambridge University Press, Cambridge, 2000.

[Neumaier 1990] Neumaier, A., *Interval Methods for Systems of Equations*, Encyclopedia of Mathematics and its Applications 37, Cambridge University Press, Cambridge, 1990.

[Neumaier 1974] *Rundungsfehleranalyse einiger Verfahren zur Summation endlicher Summen*, Z. Angew. Math. Mech. **54** (1974), 39–51.

[van der Sluis 1969] *Condition numbers and equilibration of matrices*, Numer. Math. **14** (1969), 14–23.

[Prager, Oettli 1964] *Compatibility of approximate solutions of linear equations with given error bounds for coefficients and right hand sides*, Numer. Math. **6** (1964), 405–409.

[Wilkinson 1968] *À Priori Error Analysis of Algebraic Processes*, Proc. International Congress Math.(1968), 629–639.

[Neumaier 1998] Neumaier, A., *Formstabile Interpolation*, Preprint(1998),

[Opitz 1958] *Gleichungsauflösung mittels einer speziellen Interpolation*, Z. Angew. Math. Mech. **38** (1958), 276–277.