Freie Universität Berlin

INSTITUT FÜR MATHEMATIK II

Mit Zahlen Rechnen

Computerorientierte Mathematik I

Ralf Kornhuber, Christof Schütte, Andreas Fest

5. Auflage Wintersemester 2005/2006
(Stand: 4. Januar 2006)

Stefan Geschke hat wertvolle Literaturhinweise gegeben. Sabrina Nordt hat zwei Bilder beigesteuert. Ralf Forster, Carsten Gräser, Maren Hofmann, Katrin Hornburg, Justyna Komarnicki, Eike Sommer, Bernhard Streit haben sorgfältig Korrektur gelesen und viele gute Ratschläge gegeben. **Vielen Dank!**

Inhaltsverzeichnis

Teil I

Kondition und Stabilität

1 Zahlendarstellung und Rundungsfehler

1.1 Natürliche Zahlen

1.1.1 Rechnen und Ausrechnen

Die Menge $\mathbb{N}$ der natürlichen Zahlen ist in folgender Weise charakterisiert (vgl. Ebbinghaus et al. [**?**, Kap. 1]). Es gibt ein ausgezeichnetes Element $0 \in \mathbb{N}$ und eine Abbildung $S : \mathbb{N} \to \mathbb{N}$ mit den Eigenschaften:

(S1) S ist injektiv.

(S2) $0 \notin S(\mathbb{N})$.

(S3) Ist $M \subset \mathbb{N}$ und sowohl $0 \in M$ als auch $S(M) \subset M$ so gilt $M = \mathbb{N}$.

Die Funktion S (engl. *successor*) ordnet jeder natürlichen Zahl $n \in \mathbb{N}$ ihren Nachfolger $S(n)$ zu. Axiom (S1) besagt, daß dabei jede natürliche Zahl höchstens einmal vorkommt. Aus (S2) folgt, daß dieser Zählprozess bei 0 beginnt[1]. Axiom (S3) begründet die vollständige Induktion. Wir wollen die Addition zweier natürlicher Zahlen definieren. Dazu sei $n \in \mathbb{N}$ beliebig, aber fest gewählt. Wir setzen $n+0 = n$ und fordern außerdem, daß die Rekursionsformel $n+S(m) = S(n+m)$ gelten soll. Damit sind wir fertig. Bezeichnet nämlich $M \subset \mathbb{N}$ die Menge aller $m \in \mathbb{N}$, für die $n+m \in \mathbb{N}$ nun wohldefiniert ist, so gilt $0 \in M$, wie vereinbart, und $S(m) \in M$ aufgrund der Rekursionsformel. Das bedeutet aber $M = \mathbb{N}$ wegen (S3). Setzt man übrigens $1 = S(0)$, so ergibt sich die Rechenregel $n + 1 = S(n)$. Das Produkt $n \cdot m$ sowie Kommutativität, Assoziativität und Distributivität erhält man auf ähnliche Weise (vgl. Landau [**?**, Kap. 1, § 4]). Nun können wir also mit natürlichen Zahlen rechnen.

Mit einer wesentlichen Einschränkung: *Symbolisch* können wir die Summe $s = n+m$ zweier Zahlen $n, m \in \mathbb{N}$ zwar bilden, *numerisch* ausrechnen können wir sie aber noch nicht. Dazu müsste erst jede natürliche Zahl ihren eigenen, unverwechselbaren Namen haben. Ausrechnen von $s = n + m \in \mathbb{N}$ bedeutet dann, zu den gegebenen Namen der Summanden n, m den gesuchten Namen der Summe s zu finden. Damit man sowohl Aufgabe als auch Ergebnis aufschreiben und anderen mitteilen kann, dürfen solche Namen nur aus endlich vielen Zeichen oder Symbolen bestehen. In diesem Sinne bilden Ziffernsysteme die Grundlage des numerischen Rechnens.

[1] Die hier beschriebene Charakterisierung von $\mathbb{N}$ geht auf die Axiome von G. Peano (1858 - 1932) zurück. Andere Definitionen (z.B. von G. Cantor (1845 - 1918) oder R. Dedekind (1831 - 1916)) ziehen den Beginn mit 1 vor und betrachten somit die 0 nicht als natürliche Zahl.

1.1.2 Ziffernsysteme.

Sei $\mathcal{Z}$ eine endliche Menge von Ziffern. Daraus bilden wir endliche Ketten

$$z_1\ z_2\ \cdots\ z_k, \qquad z_i \in \mathcal{Z},\ i = 1, \ldots, k,$$

wie Worte aus den Buchstaben eines Alphabets. Wir zeigen nun, daß sich jeder Zahl $n \in \mathbb{N}$ genau eine solche Ziffernkette als Name zuordnen lässt.

Satz 1.1 (Zifferndarstellung) *Ist $\mathcal{Z}$ eine endliche Menge von Ziffern und*

$$\mathcal{D}(\mathcal{Z}) = \{z_1\ z_2\ \cdots\ z_k \mid k \in \mathbb{N}, z_i \in \mathcal{Z},\ i = 1, \ldots, k\}, \tag{1.1}$$

so existiert eine bijektive Abbildung

$$\varphi : \mathbb{N} \to \mathcal{D}(\mathcal{Z}). \tag{1.2}$$

Beweis: Es reicht zu zeigen, daß sich alle Elemente von $\mathcal{D}(\mathcal{Z})$ nacheinander anordnen lassen. Gleichbedeutend damit ist die Existenz einer Nachfolger-Abbildung $\sigma : \mathcal{D}(\mathcal{Z}) \to \mathcal{D}(\mathcal{Z})$ mit den gleichen Eigenschaften wie S. Ausgehend von dem Anfangselement $d_0 \in \mathcal{D}(\mathcal{Z})$ können wir dann φ einfach wie folgt definieren

$$\varphi(0) = d_0, \qquad \varphi(S(n)) = \sigma(\varphi(n))\ .$$

Zur Anordnung von $\mathcal{D}(\mathcal{Z})$ bemerken wir zunächst, daß

$$\mathcal{D}(\mathcal{Z}) = \bigcup_{k=1}^{\infty} \mathcal{D}_k, \qquad \mathcal{D}_k = \{z_1\ z_2\ \cdots\ z_k \mid z_i \in \mathcal{Z},\ i = 1, \ldots, k\}\ .$$

Die Reihenfolge der Mengen $\mathcal{D}_k$ untereinander folgt den natürlichen Zahlen, und die endlich vielen Elemente innerhalb jeder einzelnen Menge $\mathcal{D}_k$ lassen sich natürlich ihrerseits in irgendeine Reihenfolge bringen[2]. ■

Jede spezielle Wahl von $\mathcal{Z}$ und φ erzeugt ein Ziffernsystem zur Darstellung von $\mathbb{N}$. Umgangssprachlich wird zwischen verschiedenen Ziffernsystemen und den natürlichen Zahlen selbst nicht unterschieden. So spricht man von römischen Zahlen, Dezimalzahlen, Dualzahlen und so weiter.

Eine wegen ihrer Einfachheit beliebte Wahl ist

$$\mathcal{Z} = \{\ |\ \}, \quad \mathcal{D}(\mathcal{Z}) = \{\ |,\ ||,\ |||,\ \ldots\}, \qquad \varphi(1) = |, \quad \varphi(n+1) = \varphi(n)\ |\ . \tag{1.3}$$

Die 0 bildet dabei eine Ausnahme. Das resultierende Unärsystem findet seit Jahrtausenden unter anderem auf Knochen, Kerbhölzern oder Bierdeckeln Verwendung (siehe Abbildung 1). Mehr zu diesem Thema erfährt man bei Ifrah [**?**, Kapitel 4]. Allerdings erweist sich solch eine Darstellung im Falle großer Zahlen schnell als unübersichtlich (vgl. erste Anklänge eines Quinärsystems in Abb. 1 rechts).

1.1.3 Positionssysteme.

Sei $q \in \mathbb{N}$, $q > 1$, fest gewählt. Dann erhält man zu jeder natürlichen Zahl n durch sukzessive Division mit Rest die Zerlegung

$$n = \sum_{i=0}^{k} r_i q^i \tag{1.4}$$

[2] Genauer hat $\mathcal{D}_k$ genau q^k Elemente, wobei q die Anzahl der Elemente von $\mathcal{Z}$ bedeutet.

Abbildung 1: Frühe Ziffernsysteme: Ishango-Knochen und Bierdeckel

mit eindeutig bestimmten Koeffizienten $r_i \in \{0, \ldots, q-1\} \subset \mathbb{N}$. Die Grundidee eines Positionssystems ist es nun, jede Zahl $r_i \in \{0, \ldots, q-1\}$ mit einer Ziffer zu bezeichnen und daraus den Namen von n zusammenzusetzen.

Definition 1.2 (Positionssystem zur Basis q) *Das Positionssystem zur Basis q ist charakterisiert durch eine Ziffernmenge $\mathcal{Z}$ mit q Elementen, eine bijektive Abbildung*

$$\varphi : \{0, 1, \ldots, q-1\} \ni i \mapsto z_i \in \mathcal{Z}$$

und deren durch die Zerlegung (1.4) *induzierte Erweiterung*

$$\varphi : \ \mathbb{N} \ni n \mapsto \ z_{r_k}\, z_{r_{k-1}} \cdots z_{r_0} \in \mathcal{D}(\mathcal{Z}).$$

auf ganz $\mathbb{N}$. Die so erzeugte Zifferndarstellung heißt q-adische Darstellung.

Wir sind seit Kindesbeinen an das Dezimalsystem gewöhnt. Dabei wird φ so definiert, daß der Ziffernvorrat $\mathcal{Z} = \{0, 1, 2, \ldots, 9\}$ bijektiv auf die ersten natürlichen Zahlen $\{0, S(0), S\circ S(0), \ldots, S\circ \cdots \circ S(0)\} \subset \mathbb{N}$ abgebildet wird (S bedeutet die Nachfolger-Funktion und $\circ$ die Hintereinanderausführung). Nach Konstruktion bezeichnet dann 10 den Nachfolger der natürlichen Zahl namens 9, und die Zerlegung (1.4) vererbt sich auf die entsprechenden Zifferndarstellungen. Beispielsweise ist

$$123 = 1 \cdot 10^2 + 2 \cdot 10^1 + 3 \cdot 10^0.$$

Wir identifizieren von nun an die natürlichen Zahlen mit ihrer Darstellung im Dezimalsystem.

Auch bei q-adischen Darstellungen mit $q \neq 10$ unterscheiden wir nicht mehr zwischen den Ziffern $z_i \in \mathcal{Z}$ und den natürlichen Zahlen $i \in \{0, 1, \ldots, q-1\}$. Wir schreiben also einfach[3]

$$z_k\, z_{k-1} \ldots z_{0_q} = \sum_{i=0}^{k} z_i q^i, \qquad z_i \in \mathcal{Z} = \{0, 1, 2, \ldots, q-1\}\ .$$

[3]Im Falle $q > 10$ müssten wir uns allerdings noch weitere Ziffern ausdenken. Oft werden zunächst die lateinischen Großbuchstaben verwendet, also A, B, …, Z, für 10, 11, 12, …, 35.

Falls keine Missverständnisse auftreten können, verzichten wir auf den Index q.

Unsere heutige Dezimalnotation entwickelte sich in Indien in der Zeit von 600 - 300 v. Chr. und fand über arabische Astronomen den Weg nach Europa. Das erste Positionssystem tauchte aber schon zu Beginn des zweiten Jahrtausends v. Chr. in Mesopotamien auf. Die babylonischen Mathematiker verwendeten ein Sexagesimalsystem, also die Basis $q = 60$. Im Sexagesimalsystem bedeutet

$$123_{60} = 1 \cdot 60^2 + 2 \cdot 60^1 + 3 \cdot 60^0 \ ,$$

und man rechnet schnell nach, daß dieselbe Zahl im Dezimalsystem den Namen 3723 trägt. Interessanterweise machten die Babylonier bei der Entwicklung der benötigten 60 Ziffern vom Dezimalsystem Gebrauch. Die Mayas und Azteken verwendeten ein Vigesimalsystem, also die Basis $q = 20$. Das dekadische Ziffernsystem der Griechen war kein Positionssystem. Eine Fülle weiterer Informationen zu Ursprung und Geschichte der Ziffern- und Positionssysteme findet sich bei Ebbinghaus et al. [**?**, Kap. 1] und Ifrah [**?**].

1.1.4 Dualdarstellung

Moderne Rechenmaschinen arbeiten sämtlich mit dem Dualsystem, also dem Positionssystem zur Basis $q = 2$. Es ist besonders praktisch, daß der benötigte, digitale Ziffernvorrat $\mathcal{Z} = \{0, 1\}$ auf vielfältige Weise technisch umgesetzt werden kann (vgl. Abschnitt 1.1.5). Aber schon G. W. Leibniz (1646 - 1716) schätzte das Dualsystem sehr, wenn auch, wie Laplace berichtet, aus ganz anderen Gründen[4]: „Leibniz sah in der dyadischen Arithmetik das Bild der Schöpfung. Er stellte sich vor, die Einheit stelle Gott dar und die Null das Nichts; das höchste Wesen habe alle Dinge aus dem Nichts erschaffen, ebenso wie die Einheit und die Null alle Zahlen seines Zahlensystems ausdrücken.“ Beispielsweise ist

$$1010_2 \ = \ 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 10 \ .$$

Im Dualsystem kann man mit N Stellen alle Zahlen $n \in \mathbb{N}$ mit der Eigenschaft

$$0 \leq n \leq 2^N - 1$$

darstellen. Dies folgt durch direktes Ausrechnen, aber auch aus der Summenformel für die geometrische Reihe

$$\sum_{i=0}^{N} q^i = \frac{q^{N+1} - 1}{q - 1}$$

und Einsetzen von $q = 2$.

Im Vergleich mit dem Dezimalsystem ist das Rechnen im Dualsystem sehr einfach. Zur Addition hat man sich nur $1 + 1 = 10_2$ zu merken, und das kleine Einmaleins $1 \cdot 1 = 1$ macht seinem Namen alle Ehre. Die aus der Schule bekannten Methoden zur schriftlichen Addition und Multiplikation bleiben anwendbar:

$$\begin{array}{cccccc} & 1 & 0 & 1 & 1 \\ + & {\scriptstyle 1} & 1_1 & 0_1 & 1 \\ \hline 1 & 0 & 0 & 0 & 0 \end{array} \qquad \begin{array}{cccccccc} 1 & 0 & 0 & 1 & \cdot & 1 & 1 \\ \hline & & 1 & 0 & 0 & 1 & \\ & & & 1 & 0 & 0 & 1 \\ \hline & & 1 & 1 & 0 & 1 & 1 \end{array}$$

Im Dezimalsystem hätten wir die Aufgaben $11 + 5 = 16$ und $9 \cdot 3 = 27$ allerdings im Kopf erledigt.

[4] Zitiert nach Courant und Robbins [**?**, Kap. § 1]

1.1.5 Praktische Realisierung

Eine kurze Geschichte der Rechenmaschinen

Solange Menschen rechnen, solange versuchen sie auch schon, Maschinen zu konstruieren, die ihnen diese oftmals lästige Arbeit abnehmen. Bereits lange vor unserer Zeitrechnung war mit dem Abakus oder Rechenbrett die erste mechanische Rechenhilfe bekannt, die auf einem Stellenwertsystem basiert.[5] Neben dem Abakus wurden bereits um die Zeitwende herum mechanische Zählräder als Wegmeßgeräte verwendet, wie sie auch noch um 1500 n. Chr. von Leonardo da Vinci beschrieben wurden.

Erste Rechenmaschinen, die auf einem dekadischen Zahlensystem beruhten und alle vier Grundrechenarten beherrschten, wurden u.a. 1623 von Wilhelm Schickard, 1642 von Blaise Pascal und 1673 von Gottfried Wilhelm von Leibniz entwickelt. Leibniz war es auch, der im Jahre 1679 das „dyadische Zahlensystem“, erfand, das wir heute als das Dualsystem kennen.

Der erste allein auf Dualzahlarithmetik basierende, frei programmierbare Rechner war der von Konrad Zuse 1935 bis 1938 im Wohnzimmer seiner Eltern in Berlin-Tempelhof gebaute Z1[6]. Er speicherte und verarbeitete die verwendeten Dualzahlen rein mechanisch mittels Kippschaltern. Die mechanischen Bauteile arbeiteten so unpräzise, daß einige der Funktionen nur mangelhaft realisiert wurden. So blieb dieser Rechner nur ein Versuchsgerät, das keine praktische Anwendung fand. Erst der 1941 fertiggestellte Z3, bei dem die mechanischen Schalter durch elektromechanische Relais (wie sie sonst in Telefonanlagen verwendet wurden) ersetzt wurden, kann als der erste voll funktionstüchtige, frei programmierbare, automatische Dualzahl-Rechenautomat bezeichnet werden.

Relais-Rechner wurden zu dieser Zeit ebenfalls in den USA entwickelt. Sie dienten jedoch ausschließlich dazu, im Zweiten Weltkrieg die Flugbahnen von Flugzeugen und ballistischen Geschossen schneller und genauer zu berechnen. Ein Problem bei diesen auf elekromechanischen Bauteilen basierenden Maschinen waren Kakerlaken, die regelmäßig die Relais blockierten. Um die daraus resultierenden Fehler zu beheben, ging man auf die Suche nach „bugs“ (Wanzen). Im Sprachgebrauch sucht man deshalb auch heute noch in fehlerhaften Programmen nach „bugs“.

Im englischen Cambridge wurde 1944 mit Colossos der erste auf Elektronenröhren basierende, vollständig elektronische, programmierbare Rechenautomat gebaut. Mit seiner Hilfe konnte der Enigma-Code geknackt werden, mit dem die deutsche Wehrmacht ihre Funksprüche verschlüsselte.

1947 wurde in den Bell Laboratories in Murray Hill, New Jersey, von John Bardeen, Walter Brattain, und William Shockley der Transistor erfunden, wofür sie 1956 den Physik-Nobelpreis erhielten. Der erste Computer, in dem Transistoren die bis dahin verwendeten Elektronenröhren ersetzten, wurde 1955 gebaut. Der Einsatz der Halbleitertechnik ermöglichte es, immer kleinere und dabei dennoch leistungsfähigere Computer zu bauen. Diese Entwicklung wurde durch die Einführung des Mikroprozessors 1971 durch Intel forciert. Ein aktueller Intel Pentium 4 Prescot Prozessor beherbergt 125 Millionen Transistoren auf einer Fläche von 112 mm^2.

[5] Die Ursprünge des Abakus lassen sich heute nicht mehr genau zurückverfolgen, jedoch ist es wahrscheinlich, daß er um 1000 v. Chr. in China entstand. Manche Quellen sehen seinen Ursprung jedoch in Zentralasien oder sogar in Madagaskar. Bekannt ist jedoch, daß er um die Zeitwende bereits bei den römischen Rechenmeistern (*calculatores*) sehr weit verbreitet war.

[6] Ein Nachbau befindet sich im Deutschen Technikmuseum in Berlin.

Speicherung und Verarbeitung natürlicher Zahlen im Computer

Die kleinste digitale Informationseinheit im Computer, welche die Werte 0 oder 1 annehmen kann, heißt Bit. Den Datenspeicher eines Computers kann man sich als eine sehr lange Aneinanderreihung von solchen Bits vorstellen, wobei jeweils eine gleiche Anzahl von Bits zu einer Einheit zusammengefasst werden, die urprunglich mit dem Begriff Wort bezeichnet wurde. Die Länge eines Wortes kann dabei maschinenabhängig variieren. In der Praxis hat sich für eine Folge von 8 Bit der Begriff Byte[7] durchgesetzt. Mit einem Byte lassen sich $2^8 = 256$ verschiedene Zustände speichern. Inzwischen hat sich die Bedeutung des Begriffes Wort etwas verändert. Während sich das Byte als Grundeinheit für die Aufteilung des Speichers manifestiert hat, bezeichnet man als Wort heute die Anzahl von Bits, mit denen die einzelnen Speicherzellen addressiert (also durchnummeriert, angesprochen) werden. Eine Wort hat dabei i.A. eine Länge von mehreren Bytes.

Wenn nun aber der Speicher des Computers aus einer sehr langen, ununterbrochenen Folge von Bytes besteht, wie kann der Computer dann bei der Speicherung von Zahlen feststellen, wann die Bitfolge einer Zahl aufhört und die nächste beginnt? Ist z.B. die Folge 1001110101011101 die Darstellung von $1001110101011101_2 = 40285_{10}$, oder sind hier die beiden Zahlen $10011101_2 = 157_{10}$ und $01011101_2 = 93_{10}$ gespeichert?

Dieses Problem wird gelöst, indem bei der Entwicklung des Computers festgelegt wird, wie viele Bytes verwendet werden, um eine Zahl zu speichern[8]. Dabei ist es unerheblich, wie groß oder klein der Wert der Zahl tatsächlich ist, jede Zahl verwendet genauso viele Bytes. Ist die Zahl so klein, daß sie weniger signifikante Stellen als die Maschinenzahl Bits besitzt, so werden die hochwertigen Stellen mit 0 aufgefüllt. Zahlen, die mehr signifikante Stellen besitzen, können jedoch nicht gespeichert werden.

Die Anzahl von Bytes, die zur Speicherung natürlicher Zahlen verwendet werden, ist nicht standardisiert. Moderne Computer verwenden heutzutage 8 Byte, also 64 Bit zur Darstellung von Zahlen. In älteren Rechnergenerationen sind Zahldarstellungen mit 8, 16 oder 32 Bit, also 1, 2 bzw. 4 Byte üblich. Diese Zahlentypen werden dann oftmals auch als BYTE, WORD bzw. DWORD bezeichnet, 64-Bit-Zahlen zuweilen auch als QWORD. Mit einer N Bit langen Zahl können natürliche Zahlen von 0 bis $2^N - 1$ dargestellt werden. Bei einer 64 -Bit-Zahl liegt der darstellbare Bereich also zwischen 0 und $2^{64} - 1 > 18 \cdot 10^{18} = 18$ Trillionen.

Verwendung natürlicher Zahlen im Computer

In heutigen Computern finden natürliche Zahlen ihre Anwendung hauptsächlich in der Adressierung von Speicherzellen und der Kodierung von Zeichen (ASCII-Code, Unicode).

Nummeriert man die einzelnen Bytes im Speicher des Computers durch (dies nennt man Adressierung, man kann es sich wie fortlaufende Hausnummern an den einzelnen Speicherzellen vorstellen), so können mit einer 64-Bit-Zahl Speicherbereiche bis zu einer Größe von etwa 18 Milliarden Giga-

[7] Das Kunstwort Byte wurde von IBM geprägt, die bei ihrem 1964 vorgestellten Rechner „System/360“ die Standardisierung der Wortlänge auf 8 Bit einführten. Der Begriff Byte ist an das phonetisch gleiche englische Wort „bite“ (Biss) angelehnt, daß eine klare Steigerung des Wortes „bit“ (Bisschen) darstellt.

[8] Genauer gesagt wird festgelegt, wie viele Bits verwendet werden, um eine Zahl zu speichern. Diese Anzahl von Bits wurde ursprünglich als Wort bezeichnet. Erst seit der Einführung des Bytes als Standardmaß für die Länge einer Speichereinheit werden Zahlen standardisiert in Vielfachen von 8 Bit gespeichert.

byte adressiert werden, eine heutzutage schier unglaubliche Größe. allerdings galten bei Einführung der Intel-Architektur in den 1970er Jahren auch Speichergrößen von einem Megabyte noch als rein utopisch.

Bei der Speicherung von Schriftzeichen wird ebenfalls jedem Zeichen eine natürliche Zahl zugeordnet. Der erste Standard dafür war der 1968 eingeführte ASCII-Code (American Standard Code for Information Interchange), der eine 7-Bit-Kodierung verwendet hat. Diese 128 verschiedenen Zahlen reichten aus, um die 26 Buchstaben unseres Alphabets in Klein- und Großschrift, die Ziffern unseres Zahlsystems, die wichtigsten Sonderzeichen sowie einige Steuerzeichen (z.B. Zeilenumbruch) zu kodieren. Dieser Code wurde später u.a. durch die ISO Latin-1- und Latin-9-Codes auf 8 Bit erweitert, womit auch die Sonderzeichen der meisten westeuropäischen Sprachen, wie z.B. die deutschen Umlaute, dargestellt werden können. Mit Einführung des Unicodes wurde versucht, alle Zeichen aller bekannten Schriftkulturen mit einer einheitlichen Kodierung zu versehen. Im aktuellen Unicode 4.0 werden 32-Bit-Zahlen zur Kodierung verwendet, wobei nicht der gesamte mögliche Zahlbereich als Code für Zeichen vorgesehen ist. Insgesamt definiert der Unicode zur Zeit nur 96382 Zeichen.

Zum Schluss dieses Abschnitts möchten wir die MATLAB-Benutzer unter unseren Lesern noch darauf hinweisen, daß in MATLAB mit den Funktionen `unit8(x)`, `unit16(x)` und `unit32(x)` natürliche Zahlen in 8-, 16- bzw. 32-Bit-Darstellung erzeugt werden könnnen. Dies ist jedoch nicht der Standard-Datentyp in MATLAB und wird deshalb nicht von allen arithmetischen Operationen unterstützt. Für spezielle Anwendungen kann es durchaus sinnvoll sein, diese Datentypen explizit zu verwenden, im allgemeinen ist man jedoch besser beraten, in MATLAB den Standarddatentyp für Gleitkomma-Zahlen zu verwenden, den wir in Abschnitt 1.4 besprechen werden.

1.2 Ganze Zahlen

1.2.1 Konstruktion durch Abschluß von $\mathbb{N}$ unter Subtraktion

Seien n, m natürliche Zahlen. Ist $n \leq m$, so hat die Gleichung

$$x + n = m \tag{1.5}$$

die Lösung $x = m - n \in \mathbb{N}$. Wir wollen $\mathbb{N}$ so erweitern, daß (1.5) auch für $n > m$ eine eindeutig bestimmte Lösung hat. Da die Aufgabe (1.5) durch n, $m \in \mathbb{N}$ charakterisiert ist, liegt es nahe, die zusätzlich benötigten Zahlen durch Paare (m, n) zu beschreiben. Jedes Paar (m, n) kann man sich als Pfeil vorstellen, der von n nach m zeigt (siehe Abb. 2). Nun hat $x' + n' = m'$ dieselbe Lösung

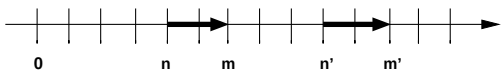


Abbildung 2: Ganze Zahlen als Paare natürlicher Zahlen

$x' = m' - n' \in \mathbb{N}$ wie (1.5), falls $n' \leq m'$ und $m' - n' = m - n$ vorliegt. Das ist gleichbedeutend mit $n' + m = n + m'$. Diese zweite Formulierung hat den Charme, daß sie auch im Falle $n > m$ oder $n' > m'$ sinnvoll ist. Auf der Menge der Paare $(m, n) \in \mathbb{N} \times \mathbb{N}$ definieren wir daher die Äquivalenzrelation

$$(m, n) \cong (m', n') \Longleftrightarrow n + m' = n' + m.$$

Die Menge $\mathbb{Z}$ der ganzen Zahlen besteht nun aus allen Äquivalenzklassen $[m, n]$ dieser Relation. Jeder Äquivalenzklasse $[m, n]$ entspricht ein Pfeil der Länge $|m - n|$, der im Falle $n < m$ nach rechts und im umgekehrten Fall $n > m$ nach links zeigt und dadurch zwei *beliebige* natürliche Zahlen verbindet[9]. Addition und Multiplikation lassen sich von $\mathbb{N}$ auf $\mathbb{Z}$ übertragen [**?**, Kap. 1] . Als Repräsentant von $[m, n]$ wählt man $(m - n, 0)$, falls $n \leq m$ und $(0, n - m)$, falls $n > m$ ist. Statt $(n, 0)$ schreibt man n, und $(0, n)$ wird mit $-n$ abgekürzt.

1.2.2 Zifferndarstellung

Betrachtet man „$-$“ als ein weiteres Element einer Ziffernmenge $\mathcal{Z}$ (welches nur am Anfang einer Ziffernkette auftauchen darf), so folgt:

Satz 1.3 *Jede Zifferndarstellung von* $\mathbb{N}$ *induziert eine Zifferndarstellung von* $\mathbb{Z}$*.*

Identifizieren wir nach wie vor die natürlichen Zahlen mit ihrer Darstellung im Dezimalsystem, so erhalten wir in gewohnter Schreibweise

$$\mathbb{Z} = \{ \ldots,\ -3,\ -2,\ -1,\ 0,\ 1,\ 2,\ 3,\ \ldots \}$$

Offenbar ist das Vorzeichen einer ganzen Zahl $z \in \mathbb{Z}$ eine digitale Größe: Statt „+“ kann man „0“ schreiben, und „-“ kann man durch „1“ ersetzen. Die resultierende Dualdarstellung der ganzen Zahlen ist dann

$$\{ \ldots,\ 111,\ 110,\ 11,\ 0,\ 01,\ 010,\ 011,\ \ldots \} \tag{1.6}$$

Die erste Stelle wird dabei immer als Vorzeichen interpretiert. Man spricht von einem Vorzeichenbit. Aus $+0 = -0 = 0$ folgt übrigens $00 = 10 = 0$. Die Darstellung der Null ist also nicht eindeutig.

1.2.3 Praktische Realisierung

Während natürliche Zahlen im Rechner hauptsächlich zur Adressierung von Speicherplätzen und zur Kodierung von Zeichen verwendet werden, stellen ganze Zahlen üblicherweise den Grunddatentyp für arithmetische Zwecke dar. Genauso wie bei der Speicherung natürlicher Zahlen steht zur Darstellung ganzer Zahlen im Rechner eine vorgegebene Anzahl von N Bits zur Verfügung, denn bei der Speicherung wird nicht zwischen beiden Zahlentypen unterschieden. Bei der konkreten Verwendung in Programmiersprachen kann jedoch zwischen natürlichen Zahlen und vorzeichenbehafteten ganzen Zahlen unterschieden werden. Dies geschieht oftmals durch einen Zusatz in der Typbezeichnung wie unsigned/signed für natürliche bzw. ganze Zahlen.

Haben nun alle gespeicherten Zahlen die gleiche, feste Anzahl an Stellen, so ist die Darstellung negativer Zahlen im sogenannten Zweierkomplement praktischer als die Vorzeichendarstellung (1.6). Dabei werden gerade die Komplemente der Koeffizienten der Potenzzerlegung verwendet. Genauer bezeichnet die Darstellung im Dualsystem mit Zweierkomplement die Kodierung

$$\boxed{1 \,\big|\, z_{N-2}\ \cdots\ z_0} \mathrel{\hat{=}} -\left(1 + \sum_{i=0}^{N-2} (1 - z_i)\ 2^i\right) .$$

[9] Solche Äquivalenzklassen kennen wir aus der Vektorrechnung.

Eine gegebene, negative Dezimalzahl kodiert man demnach durch a) Dualdarstellung, b) Umklappen der Bits und c) Addition von 1. Beispielweise ist für $N = 4$:

$$\underbrace{-3}_{\text{Dezimalzahl}} \rightarrow \underbrace{-0011}_{\text{Dualdarstellung}} \rightarrow \underbrace{-1100}_{\text{umklappen}} \rightarrow \underbrace{1101}_{\text{Zweierkomplement}} .$$

Der erste Vorteil des Zweierkomplements ist, daß sich in dieser Darstellung die Subtraktion von z auf die Addition von $-z$ zurückführen lässt. Die Aufgabe $3 - 3$ im 4-Bit-Zweierkomplement lautet somit:

$$\begin{array}{r} \boxed{0 \mid 0\,1\,1} \\ +\boxed{1 \mid 1\,0\,1} \\ \hline \boxed{0 \mid 0\,0\,0} \end{array}$$

Dabei wird der Übertrag des Vorzeichenbits einfach ignoriert. Der zweite Vorteil des Zweierkomplements ist die eindeutige Darstellung der 0. Im Dualsystem mit Zweierkomplement kann man daher mit N Bits alle ganzen Zahlen $z \in \mathbb{Z}$ mit der Eigenschaft

$$z_{\min} = -2^{N-1} \leq z \leq 2^{N-1} - 1 = z_{\max}$$

darstellen.

Das Zweierkomplement zur Darstellung negativer Zahlen ist der Standard bei der Speicherung ganzer Zahlen im Computer. Die so dargestellten Zahlen werden auch Integerzahlen genannt.

Ein Problem bei dieser Einschränkung des Zahlenbereichs tritt auf, wenn eine Addition oder Multiplikation zweier Zahlen aus dem darstellbaren Bereich herausführt, also ein sogenannter Underflow oder Overflow auftritt. Besitzen die Integerzahlen z.B. eine Größe von 8 Bit, so können Zahlen von -128 bis 127 dargestellt werden. Führt man nun die Berechnung $127 + 1$ aus, so ergibt sich 128, eine Zahl, die nicht mehr als 8-Bit-Zahl dargestellt werden kann. Stattdessen erhalten wir das Ergebnis -128. Diese anscheinend falsche Addition erklärt sich, wenn wir beachten, daß die Additionsaufgabe in binärer Schreibweise wie folgt aussieht:

$$\begin{array}{r} \boxed{0 \mid 1\,1\,1\,1\,1\,1\,1} \\ +\boxed{0 \mid 0\,0\,0\,0\,0\,0\,1} \\ \hline \boxed{1 \mid 0\,0\,0\,0\,0\,0\,0} \end{array}$$

Dabei wird korrekterweise der Übertrag der höchstwertigen Ziffernbits zu der Summe der Vorzeichenbits addiert. Die so entstandene 1 des führenden Bits wird nun aber fälschlich als negatives Vorzeichen interpretiert. Das Ergebnis der Addition entspricht einem Zurückführen der Summe in den darstellbaren Bereich -128 bis 127. Mathematisch ausgedrückt liefert die Addition oder Multiplikation zweier N-Bit-Integerzahlen a und b im Rechner als Ergebnis $(a + b) \mod 2^N$ bzw. $(a \cdot b) \mod 2^N$, wobei bei Ergebnissen, die größer als $2^N/2 - 1$ sind, schließlich noch 2^N subtrahiert wird[10].

[10] Die genaue Formel für Addition und Multiplikation lautet $[(a + b + 2^N/2) \mod 2^N] - 2^N/2$ bzw. $[(a \cdot b + 2^N/2) \mod 2^N] - 2^N/2$

Dies ist eine nicht zu verachtende Fehlerquelle in vielen Programmen. Je nach Rechnerarchitektur, Betriebssystem und verwendetem Programm kann es deshalb auch sein, daß die Berechnung mit einer Fehlermeldung (bzw. einem sogenannten Signal oder einer Exception) abgebrochen wird.

Als Ausgleich für diese in Kauf zu nehmende Einschränkung sind die Recheneinheiten der Prozessoren jedoch so gebaut, daß sie mit Integerzahlen sehr effizient rechnen können. Da für alle darstellbaren Zahlen die Anzahl der zur Berechnung relevanten Bits gleich groß ist, bleibt auch die Zeit, die der Prozessor benötigt, um zwei Zahlen zu addieren (oder zu multiplizieren), unabhängig vom Wert der Zahlen.

Wie bereits im Abschnitt über natürliche Zahlen erwähnt, gibt es keine Standards für die Anzahl der Bits bei der Verwendung von Integerzahlen, jedoch werden schon seit geraumer Zeit Integerzahlen nur noch mit einer Länge eines Vielfachen von 8 Bit gespeichert, welches einer Zweierpotenz entspricht, also mit 8, 16, 32, 64 oder 128 Bit. Dies war in der Vergangenheit durchaus anders. Diese Länge entspricht der Wortlänge des Computers, also der Anzahl der Bits, mit der auch die Speicheradressierung vorgenommen wird. Bei aktuellen Rechnern stehen Integerzahlen mit 32 oder 64 Bit Länge zur Verfügung. So können z.B. mit Prozessoren der x86- Familie wie dem Intel Pentium IV oder dem AMD Athlon XP Integerzahlen im Bereich von

$$z_{\min} = -2^{32-1} \approx -2.1 \cdot 10^9 \ , \qquad z_{\max} = 2^{32-1} - 1 \approx 2.1 \cdot 10^9 \ .$$

dargestellt werden.

Erwähnt sei noch, daß die meisten Programmiersprachen neben der von der Maschinenarchitektur gelieferten Länge für Integerzahlen noch weitere ganzzahlige Datentypen anbieten. Diese werden genauso wie eben beschrieben dargestellt, verwenden dann jedoch abweichende Bitzahlen und werden dann meist mit `word`, `short integer` oder `long integer` bezeichnet. Interessanterweise schreibt selbst der Ansi-Standard für die Programmiersprache C nicht vor, wie viele Bits welcher ganzzahlige Datentyp verwendet, sondern gibt nur Hierarchien der ganzzahligen Datentypen bezüglich ihrer Bitlängen vor, so daß die tatsächliche Umsetzung der Datentypen weiterhin maschinenabhängig erfolgen kann. Im Gegensatz dazu gibt die *Java Language Specification* die genaue Kodierungslänge der verschiedenen ganzzahligen Datentypen für alle *Java Virtual Maschines* konkret vor. Die folgende Tabelle 1 gibt eine Übersicht über die verschiedenen ganzzahligen Datentypen dieser beiden Sprachen.

In MATLAB ist es zwar durchaus möglich, ganzzahlige Datentypen der Länge von 8, 16 oder 32 Bit mittels der Funktionen `int8(x)`, `int16(x)` und `int32(x)` zu erzeugen, aber ebenso wie die entsprechenden Datentypen für natürliche Zahlen ist dies nicht der MATLAB-Standard und wird somit nicht von allen MATLAB-Funkionen unterstützt.

Ganze Zahlen variabler Länge

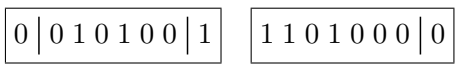
Bei der Verwendung von Computerzahlen fester Bitlänge steht man, wie eben beschrieben, vor dem Problem, daß der Bereich der darstellbaren Zahlen sehr stark eingeschränkt ist. Stellt man nur wenige Bits zur Speicherung einer Zahl zur Verfügung, so kann man durch mehrfache Anwendung von Rechenoperationen sehr schnell in Zahlenbereiche vordringen, die nicht mehr dargestellt werden können und somit zu einem Overflow mir den angesprochenen Konsequenzen führen. Stellt man andererseits ganze Zahlen mit einer sehr großen Speicherlänge dar, so bleiben oftmals viele der Bits ungenutzt, d.h. die Zahlen haben sehr viele führende Nullen. Arbeitet man insbesondere mit

Datentyp	**Speicherlänge**	$z_{\min}$	$z_{\max}$
C/C++ (angegeben ist jeweils die übliche Länge für Intel-32-Bit-Architekturen)			
char	1	-128	127
unsigned char	1	0	255
short int	2	-32.768	32.767
unsigned short	2	0	65535
int	4	-2.147.483.648	2.147.483.647
unsigned int	4	0	4.294.967.295
long	4	-2.147.483.648	2.147.483.647
unsigned long	4	0	4.294.967.295
Java			
byte	1 Byte	-128	127
short	2 Byte	-32768	32767
char	2 Byte	0	65535
int	4 Byte	-2147483648	2147483647
long	8 Byte	-9223372036854775808	9223372036854775807
char	2 Byte	0	65535

Tabelle 1: Übersicht über ganzzahlige Datentypen

sehr vielen sehr kleinen Zahlen, so wird der größte Teil des Speichers zur Speicherung von Nullen verschwendet. Insbesondere in den Zeiten, als Computerspeicher noch ein teures Gut war, konnte man es sich deshalb nicht erlauben, Zahlen in ausreichender Länge zu kodieren. Wünschenswert wäre es also, ganze Zahlen in genau der Größe zu speichern, die für die aktuelle Anwendung am besten ist, also so, daß die größte auftretende Zahl gerade noch dargestellt werden kann. Leider weiß man im allgemeinen nicht vorher, welche Zahlengrößen in der aktuellen Anwendung auftreten werden, deshalb kann man die optimale Speichergröße nicht a priori wählen.

In Computer-Algebra-Systemen wird deshalb oftmals ein Ansatz gewählt, bei dem ganze Zahlen mittels flexibler Datentypen gespeichert werden, z.B. in so genannten Byte-Listen. Man beginnt die Darstellung der Zahl mit einem Byte, dessen erstes Bit als Vorzeichenbit dient. Nun werden die sechs niedrigwertigsten Bits der darzustellenden Zahl in den nächsten sechs Bits dieses Bytes gespeichert. Das letzte Bit des Bytes wird verwendet, um zu entscheiden, ob die Zahl damit vollständig dargestellt wurde (0) oder ob weitere Stellen zur Darstellung der Zahl benötigt werden (1). Werden weitere Stellen benötigt, so gehört das folgende Byte ebenfalls zu der Zahl, und es können nun die nächsten sieben Stellen der Zahl in den ersten Bits des aktuellen Bytes gespeichert werden. Das letzte Bit entscheidet wiederum, ob die Zahl fertig ist oder weiter geht etc. Auf diese Art würde die Zahl 714, deren Binär-Darstellung 1011001010_2 ist, mit den folgenden beiden Bytes dargestellt:



Bei dieser Darstellung ist zu beachten, daß die einzelnen Ziffern der Dualdarstellung in umgekehrter Reihenfolge gespeichert werden. diese Speichermethode nennt man auch ,,Little Endian". Dies entspricht im übrigen auch der Reihenfolge, in der die einzelnen Bytes bei der Darstellung ganzer Zahlen auf verschiedenen Rechnerarchitekturen, darunter auch der Intel-x86-Architektur, gespeichert werden. Das der uns gewohnten Lesereihenfolge entsprechende ,,Big Endian"-Format fand

dagegen unter anderem in den Motorola 68000er Prozessoren Verwendung, mit denen vor allem in den 80er Jahren ganze Generationen von Heimcomputern wie der Apple McIntosh, der Atari ST oder der Commodore Amiga bestückt waren.

Eine solche variable Speicherung von Zahlen umgeht die Probleme des beschränkten Wertebereichs bzw. der Speichervergeudung, die man bei der Verwendung der maschinenabhängigen Datentypen hat. Erkauft wird diese Flexibilität mit dem Nachteil, daß bereits die Standardrechenoperationen wie Addition und Multiplikation nicht mehr mit dem einfachen Befehlssatz des Prozessors ausgeführt werden können und damit nicht mehr unabhängig von der Größe der Zahlen eine feste Anzahl an Rechnertakten benötigen, sondern die Zeit zur Durchführung dieser Rechenoperationen nun von der Kodierungslänge der Zahlen also auch von ihrem Wert abhängen. Kleine Zahlen können schnell addiert und multipliziert werden, große Zahlen benötigen mehr Zeit.

1.3 Rationale Zahlen

1.3.1 Konstruktion durch Abschluß von $\mathbb{Z}$ unter Division

Die Konstruktion der rationalen Zahlen erfolgt in völliger Analogie zum Vorgehen in Abschnitt 1.2.1. Ausgangspunkt ist nun die Tatsache, daß die Gleichung

$$b \cdot x = a \tag{1.7}$$

mit a, $b \in \mathbb{Z}$ im allgemeinen keine Lösung in $x \in \mathbb{Z}$ hat. Bei dem Versuch, die Lösung von (1.7) durch das Paar (a, b) zu charakterisieren, muß man dieses Mal beachten, daß die Gleichung $b'x = a'$ mit $b' \neq 0$ dieselbe Lösung hat, falls $a'b = b'a$ ist. Auf der Menge aller Paare $(a, b) \in \mathbb{Z} \times \mathbb{Z}$ mit $b \neq 0$ definieren wir daher die Äquivalenzrelation

$$(a, b) \cong (a', b') \Longleftrightarrow a'b = b'a.$$

Die Menge $\mathbb{Q}$ der rationalen Zahlen besteht nun aus allen Äquivalenzklassen $[a, b]$ dieser Relation. Anstelle von Paaren sprechen wir von Brüchen, und wir sind es gewohnt, $\frac{a}{b}$ anstelle von (a, b) zu schreiben. Die Zugehörigkeit verschiedener Repräsentanten $\frac{a'}{b'}$, $\frac{a}{b}$ zu einer Äquivalenzklasse $[a, b]$ bedeutet, daß $\frac{a'}{b'}$ durch Erweitern oder Kürzen in $\frac{a}{b}$ überführt werden kann[11]. Als Repräsentant wird meist $\frac{a'}{b'} \in [a, b]$ mit teilerfremden Zahlen a', b' und $b' > 0$ verwendet, also beispielsweise

$$\frac{1}{3} \in [12345, 37035]. \tag{1.8}$$

Schließlich werden noch die Abkürzungen $a = \frac{a}{1}$ und $b^{-1} = \frac{1}{b}$ eingeführt. Die Erweiterung der Addition und Multiplikation von $\mathbb{Z}$ auf $\mathbb{Q}$ erfolgt durch die bekannten Bruchrechenregeln

$$\frac{a}{b} + \frac{a'}{b'} = \frac{ab' + a'b}{bb'}\,, \qquad \frac{a}{b} \cdot \frac{a'}{b'} = \frac{aa'}{bb'}\,. \tag{1.9}$$

Kommutativität, Assoziativität und das Distributionsgesetz bleiben dabei erhalten.

[11] Hätten wir übrigens den Nenner $b = 0$ nicht von vornherein ausgeschlossen, so wäre $\frac{a}{b} = \frac{0 \cdot a}{0 \cdot b} = \frac{0}{0}$ für alle $a, b \in \mathbb{Z}$ und daher [0,0] die einzige rationale Zahl.

1.3.2 Zifferndarstellung

Betrachtet man den Bruchstrich „/“ als ein weiteres Element einer Ziffernmenge $\mathcal{Z}$ (welches in jeder Kette genau einmal und weder am Anfang noch am Ende auftauchen darf), so erhalten wir mit Blick auf Satz 1.3:

Satz 1.4 *Jede Zifferndarstellung von* $\mathbb{N}$ *induziert eine Zifferndarstellung von* $\mathbb{Q}$.

Von der entsprechenden Darstellung mit Hilfe des Dezimalsystems haben wir in (1.8) schon Gebrauch gemacht.

1.3.3 Dezimal- und Dualbrüche

In Erweiterung der Dezimaldarstellung natürlicher Zahlen definieren wir die Dezimalbrüche

$$z_n \cdots z_0,\ z_{-1} \cdots z_{-m} \ = \ \sum_{i=-m}^{n} z_i\, 10^i \ , \qquad z_i \in 0,\ldots,9, \quad n,m \in \mathbb{N}\,. \tag{1.10}$$

Vorhandene Methoden zur Addition und Multiplikation von Dezimalzahlen lassen sich im wesentlichen auf Dezimalbrüche übertragen (man muß nur auf das Komma achten). Im Vergleich zu der komplizierten Addition von Brüchen ist das ein großer Vorteil.

Offenbar ist jeder Dezimalbruch ein Bruch, denn Erweitern mit 10^m liefert

$$\sum_{i=1}^{m} z_{-i}\, 10^{-i} = \frac{\sum_{i=1}^{m} z_{-i}\, 10^{m-i}}{10^m}\ .$$

Die Umkehrung ist nicht richtig. Beispielsweise liefert sukzessive Division die Darstellung

$$\frac{1}{3} = 0,333...\ .$$

Dies ist zwar ein Ausdruck der Form (1.10) allerdings mit $m = \infty$. Es handelt sich um eine spezielle geometrische Reihe, denn es gilt ja

$$0,333... \ = \ 0,3\ (\ 1 + 10^{-1} + 10^{-2} + \cdots\ ) \ = \ 0,3\ \sum_{i=0}^{\infty} 10^{-i}\ .$$

Den Grenzwert der geometrischen Reihe erhält man für beliebige $q > 1$ aus

$$\sum_{i=0}^{\infty} q^{-i} = \lim_{m\to\infty} \sum_{i=0}^{m} q^{-i} = \lim_{m\to\infty} \frac{1 - q^{-(m+1)}}{1 - q^{-1}} = \frac{1}{1 - q^{-1}}\ .$$

Mit Hilfe der geometrischen Reihe kann man nun periodische Dezimalbrüche mit beliebiger Periodenlänge einführen. Beipielsweise ist

$$0,123123123... \ = \ 0,123(1 + 10^{-3} + 10^{-6} + \cdots) = 0,123\ \sum_{i=0}^{\infty} 10^{-3i} = 0,123\ \frac{1}{1 - 10^{-3}} = \frac{123}{999}$$

ein periodischer Dezimalbruch mit Periodenlänge 3. Wir schreiben kurz

$$0,\overline{123} = 0,123123123... \ .$$

Jeder periodische Dezimalbruch ist also eine rationale Zahl. Umgekehrt lässt sich auch jede rationale Zahl als periodischer Dezimalbruch darstellen. Der Grund liegt darin, daß bei Durchführung des Euklidischen Algorithmus nur endlich viele verschiedene Reste r auftreten können und sich bei Wiederholung eines Restes auch die vorangegangene Ziffernfolge wiederholt. Im Falle $\frac{1}{3}$ tritt z.B. nur der Rest $r = 1$ auf. Im folgenden Beispiel erhalten wir alle möglichen von Null verschiedenen Reste $r = 1, 3, 2, 6, 4, 5$ bis sich $r = 1$ wiederholt

$$\frac{1}{7} = 0.\overline{142857} \ .$$

Bleibt kein Rest, so haben wir es mit einem endlichen Dezimalbruch zu tun. In diesem Fall könnte man von der Periode $\overline{0}$ sprechen. Aber mit gleichem Recht auch von der Periode $\overline{9}$, denn es gilt

$$0,\overline{9} = 0,9 \sum_{i=0}^{\infty} 10^{-i} = 0,9 \frac{1}{1-10^{-1}} = 1,\overline{0} \ .$$

Um Eindeutigkeit zu gewährleisten, wird die Periode $\overline{0}$ zugunsten von $\overline{9}$ verboten. Die Darstellung von 1 als periodischer Dezimalbruch ist also $0,\overline{9}$. Wir fassen zusammen:

Satz 1.5 *Jede rationale Zahl lässt sich in eindeutiger Weise als periodischer Dezimalbruch darstellen.*

Man beachte, daß eine gegebene rationale Zahl x, ob als Bruch oder als periodischer Dezimalbruch geschrieben, immer durch endlich viele Ziffern charakterisiert ist.

Aus alter Gewohnheit haben wir uns auf das Dezimalsystem bezogen, obwohl wir auch ein Positionssystem zu jeder anderen Basis $q > 1$ hätten wählen können. Für $q \in \mathbb{N}$, $q > 1$ kann man nämlich in direkter Analogie zu (1.10) die q-adischen Brüche

$$z_n \ \cdots \ z_0, \ z_{-1} \ \cdots \ z_{-m} \ = \ \sum_{i=-m}^{n} z_i \, q^i \ , \qquad z_i \in \{0, \ 1, \ldots, q-1\}, \quad n, m \in \mathbb{N} \ , \tag{1.11}$$

definieren. Satz 1.5 gilt in gleicher Weise für q-adische Brüche. Man sollte allerdings nie vergessen, daß ein endlicher q_1-adischer Bruch im allgemeinen kein endlicher q_2-adischer Bruch ist. Beispielsweise gilt nämlich

$$0,1_{10} \ = \ 0,0\overline{0011}_2 \ . \tag{1.12}$$

Bei der Umrechnung von Dezimalbrüchen in Dualbrüche ist also Vorsicht geboten.

1.3.4 Praktische Realisierung

Entsprechend der in Abschnitt 1.3.1 vorgestellten Definition von Bruchzahlen als Äquivalenzklassen von Zahlenpaaren lassen sich rationale Zahlen im Rechner in Form zweier Integerzahlen speichern. Die Einschränkungen des Zahlenbereichs von Integerzahlen implizieren natürlich auch eine Einschränkung des entsprechenden Zahlenbereichs der so dargestellten Bruchzahlen. Anders als bei

Integerzahlen gibt es bei so dargestellten rationalen Zahlen nicht einmal ein $z_{\min}$ und $z_{\max}$, für die gilt, daß alle rationalen Zahlen q mit $z_{\min} < q < z_{\max}$ als Paar von Integerzahlen dargestellt werden können. Verwendet man N-Bit-Integerzahlen, so kann schon der Bruch $\frac{1}{2^N+1}$ nicht dargestellt werden.

Die Darstellung von rationalen Zahlen hat sich aus diesem und aus weiteren Gründen als nicht praktikabel erwiesen, so daß es in üblichen Computern keine hardware-seitige Unterstützung von rationalen Zahlen in entsprechender Form gibt. Stattdessen verwendet man die wesentlich leichter zu handhabenden Gleitkommazahlen, die in Abschnitt 1.4.4 vorgestellt werden.

Insbesondere für spezielle mathematische Softwarepakete ist die exakte Darstellbarkeit von Bruchzahlen jedoch entscheidend, so zum Beispiel in Computer-Algebra-Systemen wie MATHEMATICA oder MAPLE. Dort werden mittels entsprechender Softwarelösungen Brüche ebenfalls als Paare ganzer Zahlen dargestellt, wobei dann wiederum die in Abschnitt 1.2.3 erwähnten Zahlen variabler Länge verwendet werden. Einige der dabei auftretenden Probleme behandeln wir später im zweiten Teil.

Vorweg wollen wir nur zusammenfassend feststellen, daß sich das Rechnen mit Bruchzahlen als wesentlich aufwendiger gestaltet als das Rechnen mit anderen im Computer dargestellten Datentypen, so wie uns auch unsere alltägliche Erfahrung sagt, daß die Bruchrechnung wesentlich aufwendiger ist als das Rechnen mit Dezimalbrüchen. Deshalb wollen wir im Folgenden darlegen, wie Dezimalbrüche im Computer dargestellt werden.

1.4 Reelle Zahlen

1.4.1 Konstruktion durch Vervollständigung von $\mathbb{Q}$

„So wie eine unendliche Zahl keine Zahl ist, so ist eine irrationale Zahl keine wahre Zahl, weil sie sozusagen unter einem Nebel der Unendlichkeit verborgen ist“, schreibt Stifel im Jahre 1544 in seiner Arithmetica integra [**?**, Buch II, Kap. 1]. Das sind überraschend große Worte, wo doch rationale wie irrationale Zahlen als Punkte auf der Zahlengeraden anscheinend mit bloßen Augen zu sehen sind. Doch Anschauung allein kann trügen. Die aller damaliger Anschauung widersprechende Entdeckung gewisser *inkommensurabler Streckenverhältnisse*, also irrationaler Zahlen, durch Hippasus von Metapont im 5. vorchristlichen Jahrhundert hat einen politischen Skandal ausgelöst, der mit dem Aufstand gegen die bis 445 ganz Unteritalien beherrschenden Pythagoräer in Zusammenhang gebracht wird. Noch für B. Bolzano (1781 - 1848), Professor für Philosophie und Religion in Prag und ein Wegbereiter der modernen Analysis, bedeutete die Begründung der reellen Zahlen eine Annäherung an Gott. Die präzisen Begriffsbildungen des 19. Jahrhunderts verraten dagegen wenig von den Weltanschauungskrisen früherer Jahre. Wir skizzieren im folgenden beispielhaft den Zugang von G. Cantor (1845 - 1918) und C. Meray (1835 - 1911) zur Konstruktion von $\mathbb{R}$ durch Vervollständigung der rationalen Zahlen. Für eine ausführliche Darstellung und umfangreiche historische Bemerkungen verweisen wir wieder auf Ebbinghaus et al. [**?**, Kap. 2].

Wir beginnen zunächst wieder mit der Invertierbarkeit von Rechenoperationen. Nach Summen und Produkten in den Abschnitten 1.2 und 1.3 sind nun Potenzen an der Reihe, also Aufgaben der Form

$$x^n - m = 0 \tag{1.13}$$

mit $n, m \in \mathbb{N}$ und $n, m \geq 2$. Bekanntlich hat (1.13) schon für $n = m = 2$ keine Lösung $x \in \mathbb{Q}$ (vgl.

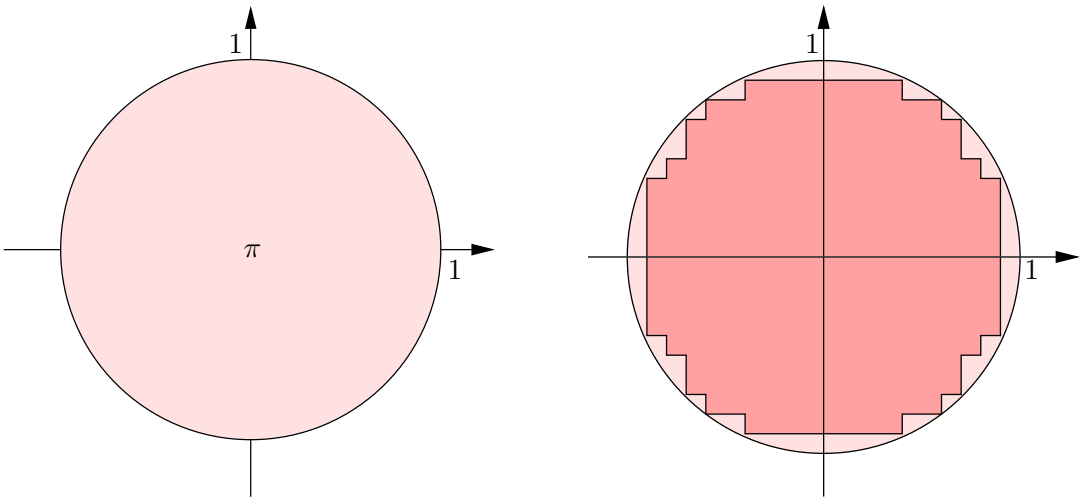


Abbildung 3: Auf dem Weg zu π

z.B. Courant und Robbins [**?**, Kap. 2, § 2]). Analog zu den Abschnitten 1.2 und 1.3 können wir die Lösung von (1.13) durch Paare (n, m) beschreiben, Äquivalenzklassen bilden und so weiter. Aber das reicht nicht. Selbst wenn wir anstelle von $x^n - m$ die Nullstellen beliebiger Polynome

$$\sum_{i=1}^{n} a_i x^i = 0$$

mit ganzzahligen Koeffizienten a_i, die sogenannten *algebraischen Zahlen*,ins Spiel bringen, so bleiben auf der Zahlengeraden immer noch Löcher übrig.

Wir betrachten dazu eine klassische Aufgabe der Mathematik, die Quadratur des Einheitskreises. Dabei ist die Frage, wie man ein Quadrat konstruieren kann, dessen Fläche genauso groß ist wie die des Einheitskreises.[12] Aus der Schule wissen wir, wie groß diese Fläche ist: π. Aber was ist π? Zunächst ist π nur ein Name für den Flächeninhalt des Einheitskreises, dessen genauen Wert wir nicht kennen. Bereits vor viertausend Jahren wurde von den Ägyptern und den Babyloniern versucht, den Wert von π anzugeben. Dabei verwendete man Bruchzahlen, um den Wert zu approximieren. Im Laufe der Jahrhunderte wurden die Approximationen zwar immer besser, aber man versuchte stets weiterhin, π durch Brüche, später auch durch Wurzelterme, anzunähern. Eine kleine Auswahl der Erfolge und Mißerfolge auf diesem Weg ist in Tabelle 2 dargestellt. Seit der gefeierten Arbeit von Lindemann [**?**] aus dem Jahre 1882 wissen wir jedoch, daß π keine algebraische Zahl, also erst recht nicht rational sein kann[13]. π ist vorläufig eines der oben angesprochenen „Löcher“. Eine Möglichkeit, π auf die Spur zu kommen, besteht darin, die Fläche des Einheitskreises zu approximieren. Das kann man wiederum auf vielfältige Weise tun. Eine einfache Möglichkeit besteht darin, die Ebene mit einem Gitternetz aus Quadraten mit Kantenlänge $h_n = 1/10^n$, $n \in \mathbb{N}$, zu überziehen und jeweils die Summe $x_n \in \mathbb{Q}$ der Flächen all solcher Gitterzellen zu bilden, die ganz im Einheitskreis liegen (vgl. Abb 3). Eine „kleine Nebenrechnung“ ergibt

$$x_0 = 0,\ x_1 = 2,68,\ x_2 = 3,1,\ x_3 = 3,137524,\ x_4 = 3,14119020,\ x_5 = 3,1415525416,\ldots \quad (1.14)$$

[12] Über viele Jahrhunderte haben Mathematiker versucht, ein solches Quadrat allein mit den geometrischen Grundkonstruktionen mittels Zirkel und Lineal zu konstruieren. Heute wissen wir, daß dies wegen der Transzendenz von π nicht möglich ist.

[13] C.L.F. von Lindemann (1852 - 1939) war der Lehrer des Königsbergers David Hilbert (1862 - 1943), eines der bedeutendsten Mathematiker aller Zeiten.

Die Differenz $|x_m - x_n|$ ist gerade der durch Verfeinerung der Schrittweite bewirkte Zuwachs der einbeschriebenen Fläche. Der wird beliebig klein, falls n, m genügend groß sind, also gilt das auch für $|x_m - x_n|$. Folgen (x_n) mit dieser Eigenschaft nennt man *Cauchy-Folgen* oder *Fundamentalfolgen*. Offenbar kommt x_n für genügend große n dem Flächeninhalt namens π beliebig nahe. π selbst ist aber keine rationale Zahl. Die Cauchy-Folge $(x_n) \subset \mathbb{Q}$ hat also keinen Grenzwert in $\mathbb{Q}$. Ziel ist es, $\mathbb{Q}$ zu *vervollständigen*, d.h. so zu erweitern, daß alle Cauchy-Folgen, insbesondere also auch (x_n), einen Grenzwert aus dieser erweiterten Menge haben. Am einfachsten wäre es, π direkt mit der Folge (x_n) zu identifizieren. Allerdings gibt es noch beliebig viele andere Folgen rationaler Zahlen, welche π beliebig genau approximieren, beispielsweise die Folge

$$d_0 = 3,\ d_1 = 3,1,\ d_2 = 3,14,\ d_3 = 3,141,\ d_4 = 3,1415,\ d_5 = 3,14159\ldots\ .$$

Ein ähnliches Problem tauchte auch schon bei der Konstruktion von $\mathbb{Z}$ und $\mathbb{Q}$ auf, und wir lösen es in bewährter Weise. Offenbar wird in diesem Fall $|x_n - d_n|$ beliebig klein, falls n genügend groß gewählt ist. Je zwei Cauchy-Folgen (x_n), (d_n) mit dieser Eigenschaft nennen wir äquivalent und schreiben $(x_n) \cong (d_n)$. Die Menge $\mathbb{R}$ der reellen Zahlen besteht nun aus allen Äquivalenzklassen $x = [(x_n)]$ von Cauchy-Folgen $(x_n) \subset \mathbb{Q}$ bezüglich dieser Relation.

Die reellen Zahlen *sind* also Grenzprozesse. Addition, Multiplikation, Anordnung und die entsprechenden Rechenregeln vererben sich von den Folgengliedern. Als Vertreter jeder Äquivalenzklasse $x \geq 0$ wählen wir die Folge endlicher Dezimalbrüche

$$d_0 = z_n \cdots z_0,\ \ d_1 = z_n \cdots z_0, z_{-1},\ \ d_2 = z_n \cdots z_0, z_{-1}z_{-2},\ \ d_3 = z_n \cdots z_0, z_{-1}z_{-2}z_{-3},\ \ldots$$

mit $z_i \in \mathcal{Z} = \{0, 1, \ldots, 9\}$ und schreiben x als unendlichen Dezimalbruch

$$x = z_n \cdots z_0, z_{-1}z_{-2}z_{-3}z_{-4}\ldots\ .$$

In diesem Sinne ist

$$\pi = 3,1415\ldots \in \mathbb{R}\ .$$

Die *Darstellung* der einzelnen Folgenglieder x_n oder d_n spielte bei unseren Überlegungen keine Rolle. Wir hätten anstelle von Dezimalbrüchen genauso gut auch eine andere q-adische Darstellung wählen können. So erhält man beispielsweise im Dualsystem

$$\pi = 11,0010\ 0100\ 0011\ 1111\ldots_2\ . \tag{1.15}$$

1.4.2 Abzählbarkeit und Zifferndarstellung

Die Voraussetzung für numerisches Rechnen mit reellen Zahlen ist deren Darstellung durch ein Ziffernsystem (vgl. die Abschnitte 1.1.1 und 1.1.2). Betrachtet man reelle Zahlen als unendliche Dezimalbrüche, so kommt man zwar mit endlich vielen Ziffern aus, die entsprechenden Ziffernketten sind aber unendlich lang. Die unendlichen Dezimalbrüche sind also noch kein Ziffernsystem im Sinne von Absatz 1.1.2. Wir wollen untersuchen, ob es solch eine Zifferndarstellung der reellen Zahlen überhaupt geben kann.

Definition 1.6 (Abzählbarkeit) *Eine Menge M mit unendlich vielen Elementen heißt abzählbar unendlich oder kurz abzählbar, wenn eine bijektive Abbildung $\varphi : \mathbb{N} \to M$ existiert.*

Wann	**Wer**	**Approximation von π**	**Anzahl der korrekten Nachkommastellen**
ca. 2000 v. Chr.	Babylonier	$3\frac{1}{8}$	1
	Ägypter	$\frac{256}{81}$	1
ca. 1100 v.Chr.	Chinesen	3	0
ca. 550 v. Chr.	Altes Testament	3	0
3. Jhd v.Chr.	Archimedes	$3\frac{10}{71} < \pi < 3\frac{1}{7}$	2–3
2. Jhd. n.Chr.	Ptolemäus	$\frac{377}{120}$	3
3. Jhd. n.Chr.	Wang Fau	$\frac{142}{45}$	1
263 n.Chr.	Liu Hui	$\frac{157}{50} = 3,14$	2
ca. 450	Tsu Ch'ung Chi	$\frac{355}{113}$	6
ca. 530	Aray-Bhata	$\frac{62832}{20000}$	3
ca. 650	Brahmagupta	$\sqrt{10}$	1

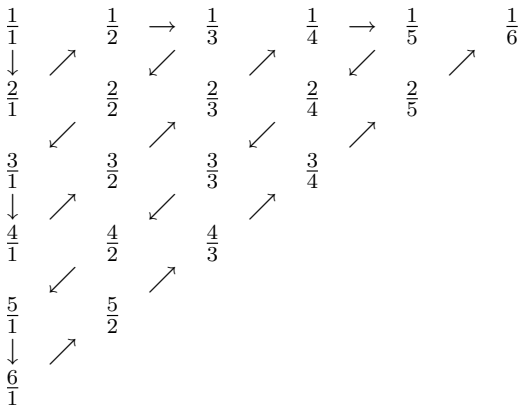
Tabelle 2: Approximation von π im Laufe der Zeit

Anschaulich bedeutet das: Jede abzählbare Menge M lässt sich durchnummerieren, also in der Form

$$M = \{m_0,\ m_1,\ m_2,\ \ldots\}$$

aufschreiben. Trivialerweise ist $\mathbb{N}$ abzählbar. Nach Satz 1.1 ist auch die Menge $\mathcal{D}(\mathcal{Z})$ aller Ziffernketten aus einer endlichen Ziffernmenge $\mathcal{Z}$ abzählbar. Infolgedessen ist eine Menge von Zahlen genau dann abzählbar, wenn sie eine Zifferndarstellung besitzt. Aus Satz 1.3 und Satz 1.4 folgt also die Abzählbarkeit von $\mathbb{Z}$ und $\mathbb{Q}$.

Man kann übrigens auch alle rationalen Zahlen direkt durchnummerieren. Dazu listet man zunächst alle positiven rationalen Zahlen entsprechend dem folgenden Dreiecksschema auf.



Mehrfach auftauchende Zahlen, wie etwa $1 = \frac{2}{2} = \frac{3}{3}, \ldots$, lässt man einfach weg. Dann schreibt man noch 0 an den Anfang und $-\frac{m}{n}$ jeweils direkt hinter $\frac{m}{n}$. So erhält man schließlich die Aufzählung

$$\mathbb{Q} = \left\{0,\ 1,\ -1,\ 2,\ -2,\ \frac{1}{2},\ -\frac{1}{2},\ \frac{1}{3},\ -\frac{1}{3},\ 3,\ -3,\ 4,\ -4,\ \frac{3}{2},\ -\frac{3}{2},\ \ldots\right\}.$$

Diese Methode geht auf G. Cantor (1845 - 1918) zurück. Eine andere Aufzählung von Calkin und Wilf ist bei Aigner und Ziegler [**?**, Kap. 16] beschrieben. Und nun der Hammer:

Satz 1.7 (Überabzählbarkeit von $\mathbb{R}$) *Die Menge $\mathbb{R}$ der reellen Zahlen ist nicht abzählbar.*

Beweis: Wäre $\mathbb{R}$ abzählbar, so hätte auch jede unendliche Teilmenge diese Eigenschaft. Es reicht daher, eine unendliche Teilmenge $M \subset \mathbb{R}$ zu finden, die nicht abzählbar ist. Wir wählen $M = \{x \in \mathbb{R} \mid 0 < x \leq 1\}$. Im Widerspruch zur Behauptung sei M abzählbar, also $M = \{x_0,\ x_1,\ x_2,\ \ldots\}$. Wir schreiben nun jedes Element $x_n \in M$ als unendlichen Dezimalbruch, also beispielsweise $0,\overline{9}$ statt 1. Auf diese Weise erhalten wir das unendliche Schema

$$\begin{array}{ccc} x_0 & = & 0, z_{00}\ z_{01}\ z_{02}\ldots \\ x_1 & = & 0, z_{10}\ z_{11}\ z_{12}\ldots \\ \vdots & \vdots & \vdots \\ x_n & = & 0, z_{n0}\ z_{n1}\ z_{n2}\ldots \\ \vdots & & \vdots \end{array}$$

Zu jedem $n \in \mathbb{N}$ wählen wir ein $a_n \in \{1, \ldots, 8\}$ mit $a_n \neq z_{nn}$ und setzen $a = 0, a_1 a_2 \ldots$. Offenbar ist $a \in M$, also $a = x_n$ für ein geeignetes $n \in \mathbb{N}$. Das kann aber nicht sein, denn nach Konstruktion ist $a_n \neq z_{nn}$ und daher unterscheidet sich a von jedem x_n. Widerspruch. ■

Wir formulieren eine direkte Konsequenz aus Satz 1.7.

Satz 1.8 *Es gibt kein Ziffernsystem zur Darstellung von $\mathbb{R}$.*

Ein möglicher Ausweg wäre die Erweiterung der Ziffernmenge $\mathcal{Z}$ um abzählbar viele zusätzliche Symbole, etwa $\sqrt[n]{m}$ mit geeigneten $n, m \in \mathbb{N}$, die Kreiszahl π, die Eulersche Zahl e und so weiter. Aber auch das hilft nicht. Denn im Falle einer abzählbaren Ziffernmenge ist die Menge $\mathcal{D}_n(\mathcal{Z})$ aller Ziffernketten der Länge n abzählbar und $\mathcal{D}(\mathcal{Z})$ als Vereinigung abzählbar vieler abzählbarer Mengen wieder abzählbar (Übung). Eine überabzählbare Ziffernmenge $\mathcal{Z}$ würde auf ein eigenes Symbol für jede reelle Zahl hinauslaufen. Das ist nicht praktikabel[14]. Es führt kein Weg daran vorbei:

Numerisches Rechnen mit reellen Zahlen ist nicht möglich!

Was tun? Beim numerischen Rechnen mit reellen Zahlen müssen wir uns mit Approximationen aus einer geeigneten Menge $\mathbb{G} \subset \mathbb{R}$ von darstellbaren Zahlen begnügen. Dies liegt in der Natur der

[14] Ein Ausweg besteht darin, den endlich vielen, in einer konkreten Rechnung auftretenden irrationalen Zahlen und den daraus abgeleiteten Größen jeweils lokale, nur in dieser Rechnung gültige Namen zuzuordnen und damit *symbolisch* zu rechnen. Das ist nichts anderes als das altbekannte „Buchstabenrechnen“ aus der Schule. Es gibt mittlerweile hochentwickelte Symbolik-Programmsysteme wie MAPLE, MATHEMATICA, die auf Grundlage faszinierender Verbindungen zwischen Analysis und Algebra weit mehr als nur Bruchrechnung beherrschen. Symbolisches Rechnen stößt allerdings an natürliche Grenzen, sobald die gesuchte Lösung aus prinzipiellen Gründen nicht mehr geschlossen angegeben werden kann. Das ist leider bei einer Vielzahl von Aufgaben die Regel, beispielsweise bei der Lösung von Differentialgleichungen. Wir verfolgen das symbolische Rechnen an dieser Stelle nicht weiter.

reellen Zahlen. Auch kommende Computer-Generationen können daran nichts ändern. Es treten grundsätzlich *Rundungsfehler* auf, mit denen wir uns zu beschäftigen haben.

1.4.3 Absoluter und relativer Fehler

Es sei $\tilde{x}$ eine Approximation von $x \in \mathbb{R}$. Ein natürliches Maß für den Fehler ist der Abstand von x und $\tilde{x}$ auf der Zahlengeraden. Dieser Abstand heißt

$$\underline{\text{absoluter Fehler}}: \qquad |x - \tilde{x}| \; .$$

Zum Beispiel approximiert $\tilde{x} = 3$ die Zahl $x = \pi$ bis auf den absoluten Fehler $|\pi - 3| < 2 \cdot 10^{-1}$.

Der absolute Fehler gibt keinen Aufschluß über die Anzahl der gültigen Stellen von $\tilde{x}$. So führt beispielsweise die Approximation von a) $x = 4$ durch $\tilde{x} = 0$ und von b) $x = 12344$ durch $\tilde{x} = 12340$ auf denselben absoluten Fehler $|x - \tilde{x}| = 4$, obwohl im Fall b) die ersten vier Stellen von x und $\tilde{x}$ übereinstimmen und im Fall a) keine einzige. Sehr verschieden sind jeweils die Größenverhältnisse von approximierter Zahl und absolutem Fehler. Das Verhältnis von absolutem Fehler und $|x|$ heißt

$$\underline{\text{relativer Fehler}}: \qquad \frac{|x - \tilde{x}|}{|x|} \; , \qquad x \neq 0 \; .$$

Im Falle $x = 0$ ist der relative Fehler nicht definiert. Im obigen Beispiel ist

$$\frac{4 - 0}{4} = 10^0, \qquad \frac{12344 - 12340}{12344} \approx 3 \cdot 10^{-4} \; .$$

Bisweilen wird der relative Fehler in Prozent als $100 \cdot |x - \tilde{x}|/|x|\%$ angegeben. In Beispiel a) haben wir offenbar einen Fehler von 100 Prozent, während im Fall b) ein Fehler von weniger als 1 Promille vorliegt.

1.4.4 Gleitkommazahlen und Rundungsfehler

Wir wollen die reellen Zahlen durch *endliche* q-adische Brüche approximieren. Die Anzahl der verfügbaren Stellen bezeichnen wir mit ℓ.

Zunächst gehen wir von einer *festen* Anzahl von n Vor- und m Nachkommastellen aus. Die resultierenden Festkommazahlen haben die Gestalt

$$z_{n-1}\, z_{n-2} \cdots z_0, z_{-1} \cdots z_{-m} = \sum_{i=-m}^{n-1} z_i \, q^i, \qquad z_i \in \{0, \ldots, q-1\} \; .$$

Als Beispiel betrachten wir das Dezimalsystem, also $q = 10$, und $\ell = 4$. Bei Wahl von $n = 3$, $m = 1$ erhalten wir als Approximation von $x = \pi$ durch Runden auf eine Nachkommastelle

$$\tilde{x} = 3,1, \qquad |\pi - \tilde{x}| > 4 \cdot 10^{-2} \; .$$

Hätten wir $n = 1$ und $m = 3$ gewählt, wäre die Approximation durch bessere Ausnutzung der vorhandenen $\ell = 4$ Stellen mit

$$\tilde{x} = 3,142, \qquad |\pi - \tilde{x}| < 5 \cdot 10^{-4}$$

um zwei Größenordnungen besser ausgefallen. Bei Wahl von $x = 123,4$ hätte allerdings alles wieder ganz anders ausgesehen.

Im Sinne einer optimalen Ausnutzung der vorgegebenen ℓ Stellen wollen wir die *Anzahl der Vor- und Nachkommastellen variabel* halten. Dazu führen wir nun die Gleitkommazahlen ein.

Definition 1.9 (Gleitkommazahlen) *Jede in der Form*

$$\tilde{x} = (-1)^s a \cdot q^e \tag{1.16}$$

mit Vorzeichenbit $s \in \{0,1\}$, Exponent $e \in \mathbb{Z}$ und Mantisse

$$a = 0, a_1 \cdots a_\ell = \sum_{i=1}^{\ell} a_i \, q^{-i} \, , \qquad a_i \in \{0, \ldots, q-1\} \, , \; a_1 \neq 0 \, ,$$

oder $a = 0$ darstellbare Zahl $\tilde{x}$ heißt Gleitkommazahl mit Mantissenlänge $\ell \in \mathbb{N}$, $\ell \geq 1$. Die Menge all dieser Zahlen heißt $\mathbb{G}(q,\ell)$. Die Darstellung (1.16) *heißt normalisierte Gleitkommadarstellung.*

Die Menge $\mathbb{G}(10,4)$ besteht aus allen Dezimalbrüchen, bei denen höchstens die $\ell = 4$ führenden Ziffern von Null verschieden sind. Beispielsweise ist

$$0,000001234, \quad 3,142, \quad 123,4, \quad 12340000000 \in \mathbb{G}(10,4) \, .$$

Im Extremfall erhält man

$$\mathbb{G}(2,1) = \{0, \ldots, \, \pm 0,001, \, \pm 0,01, \, \pm 0,1, \, \pm 1, \, \pm 10, \, \pm 100, \, \pm 1000, \, \ldots\} \, .$$

Die Gleitkommazahlen häufen sich in 0 und dünnen mit wachsendem Betrag aus. Genauer gilt

$$|\tilde{x} - \tilde{y}| \geq |\tilde{x}| q^{-\ell} \qquad \forall \tilde{x}, \tilde{y} \in \mathbb{G}(q,\ell) \, . \tag{1.17}$$

Als nächstes wollen wir untersuchen, mit welcher Genauigkeit wir eine *gegebene reelle Zahl x* durch eine Gleitkommazahl $\tilde{x}$ approximieren können. Dazu beschreiben wir das Runden reeller Zahlen. Sei $x \in \mathbb{R}$ und $x > 0$. Wir schreiben x in der Form

$$x = a^* q^e$$

und wählen dabei $e \in \mathbb{Z}$ so, daß $q^{-1} \leq a^* < 1$ gilt. In der q-adischen Darstellung

$$a^* = 0, a_1 \, a_2 \cdots a_\ell \, a_{\ell+1} ... = \sum_{i=1}^{\infty} a_i q^{-i} \, , \qquad a_i \in \{0, \ldots, q-1\} \, , \tag{1.18}$$

ist also $a_1 \neq 0$. Um Eindeutigkeit zu erzielen, schließen wir aus, daß von einem gewissen Index an für alle i durchweg $a_i = q-1$ gilt. Wir wählen also beipielsweise $a^* = 0,124$ anstelle von $0,123\overline{9}$. Runden von a^* auf ℓ Stellen liefert dann

$$a = \sum_{i=1}^{\ell} a_i q^{-i} + \begin{cases} 0 & \text{falls } a_{\ell+1} < \frac{1}{2} q \\ q^{-\ell} & \text{falls } a_{\ell+1} \geq \frac{1}{2} q \end{cases} \, . \tag{1.19}$$

Wir setzen

$$\mathrm{rd}(x) = a q^e \, . \tag{1.20}$$

Nach Konstruktion ist a ein höchstens ℓ-stelliger q-adischer Bruch, also ist $\mathrm{rd}(x) \in \mathbb{G}(q,\ell)$. Im Falle $x < 0$ definieren wir $\mathrm{rd}(x) = -\mathrm{rd}(-x)$ und schließlich wird $\mathrm{rd}(0) = 0$ vereinbart.

Die so erklärte Abbildung $\mathrm{rd} : \mathbb{R} \to \mathbb{G}(q,\ell)$ verallgemeinert das bekannte Runden von Dezimalzahlen. Für $q = 2$ und $\ell = 4$ erhalten wir beispielsweise

$$\mathrm{rd}(101110_2) = 110000_2, \qquad \mathrm{rd}(11,010110_2) = 11,01_2 \ .$$

Wir haben die Abbildung rd gerade so konstruiert, daß für jedes feste $x \in \mathbb{R}$ gilt

$$|x - \mathrm{rd}(x)| \le |x - \tilde{x}| \qquad \forall \tilde{x} \in \mathbb{G}(q,\ell) \ . \tag{1.21}$$

Insbesondere ist $\mathrm{rd} \circ \mathrm{rd} = \mathrm{rd}$ ($\circ$ bedeutet die Hintereinanderausführung). Abbildungen mit dieser Eigenschaft heißen Projektion. Die Rundung rd projeziert also jede Zahl $x \in \mathbb{R}$ auf ihre Bestapproximation $\mathrm{rd}(x)$ aus $\mathbb{G}(q,\ell)$.

Nun wollen wir wissen, ob wir obere Schranken für den Rundungsfehler finden können, die *gleichmäßig* in $x \in \mathbb{R}$ gültig sind. Die Antwort wird davon abhängen, welches Fehlermaß wir verwenden. Für den absoluten Rundungsfehler $|x - rd(x)|$ gibt es eine gleichmäßige Fehlerschranke nicht:

Satz 1.10 *Zu jedem $N \in \mathbb{N}$ gibt es ein $x \in \mathbb{R}$, so daß*

$$|x - \mathrm{rd}(x)| \ge q^N \ .$$

Beweis: Die Behauptung folgt durch Wahl von $x = 0, z_1 \cdots z_\ell \ z_{\ell+1} \cdot q^{\ell+1+N}$ mit $z_1,\ z_{\ell+1} \neq 0$ und beliebigem $N \in \mathbb{N}$. ■

Der absolute Rundungsfehler kann also beliebig groß werden. In $\mathbb{G}(10,4)$ erhält man beispielsweise für $x = 0,10001 \cdot 10^{15}$ den absoluten Rundungsfehler $|x - \mathrm{rd}(x)| = 10^{10}$.

Anders sieht es für den relativen Rundungsfehler $|x - \mathrm{rd}(x)|/|x|$ aus:

Satz 1.11 *Es sei q eine gerade Zahl. Dann gilt*

$$\frac{|x - \mathrm{rd}(x)|}{|x|} \le \frac{q}{2} q^{-\ell} = eps(q,\ell) \qquad \forall x \in \mathbb{R},\ x \neq 0 \ . \tag{1.22}$$

Die Zahl $eps(q,\ell)$ heißt Maschinengenauigkeit.

Beweis: Es sei $x = a^* q^e$ und $\mathrm{rd}(x) = a q^e$ mit $q^{-1} \le a^* < 1$ wie in (1.18) und gerundeter Mantisse a aus (1.19). Wir zeigen

$$|a - a^*| \le \frac{1}{2} q^{-\ell} \tag{1.23}$$

und betrachten dazu zwei Fälle.

Im ersten Fall ist $a_{\ell+1} < \frac{1}{2}q$. Es wird also abgerundet. Aus $\frac{1}{2}q \in \mathbb{N}$ folgt zunächst $a_{\ell+1} + 1 \le \frac{1}{2}q$. Weiter gilt

$$|a^* - a| = \sum_{i=\ell+1}^{\infty} a_i q^{-i} = \left(a_{\ell+1} + \sum_{j=1}^{\infty} a_{j+\ell+1} q^{-j} \right) q^{-(\ell+1)} < (a_{\ell+1} + 1) q^{-(\ell+1)} \le \frac{1}{2} q^{-\ell} \ .$$

Das Kleinerzeichen ist richtig, weil nach Voraussetzung nicht durchweg $a_{j+\ell+1} = q - 1$ sein darf.

Im zweiten Fall ist $a_{\ell+1} \geq \frac{1}{2}q \geq 1$. Es wird also aufgerundet. Nun erhält man

$$|a^* - a| = a - a^* = q^{-\ell} - \sum_{i=\ell+1}^{\infty} a_i q^{-i} = \left(1 - \sum_{j=1}^{\infty} a_{j+\ell} q^{-j}\right) q^{-\ell} \leq \left(1 - a_{\ell+1} q^{-1}\right) q^{-\ell} \leq \left(1 - \frac{1}{2}\right) q^{-\ell} = \frac{1}{2} q^{-\ell} .$$

Damit ist (1.23) bewiesen. Daraus folgt unmittelbar

$$|x - \mathrm{rd}(x)| = |a^* - a| q^e \leq \frac{1}{2} q^{e-\ell} ,$$

und wegen $a_1 \geq 1$ gilt

$$|x| = |a^*| q^e \geq a_1 q^{-1} q^e \geq q^{e-1} .$$

Insgesamt erhält man

$$\frac{|x - \mathrm{rd}(x)|}{|x|} \leq \frac{1}{2} \frac{q^{e-\ell}}{q^{e-1}} = \frac{1}{2} q^{-(\ell-1)} = \mathrm{eps}(q, \ell)$$

und damit die Behauptung. ■

Wir gehen im folgenden davon aus, daß q geradzahlig ist, wie etwa $q = 2$ oder $q = 10$.

Der relative Rundungsfehler ist also *gleichmäßig* in $x \in \mathbb{R}$ durch die Maschinengenauigkeit $eps(q, \ell)$ beschränkt. Gleichbedeutend damit ist

$$\mathrm{rd}(x) \in B(x, eps(q, \ell)) = \{\tilde{x} \in \mathbb{R} \mid \tilde{x} = x(1 + \varepsilon),\ |\varepsilon| \leq eps(q, \ell)\} \qquad \forall x \in \mathbb{R} , \tag{1.24}$$

denn $B(x, \rho)$ ist gerade die Menge aller Approximationen $\tilde{x}$ von x mit maximalem relativem Fehler $\rho \geq 0$.

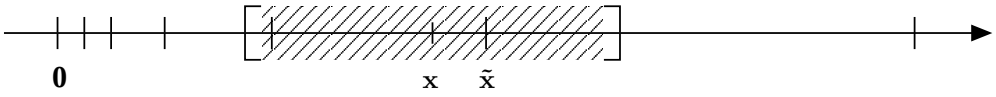


Abbildung 4: Die Menge $B(x, eps(q, \ell))$.

Sofern $\tilde{x} \in B(x, eps(q, \ell))$ vorliegt, sagt man, daß x durch $\tilde{x}$ bis auf ℓ gültige Stellen approximiert wird. Man beachte, daß nur beim Abrunden die führenden ℓ Ziffern dann auch tatsächlich übereinstimmen. So wird im Falle $q = 10$ (Dezimalsystem), $\ell = 4$ und

$$x = 12344,\ \tilde{x} = 12340, \qquad x = 12,995,\ \tilde{x} = 13,00$$

jeweils x durch $\tilde{x}$ bis auf $\ell = 4$ gültige Stellen approximiert.

Nun wechseln wir den Blickwinkel. Beim praktischen Rechnen haben wir nur die Gleitkommazahlen zur Verfügung. Eine *gegebene Gleitkommazahl* $\tilde{x}$ repräsentiert dann alle reellen Zahlen, die auf $\tilde{x}$ gerundet werden, also die Menge

$$R(\tilde{x}) = \{x \in \mathbb{R} \mid \tilde{x} = \mathrm{rd}(x)\} .$$

Beispielsweise ist $R(1) = [1 - q^{-1} eps, 1 + eps)$. Beim Rechnen mit Gleitkommazahlen können wir nicht zwischen verschiedenen $x, y \in R(\tilde{x})$ unterscheiden. Tatsächlich sind im Sinne der Äquivalenzrelation

$$x \cong y \Longleftrightarrow \mathrm{rd}(x) = \mathrm{rd}(y) \tag{1.25}$$

alle Zahlen $x \in R(\tilde{x})$ äquivalent. Wir wollen genau wissen, wie die Äquivalenzklassen $R(\tilde{x})$ aussehen.

Satz 1.12 *Es sei*

$$\tilde{x} = aq^e \in \mathbb{G}(q,\ell), \quad q^{-1} < a_0, a_1 \cdots a_\ell \leq 1 \ .$$

Dann gilt $R(\tilde{x}) = [\ \alpha(\tilde{x}),\ \beta(\tilde{x})\ )$ *mit*

$$\alpha(\tilde{x}) = \tilde{x} - q^{e-1}eps \ , \ \ \beta(\tilde{x}) = \tilde{x} + q^{e-1+a_0}eps \ . \tag{1.26}$$

Beweis: Ist $a < 1$, so wird $x \in \mathbb{R}$ gemäß (1.19) genau dann auf $\tilde{x}$ abgerundet, wenn die Ungleichungen

$$\tilde{x} \leq x < \tilde{x} + \tfrac{1}{2}q^{e-\ell} = 0, a_1 \cdots a_\ell \ a_{\ell+1} \cdot q^e \ , \quad a_1 \neq 0,\ a_{\ell+1} = \tfrac{q}{2} \ ,$$

erfüllt sind. Gilt $a_0 = a = 1$, so ist dies gerade für alle x mit der Eigenschaft

$$\tilde{x} \leq x < \tilde{x} + \tfrac{1}{2}q^{e-(\ell-1)} = 1, 0 \cdots 0 \ a_\ell \cdot q^e \ , \quad a_\ell = \tfrac{q}{2} \ ,$$

der Fall. Nach (1.19) wird x genau dann aufgerundet, wenn

$$\tilde{x} \geq x \geq \tilde{x} - q^{e-\ell} + \tfrac{1}{2}q^{e-\ell} = 0, a_1^* \cdots a_\ell^* \ a_{\ell+1}^* \cdot q^e, \qquad a_1^* \neq 0,\ a_{\ell+1}^* = \tfrac{q}{2} \ ,$$

vorliegt. ■

Wie erwartet, wird die Länge $\beta(\tilde{x}) - \alpha(\tilde{x}) \geq q^{e-\ell}$ des Intervalls $R(\tilde{x})$ mit wachsendem $|x|$ immer größer. Außerdem bemerken wir

$$B(\tilde{x}, q^{-1}eps) \subset R(\tilde{x}) \subset B(\tilde{x}, q\ eps) \ .$$

Zum Schluß wollen wir feststellen, ob und wie die Äquivalenz zweier reeller Zahlen x, y im Sinne von (1.25) mit ihrem Abstand zusammenhängt.

Satz 1.13 *Es gilt*

$$\mathrm{rd}(x) = \mathrm{rd}(y) \Longrightarrow |x - y| \leq (|x| + |y|)eps \ .$$

Umgekehrt gibt es zu jedem $\varepsilon > 0$ *reelle Zahlen* x, y *mit den Eigenschaften* $|x - y| \leq \varepsilon$ *und* $\mathrm{rd}(x) \neq \mathrm{rd}(y)$.

Beweis: Der Fall $x = y = 0$ ist trivial. Es sei also $x, y \neq 0$ und $\tilde{x} = \mathrm{rd}(x) = \mathrm{rd}(y)$. Setzt man

$$\varepsilon_x = \frac{\tilde{x} - x}{x} \ , \qquad \varepsilon_y = \frac{\tilde{x} - y}{y} \ ,$$

so folgt $|\varepsilon_x|, |\varepsilon_y| \leq eps$ aus (1.24). Aus $x(1 + \varepsilon_x) = \tilde{x} = y(1 + \varepsilon_y)$ erhält man nun

$$|x - y| = |y\varepsilon_y - x\varepsilon_x| \leq (|x| + |y|)eps \ .$$

Offenbar wird beispielsweise

$$x = 0, a_1 \cdots a_\ell \ a_{\ell+1} \ , \qquad a_1 \neq 0 \ , a_{\ell+1} = \tfrac{1}{2}q \ ,$$

aufgerundet und $y = x - \varepsilon$ für beliebig kleine $\varepsilon > 0$ abgerundet. ■

Wie wir wissen, ist der relative Abstand von zwei verschiedenen Gleitkommazahlen $\tilde{x}$, $\tilde{y}$ mindestens $q^{-\ell} = 2q^{-1}eps$. Ein *beliebig kleiner* relativer Abstand zweier reeller Zahlen x, y kann also durch Runden auf $\tilde{x}$, $\tilde{y}$ zu einem relativen Abstand von mehr als $2q^{-1}eps$ vergrößert werden.

1.4.5 Praktische Realisierung

Die Speicherung einer Gleitkommazahl im Computer erfolgt in der normalisierten Darstellung (1.16) mit $q = 2$. Somit sind neben dem Vorzeichen s noch die Mantisse a, also $a_1, \cdots, a_\ell$, und der Exponent e (in Form einer ganzen Zahl) zu speichern. Es ist also wiederum erforderlich, sich darüber Gedanken zu machen, wie viele Bits zur Speicherung zur Verfügung gestellt werden sollen. Diese Anzahl an Bits muss nun jedoch geeignet auf das Vorzeichen, das in Form eines Vorzeichenbits gespeichert wird, die Mantisse und den Exponenten aufgeteilt werden. Das heißt, daß bei der praktischen Realisierung von Gleitkommazahlen nicht nur (wie bei den ganzen Zahlen) die Mantissenlänge, sondern auch der Bereich darstellbarer Exponenten auf

$$e \in [e_{\min}, e_{\max}] \cap \mathcal{Z}$$

eingeschränkt wird. Offenbar ist damit auch der Bereich darstellbarer Zahlen eingeschränkt. Für jede Gleitkommazahl $\tilde{x}$ gilt nämlich

$$x_{\min} := q^{e_{\min}-1} \leq |\tilde{x}| \leq (1 - q^{-\ell}) q^{e_{\max}} =: x_{\max} \ .$$

In diesem Fall bleibt Satz 1.11 richtig, falls die zusätzlichen Bedingungen $x \in [x_{\min}, x_{\max}]$ und $|x| \geq q^{e_{\min}}$ erfüllt sind.

Es sei noch erwähnt, daß bei $q = 2$ schon eine Exponentenlänge von $M = 4$ ausreicht, um genauso große Zahlen darstellen zu können wie mit einer 32–Bit–Integer–Zahl. Diese Vergrößerung des darstellbaren Bereichs wird jedoch damit erkauft, daß nun nicht mehr alle ganzen Zahlen x mit $x_{\min} \leq x \leq x_{\max}$ exakt darstellbar sind.

Im Gegensatz zur Darstellung von ganzen Zahlen gibt es für die Speicherung von Gleitkommazahlen eine Industrienorm, die Norm `IEEE 754`, die weitestgehend mit der Art der Darstellung von Gleitkommazahlen übereinstimmt, die bereits Konrad Zuse in seinen ersten Rechenmaschinen Z1 und Z3 realisiert hat.

Der Standard `IEEE 754` sieht vor, daß Gleitkommazahlen in der Form

$$\boxed{s \,\big|\, e_{M-1} e_{M-2} \ \cdots \ e_0 \,\big|\, a_2 \ \cdots \ a_\ell} \ \hat{=} \ (-1)^s a 2^e \ .$$

gespeichert werden. Die beiden wichtigsten in der Norm festgelegten Formate sind Gleitkommazahlen mit einfacher Genauigkeit (*single precision*, oftmals als Datentyp *float* bezeichnet) und mit doppelter Genauigkeit(*double precision*, z.B. als Datentyp *double* in den Programmiersprachen C/C++ und Java). Einfache Floats habe eine Länge von 32 Bit, wobei 8 Bit auf den Exponenten und 23 Bit auf die Mantisse entfallen. Bei Double Floats teilen sich die vorgesehenen 64 Bit in eine Mantisse von 52 Bit und einen Exponenten von 11 Bit auf. Zusätzlich ist selbstverständlich jeweils noch ein Vorzeichenbit vorgesehen. Neben diesen beiden Formaten definiert die Norm zusätzlich noch erweiterte Formate, die in der heutigen Praxis für den Anwender jedoch nur eine untergeordnete Rolle spielen.

Das Vorzeichenbit s wird jeweils mit dem Wert 0 für ein positives und dem Wert 1 für ein negatives Vorzeichen repräsentiert.

Der Exponent $e = e_{M-1} e_{M-2} \ \cdots \ e_0$ wird als ganze Zahl nicht im Zweierkomplement sondern in einer so genannten Biased-Form dargestellt. Dazu wird der Wert des Exponenten um einen festen Wert, den Bias, verschoben, so daß alle möglichen Werte des Exponenten positiv werden. Dies

erleichtert u.a. den Größenvergleich zweier verschiedener Gleitkommawerte. Bei Gleitkommazahlen mit einfacher Genauigkeit beträgt der Bias 127, bei doppelter Genauigkeit 1023. Da die Bitfolgen $000\ldots000$ und $111\ldots111$ des Exponenten für spezielle Bedeutungen reserviert sind, lassen sich so Exponenten im Bereich von $e_{\min} = -126$ (bzw. $e_{\min} = -1022$) bis $e_{\max} = 128$ bei einfacher Genauigkeit (bzw. $e_{\max} = 1024$ bei doppelter Genauigkeit) darstellen.

Wegen der Forderung der Normalisierung der Mantisse a einer Gleitkommazahl $x \neq 0$ auf $q^{-1} \leq a < 1$ gilt bei $q = 2$ stets $a = 0,1a_2\cdots a_\ell$. Da also stets $a_1 = 1$ gilt, ist es nicht nötig, das a_1-Bit mit zu speichern, es brauchen nur die Bits der Ziffern a_2 bis a_ℓ gespeichert werden. Man spricht von der Hidden-Bit-Darstellung, die im Standard `IEEE 754` vorgesehen ist. Auf diese Weise lassen sich mit den beiden Standarddatentypen normalisierte Gleitkommazahlen x im Bereich von etwa $1,18\cdot10^{-38} \leq |x| \leq 3,40\cdot10^{+38}$ bei einfacher Genauigkeit und im Bereich von etwa $2,23\cdot10^{-308} \leq |x| \leq 1,80\cdot10^{+308}$ bei doppelter Genauigkeit darstellen.

Allerdings kann durch diese Hidden-Bit-Darstellung die Null nicht mehr wie beschrieben repräsentiert werden. Um dennoch auch die Null als Gleitkommazahl speichern zu können, sind für den Wert $x = 0$ die speziellen Bitfolgen reserviert, bei denen (mit beliebigem Vorzeichenbit s) alle Bits des Exponenten und der Mantisse den Wert 0 haben. Neben der absoluten Zahl $Null$ werden auch alle Zahlen x mit $|x| < x_{\min}$ zu 0.0 gerundet, wobei das Vorzeichen der Zahl x erhalten bleibt. Beim Vergleich zweier Gleitkommazahlen werden jedoch $+0.0$ und -0.0 als gleich angesehen.

Neben der speziellen Darstellung der Null gibt es noch weitere reservierte Werte für $+\infty$, $-\infty$ und NaN (*Not a Number*, keine Zahl). So ist z.B. festgelegt, daß die Berechnung von $1.0/0.0$ zum Ergebnis $+\infty$ führt. Der Ausdruck $0.0/0.0$ ist jedoch unbestimmt und liefert deshalb NaN.

Eine Übersicht über die Darstellung verschiedener Zahlen und Werte als Gleitkommazahlen in der Norm `IEEE 754` liefert Tabelle 3. Interessant ist noch zu erwähnen, daß auch Speicherbelegungen,

Exponent	**Mantisse**	**Bedeutung**
$111\ldots111$	$000\ldots000$	$+/-\infty$
$111\ldots111$	$\neq 000\ldots000$	NaN, „Not a Number" (keine Zahl)
$000\ldots000$	$000\ldots000$	$+/-0.0$, Null
$000\ldots000$	$\neq 000\ldots000$	Denormalisierte Fließkommazahl
$000\ldots001$ bis $111\ldots110$	beliebig	Normalisierte Fließkommazahl

Tabelle 3: Darstellbare Werte für Gleitkommazahlen in der Norm `IEEE 754`.

bei denen der Exponent durch $000\ldots000$ und die Mantisse durch eine Bitfolge $\neq 000\ldots000$ dargestellt wird, eine Bedeutung zugemessen wird. Bei unserer bisherigen Beschreibung von Gleitkommazahlen wurden solche Zahlen noch nicht berücksichtigt. Solche Zahlen werden als denormalisierte Gleitkommazahlen interpretiert. Sie dienen dazu, die Lücke zwischen der kleinsten darstellbaren normalisierten Gleitkommazahl und der Null zu verkleinern. So hat die kleinste, positive, darstellbare denormalisierte Gleitkommazahl bei einfacher Genauigkeit einen Wert von etwa $x_{\min} = 1,4\cdot10^{-45}$ und bei doppelter Genauigkeit von $x_{\min} = 4,9\cdot10^{-324}$. Die relativen Rundungsfehler, die bei der Speicherung einer Zahl x in diesem Bereich gemacht werden, sind sehr groß, da zwei benachbarte denormalisierte Gleitkommazahlen stets den gleichen Abstand haben.

Auf eine genaue Beschreibung der denormalisierten Gleitkommazahlen wollen wir verzichten, da

sie für unsere folgenden Überlegungen nicht von zentraler Bedeutung sind. Durch die künstliche Verfeinerung des Zahlenbereichs in der Nähe der Null wird eine Exaktheit vorgegaukelt, die beim numerischen Rechnen mit den denormalisierten Gleitkommazahlen nicht erreicht werden kann. Insbesondere ist es durch die auftretenden Ungenauigkeiten in der Praxis nicht sinnvoll, zwei sehr kleine denormalisierte Gleitkommazahlen zu vergleichen. Die Gründe dafür liegen in Prinzipien, die wir erst im Abschnitt 2 über Gleitkommaarithmetik beschreiben werden. Wichtig ist es nur, sich zu merken, daß ein Vergleich einer Zahl x mit der Null 0.0 in der Praxis nicht sinnvoll ist. Stattdessen testet man, ob die Zahl x nahe an der Null liegt: $|x| < eps$, wobei die Maschinengenauigkeit eps für Gleitkommazahlen mit einfacher Genauigkeit bei etwa $eps = 5,96 \cdot 10^{-8}$ und für solche mit doppelter Genauigkeit bei $eps = 1,11 \cdot 10^{-16}$ liegt.

In Tabelle 4 werden noch einmal die Charakteristika der beiden wichtigsten Datentypen für Gleitkommazahlen zusammengestellt.

	Float	Double
Länge in Bits	32	64
Exponent		
Bits	8	11
Bias	127	1023
$e_{\min}$	-126	-1022
$e_{\max}$	128	1024
Mantisse		
Bits	23	52
normalisierte Gleitkommazahl (Exponent = $00\ldots01$ bis $11\ldots10$)		
$x_{\min}$	$1,2 \cdot 10^{-38}$	$2,2 \cdot 10^{-308}$
$x_{\max}$	$3,4 \cdot 10^{+38}$	$1,8 \cdot 10^{+308}$
Maschinengenauigkeit eps	$6,0 \cdot 10^{-8}$	$1,1 \cdot 10^{-16}$
denormalisierte Gleitkommazahl (Exponent = $00\ldots00$, Mantisse $\neq$ $00\ldots00$)		
$x_{\min}$	$1,4 \cdot 10^{-45}$	$4,9 \cdot 10^{-324}$

Tabelle 4: Zusammenfassung der Eigenschaften der beiden Gleitkomma-Datentypen Float und Double

Als Beispiel für die Umwandlung einer (endlichen oder unendlichen) Dezimalzahl in eine duale Gleitkommazahl einfacher Genauigkeit wollen wir die Zahl 2π betrachten. Aufgrund der Darstellung (1.15) von π können wir die Dualdarstellung von 2π direkt angeben:

$$2\pi = 110,010\ 0100\ 0011\ 1111\ldots_2 .$$

Das Komma ist einfach um eine Stelle nach rechts verschoben. Um nun die Darstellung als Float zu erhalten, bräuchten wir nur noch die Kenntnis weiterer Nachkommastellen und könnten dann die entsprechende Gleitkommazahl direkt ablesen. Es bleibt jedoch die Frage offen, wie man die Dualdarstellung aus der Dezimaldarstellung der Zahl gewinnen kann. Deshalb wollen wir noch einmal ganz von vorne anfangen.

Zunächst nähern wir die Zahl 2π noch einmal mit einer ausreichenden Anzahl an Nachkommastellen

im Dezimalsystem an. Dazu benötigen wir mindestens $\lceil \log_{10}(2^\ell) \rceil = \lceil \log_{10}(2^{23}) \rceil = 7$ Nachkommastellen:

$$2\pi = 6,283\ 185\ 3..._{10}\ .$$

Nun liest man zuerst das Vorzeichenbit s ab, welches in diesem Falle für die positive Zahl 2π ein 0-Bit ist.

Zur Bestimmung des Exponenten ist der duale Logarithmus der darzustellenden Zahl zu bestimmen, also

$$e_{10} = \lfloor \log_2(2\pi) \rfloor = \lfloor 2,65..._{10} \rfloor = 2_{10}.$$

Zu dieser Zahl muss noch der Bias 127_{10} addiert werden, und das Ergebnis 129_{10} in eine 8-Bit-Dualzahl umgewandelt werden:

$$e_2 = 10000001_2.$$

Am kompliziertesten gestaltet sich die Berechnung der Mantisse. Diese ergibt sich aus der Formel

$$\left\lfloor \left(\frac{x}{2^e} - 1 \right) \cdot 2^{\ell-1} \right\rfloor,$$

wobei dieses ganzzahlige Ergebnis noch in eine $(\ell-1)$-stellige Dualzahl umgewandelt werden muss. Es ist zu beachten, daß bei dieser Berechnung das führende a_1-Bit mit dem Wert 1_2 bereits herausgerechnet wurde. Angewendet auf unser Beispiel mit $x = 2\pi$, $e = 2$ und $\ell = 24$ ergibt sich:

$$\bar{a} = \left\lfloor \left(\frac{2\pi}{2^2} - 1 \right) \cdot 2^{23} \right\rfloor = \lfloor (1,5707963... - 1) \cdot 2^{23} \rfloor = \lfloor 4788186,6..._{10} \rfloor = 4788186_{10}.$$

Nach der Umwandlung in eine Dualzahl erhalten wir die abzuspeichernden Bits der Mantisse als

$$a_2 \ldots a_\ell = 10010010000111111011010.$$

Insgesamt wird also die Zahl 2π als 32-Bit-Gleitkommazahl in der Form

$$2\pi \hat{=} \boxed{0 \mid 10000001 \mid 10010010000111111011010}$$

gespeichert.

Während in älteren Rechnergenerationen oftmals aus Kostengründen auf die hardwareseitige Implementierung von Gleitkommazahlen verzichtet wurde[15], sind heutige Hauptprozessoren meist mit einer Floating Point Unit (FPU, Gleitkomma-Einheit) ausgestattet, die zumindest die beiden Standarddatentypen für Gleitkommazahlen der `IEEE 754` unterstützen. Durch die Implementierung der Floating Point Unit kann die Gleitkommaarithmetik in heutigen Rechnergenerationen sehr effizient durchgeführt werden. Zwar benötigen die Standardoperationen wie Addition und Multiplikation noch immer mehr Prozessortakte als die entsprechenden Operationen bei Integerarithmetik, jedoch gilt für die Gleitkommaarithmetik ebenso — entsprechend dem in Abschnitt 1.2.3 für Integerzahlen Gesagten —, daß die tatsächliche Anzahl der Rechnertakte und damit die Rechenzeit, die zur Durchführung der Standardoperationen benötigt werden, nicht vom Wert der Operanden abhängt, sondern nur von dem verwendeten Datentyp.

[15] Bei Intel-Prozessoren älterer Generationen bis zum 486SX konnte ein mathematischer Coprozessor nachgerüstet werden, der die Floating Point Unit (FPU) implementierte. Im Nachfolgemodell 486DX war dann dieser Coprozessor schon integriert, was bei späteren Prozessorgenerationen ab dem Intel Pentium zum Standard wurde.

Der Gleitkommatyp Double ist auch der von dem Numerikpaket MATLAB verwendete Standarddatentyp. Jede in MATLAB erzeugte Variable ist standardmäßig (eine Matrix) vom Typ Double. Will man andere Datentypen verwenden, so muß dies explizit durch eine Typkonvertierung angegeben werden. Neben verschiedenen ganzzahligen Datentypen lassen sich durch explizite Konvertierung in MATLAB auch Gleitkommazahlen einfacher Genauigkeit erzeugen. Dazu verwendet man die Funktion `single()`. Dies ist vor allem dann empfehlenswert, wenn man mit sehr großen Datenmengen arbeitet, bei denen nicht die volle Genauigkeit des Double-Datentyps benötigt wird, da auf diese Weise Speicherplatz gespart werden kann.

2 Gleitkommaarithmetik

2.1 Kondition der Grundrechenarten

In diesem Abschnitt wollen wir feststellen, wie sich relative Fehler in den Eingabedaten x und y auf deren Summe, Differenz, Produkt und Quotienten auswirken. Wir gehen also davon aus, daß anstelle von x und y nur die Approximationen

$$\tilde{x} = x(1+\varepsilon_x)\ , \qquad \tilde{y} = y(1+\varepsilon_y) \tag{2.27}$$

zur Verfügung stehen. Dann ist

$$\frac{|x-\tilde{x}|}{|x|} = |\varepsilon_x|\ , \quad \frac{|y-\tilde{y}|}{|y|} = |\varepsilon_y|\ ,$$

und wir bezeichnen ε, definiert durch

$$\varepsilon = \max\{|\varepsilon_x|, |\varepsilon_y|\}\ ,$$

als relativen Eingabefehler. Handelt es sich um die gerundeten Eingabedaten $\tilde{x} = \mathrm{rd}(x), \tilde{y} = \mathrm{rd}(y) \in \mathbb{G}(q,\ell)$, so ist nach Satz 1.11 der relative Eingabefehler durch die Maschinengenauigkeit beschränkt, also $\varepsilon \le eps(q,\ell)$.

Wir wollen herausfinden, um welchen Faktor der relative Eingabefehler durch Anwendung der Grundrechenarten schlimmstenfalls vergrößert werden kann. Diesen Faktor κ nennen wir relative Kondition oder kurz Kondition. Zur Konzentration aufs Wesentliche führen wir noch eine Abkürzung ein.

Definition 2.1 (Landau-Symbol o) *Sei $f : (-a,a) \to \mathbb{R}$ mit $a > 0$ eine Funktion. Wir verabreden die Schreibweise*

$$\lim_{\varepsilon\to 0} \frac{f(\varepsilon)}{\varepsilon} = 0 \quad \Longleftrightarrow \quad f(\varepsilon) = o(\varepsilon)\ .$$

Beispielsweise ist $3\varepsilon^2 + 12\varepsilon^4 + 9\varepsilon^5 = o(\varepsilon)$. Man bezeichnet $o(\varepsilon)$ auch oft als Terme höherer Ordnung.

Als erstes untersuchen wir die Addition.

Satz 2.2 (Addition) *Unter der Voraussetzung $x, y > 0$ gilt*

$$\frac{|(x+y)-(\tilde{x}+\tilde{y})|}{|x+y|} \le \varepsilon\ . \tag{2.28}$$

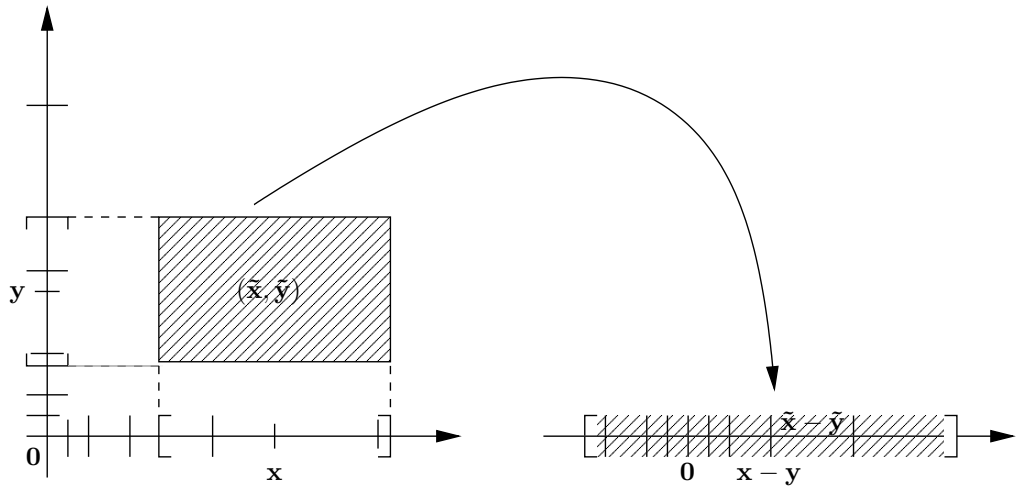


Abbildung 5: Fehlerverstärkung durch Auslöschung.

Beweis: Mit der Dreiecksungleichung erhalten wir aus (2.27) die Abschätzung

$$|(x+y)-(\tilde{x}+\tilde{y})| = |(x-\tilde{x})+(y-\tilde{y})| = |x\varepsilon_x + y\varepsilon_y| \le |x||\varepsilon_x| + |y||\varepsilon_y| \le (|x|+|y|)\varepsilon = |x+y|\varepsilon\ . \tag{2.29}$$

Im letzten Schritt haben wir $x, y > 0$ ausgenutzt. ■

Diese Abschätzung ist scharf, d.h. sie kann nicht verbessert werden. Im Falle $\varepsilon_x = \varepsilon_y = \varepsilon > 0$ erhält man nämlich sofort $|(x+y)-(\tilde{x}+\tilde{y})| = |x+y|\varepsilon$. Die Kondition der Addition ist also $\kappa = 1$. Die Addition positiver Zahlen ist somit ausgesprochen gutmütig: Schlimmstenfalls überträgt sich der Eingabefehler in den Summanden verstärkungsfrei auf die Summe.

Satz 2.3 (Subtraktion) *Unter den Voraussetzungen $x, y > 0$ und $x \neq y$ gilt*

$$\frac{|(x-y)-(\tilde{x}-\tilde{y})|}{|x-y|} \le \left(\frac{|x|+|y|}{|x-y|}\right)\varepsilon\ . \tag{2.30}$$

Beweis: Wir gehen genauso vor wie im Beweis zu Satz 2.2 und erhalten anstelle von (2.29) diesmal

$$|(x-y)-(\tilde{x}-\tilde{y})| = |(x-\tilde{x})+(\tilde{y}-y)| = |x\varepsilon_x - y\varepsilon_y| \le |x||\varepsilon_x| + |y||\varepsilon_y| \le (|x|+|y|)\varepsilon\ .$$

■

Genau wie (2.28) so ist auch die Abschätzung (2.30) scharf. Zunächst bemerken wir, daß bei der Subtraktion positiver Zahlen die Kondition $\kappa = \kappa(x,y)$ von x und y abhängt. Insbesondere kann $\kappa(x,y)$ beliebig groß werden, falls sich x und y wenig unterscheiden. So gilt beispielsweise mit beliebigem $m \in \mathbb{N}$

$$x = 1 \approx y = 1 + 10^{-m} \quad \Longrightarrow \quad \frac{|x|+|y|}{|x-y|} > 2 \cdot\ 10^m\ .$$

Die Subtraktion positiver Zahlen ist beliebig schlecht konditioniert.

Die praktischen Auswirkungen wollen wir anhand eines einfachen Zahlenbeispiels illustrieren. Wir gehen dabei von $\ell = 4$ gültigen Stellen im Dezimalsystem aus. Im Falle $x = \frac{1}{3}$ und $y = 0,3332$ ist dann $\tilde{x} = 0.3333$ und $\tilde{y} = y$. Wir erhalten

$$\tilde{x} - \tilde{y} = 0.0001 = 0.1XYZ \cdot 10^{-3} \ ,$$

also anstelle der *vier richtigen Stellen* in den Eingabedaten nur noch *eine richtige Stelle* im Ergebnis! Dieses Phänomen heißt Auslöschung. Anstelle von x, Y, Z können irgenwelche Zahlen stehen[16]. Man kann nachrechnen, daß in unserem Beispiel die Verstärkung des relativen Fehlers bei $\frac{1}{4}10^4$ liegt.

Wir kommen zur Punktrechnung.

Satz 2.4 (Multiplikation) *Unter den Voraussetzungen $x \neq 0$ und $y \neq 0$ gilt*

$$\frac{|(x \cdot y) - (\tilde{x} \cdot \tilde{y})|}{|x \cdot y|} \leq 2\,\varepsilon + o(\varepsilon) \ . \tag{2.31}$$

Beweis: Einsetzen von (2.27) nebst Dreiecksungleichung liefert die Abschätzung

$$|x \cdot y - \tilde{x} \cdot \tilde{y}| = |xy||\varepsilon_x + \varepsilon_y + \varepsilon_x\varepsilon_y| \leq |xy|(2\varepsilon + \varepsilon^2)$$

und offenbar ist $\varepsilon^2 = o(\varepsilon)$. ■

Wieder gilt Gleichheit für $\varepsilon_x = \varepsilon_y = \varepsilon > 0$. Der Eingabefehler ε wird also um den Faktor 2 verstärkt und dann kommen noch der Term $o(\varepsilon) = \varepsilon^2$ von höherer Ordnung hinzu. Nach Definition 2.1 ist

$$2\varepsilon \gg o(\varepsilon)$$

für genügend kleine ε[17]. Wir *ignorieren daher den Fehleranteil* $o(\varepsilon)$. Damit ist $\kappa = 2$ die Kondition der Multiplikation.

Satz 2.5 (Division) *Unter den Voraussetzungen $x \neq 0$ und $y \neq 0$ gilt*

$$\frac{\left|\frac{x}{y} - \frac{\tilde{x}}{\tilde{y}}\right|}{\left|\frac{x}{y}\right|} \leq 2\,\varepsilon + o(\varepsilon) \ . \tag{2.32}$$

Beweis: Einsetzen von (2.27) ergibt

$$\left|\frac{x}{y} - \frac{\tilde{x}}{\tilde{y}}\right| = \left|1 - \frac{1+\varepsilon_x}{1+\varepsilon_y}\right|\left|\frac{x}{y}\right| = \left|\frac{\varepsilon_y - \varepsilon_x}{1+\varepsilon_y}\right|\left|\frac{x}{y}\right| \leq \frac{2\varepsilon}{1-\varepsilon}\left|\frac{x}{y}\right| = \left(2\varepsilon + \frac{2\varepsilon^2}{1-\varepsilon}\right)\left|\frac{x}{y}\right| \ ,$$

und offenbar ist $2\varepsilon^2/(1-\varepsilon) = o(\varepsilon)$. ■

Auch diese Abschätzung ist scharf. Analog zur Multiplikation ist also $\kappa = 2$ die Kondition der Division.

[16] Welche Zahlen das sind, ist rechnerabhängig. Man sollte nicht $X = Y = Z = 0$ erwarten.
[17] Bei einfacher Genauigkeit ist $eps = 5,96 \cdot 10^{-8} \gg eps^2 = 3.55 \cdot 10^{-15}$.

2.2 Algebraische Eigenschaften

Die Menge $\mathbb{R}$ der reellen Zahlen, versehen mit Addition und Multiplikation, ist ein Körper. Die reellen Zahlen genügen also den folgenden Körperaxiomen (siehe z.B. Bosch [**?**, Kap. 2.1]).

(K1) $x + (y + z) = (x + y) + z$ (Assoziativität der Addition)

(K2) $x + y = y + x$ (Kommutativität der Addition)

(K3) Es gibt ein Element $0 \in \mathbb{R}$ mit $0 + x = x$ für alle $x \in \mathbb{R}$. (neutrales Element der Addition)

(K4) Zu jedem $x \in \mathbb{R}$ existiert ein Element $-x \in \mathbb{R}$ mit $(-x) + x = 0$. (Inverses Element der Add.)

(K5) $x(yz) = (xy)z$ (Assoziativität der Multiplikation)

(K6) $xy = yx$ (Kommutativität der Multiplikation)

(K7) Es gibt ein Element $1 \in \mathbb{R}$ mit $1x = x$ für alle $x \in \mathbb{R}$. (neutrales Element der Multiplikation)

(K8) Zu jedem $x \in \mathbb{R} \setminus \{0\}$ existiert ein Element $x^{-1} \in \mathbb{R}$ mit $x^{-1}x = 1$. (Inv. Element der Mult.)

(K9) x(y+z)=xy +xz (Distributivität)

(K10) $1 \neq 0$ (Nullteilerfreiheit)

Diese zehn Eigenschaften bilden die Grundlage für alle äquivalenten algebraischen Umformungen, die wir aus der Schule gewohnt sind. Beispielsweise gewinnt man die binomische Formel $(a+b)^2 = a^2 + 2ab + b^2$ aus (K6) und dreimaliger Anwendung von (K9). Wir wollen untersuchen, in welchem Umfang sich die Körpereigenschaften (K1) – (K10) auf die Teilmenge $\mathbb{G} \subset \mathbb{R}$ der Gleitkommazahlen vererben.

Gleich am Anfang stoßen wir auf eine grundsätzliche Schwierigkeit: Die Grundrechenarten können aus $\mathbb{G}$ herausführen. So ist offenbar $\tilde{x} = 1234 \in \mathbb{G}(10, 4)$ und $\tilde{y} = 12,34 \in \mathbb{G}(10, 4)$, für die Summe gilt aber $\tilde{x} + \tilde{y} = 1246,34 \notin \mathbb{G}(10, 4)$ und auch die Multiplikation liefert $15227,56 \in \mathbb{G}(10, 4)$. Im allgemeinen gilt also

$$\tilde{x}, \tilde{y} \in \mathbb{G} \quad \not\Longrightarrow \quad \tilde{x} + \tilde{y},\ \tilde{x} \cdot \tilde{y} \in \mathbb{G}\ .$$

Um die Grundrechenarten auf $\mathbb{G}$ zu beschränken, müssen wir nach jeder Rechenoperation das Ergebnis auf $\mathbb{G}$ zurückprojezieren, also *runden*.

Definition 2.6 (Gleitkommaarithmetik) *Auf den Gleitkommazahlen $\mathbb{G}$ definieren wir die gerundeten Grundrechenarten $\tilde{+}$, $\tilde{-}$, $\tilde{*}$ und $\tilde{:}$ durch*

$$\tilde{x}\tilde{+}\tilde{y} = \mathrm{rd}(\tilde{x} + \tilde{y}), \quad \tilde{x}\tilde{-}\tilde{y} = \mathrm{rd}(\tilde{x} - \tilde{y}), \quad \tilde{x}\tilde{*}\tilde{y} = \mathrm{rd}(\tilde{x}\tilde{y}), \quad \tilde{x}\tilde{:}\tilde{y} = \mathrm{rd}(\tilde{x} : \tilde{y}),\ \tilde{y} \neq 0, \qquad \tilde{x}, \tilde{y} \in \mathbb{G}$$

Wir wollen sehen, ob die Gleitkommazahlen, versehen mit gerundeter Addition und Multiplikation, die Axiome (K1)-(K10) erfüllen. Die meisten Axiome postulieren die *Gleichheit* gewisser Ausdrücke. Wie wir am Ende des vorigen Abschnitts gesehen haben, sind Gleichheitsabfragen von Gleitkommazahlen aber im allgemeinen sinnlos. Das lässt nichts Gutes erwarten.

Und schon mit (K1) gibt es Schwierigkeiten. In $\mathbb{G}(10, 4)$ gilt nämlich

$$(1234\tilde{+}0,4)\tilde{+}0,4 = 1234\tilde{+}0,4 = 1234 \neq 1235 = 1234\tilde{+}0,8 = 1234\tilde{+}(0,4\tilde{+}0,4)\ ,$$

und dieses Beispiel lässt sich leicht auf beliebige Basis q und Mantissenlänge ℓ übertragen. Die gerundete Addition ist also nicht assoziativ. Unser Beispiel zeigt übrigens auch, daß bei Gleitkommazahlen aus $\tilde{a} + \tilde{x} = \tilde{a}$ nicht notwendig $\tilde{x} = 0$ folgen muß.

Nun zur Abwechslung eine gute Nachricht: Die gerundete Addition ist kommutativ. Dies folgt aus

$$\tilde{x}\tilde{+}\tilde{y} = \mathrm{rd}(a_x q^{e_x} + a_y q^{e_y}) = \mathrm{rd}(a_x + a_y q^{e_y - e_x}) q^{e_x} = \mathrm{rd}(a_y + a_x q^{e_x - e_y}) q^{e_y} = \tilde{y}\tilde{+}\tilde{x} \; . \tag{2.33}$$

Wegen $0 \in \mathbb{G}$ und $\tilde{x} \in \mathbb{G} \Rightarrow -\tilde{x} \in \mathbb{G}$ sind (K3) und (K4) klar.

Wir kommen zur Multiplikation. In $\mathbb{G}(10, 4)$ gilt offenbar

$$(1234\tilde{*}0,9996)\tilde{*}0,9999 = 1234\tilde{*}0,9999 = 1234 \neq 1233 = 1234\tilde{*}0,9995 = 1234\tilde{*}(0,9996\tilde{*}0,9999) \; .$$

Wieder lässt sich dieses Beispiel auf $\mathbb{G}$ verallgemeinern. Die Multiplikation ist also nicht assoziativ. Wieder sieht man auch, daß sich $\tilde{a}\tilde{x} = \tilde{a}$, $\tilde{a} \neq 0$, nicht äquivalent in $\tilde{x} = 1$ umformen lässt.

Wie schon die Addition, so ist auch die gerundete Multiplikation kommutativ. Der Nachweis erfolgt analog zu (2.33).

Wegen $1 \in \mathbb{G}$ ist (K7) kein Problem, aber schon (K8) macht wieder Schwierigkeiten. In $\mathbb{G}(10, 4)$ gilt nämlich beispielsweise

$$0,1111\tilde{*}9 = 0,9999 < 1 < 1,001 = 0,1112\tilde{*}9 \; .$$

Da die gerundete Multiplikation mit 9 streng monoton ist, also $9\tilde{y} < 9\tilde{z}$ aus $\tilde{y} < \tilde{z}$ folgt, und da es kein $\tilde{y} \in \mathbb{G}(10, 4)$ mit $0,1111 < \tilde{y} < 0,1112$ gibt, folgt daraus, daß $\tilde{x} = 9$ kein inverses Element $\tilde{x}^{-1} \in \mathbb{G}(10, 4)$ hat. Die gerundete Multiplikation ist also im allgemeinen nicht invertierbar.

An dieser Stelle wird schon niemand mehr erwarten, daß das Distributivgesetz gilt. Tatsächlich findet man schnell Gegenbeispiele. Die gerundete Addition und Multiplikation sind nicht distributiv[18].

Wir haben uns also wohl oder übel darauf einzustellen, daß beim gerundeten Rechnen mit Gleitkommazahlen elementare Rechenregeln verletzt werden. Umformungen, die bei exaktem Rechnen in $\mathbb{R}$ zu äquivalenten Ausdrücken führen, liefern beim gerundeten Rechnen verschiedene Resultate. Beispielsweise gilt die binomische Formel für Gleitkommazahlen im allgemeinen nicht. Sind $\tilde{a}, \tilde{b} \in \mathbb{G}$ Approximationen von $a, b \in \mathbb{R}$, so sind die Ausdrücke $(\tilde{a} + \tilde{b})^2$ und $\tilde{a}^2 + 2\tilde{a}\tilde{b} + \tilde{b}^2$ im allgemeinen *verschiedene* Approximationen von $(a + b)^2 = a^2 + 2ab + b^2$. Welche ist die bessere? Diese Frage führtauf den Begriff der *Stabilität*, mit dem wir uns in Kapitel 4 ausführlich beschäftigen werden. wenn wir die Stabilität Einen kleinen Vorgeschmack geben die nächsten beiden Abschnitte.

2.3 Gleichheitsabfragen

Es ist numerisch nicht möglich, die Gleichheit zweier beliebiger reeller Zahlen x, y festzustellen. Dazu wäre ja der Vergleich aller unendlich vielen Ziffern der entsprechenden q-adischen Darstellung

[18] Vor dem Hintergrund all dieser Negativresultate stellt sich die Frage, warum zur Approximation von $\mathbb{R}$ nicht besser die rationalen Zahlen herangezogen werden. Schließlich ist $\mathbb{Q}$, versehen mit Addition und Multiplikation, nach wie vor ein Körper, in dem man, anders als in $\mathbb{G}$, die gewohnten algebraischen Umformungen vornehmen kann. Der entscheidende Grund liegt in dem hohen Zeitaufwand für das Rechnen mit Brüchen (vgl. Abschnitt 1.3.4). Angesichts exorbitant hoher Rechenzeiten wird die rationale Arithmetik derzeit nur in einzelnen Spezialanwendungen eingesetzt.

notwendig. Ersatzweise bleibt uns nur übrig zu prüfen, ob die gerundete Version $\mathrm{rd}(x) = \mathrm{rd}(y)$ richtig ist. Trivialerweise gilt

$$x = y \qquad \Longrightarrow \qquad \mathrm{rd}(x) = \mathrm{rd}(y) \; .$$

Mehr können wir auch nicht erwarten. Wir können insbesondere nicht ausschließen, daß $\mathrm{rd}(x) = \mathrm{rd}(y)$ vorliegt, obwohl $x \neq y$ ist. Nach Möglichkeit sollte man Gleichheitsabfragen von Gleitkommazahlen also vermeiden.

Die Sache wird noch etwas komplizierter, wenn Rechenergebnisse ins Spiel kommen. Als Beispiel betrachten wir die exakte Aussage „$s = x + y$“ mit positiven Zahlen s, x, $y \in \mathbb{R}$. Es liegt nahe, wieder die gerundete Version „$\tilde{s} = \tilde{x} \tilde{+} \tilde{y}$“ mit $\tilde{s} = \mathrm{rd}(s)$, $\tilde{x} = \mathrm{rd}(x)$, $\tilde{y} = \mathrm{rd}(y)$ zu verwenden. Überraschenderweise ist nun aber

$$s = x + y \qquad \not\Longrightarrow \qquad \tilde{s} = \tilde{x} \tilde{+} \tilde{y} \; .$$

Dies bestätigt man leicht durch Gegenbeispiele, etwa durch Wahl von $q = 10$, $\ell = 4$ und

$$x = y = 0,10004 \; , \quad s = x + y = 0,20008 \; , \qquad 0,2001 = \tilde{s} \neq \tilde{x} \tilde{+} \tilde{y} = 0,2 \; .$$

Zwischen der exakten und der gerundeten Aussage besteht also kein Zusammenhang! Bei Verwendung von Gleitkommaarithmetik gilt:

Gleichheitsabfragen von Rechenergebnissen sind sinnlos!

An die Stelle von Gleichheitsabfragen treten *Ungleichungsbedingungen*, welche sich im allgemeinen aus Konditions- und Stabilitätsüberlegungen ergeben (vgl. Satz 4.4 in Abschnitt 4.2). Das prinzipielle Vorgehen wollen wir am vorliegenden Beispiel erläutern. Zunächst gilt offenbar

$$|\tilde{s} - s| \leq |s| \; eps \; , \qquad |(\tilde{x} \tilde{+} \tilde{y}) - (\tilde{x} + \tilde{y})| \leq |\tilde{x} + \tilde{y}| \; eps \; ,$$

und Satz 2.2 liefert

$$|(\tilde{x} + \tilde{y}) - (x + y)| \leq |x + y| \; eps \; .$$

Ist nun $s = x + y$, so erhält man daraus mit der Dreiecksungleichung

$$|\tilde{s} - (\tilde{x} \tilde{+} \tilde{y})| \leq 3|s|eps + |s|eps^2 \leq (3|\tilde{s}| + 4|s|eps) \; eps = 3|\tilde{s}|eps + o(eps) \; .$$

Kompensiert man die Terme höherer Ordnung durch einen Sicherheitsfaktor $\rho \geq 4|s|eps$, so ist die resultierende Ungleichungsbedingung

$$|\tilde{s} - (\tilde{x} \tilde{+} \tilde{y})| \leq (3|\tilde{s}| + \rho) \; eps$$

im Falle $s = x + y$ erfüllt und insofern sinnvoll. Nach wie vor gilt die Umkehrung natürlich nicht.

2.4 Stabilität von Summationsalgorithmen

Unsere Aufgabe besteht in der Summation von n positiven reellen Zahlen, also der Auswertung von

$$s_n = \sum_{i=1}^{n} x_i = x_1 + x_2 + \cdots + x_n \ .$$

Genauer gesagt, wollen wir einen Algorithmus finden, der s_n durch *sukzessive gerundete Addition* möglichst genau approximiert. Diese Problematik hat praktische Bedeutung und ist daher recht genau untersucht. Ueberhuber [**?**, Kapitel 5.8] widmet dem Thema eine Fallstudie. Einen umfassenden Überblick über Summationsalgorithmen gibt Highham [**?**, Kapitel 4].

Eine naheliegende Möglichkeit s_n auszuwerten besteht darin, die Summanden x_1 bis x_n einfach nacheinander zu addieren. Man beachte, daß dadurch eine bestimmte Klammerung festgelegt wird, nämlich beispielsweise

$$s_5 = (((x_1 + x_2) + x_3) + x_4) + x_5 \ .$$

Die entsprechende Rekursionsformel

$$s_1 = x_1 \ , \quad s_i = s_{i-1} + x_i \ , \qquad i = 2, \ldots, n \ ,$$

lässt sich direkt in einen Algorithmus umsetzen.

Algorithmus 2.7 (Rekursive Summation)

```
s=x(1);
for i = 2:n
  s=s+x(i);
end
```

Gleichbedeutend mit der Anwendung von Algorithmus 2.7 ist die Auswertung der *gestörten Rekursion*

$$\tilde{s}_1 = x_1 \ , \qquad \tilde{s}_i = \tilde{s}_{i-1} \tilde{+} x_i \ , \quad i = 2, \ldots, n \ , \tag{2.34}$$

bei der wir die exakte Addition durch die Gleitkommaaddition ersetzt haben. Wir wollen feststellen, wie stabil sich die Berechnungsvorschrift (2.34) gegenüber dieser Störung verhält. Die Frage ist also, wie stark sich die Lawine von Rundungsfehlern, die bei den vielen Gleitkommaadditionen entsteht, auf das Endergebnis auswirkt. Vereinfachend setzen wir dabei voraus, daß

$$x_i = \tilde{x}_i > 0 \in \mathbb{G}, \quad i = 1, \ldots, n \ . \tag{2.35}$$

Es gibt also *keine Eingabefehler*, und es kann *keine Auslöschung* auftreten.

Satz 2.8 *Unter der Voraussetzung* (2.35) *gilt für die rekursive Summation* (2.34) *die Fehlerabschätzung*

$$|s_n - \tilde{s}_n| \leq \sum_{i=1}^{n} x_i (2^{n+1-i} - 1) eps \ . \tag{2.36}$$

Dabei bedeutet $eps = eps(q, \ell)$ die Maschinengenauigkeit.

Beweis: Zur Vorbereitung erinnern wir an die binomische Formel

$$(a+b)^k = \sum_{l=0}^{k} \left(\begin{array}{c} k \\ l \end{array} \right) a^l b^{k-l} , \qquad \left(\begin{array}{c} k \\ l \end{array} \right) = \frac{k!}{l!(k-l)!} . \tag{2.37}$$

Setzt man $a = b = 1$, so folgt

$$\sum_{l=0}^{k} \left(\begin{array}{c} k \\ l \end{array} \right) = 2^k . \tag{2.38}$$

Doch nun zum eigentlichen Beweis. Wegen Satz 1.11 beziehungsweise (1.24) ist

$$\tilde{s}_1 = x_1(1+\varepsilon_1) , \qquad \tilde{s}_i = \mathrm{rd}(\tilde{s}_{i-1} + x_i) = (\tilde{s}_{i-1} + x_i)(1+\varepsilon_i) , \quad i = 2, \ldots, n ,$$

mit $\varepsilon_1 = 0$ und $|\varepsilon_i| \leq eps$, $i = 2, \ldots, n$. Daraus folgt mit vollständiger Induktion die Darstellung

$$\tilde{s}_n = \sum_{i=1}^{n} x_i \prod_{j=i}^{n} (1+\varepsilon_j) .$$

Mit Hilfe der Dreiecksungleichung erhält man daraus

$$|s_n - \tilde{s}_n| \leq \sum_{i=1}^{n} x_i K_i , \qquad K_i = \left| \prod_{j=i}^{n} (1+\varepsilon_j) - 1 \right| . \tag{2.39}$$

Wir haben nun die Faktoren K_i abzuschätzen. Nach Ausmultiplizieren des Produkts heben sich die Summanden 1 und -1 weg. Dann wendet man die Dreiecksungleichung an, fügt die Summanden 1 und -1 wieder hinzu und macht die Ausmultiplikation rückgängig. Auf diese Weise ergibt sich

$$K_i = \left| \prod_{j=i}^{n} (1+\varepsilon_j) - 1 \right| \leq \prod_{j=i}^{n} (1+|\varepsilon_j|) - 1 \leq (1+eps)^{n+1-i} - 1 . \tag{2.40}$$

Nun liefert die binomische Formel wegen $eps < 1$ und (2.38) für $k = n+1-i$ die Abschätzung

$$(1+eps)^k - 1 = \sum_{l=1}^{k} \left(\begin{array}{c} k \\ l \end{array} \right) eps^l < eps \sum_{l=1}^{k} \left(\begin{array}{c} k \\ l \end{array} \right) = (2^k - 1)eps . \tag{2.41}$$

Insgesamt gilt also

$$K_i \leq (2^{n-i+1} - 1)eps , \qquad i = 1, \ldots, n .$$

Durch Einsetzen dieser Abschätzung in (2.39) erhalten wir schließlich die Behauptung. ■

In Satz 2.8 haben wir eine *absolute Fehlerabschätzung* bewiesen. Offenbar liefert (2.36) aber unmittelbar auch die folgende Abschätzung des *relativen Fehlers*

$$\frac{|s_n - \tilde{s}_n|}{|s_n|} \leq (2^n - 1)eps . \tag{2.42}$$

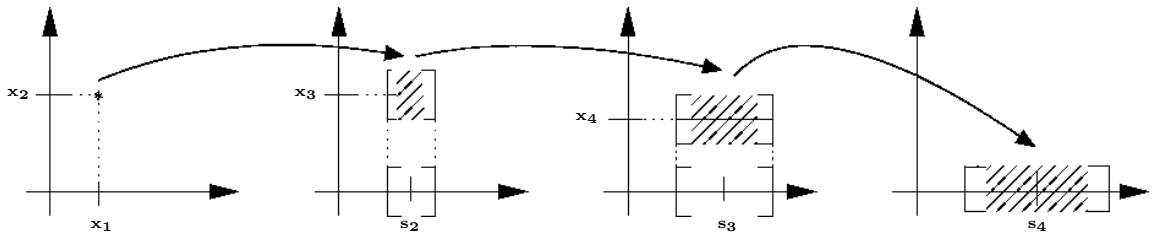


Abbildung 6: Sukzessive Akkumulation von Rundungsfehlern.

Wir stellen erschreckt fest, daß sich die auftretenden Rundungsfehler von der Größenordnung *eps* im Laufe der Summation derart aufschaukeln können, daß sie in ihrer Auswirkung auf das Ergebnis

um einen exponentiell wachsenden Faktor verstärkt werden. Nun, ganz so schlimm muß es nicht kommen, denn (2.36) und (2.42) stellen nur obere Schranken dar, welche den tatsächlichen Fehler im allgemeinen erheblich überschätzen.[19] Ein deutlicher Hinweis auf mangelnde Stabilität von Algorithmus 2.7 sind die Abschätzungen (2.36) und (2.42) aber auf jeden Fall.

Es lohnt sich, die Fehlerabschätzung (2.36) etwas genauer zu betrachten. Die Verstärkungsfaktoren $(2^{n-i+1}-1)$ werden mit wachsendem i immer kleiner. Wenn nun die zugehörigen Summanden x_i so geordnet sind, daß sie mit wachsendem i immer größer werden, so ist die entsprechende Fehlerschranke unter allen Permutationen der Summanden minimal. Wir können also die Stabilität der rekursiven Summation vergrößern, indem wir eine aufsteigende Reihenfolge der Summanden wählen. Eine Veränderung der Reihenfolge entspricht übrigens einer anderen Klammerung ($\tilde{+}$ ist nicht assoziativ!).

Algorithmus 2.9 (Aufsteigende Summation)

```
y=sort(x)

s=y(1);
for i = 2:n
  s=s+y(i);
end
```

Aus der Fehlerabschätzung (2.34) können wir noch mehr lernen. Offenbar variieren die zu x_i gehörenden Verstärkungsfaktoren sehr stark, nämlich zwischen 2^n-1 für $i=1$ und 1 für $i=n$. Der Grund liegt darin, daß x_1 an allen $n-1$ Additionen beteiligt ist, der letzte Summand x_n dagegen nur an einer. Könnten wir hier nicht für mehr Ausgeglichenheit sorgen und so vielleicht den maximalen Verstärkungsfaktor reduzieren?

Dazu nehmen wir der Einfachheit halber an, daß $n=2^J$ eine Zweierpotenz ist[20]. Setzt man nun die Klammern, beispielsweise für $n=8=2^3$, wie folgt,

$$s_8 = ((x_1+x_2)+(x_3+x_4))+((x_5+x_6)+(x_7+x_8)) \;,$$

so ist jeder Summand genau an $J=3$ Additionen beteiligt. Das Prinzip besteht darin, in jedem Schritt die benachbarten Summanden zu addieren und dadurch eine neue Summe mit halbierter Anzahl von Summanden zu erzeugen. Die hierarchische Struktur dieser Vorgehensweise wird in Abbildung 7 deutlich.

Aus Abbildung 7 lässt sich auch die folgende Rekursion ablesen,

$$s_l^{(0)} = x_l \;, \quad l=1,\ldots,2^J \;, \qquad s_l^{(k)} = s_{2l-1}^{(k-1)} + s_{2l}^{(k-1)} \;, \quad l=1,\ldots 2^{J-k}, \;\; k=1,\ldots,J \;. \tag{2.43}$$

Das gesuchte Ergebnis ist dann $s_n = s_1^{(J)}$. Die Berechnungsvorschrift (2.43) lässt sich direkt algorithmisch umsetzen:

Algorithmus 2.10 (Hierarchische Summation)

[19] Das liegt unter anderem an der sehr pessimistischen Abschätzung (2.41).

[20] Theoretisch können wir das natürlich immer erreichen, indem wir gegebenenfalls zusätzliche Nullen addieren. Praktisch wird man effizientere Modifikationen finden.

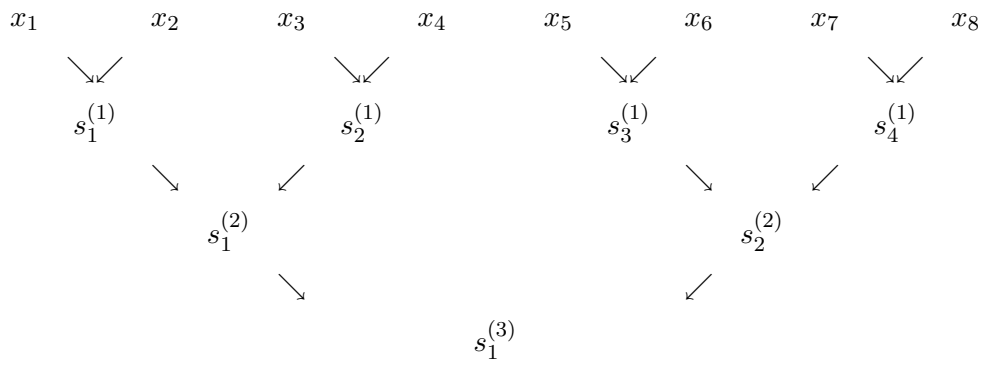


Abbildung 7: Hierarchische Summation

```
s=x;

for k = 1:J
  for l=1:2**(J-k)
     s(l)=s(2*l-1)+s(2*l);
  end
end
```

Eleganter wäre allerdings eine rekursive Implementierung.

Die Auswertung von Algorithmus 2.10 in Gleitkommaarithmetik ist gleichbedeutend mit der gestörten Berechnungsvorschrift

$$\tilde{s}_l^{(0)} = x_l \ , \quad l = 1, \ldots, 2^J \ , \qquad \tilde{s}_l^{(k)} = \tilde{s}_{2l-1}^{(k-1)} \tilde{+} \tilde{s}_{2l}^{(k-1)} \ , \quad l = 1, \ldots 2^{J-k}, \ \ k = 1, \ldots, J \ , \tag{2.44}$$

und dem Endergebnis $\tilde{s}_n = \tilde{s}_1^J$. Im Vergleich mit Algorithmus 2.7 haben wir nur die Klammerung geändert. Wir wollen sehen, ob es uns gelungen ist, auf diese Weise die Stabilität zu verbessern.

Satz 2.11 *Unter der Voraussetzung* (2.35) *gilt für die hierarchische Summation* (2.44) *die Fehlerabschätzung*

$$\frac{|s_n - \tilde{s}_n|}{|s_n|} \leq (n-1)eps \ . \tag{2.45}$$

Beweis: Nach (1.24) führt (2.44) auf die Rekursion

$$\tilde{s}_l^{(0)} = x_l \ , \quad l = 1, \ldots, 2^J \ , \qquad \tilde{s}_l^{(k)} = (\tilde{s}_{2l-1}^{(k-1)} + \tilde{s}_{2l}^{(k-1)})(1 + \varepsilon_l^{(k)}) \ , \quad l = 1, \ldots, 2^{J-k} \ , \quad k = 1, \ldots, J \ ,$$

mit $|\varepsilon_l^{(k)}| \leq eps$. Mit Hilfe dieser Vorschrift kann man durch Induktion über k auf erstaunlich einfache Weise die etwas kompliziertere Darstellung

$$\tilde{s}_l^{(k)} = \sum_{i=(l-1)2^k+1}^{l2^k} x_i \prod_{j=1}^{k} \left(1 + \varepsilon_{1+[(i-1)/2^j]}^{(j)}\right) \ , \qquad l = 1, \ldots, 2^{J-k} \ , \quad k = 1, \ldots, J \ ,$$

bestätigen. Dabei bedeutet $[a] = \max\{m \in \mathbb{N} \mid m \leq a\}$. Insbesondere erhält man

$$\tilde{s}_n = s_1^{(J)} = \sum_{i=1}^{n} x_i \prod_{j=i}^{J} \left(1 + \varepsilon_{1+[(i-1)/2^j]}^{(j)}\right) \ .$$

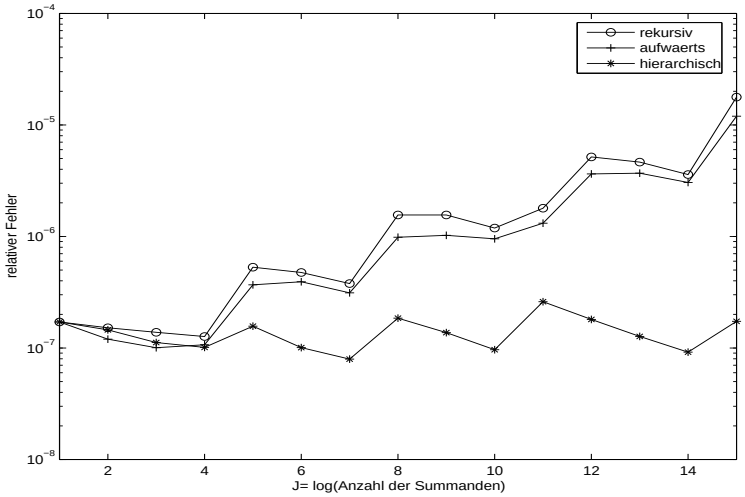


Abbildung 8: Vergleich von rekursiver, aufsteigender und hierarchischer Summation

Nun geht es genauso weiter wie im Beweis zu Satz 2.8. Offenbar gilt

$$|s_n - \tilde{s}_n| \leq \sum_{i=1}^{n} x_i \left| \prod_{j=1}^{J} \left(1 + \varepsilon^{(j)}_{1+[(i-1)/2^j]}\right) - 1 \right| ,$$

und die Abschätzung

$$\left| \prod_{j=1}^{J} \left(1 + \varepsilon^{(j)}_{1+[(i-1)/2^j]}\right) - 1 \right| \leq (1+eps)^J - 1 \leq (2^J - 1)eps = (n-1)eps$$

folgt wie (2.40) und (2.41). ∎

In theoretischer Hinsicht ist der Fortschritt in der Tat dramatisch ($\tilde{+}$ ist wirklich nicht assoziativ!). Während beim rekursiven Summationsalgorithmus 2.7 laut (2.42) eine *exponentielle* Fehlerverstärkung um den Faktor $2^n - 1$ möglich ist, kann beim hierarchischen Algorithmus 2.10 höchstens der *lineare* Faktor $n - 1$ auftreten.

Abschließend wollen wir anhand numerischer Experimente feststellen, inwieweit sich unsere theoretischen Resultate auch im praktischen Verhalten der Algorithmen widerspiegeln. Da sich Rundungsfehler zufällig verstärken oder auslöschen können, sollen unsere Summationsverfahren anhand mehrerer Tests verglichen werden. Für jeden Test wählen wir jeweils $n_J = 2^J$ Zufallszahlen $x_i \in (0,1)$ mit $\ell = 7$ Stellen und $J = 1, \ldots, 15$. Deren Summe berechnen wir dann approximativ mittels rekursiver, aufsteigender und hierarchischer Summation bei 7-stelliger Gleitkommaaddition. Zum Vergleich wird noch die „exakte“ Summe s_n durch hierarchische Summation mit doppelt genauer Arithmetik (15 Stellen) ausgerechnet. Abbildung 8 zeigt die Mittelwerte der relativen Fehler von 100 solcher Tests in Abhängigkeit von der Anzahl $2^1, \ldots, 2^{15}$ der Summanden. Zunächst fällt auf, daß bei allen Verfahren der Fehler deutlich geringer ausfällt als nach unserer Stabilitätsanalyse zu befürchten war. Qualitativ werden unsere theoretischen Untersuchungen aber nachhaltig bestätigt: Die Verbesserung der rekursiven Summation 2.7 durch aufsteigende Anordnung der Summanden ist sichtbar, aber nicht gravierend. Dagegen entspricht die Überlegenheit der hierarchischen Summation 2.10 dem dramatisch besseren Stabilitätsresultat in Satz 2.11. Während bei rekursiver und aufsteigender Summation der relative Fehler um zwei Größenordnungen anwächst, ist bei der hierarchischen Summation auch für $n_{15} = 32\,768$ Summanden kein wesentlicher Einfluß der Rundungfehler sichtbar.

2.5 Praktische Realisierung

Die `IEEE 754`-Norm fordert bei allen Operationen exaktes Runden, d.h. die Ergebnisse der Operationen müssen dieselben sein, die man bei zunächst exakter Ausführung der Rechenoperation und anschließendem Runden erhielte. In der Praxis wird dies erreicht, indem ein interner Zwischenspeicher mit erhöhter Genauigkeit für die Speicherung der Zwischenergebnisse verwendet wird. Aus diesen Ergebnissen mit erhöhter Genauigkeit wird am Ende der Rechnung auf die übliche (einfache bzw. doppelte) Genauigkeit gerundet.

3 Funktionsauswertungen

3.1 Einlochen auf ebenem und unebenem Grün

Wir befinden uns auf einem Golfplatz, genauer gesagt auf einem kreisförmigen Grün, und stehen vor der Aufgabe, den vor uns liegenden Golfball einzulochen. Das Loch ist $d = 3$ m entfernt und hat den Radius $r_L = 5,4$ cm. Es gilt also, den Golfball mit wohldosierter Kraft in die richtige Richtung zu schlagen. Das sind zwei unabhängige Parameter. Wir gehen im folgenden davon aus, daß die Dosierung stimmt und konzentrieren uns nur auf die Richtung. Diese ist durch den Abschlagswinkel x zur Verbindungsgeraden zwischen Golfball- und Lochmittelpunkt charakterisiert (vgl. Abb. 9). Je nach Wahl von x nimmt die Kugel ihren Lauf und erreicht dabei einen minimalen Abstand von $f(x)$ Metern zum Lochmittelpunkt. Auf diese Weise ist eine Funktion

$$f: \quad [-\tfrac{\pi}{2}, \tfrac{\pi}{2}] = \{x \in \mathbb{R} \mid -\tfrac{\pi}{2} \le x \le \tfrac{\pi}{2}\} \to \mathbb{R}$$

erklärt[21]. Natürlich sind wir an Winkeln x mit der Eigenschaft $f(x) < r_L$ interessiert, denn dann geht der Ball ins Loch. Leider ist die Funktion f im allgemeinen nicht in geschlossener Form gegeben. Jede Funktionsauswertung erfordert einen Versuch (oder dessen Simulation auf dem Rechner). Im Falle eines völlig ebenen Grüns liegen die Dinge einfacher. Angesichts der linearen Bahn des

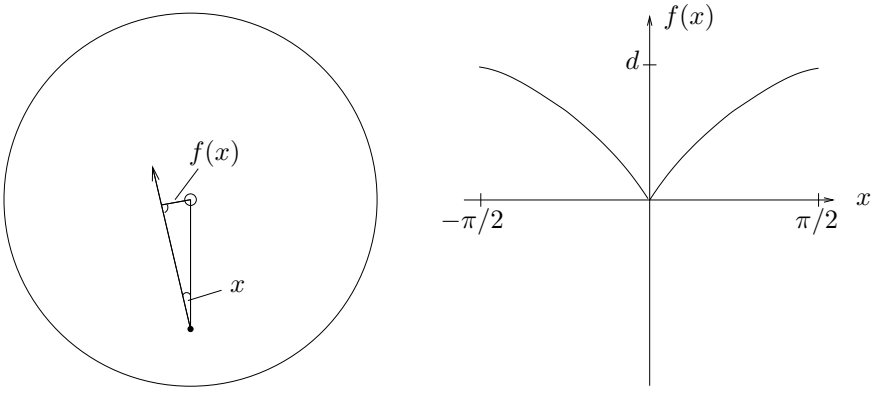


Abbildung 9: Das ebene Grün: Bahn des Golfballs und Graph von f

Golfballs gilt dann nämlich

$$f(x) = d|\sin(x)| \,.$$

[21] Wir sollten zumindest in Richtung des Lochs schlagen, daher die Einschränkung auf $x \in [-\frac{\pi}{2}, \frac{\pi}{2}] \subset [-\pi, \pi]$.

Offenbar liefert $x_0 = 0$ den Abstand $f(x_0) = 0$ und ist damit optimal. Aber wir können uns auch kleine Fehler erlauben und trotzdem erfolgreich einlochen. Dazu reicht jeder Abschlagswinkel x (gemessen im Bogenmaß) mit der Eigenschaft

$$|x_0 - x| < |\arcsin(r_L/d)| = 0,01800097214174 \; .$$

Das ist eine Abweichung von etwa einem Grad. Andernfalls geht der Ball am Loch vorbei.

Unsere Aufgabe wird deutlich einfacher, wenn das Grün konzentrisch gesenkt ist. In diesem Fall führt jeder Abschlagswinkel zum Ziel, wie Abbildung 10 illustriert.

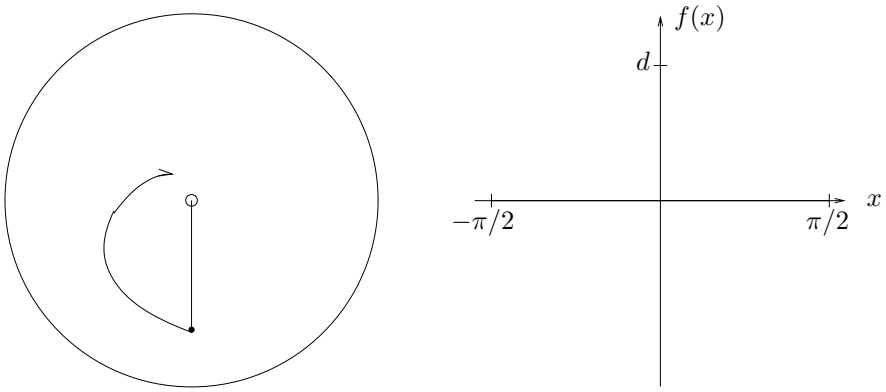


Abbildung 10: Die Senke: Bahn des Golfballs und Graph von f

Umgekehrt wird die Sache bei einem konzentrisch gewölbten Grün schwieriger: Mit zunehmender Krümmung werden Störungen $|x_0 - x|$ in zunehmendem Maße verstärkt. Schon kleinste Abweichungen führen zum Misserfolg.

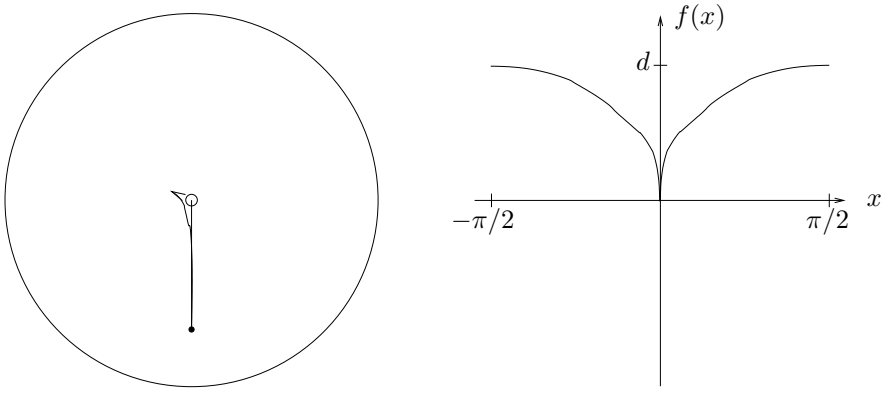


Abbildung 11: Der Hügel: Bahn des Golfballs und Graph von f

Die Wirklichkeit hält natürlich eine weit größere Vielfalt an unterschiedlichen Bedingungen bereit. Wer will, kann sich beim Platzwart beschweren.

3.2 Die Kondition von Funktionsauswertungen

Gegeben sei ein Intervall $I \subset \mathbb{R}$, eine Funktion $f : I \to \mathbb{R}$ und $x_0 \in I$. Wir betrachten das Problem der

$$\text{Auswertung von } f \text{ an der Stelle } x_0 \,. \tag{3.46}$$

Es interessiert uns, wie sich Fehler in der Eingabe x_0 auf die Ausgabe $f(x_0)$ auswirken[22]. Der maximale Verstärkungsfaktor ist die Kondition des Problems (3.46). Zuvor müssen wir Ein- und Ausgabefehler quantifizieren, das heißt, wir müssen uns für ein Fehlermaß entscheiden. Wir beginnen mit dem absoluten Fehler.

3.2.1 Absolute Kondition

Zum Aufwärmen betrachten wir die affin lineare Funktion

$$f(x) = ax + b$$

mit a, $b \in \mathbb{R}$. Offenbar gilt

$$|f(x_0) - f(x)| = |a||x_0 - x| \,.$$

Die absolute Kondition von (3.46) ist in diesem Fall also $\kappa_{\text{abs}} = |a|$. Sie hängt nicht von x_0 ab.

Nun zu folgendem Polynom zweiten Grades

$$f(x) = ax^2 + bx + c \,.$$

Mit Hilfe der binomischen Formel $a^2 - b^2 = (a + b)(a - b)$ bestätigt man

$$f(x_0) - f(x) = (2ax_0 + b)(x_0 - x) - a(x_0 - x)^2 \,,$$

und es folgt

$$|f(x_0) - f(x)| = |2ax_0 + b||x_0 - x| + o(|x_0 - x|) \,. \tag{3.47}$$

Dabei haben wir von dem Landau-Symbol o Gebrauch gemacht (vgl. Definition 2.1) und insbesondere

$$|\kappa\varepsilon + o(\varepsilon)| = |\kappa||\varepsilon| + o(\varepsilon) \qquad \forall \kappa \in \mathbb{R}$$

ausgenutzt. Für kleine Fehler $|x_0 - x|$ wird die rechte Seite von (3.47) durch den ersten Summanden dominiert. Vernachlässigt man den Term höherer Ordnung $o(|x_0 - x|)$, so erhält man die absolute Kondition $\kappa_{\text{abs}} = |2ax_0 + b|$.

Im allgemeinen sind wir mit einer oberen Schranke für den Ausgabefehler zufrieden.

Definition 3.1 (Absolute Kondition) *Die absolute Kondition κ_{abs} von (3.46) ist die kleinste Zahl mit der Eigenschaft*

$$|f(x_0) - f(x)| \leq \kappa_{\text{abs}}|x_0 - x| + o(|x_0 - x|) \,. \tag{3.48}$$

Liegt (3.48) für keine reelle Zahl κ_{abs} vor, so wird $\kappa_{\text{abs}} = \infty$ gesetzt.

[22] Dabei spielt es keine Rolle, ob es sich um Rundungsfehler handelt oder nicht.

Die Kondition $\kappa_{\text{abs}} = \kappa_{\text{abs}}(f, x_0)$ hängt nur von der auszuwertenden Funktion f und dem Argument x_0 ab. Die Kondition ist eine Eigenschaft des betrachteten Problems.

Auf elegantere, aber etwas anspruchsvollere Weise kann man die absolute Kondition auch wie folgt charakterisieren:

$$\kappa_{\text{abs}} = \limsup_{x \to x_0} \frac{|f(x_0) - f(x)|}{|x_0 - x|} \ . \tag{3.49}$$

Es kann sein, daß es kein $\rho > 0$ gibt, so daß $f(\tilde{x}_0)$ für alle $\tilde{x}_0$ mit $|x_0 - \tilde{x}_0| \leq \rho$ auswertbar ist. Dann ist $\kappa_{\text{abs}} = \infty$.

Ist die Funktion f unstetig in x_0, so gilt $\kappa_{\text{abs}} = \infty$. Als Beispiel betrachten wir die Funktion f,

$$f(x) = \begin{cases} 0 & \text{falls } x < 0 \ , \\ 1 & \text{falls } x \geq 0 \end{cases}$$

und $x_0 = 0$. Auch für beliebig kleine Eingabefehler ist im Falle $x < x_0$ immer $|f(x_0) - f(x)| = 1$.

Aber auch bei stetigen Funktionen kann $\kappa_{\text{abs}} = \infty$ vorliegen. Sei nämlich $f(x) = \sqrt{x}$ und $I = [0, \infty)$. Mit der binomischen Formel folgt dann für $x_0 \neq 0$

$$f(x_0) - f(x) = \frac{1}{2\sqrt{x_0}}(x_0 - x) + \frac{\sqrt{x_0} - \sqrt{x}}{2(x_0 + \sqrt{x_0 x})}(x_0 - x)$$

und somit $\kappa_{\text{abs}} = \frac{1}{2\sqrt{x_0}}$. Nun sei $x_0 = 0$. Ist κ_{abs} eine Zahl mit der Eigenschaft

$$\sqrt{x} = |f(0) - f(x)| \leq \kappa_{\text{abs}} |x_0 - x| + o(|x_0 - x|) = \kappa_{\text{abs}} x + o(x) \qquad \forall x > 0 \ ,$$

so ergibt Division durch $\sqrt{x}$ die Abschätzung

$$1 \leq \kappa_{\text{abs}} \sqrt{x} + \frac{o(x)}{x} \sqrt{x} \ .$$

Grenzübergang $x \to 0$ führt auf den Widerspruch $1 \leq 0$. Damit ist $\kappa_{\text{abs}} = \infty$ falls $x_0 = 0$.

Satz 3.2 *Ist f differenzierbar in x_0, so gilt $\kappa_{\text{abs}} = |f'(x_0)|$.*

Beweis: Nach Definition der Ableitung gilt

$$\lim_{x \to x_0} \frac{f(x_0) - f(x)}{x_0 - x} = f'(x_0)$$

oder gleichbedeutend

$$\lim_{x \to x_0} \frac{f(x_0) - f(x) - f'(x_0)(x_0 - x)}{x_0 - x} = 0 \ .$$

Daraus erhält man unmittelbar

$$f(x_0) - f(x) = f'(x_0)(x_0 - x) + o(x_0 - x) \ .$$

Nimmt man nun auf beiden Seiten den Betrag und verwendet die Rechenregel $|a + o(\varepsilon)| = |a| + o(\varepsilon)$, so folgt die Behauptung. ■

Alle unsere bisherigen Beispiele mit endlicher Kondition waren verdeckte Anwendungen von Satz 3.2. Die Differenzierbarkeit von f ist aber kein notwendiges Kriterium für $\kappa_{\text{abs}} < \infty$. Beispielsweise hat (3.46) im Falle $f(x) = |x|$ und $x_0 = 0$ die Kondition $\kappa_{\text{abs}} = 1$.

Definition 3.3 (Lipschitz-Stetigkeit) *Die Funktion $f : I \to \mathbb{R}$ heißt Lipschitz-stetig mit Lipschitz-Konstante L, falls*

$$|f(x) - f(y)| \leq L|x - y| \qquad \forall x, y \in I \; .$$

Beispielsweise sind $f(x) = x$, $f(x) = |x|$ und $f(x) = \sin(x)$ Lipschitz-stetig mit Lipschitz-Konstante $L = 1$.

Aus der Definition folgt unmittelbar:

Satz 3.4 *Ist $f : I \to \mathbb{R}$ Lipschitz-stetig mit Lipschitz-Konstante L, so genügt die absolute Kondition κ_{abs} von* (3.46) *der Abschätzung*

$$\kappa_{\text{abs}} \leq L \; .$$

Wir betrachten nun die Kondition von geschachtelten Funktionen.

Satz 3.5 *Es sei $f(x) = g \circ h(x) = g(h(x))$. Es bezeichne $\kappa_{\text{abs}}(h, x_0)$ die absolute Kondition der Auswertung von h an der Stelle x_0 und $\kappa_{\text{abs}}(g, y_0)$ die absolute Kondition der Auswertung von g an der Stelle $y_0 = h(x_0)$. Dann ist*

$$\kappa_{\text{abs}} \leq \kappa_{\text{abs}}(g, y_0) \; \kappa_{\text{abs}}(h, x_0) \; . \tag{3.50}$$

Ist h differenzierbar in x_0 und g differenzierbar in y_0, so liegt in (3.50) *Gleichheit vor.*

Beweis: Wir brauchen nur den Fall $\kappa_{\text{abs}}(h, x_0) < \infty$ und $\kappa_{\text{abs}}(g, y_0) < \infty$ zu betrachten. Nach Definition gilt

$$\begin{aligned} |f(x_0) - f(x)| &\leq \kappa_{\text{abs}}(g, y_0)|h(x_0) - h(x)| + o(|h(x_0) - h(x)|) \\ &\leq \kappa_{\text{abs}}(g, y_0)\kappa_{\text{abs}}(h, x_0)|x_0 - x| + \kappa_{\text{abs}}(g, y_0)o(|x_0 - x|) + o\left(\kappa_{\text{abs}}(h, x_0)|x_0 - x| + o(|x_0 - x|)\right) \\ &= \kappa_{\text{abs}}(g, y_0)\kappa_{\text{abs}}(h, x_0)|x_0 - x| + o(|x_0 - x|) \; . \end{aligned}$$

Der zweite Teil der Behauptung folgt aus Satz 3.2 und der Kettenregel. ■

Als abschließendes Beispiel betrachten wir das Problem vom Golfplatz (vgl. Abschnitt 3.1), also

$$f(x) = d|\sin(x)| \; , \qquad x_0 = 0 \; .$$

Setzt man $g(y) = d|y|$ und $h(x) = \sin(x)$, so folgt aus Satz 3.5 nebst Satz 3.2 sofort $\kappa_{\text{abs}} \leq d$. Obwohl g nicht differenzierbar ist, gilt in diesem Fall sogar $\kappa_{\text{abs}} = d$. Die Fehlerverstärkung wächst also linear mit dem Abstand d zum Loch.

3.2.2 Relative Kondition

Der relative Fehler in Ein- oder Ausgabe ist für $x_0 = 0$ oder $f(x_0) = 0$ nicht definiert. Es sei also von nun an $x_0 \neq 0$ und $f(x_0) \neq 0$. Wir definieren die relative Kondition in vollkommener Analogie zur absoluten Kondition.

Definition 3.6 (Relative Kondition) *Die relative Kondition κ_{rel} von* (3.46) *ist die kleinste Zahl mit der Eigenschaft*

$$\frac{|f(x_0) - f(x)|}{|f(x_0)|} \leq \kappa_{\text{rel}} \frac{|x_0 - x|}{|x_0|} + o(|x_0 - x|) \; . \tag{3.51}$$

Ist (3.51) *für keine reelle Zahl κ_{rel} richtig, so wird $\kappa_{\text{rel}} = \infty$ gesetzt.*

Zwischen relativer und absoluter Kondition besteht ein enger Zusammenhang.

Satz 3.7 *Es gilt*

$$\kappa_{\text{rel}} = \frac{|x_0|}{|f(x_0)|}\kappa_{\text{abs}} \ . \tag{3.52}$$

Beweis: Offenbar ist (3.48) äquivalent mit

$$\frac{|f(x_0) - f(x)|}{|f(x_0)|} \leq \frac{|x_0|}{|f(x_0)|}\kappa_{\text{abs}}\frac{|x_0 - x|}{|x_0|} + o(|x_0 - x|) \ ,$$

und nach Definition ist κ_{abs} die kleinste Zahl, mit der diese Abschätzung gilt. ■

Absolute und relative Kondition können völlig unterschiedliche Größenordnungen haben. Ein einfaches Beispiel ist $f(x) = ax$ mit a, $x_0 \neq 0$. Aus Satz 3.2 und Satz 3.7 folgt

$$\kappa_{\text{abs}} = |f'(x_0)| = a \ , \qquad \kappa_{\text{rel}} = \frac{|x_0||f'(x_0)|}{|f(x_0)|} = 1 \ .$$

3.3 3-Term-Rekursionen

Wir betrachten die Drei-Term-Rekursion

$$x_{k+1} + a_k x_k + b_k x_{k-1} + c_k = 0 \ , \qquad k = 0, 1, 2, \dots \ , \tag{3.53}$$

mit gegebenen Koeffizienten $a_k, b_k, c_k \in \mathbb{R}$. Gibt man die Anfangswerte $x_{-1}, x_0 \in \mathbb{R}$ vor, so lassen sich die anderen Folgenglieder $x_1, x_2, \dots$ offenbar rekursiv berechnen. Drei-Term-Rekursionen spielen in der Angewandten Analysis wie auch in der Numerischen Mathematik eine wichtige Rolle. Beispielsweise genügen die sogenannten *Bessel-Funktionen*[23] J_k der Drei-Term-Rekursion

$$J_{k+1}(x) - \frac{2k}{x}J_k(x) + J_{k-1}(x) = 0 \ , \qquad k = 1, 2, \dots \ . \tag{3.54}$$

Für weitere Beispiele und eine ausführliche Untersuchung der numerischen Aspekte von Drei-Term-Rekursionen verweisen wir auf Deuflhard und Hohmann [**?**, Kapitel 6].

Wir beschränken uns im folgenden auf den Fall

$$x_{k+1} + a x_k + b x_{k-1} = 0 \ , \qquad k = 0, 1, 2, \dots \ , \tag{3.55}$$

also auf homogene Drei-Term-Rekursionen mit konstanten Koeffizienten a, $b \in \mathbb{R}$. Außerdem gehen wir davon aus, daß x_{-1} fest vorgegeben ist und nur x_0 variiert. Dann wird durch (3.55) für jedes $k = 1, 2, \dots$ die Funktion

$$f_k : \mathbb{R} \ni x_0 \mapsto f_k(x_0) = x_k$$

definiert. Wir wollen wissen, wie die Kondition der Auswertung von f_k an der Stelle x_0 von den Parametern a, b, x_{-1} und k abhängt. Dazu verschaffen wir uns zunächst unter zusätzlichen Voraussetzungen eine geschlossene Darstellung der Funktionen f_k.

[23] Die Bessel-Funktionen J_k sind Lösungen der Besselschen Differentialgleichung $x^2 J_k'' + x J_k' + (x^2 - k^2)J_k = 0$. Einzelheiten finden sich etwa bei Kamke [**?**].

Lemma 3.8 *Das charakteristische Polynom*[24]

$$\lambda^2 + a\lambda + b = 0 \tag{3.56}$$

habe die zwei verschiedenen, reellen Nullstellen λ_2, λ_1, und es sei $|\lambda_2| > |\lambda_1|$. Dann gilt

$$f_k(x_0) = \alpha(x_0)\lambda_1^{k+1} + \beta(x_0)\lambda_2^{k+1} , \qquad \alpha(x_0) = \frac{\lambda_2 x_{-1} - x_0}{\lambda_2 - \lambda_1} , \quad \beta(x_0) = x_{-1} - \alpha(x_0) .$$

Beweis: Der Beweis erfolgt mit vollständiger Induktion. Die Fälle $k = 1, 2$ bestätigt man durch Einsetzen. Die Behauptung gelte nun für $k, k-1$ mit $k \geq 0$. Dann folgt mit $\alpha = \alpha(x_0)$ und $\beta = \beta(x_0)$

$$\alpha\lambda_1^{k+2} + \beta\lambda_2^{k+2} + a(\alpha\lambda_1^{k+1} + \beta\lambda_2^{k+1}) + b(\alpha\lambda_1^k + \beta\lambda_2^k) = \alpha\lambda_1^k(\underbrace{\lambda_1^2 + a\lambda_1 + b}_{=0}) + \beta\lambda_2^k(\underbrace{\lambda_2^2 + a\lambda_2 + b}_{=0}) = 0$$

und damit die Behauptung. ■

Nun brauchen wir nur noch die Sätze 3.7 und 3.2 anzuwenden.

Satz 3.9 *Es sei $x_0 \neq 0$ und $f_k(x_0) \neq 0$. Unter den Voraussetzungen von Lemma 3.8 ist dann für $k = 1, 2, \ldots$ die relative Kondition κ_{rel}^k der Auswertung von f_k an der Stelle x_0 gegeben durch*

$$\kappa_{\text{rel}}^k = \frac{|x_0|}{|x_0 - \lambda_1 R_k x_{-1}|} , \qquad R_k = \frac{\lambda_2^{k+1} - \lambda_2\lambda_1^k}{\lambda_2^{k+1} - \lambda_1^{k+1}}.$$

Beweis: Es gilt

$$f_k'(x_0) = \alpha'(x_0)\lambda_1^{k+1} + \beta'(x_0)\lambda_2^{k+1} = \frac{\lambda_2^{k+1} - \lambda_1^{k+1}}{\lambda_2 - \lambda_1} .$$

Durch Ausschreiben von $\alpha(x_0)$, $\beta(x_0)$ nebst elementaren Umformungen bestätigt man

$$f_k(x_0) = \frac{1}{\lambda_2 - \lambda_1}\left((\lambda_2 x_{-1} - x_0)\lambda_1^{k+1} - (\lambda_1 x_{-1} - x_0)\lambda_2^{k+1}\right) = f_k'(x_0)\,(x_0 - \lambda_1 R_k x_{-1}) , \tag{3.57}$$

und Einsetzen in (3.52) liefert schließlich

$$\kappa_{\text{rel}}^k = \frac{|x_0||f_k'(x_0)|}{|f_k(x_0)|} = \frac{|x_0|}{|x_0 - \lambda_1 R_k x_{-1}|} .$$

■

Wir wollen untersuchen, wie sich Eingabefehler in x_0 für große k auf die Folgenglieder $x_k = f_k(x_0)$ auswirken. Offenbar gilt wegen $|\lambda_2| > |\lambda_1|$

$$\lim_{k\to\infty} R_k = 1 .$$

Daher ist

$$\lim_{k\to\infty} \kappa_{\text{rel}}^k = \begin{cases} \dfrac{|x_0|}{|x_0 - \lambda_1 x_{-1}|} , & \text{falls } x_0 \neq \lambda_1 x_{-1} , \\ \infty , & \text{falls } x_0 = \lambda_1 x_{-1} . \end{cases}$$

[24] Die Nullstellen sind gerade die Eigenwerte der Matrix $\begin{pmatrix} -a & -b \\ 1 & 0 \end{pmatrix}$.

Wir wollen zunächst den generischen Fall $x_0 \neq \lambda_1 x_{-1}$ anhand eines Zahlenbeispiels illustrieren. Dazu betrachten wir die Drei-Term-Rekursion

$$4x_{k+1} - 4x_k - 3x_{k-1} = 0 \,, \qquad x_{-1} = 1 \,, \tag{3.58}$$

für $x_0 = 1$. Nach Lemma 3.8 sind die Folgenglieder x_k gegeben durch

$$x_k = f_k(x_0) = \tfrac{1}{4}\left(\lambda_1^{k+1} + 3\lambda_2^{k+1}\right) \,, \qquad \lambda_1 = -\tfrac{1}{2} \,, \;\; \lambda_2 = \tfrac{3}{2} \,,$$

und es gilt $\kappa_{\text{rel}}^k \to \frac{2}{3}$ für $k \to \infty$. Eingabefehler werden also im Sinne des relativen Fehlermaßes kaum verstärkt. Die folgende Tabelle bestätigt diesen Sachverhalt auch numerisch: Die Auswirkungen der Störung $\tilde{x}_0 = x_0(1 + 10^{-5})$ in der 6. Stelle bleiben durchweg auf die 6. Stelle beschränkt.[25]

k	10	100	1000
$x_k = f_k(x_0)$	$6.487304 \cdot 10^1$	$4.573813 \cdot 10^{17}$	$1.388070 \cdot 10^{176}$
$\tilde{x}_k = f_k(\tilde{x}_0)$	$6.487347 \cdot 10^1$	$4.573843 \cdot 10^{17}$	$1.388079 \cdot 10^{176}$
rel. Fehler	$0.7 \cdot 10^{-5}$	$0.7 \cdot 10^{-5}$	$0.7 \cdot 10^{-5}$

Tabelle 5: Gute relative Kondition der Drei-Term-Rekursion im generischen Fall

Im Falle $x_0 = \lambda_1 x_{-1}$ wächst κ_{rel}^k über alle Grenzen für $k \to \infty$. Wir wollen verstehen, was dahintersteckt. Zunächst bestätigt man durch Einsetzen, daß beliebige Linearkombinationen $x_k = \alpha p_k + \beta q_k$ der Folgen

$$p_k = \lambda_1^{k+1} \,, \quad q_k = \lambda_2^{k+1} \,, \qquad k = -1, 0, 1, 2, \ldots \,,$$

Lösungen von (3.55) sind. Die Koeffizienten α, β werden gerade durch die beiden Anfangsbedingungen festgelegt (vgl. Lemma 3.8). Aus $|\lambda_2| > |\lambda_1|$ folgt

$$\lim_{k\to\infty} \frac{p_k}{q_k} = 0 \,. \tag{3.59}$$

Also ist $q_k \gg p_k$ für $k \gg 1$. Man nennt daher q_k dominant und p_k rezessiv. Für die Folgenglieder x_k hat (3.59) die Konsequenz

$$x_k \approx \begin{cases} q_k \gg p_k & \text{falls } \beta \neq 0 \,, \\ \\ p_k & \text{falls } \beta = 0 \,. \end{cases}$$

Nun ist $x_0 = \lambda_1 x_{-1}$ gleichbedeutend mit $\beta(x_0) = 0$. In diesem Fall enthält $x_k = \alpha(x_0) p_k$ also nur den rezessiven Lösungsanteil. Jede Approximation $\tilde{x}_0 \neq x_0$ führt aber zu $\beta(\tilde{x}_0) \neq 0$. Die gestörten Folgenglieder $\tilde{x}_k = \alpha(\tilde{x}_0) p_k + \beta(\tilde{x}_0) q_k$ enthalten damit den dominanten Anteil q_k, und der überlagert die exakte Lösung für genügend große k.

Zur Illustration betrachten wir wieder die Drei-Term-Rekursion (3.58) und wählen diesmal gerade $x_0 = \lambda_1 x_{-1} = -\frac{1}{2}$. Die exakte Lösung klingt exponentiell ab, während die gestörte Lösung auch für beliebig kleine Störungen exponentiell wächst. Die folgende Tabelle zeigt, wie schnell die 10 gültigen Stellen von $\tilde{x}_0 = x_0(1 + 10^{-10})$ verloren gehen. Drei-Term-Rekursionen mit variablen Koeffizienten zeigen ein ähnliches Verhalten. Beispielsweise führt die Verwendung der Rekursion (3.54) sehr schnell in den Besselschen Irrgarten. Neugierige verweisen wir auf Deuflhard und Hohmann [**?**, Kapitel 6.2].

[25] Man beachte, daß der *absolute Fehler* dramatisch anwächst und für $k = 1000$ den Wert $|x_k - \tilde{x}_k| = 0.9 \cdot 10^{171}$ erreicht.

k	5	10	25
$x_k = f_k(x_0)$	$1.5625000000 \cdot 10^{-2}$	$-4.8828125000 \cdot 10^{-4}$	$1.49011611938 \cdot 10^{-8}$
$\tilde{x}_k = f_k(\tilde{x}_0)$	$1.5625000569 \cdot 10^{-2}$	$-4.8827692509 \cdot 10^{-4}$	$1.90873893994 \cdot 10^{-6}$
rel. Fehler	$0.3 \cdot 10^{-7}$	$0.9 \cdot 10^{-5}$	$0.1 \cdot 10^{+3}$

Tabelle 6: Schlechte relative Kondition der Drei-Term-Rekursion bei rezessiven Lösungen

3.4 Ausblick

Wir haben die Kondition der Grundrechenarten und der Auswertung von Funktionen $f : I \to \mathbb{R}$ definiert und untersucht. Die Konzeption der Kondition ist nicht auf diese Aufgabenstellungen beschränkt. Auf zwei beliebigen Mengen X und Y brauchen wir nur Fehlermaße err_X und err_Y festzulegen. Dann können wir in direkter Analogie zu den Definitionen 3.1 und 3.6 die Kondition der Auswertung einer Abbildung $f : X \to Y$ an der Stelle x_0 als kleinste obere Schranke für den Fehlerverstärkungsfaktor erklären. Der Fehlerverstärkungsfaktor ist, bis auf Terme höherer Ordnung, das Verhältnis von Ausgabefehler in $f(x_0) \in Y$ zu Eingabefehler in $x_0 \in X$. X und Y

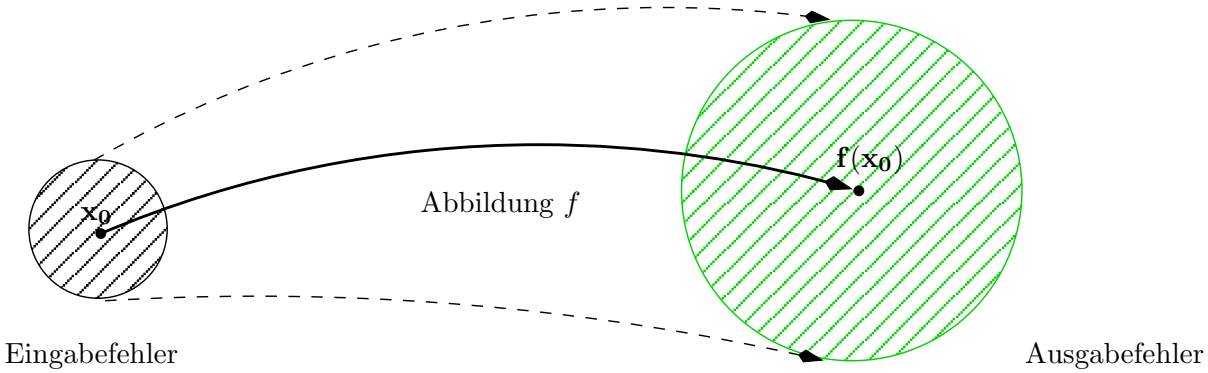


Abbildung 12: Die Kondition der Auswertung von $f : X \to Y$ an der Stelle x_0

können Mengen von Zahlen, Vektoren oder sogar Funktionen sein. Im letzten Teil dieses Buches werden wir die Kondition von linearen Gleichungssystemen untersuchen. Dann ist etwa X die Menge aller Koeffizientenmatrizen A und rechter Seiten b, und $f(A, b) = y$ ist die Lösung von $Ay = b$.

4 Rekursive Funktionsauswertungen

4.1 Ein Polynom-Desaster

Gegeben seien das Polynom

$$f(x) = x^3 + 12a^2x - 6ax^2 - 8a^3 \,, \tag{4.60}$$

der Parameter $a = 4\,999\,999$ und das Argument

$$x_0 = 10\,000\,000 \,.$$

Gesucht ist der Funktionswert $f(x_0)$.

Um diese Aufgabe zu lösen, liegt es nahe, die Berechnungsvorschrift (4.60) direkt zu implementieren.

Algorithmus 4.1 (Auswertung von $f = x^3 + 12a^2x - 6ax^2 - 8a^3$)

```
a =  4999999;
x = 10000000;
f = x^3 + 12*a^2*x - 6*a*x^2 - 8*a^3
```

Algorithmus 4.1 liefert:

```
f =
    393216
```

also das Ergebnis $f(x_0) = 393216$. Nun könnten wir uns zufrieden zurücklehnen.

Besser nicht. Unter Verwendung der binomischen Formel

$$(a+b)^3 = a^3 + 3a^2b + 3ab^2 + b^3$$

lässt sich (4.60) äquivalent umschreiben in

$$f(x) = (x - 2a)^3 \ . \tag{4.61}$$

Die andere Darstellung erzeugt einen anderen Algorithmus.

Algorithmus 4.2 (Auswertung von $f = (x - 2a)^3$)

```
a =  4999999;
x = 10000000;
f = (x - 2*a)^3
```

Dieser Algorithmus liefert:

```
f =
    8
```

und wegen $x - 2a = 2$ ist das offenbar auch das richtige Ergebnis.

Wir wollen verstehen, weshalb der erste Algorithmus 4.1 derart versagt. An der Kondition kann es nicht liegen, denn Eingabefehler treten nicht auf.

4.2 Stabilität rekursiver Funktionsauswertungen

Nachdem wir im vorigen Kapitel 3 das Problem der Auswertung einer Funktion $f : I \to \mathbb{R}$ an einer Stelle $x_0 \in I \subset \mathbb{R}$ untersucht haben, wollen wir uns nun mit Algorithmen zu dessen Lösung beschäftigen. Unter einem Algorithmus wollen wir dabei eine Zerlegung

$$f(x_0) = g_n \circ g_{n-1} \circ \cdots \circ g_1(x_0) \tag{4.62}$$

der Funktion f in elementare Funktionen g_i, $i = 1, \ldots, n$, verstehen. Dabei bedeutet $\circ$ die Hintereinanderschaltung, also $g_i \circ g_{i-1}(y) = g_i\,(g_{i-1}(y))$. Verschiedene Zerlegungen charakterisieren verschiedene Algorithmen. Beispielsweise ist

$$f(x) = ax + b = g_2 \circ g_1(x)\ , \qquad g_1(y) = ay\ , \quad g_2(y) = y + b\ . \tag{4.63}$$

Unter der Voraussetzung $a \neq 0$ kann man f aber auch in der folgenden Form schreiben

$$f(x) = ax + b = h_2 \circ h_1(x)\ , \qquad h_1(y) = y + \frac{b}{a}\ , \quad h_2(y) = ay\ . \tag{4.64}$$

Solange alle Elementarfunktionen exakt ausgewertet werden, liefern beide Zerlegungen dasselbe Ergebnis. Das ist für alle Algorithmen der Form (4.62) nicht anders. Oft können aber nur Approximationen $\tilde{g}_i(y)$ von g_i praktisch realisiert werden. Beispielsweise führt die Berücksichtigung von Rundungsfehlern auf die Approximation

$$\tilde{g}_i(y) = \mathrm{rd}\,(g_i(y)) = g_i(y)(1 + \varepsilon_i)\ , \qquad |\varepsilon_i| \leq eps\ . \tag{4.65}$$

Dann erhält man auch anstelle der Funktion f nur eine Approximation, nämlich

$$\tilde{f}(\varepsilon, x) = \tilde{g}_n \circ \tilde{g}_{n-1} \circ \cdots \circ \tilde{g}_1(x)\ . \tag{4.66}$$

Die Approximation $\tilde{f}(\varepsilon, x)$ hängt offenbar von den Störungen ε_i aller Elementarfunktionen g_i ab, die in dem Vektor $\varepsilon = (\varepsilon_1, \ldots, \varepsilon_n)$ aufgehoben sind.

Verschiedene Algorithmen führen auf *verschiedene Approximationen*. Beispielsweise führt Algorithmus (4.63) bei gerundeter Arithmetik auf die Approximation

$$\tilde{f}_g(\varepsilon, x) = (ax(1 + \varepsilon_1) + b)\,(1 + \varepsilon_2)\ ,$$

während (4.64) eine andere Approximation, nämlich

$$\tilde{f}_h(\varepsilon, x) = (ax + b)(1 + \varepsilon_1)(1 + \varepsilon_2)\ ,$$

ergibt. Welche ist besser? Dazu schauen wir uns den jeweiligen Auswertungsfehler $f(x_0) - \tilde{f}(\varepsilon, x_0)$ an. Zu dessen Quantifizierung verwenden wir das *relative Fehlerkonzept*[26]. Wir erhalten einerseits

$$\frac{|f(x_0) - \tilde{f}_g(\varepsilon, x_0)|}{|f(x_0)|} = \frac{|ax_0\varepsilon_1 + (ax_0 + b)\varepsilon_2|}{|ax_0 + b|} + o(\max\{|\varepsilon_1|, |\varepsilon_2|\})$$

und andererseits

$$\frac{|f(x_0) - \tilde{f}_h(\varepsilon, x_0)|}{|f(x_0)|} = |1 - (1 + \varepsilon_1)(1 + \varepsilon_2)| = |\varepsilon_1 + \varepsilon_2| + o(\max\{|\varepsilon_1|, |\varepsilon_2|\})\ .$$

Die Version (4.64) erlaubt also insbesondere im Falle $\frac{|ax_0|}{|ax_0+b|} \gg 1$ (Auslöschung!) deutlich bessere Fehlerschranken als (4.63).

Vor diesem Hintergrund wollen wir jetzt die Stabilität von Algorithmen zur Funktionsauswertung definieren. Dabei betrachten wir nur Störungen der Form (4.65), also Rundungsfehler. Zu deren Quantifizierung verwenden wir die Abkürzung

$$\|\varepsilon\| = \max_{i=1,\ldots,n} |\varepsilon_i|\ , \qquad \varepsilon = (\varepsilon_1, \ldots, \varepsilon_n)\ .$$

Bekanntlich gilt $\|\varepsilon\| \leq eps$ nach Satz 1.11 in Abschnitt 1.4.4.

[26] Die Betrachtung von Rundungsfehlern (4.65) ist in natürlicher Weise mit dem relativen Fehlerkonzept verbunden (vgl. Satz 1.11 in Abschnitt 1.4.4).

Definition 4.3 (Relative Stabilität) *Es sei $f(x_0) \neq 0$. Dann ist die relative Stabilität $\sigma_{\rm rel}$ von Algorithmus* (4.62) *gegenüber Rundungsfehlern* (4.65) *die kleinste Zahl $\sigma_{\rm rel}$ mit der Eigenschaft*

$$\frac{|f(x_0) - \tilde{f}(\varepsilon, x_0)|}{|f(x_0)|} \leq \sigma_{\rm rel}\|\varepsilon\| + o(\|\varepsilon\|) \; . \tag{4.67}$$

Liegt (4.67) *für keine reelle Zahl $\sigma_{\rm rel}$ vor, so wird $\sigma_{\rm rel} = \infty$ gesetzt.*

Die Stabilität $\sigma_{\rm rel} = \sigma_g(x_0)$ hängt nur von der Zerlegung (4.62) in Elementarfunktionen $g = (g_1, \ldots, g_n)$ und dem Argument x_0 ab. Die Stabilität ist eine Eigenschaft des zur Auswertung von f verwendeten Algorithmus.

Analog zu (3.49) ist Definition 4.3 gleichbedeutend mit

$$\sigma_{\rm rel} = \frac{1}{|f(x_0)|} \limsup_{\|\varepsilon\| \to 0} \frac{|f(x_0) - \tilde{f}(\varepsilon, x_0)|}{\|\varepsilon\|} \; .$$

Es kann sein, daß es kein $\rho > 0$ gibt, so daß $\tilde{f}(\varepsilon, x_0)$ für alle ε mit $\|\varepsilon\| \leq \rho$ auswertbar ist. Dann ist $\sigma_{\rm rel} = \infty$.

Allgemein gilt

$$\sigma_{\rm rel} \geq 1 \; , \tag{4.68}$$

denn bei Wahl von $\varepsilon_1 = \cdots = \varepsilon_{n-1} = 0$ ist offenbar

$$\frac{|f(x_0) - \tilde{f}(\varepsilon, x_0)|}{|f(x_0)|} = \frac{|f(x_0) - (1+\varepsilon_n)f(x_0)|}{|f(x_0)|} = |\varepsilon_n| = \|\varepsilon\| \; .$$

Beispielsweise hat der *triviale Algorithmus*

$$f(x_0) = g_1(x_0)$$

wegen

$$\frac{|f(x_0) - \tilde{f}(\varepsilon, x_0)|}{|f(x_0)|} = \frac{|g_1(x_0) - (1+\varepsilon)g_1(x_0)|}{|g_1(x_0)|} = |\varepsilon|$$

die Stabilität $\sigma_{\rm rel} = 1$.

Die Stabilität σ_g und σ_h der Algorithmen (4.63) und (4.64) ist gegeben durch

$$\sigma_g = 1 + \frac{|ax_0|}{|ax_0 + b|} \; , \qquad \sigma_h = 2 \; .$$

Im allgemeinen hat man es mit gestörte Eingabedaten $\tilde{x}_0$ zu tun. Die Stabilität von Algorithmus (4.62) ist dann $\sigma_{\rm rel}(\tilde{x}_0)$. Eingabefehler und Auswertungsfehler bewirken zusammen den Gesamtfehler

$$\frac{|f(x_0) - \tilde{f}(\varepsilon, \tilde{x}_0)|}{|f(x_0)|} \; .$$

Den Gesamtfehler können wir mit Hilfe der Kondition $\kappa_{\rm rel}$ des Problems (3.46) und der Stabilität $\sigma_{\rm rel}(\tilde{x}_0)$ kontrollieren.

Satz 4.4 *Es sei $x_0 \neq 0$ und $f(x_0) \neq 0$. Dann genügt der Gesamtfehler der Abschätzung*

$$\frac{|f(x_0) - \tilde{f}(\varepsilon, \tilde{x}_0)|}{|f(x_0)|} \leq \kappa_{\rm rel} \frac{|x_0 - \tilde{x}_0|}{|x_0|} + \sigma_{\rm rel}(\tilde{x}_0)\|\varepsilon\| + o(|x_0 - \tilde{x}_0| + \|\varepsilon\|) \; . \tag{4.69}$$

Beweis: Unter Verwendung der Definitionen von relativer Kondition und Stabilität berechnet man

$$\begin{aligned}
\frac{|f(x_0) - \tilde{f}(\varepsilon, \tilde{x}_0)|}{|f(x_0)|} &\leq \frac{|f(x_0) - f(\tilde{x}_0)|}{|f(x_0)|} + \frac{|f(\tilde{x}_0) - \tilde{f}(\varepsilon, \tilde{x}_0)|}{|f(\tilde{x}_0)|} \, \frac{|f(\tilde{x}_0)|}{|f(x_0)|} \\
&\leq \kappa_{\rm rel} \frac{|x_0 - \tilde{x}_0|}{|x_0|} + \sigma_{\rm rel}(\tilde{x}_0)\|\varepsilon\| \left(1 + \frac{|f(x_0) - f(\tilde{x}_0)|}{|f(x_0)|}\right) + o(|x_0 - \tilde{x}_0|) + o(\|\varepsilon\|) \\
&\leq \kappa_{\rm rel} \frac{|x_0 - \tilde{x}_0|}{|x_0|} + \sigma_{\rm rel}(\tilde{x}_0)\|\varepsilon\| + o(|x_0 - \tilde{x}_0| + \|\varepsilon\|) \; .
\end{aligned}$$

■

Eine obere Schranke für den Gesamtfehler ist also die Summe aus Eingabefehler, verstärkt durch die Kondition, und Auswertungsfehler, verstärkt durch die Stabilität.

Wir haben bislang nicht ausdrücklich ausgeschlossen, daß die elementaren Funktionen g_i vektorwertig sind oder Vektoren als Argumente haben. Dadurch sind insbesondere die Grundrechenarten $+$, $-$, $\cdot$, : mit jeweils zwei Argumenten zugelassen. Beispielsweise kann die Zerlegung

$$f(x) = ax^2 + bx = g_1(x) + g_2(x) \; , \qquad g_1(x) = ax^2 \; , \quad g_2(x) = bx \tag{4.70}$$

explizit in der Form (4.62) geschrieben werden, indem man

$$f(x) = h_2 \circ h_1(x) \; , \qquad h_1(x) = (g_1(x), g_2(x)) \; , \quad h_2(y, z) = y + z$$

setzt. Der Übersichtlichkeit halber bleiben wir aber bei der üblichen Schreibweise (4.70). Außerdem vereinbaren wir, daß von nun an alle elementaren Funktionen g_i oder h_i reellwertig sind.

Wir kommen nun zu Stabilitätsabschätzungen.

Satz 4.5 *Es sei*

$$h(x_0) = g \circ f(x_0) = g \circ g_n \circ g_{n-1} \circ \cdots g_1(x_0) \tag{4.71}$$

*ein Algorithmus zur Auswertung von $h(x_0)$. Bezeichnet dann κ_g die Kondition von g and der Stelle $y = f(x_0)$ und σ_f die Stabilität von Algorithmus (**??**), so für die Stabilität σ_h von (**??**) die Abschätzung*

$$\sigma_h leq \kappa_g \sigma_f + 1 \tag{4.72}$$

Beweis: Berücksichtigung der Auswertungsfehler ergibt

$$\tilde{h}(\varepsilon, x_0) = (1 + \varepsilon_g) g(\tilde{f}(\varepsilon_f, x_0))$$

mit $\varepsilon = (\varepsilon_f, \varepsilon_g)$, $\varepsilon_f = (\varepsilon_1, \ldots, \varepsilon_n)$ und $\varepsilon_g \in \mathbb{R}$. Verwendet man nun jeweils die Definitionen von Kondition und Stabilität, so erhält man

$$\begin{aligned}
\frac{|h(x_0) - \tilde{h}(\varepsilon, x_0)|}{h(x_0)} &\leq \frac{|g(f(x_0)) - g(\tilde{f}(\varepsilon_f, x_0))|}{g(f(x_0))} + |\varepsilon_g| \frac{|g(\tilde{f}(\varepsilon, x_0)|}{g(f(x_0))} \\
&\leq \kappa_g \frac{|f(x_0) - \tilde{f}(\varepsilon_f, x_0)|}{f(x_0)} + o\left(\frac{|f(x_0) - \tilde{f}(\varepsilon_f, x_0)|}{f(x_0)}\right) + |\varepsilon_g| + |\varepsilon_g| \frac{|g(f(x_0) - g(\tilde{f}(\varepsilon, x_0)|}{g(f(x_0))} \\
&\leq \kappa_g \sigma_f \|\varepsilon_f\| + |\varepsilon_g| + o(\|\varepsilon\|) \\
&\leq (\kappa_g \sigma_f + 1)\|\varepsilon\| + o(\|\varepsilon\|)
\end{aligned}$$

und damit die Behauptung. ■

Mit Hilfe von Satz **??** wollen wir nun die Stabilität der Approximation (4.66) durch die Kondition der elementaren Funktionen g_i abschätzen. Als Vorbereitung notieren wir folgendes Lemma.

Lemma 4.6 *Gegeben sei die inhomogene Differenzengleichungen erster Ordnung*

$$x_i = \alpha_i x_{i-1} + \beta_i \ , \quad i = 1, 2, \dots \ , \qquad x_0 = 0 \ , \tag{4.73}$$

mit nicht-negativen Koeffizienten $\alpha_i, \beta_i \in \mathbb{R}$, $i = 1, 2 \dots$. Dann genügen alle Folgen e_i mit der Eigenschaft

$$e_i \leq \alpha_i e_{i-1} + \beta_i \ , \quad i = 1, 2 \dots \ , \qquad e_0 = 0 \ , \tag{4.74}$$

der Abschätzung[27]

$$e_i \leq x_i = \sum_{j=1}^{i} K_{ij} \beta_j \ , \quad K_{ij} = \prod_{k=j+1}^{i} \alpha_k \ , \qquad i = 1, 2, \dots \ . \tag{4.75}$$

Beweis: Der Beweis erfolgt durch vollständige Induktion. Für $i = 1$ bestätigt man die Gültigkeit von (4.75) durch Einsetzen. Ist (4.75) für ein $i \geq 1$ richtig, so folgt

$$x_{i+1} = \alpha_{i+1} \sum_{j=1}^{i} K_{ij} \beta_j + \beta_{i+1} = \sum_{j=1}^{i+1} K_{i+1,j} \beta_j$$

durch Einsetzen in (4.73). Schließlich erhält man aus (4.74) und α_{i+1}, $\beta_{i+1} \geq 0$ sofort

$$e_{i+1} \leq \alpha_{i+1} e_i + \beta_{i+1} \leq \alpha_{i+1} x_i + \beta_{i+1} = x_{i+1}$$

und damit die Behauptung. ■

Satz 4.7 *Es bezeichne κ_i die relative Kondition der Auswertung der elementaren Funktion g_i an der Stelle y_{i-1}, und es sei $f(x_0) = g_n \circ \cdots \circ g_1(x_0)$ mit*

$$y_i = g_i(y_{i-1}) \ , \quad i = 1, \dots, n \ , \qquad y_0 = x_0 \ .$$

Dann gilt

$$\sigma_{\text{rel}} \leq \sum_{j=1}^{n} \prod_{i=j+1}^{n} \kappa_i \ = 1 + \kappa_n (1 + \kappa_{n-1} (1 + \dots \kappa_3 (1 + \kappa_2) \dots)) \ .$$

Beweis: Setzt man

$$f_i(x_0) = g_i \circ \cdots \circ g_1(x_0) \ , \qquad i = 1, \dots, n, \tag{4.76}$$

so ist offenbar $f_i(x_0) = g_i \circ f_{i-1}(x_0)$ und $f_n(x_0) = f(x_0)$. Anwendung von Satz 4.5 ergibt für die Stabilität σ_i von Algorithmus (4.76) die Ungeichungsrekursion

$$\sigma_i \leq \kappa_i \sigma_{i-1} + 1 \qquad i = 2, \dots, n \ .$$

Die Auswertung von g_1 an der Stelle x_0 hat wegen

$$\frac{|g_1(x_0) - \tilde{g}_1(\varepsilon_1, x_0)|}{|g_1(x_0)|} = |\varepsilon_1|$$

[27] Man beachte $\prod_{k=2}^{1} \alpha_k = 1$.

die Stabilität $\sigma_1 = 1$. Setzt man $\sigma_0 = 0$, so ist die Rekursion (**??**) auch für $i = 1$ richtig. Setzt man $\alpha_i = \kappa_i$ und $\beta_i = 1$, so folgt unmittelbar aus Lemma 4.6 die Abschätzung

$$\sigma_n \leq \sum_{j=1}^{i} K_{nj} , \qquad K_{nj} = \prod_{k=j+1}^{n} \kappa_k .$$

und damit die Behauptung. ■

Aus Satz 4.7 kann man zwei wichtige Regeln ablesen, die weit über den vorliegenden Fall hinaus allgemeine Gültigkeit haben.

> Schlecht konditionierte Elementarfunktionen vermeiden!

Als Beispiel betrachten wir die Funktion

$$f(x) = a - \sqrt{a^2 - x} , \qquad x \in (\infty, a^2]$$

mit gegebenem $a > 0$. Offenbar ist $y = f(x)$ gerade eine Nullstelle des quadratischen Polynoms

$$p(y) = y^2 - 2ay + x .$$

Einen naheliegenden Algorithmus zur Auswertung von f an der Stelle x_0 liefert die Zerlegung

$$f(x_0) = g_3 \circ g_2 \circ g_1(x_0) , \qquad g_1(x_0) = a^2 - x_0 , \quad g_2(y_1) = \sqrt{y_1} , \quad g_3(y_2) = a - y_2 . \tag{4.77}$$

Unter Verwendung von Satz 4.7 und

$$\kappa_2 = \frac{1}{2} , \qquad \kappa_3 = \frac{\sqrt{a^2 - x_0}}{|a - \sqrt{a^2 - x_0}|}$$

lässt sich die Stabilität σ_g von (4.77) wie folgt abschätzen:

$$\sigma_g \leq 1 + \kappa_3(1 + \kappa_2) = 1 + \frac{3\sqrt{a^2 - x_0}}{2|a - \sqrt{a^2 - x_0}|} .$$

Offenbar wird diese Schranke für $x_0 \to 0$ beliebig groß. Der Grund dafür ist die schlechte Kondition von g_3 (Auslöschung!). Zu deren Vermeidung schreiben wir f äquivalent um. Mit Hilfe der binomischen Formel $a^2 - b^2 = (a + b)(a - b)$ folgt nämlich

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

Der entsprechende Algorithmus lautet

$$f(x_0) = \frac{x_0}{h_3 \circ h_2 \circ h_1(x_0)} , \qquad h_1(x) = a^2 - x , \quad h_2(y_1) = \sqrt{y_1} , \quad h_3(y_2) = a + y_2 . \tag{4.78}$$

Auf ähnliche Weise wie zuvor erhalten wir nun aus Satz 4.8 und Satz 4.7 die obere Schranke

$$\sigma_h \leq 2 \left(\max \left\{ 1, 1 + \frac{3\sqrt{a^2 - x_0}}{2(a + \sqrt{a^2 - x_0})} \right\} \right) < 5 .$$

Satz 4.8 *Es sei $f(x_0) \neq 0$ sowie $g(x_0)$, $h(x_0) \neq 0$ und*

$$g(x_0) = g_n \circ g_{n-1} \circ \cdots \circ g_1(x_0) \ , \quad h(x_0) = h_m \circ h_{m-1} \circ \cdots \circ h_1(x_0)$$

Algorithmen zur Auswertung von $g(x_0)$ und $h(x_0)$ mit der relativen Stabilität σ_g, σ_h. Dann gilt jeweils

$$\begin{aligned}
f(x_0) = g(x_0) + h(x_0) \quad &: \qquad \sigma_{\text{rel}} \leq \max\{\sigma_g, \sigma_h\} \ , \\
f(x_0) = g(x_0) - h(x_0) \quad &: \qquad \sigma_{\text{rel}} \leq \frac{|g(x_0)| + |h(x_0)|}{|g(x_0) - h(x_0)|} \max\{\sigma_g, \sigma_h\} \ , \\
f(x_0) = g(x_0) \cdot h(x_0) \quad &: \qquad \sigma_{\text{rel}} \leq 2 \max\{\sigma_g, \sigma_h\} \ , \\
f(x_0) = g(x_0) / h(x_0) \quad &: \qquad \sigma_{\text{rel}} \leq 2 \max\{\sigma_g, \sigma_h\} \ ,
\end{aligned}$$

wobei in den ersten beiden Fällen $g(x_0)$, $h(x_0) > 0$ vorausgesetzt ist.

Beweis: Ist $\sigma_g = \infty$ oder $\sigma_h = \infty$, so bleibt nichts zu zeigen. Es sei also σ_g, $\sigma_h < \infty$. Wir betrachten nur den Fall $f(x_0) = g(x_0) \cdot h(x_0)$. Wir setzen

$$\tilde{f}(\varepsilon, x_0) = \tilde{g}(\varepsilon_g, x_0) \cdot \tilde{h}(\varepsilon_h, x_0)(1 + \varepsilon_f)$$

mit $\varepsilon = (\varepsilon_g, \varepsilon_h, \varepsilon_f)$. Mit den Abkürzungen

$$f = f(x_0) \ , \quad \tilde{f} = \tilde{f}(\varepsilon, x_0) \ , \quad g = g(x_0) \ , \quad \tilde{g} = \tilde{g}(\varepsilon_g, x_0) \ , \quad h = h(x_0) \ , \quad \tilde{h} = \tilde{h}(\varepsilon_h, x_0) \ ,$$

erhält man unter Verwendung von Satz 2.4 und (4.68) die Abschätzung

$$\begin{aligned}
\tfrac{|f-\tilde{f}|}{|f|} &\leq |\varepsilon_f| \left(1 + \tfrac{|g \cdot h - \tilde{g} \cdot \tilde{h}|}{|g \cdot h|}\right) + \tfrac{|g \cdot h - \tilde{g} \cdot \tilde{h}|}{|g \cdot h|} \leq |\varepsilon_f| + 2(1 + |\varepsilon_f|) \max\left\{\tfrac{|g-\tilde{g}|}{|g|}, \tfrac{|h-\tilde{h}|}{|h|}\right\} + o\left(\max\left\{\tfrac{|g-\tilde{g}|}{|g|}, \tfrac{|h-\tilde{h}|}{|h|}\right\}\right) \\
&\leq |\varepsilon_f| + 2(1 + |\varepsilon_f|) \max\left\{\sigma_g \|\varepsilon_g\| + o(\|\varepsilon_g\|), \sigma_h \|\varepsilon_h\| + o(\|\varepsilon_h\|)\right\} + o(\max\{\sigma_g \|\varepsilon_g\| + o(\|\varepsilon_g\|), \sigma_h \|\varepsilon_h\| + o(\|\varepsilon_h\|)\}) \\
&\leq 2 \max\left\{1, \sigma_g, \sigma_h\right\} \|\varepsilon\| + o(\|\varepsilon\|) = 2 \max\left\{\sigma_g, \sigma_h\right\} \|\varepsilon\| + o(\|\varepsilon\|)
\end{aligned}$$

und damit die Behauptung.

Die übrigen Aussagen folgen analog unter Verwendung der Sätze 2.2, 2.3 und 2.5. ■

Wir werfen noch einen Blick auf die Stabilitätsabschätzung in Satz 4.7. Offenbar taucht die Kondition κ_1 der Auswertung von $g_1(x_0)$ dort überhaupt nicht auf. Diese Beobachtung führt zu der zweiten Regel.

Unvermeidbare, schlecht konditionierte Elementarfunktionen an den Anfang!

Ein erstes Beispiel haben wir mit den Algorithmen (4.63) und (4.64) schon gleich zu Beginn dieses Abschnitts kennengelernt.

Wir wollen zum Abschluß das Polynomdesaster aus Abschnitt 4.1 genauer analysieren. Die in Algorithmus 4.1 implementierte Darstellung $f(x) = x^3 + 12a^2x - 6ax^2 - 8a^3$ basiert auf der Zerlegung[28]

$$f(x_0) = ((g_1(x_0) + g_2(x_0)) - g_3(x_0)) - g_4(x_0) \tag{4.79}$$

[28] Die Klammerung wählt MATLAB automatisch.

mit

$$g_1(x) = x^3 \ , \ \ g_2(x_0) = 12a^2x \ , \ \ g_3(x_0) = 6ax^2 \ , \ \ g_4(x_0) = 8a^3 \ .$$

Wir ignorieren also die bei der Auswertung der Elementarfunktionen g_i intern auftretenden Rundungsfehler. Um (dreimal) Satz 4.8 anwenden zu können, schreiben wir (4.79) rekursiv in der Form

$$g_{12}(x_0) = g_1(x_0) + g_2(x_0) \ , \quad g_{123}(x_0) = g_{12}(x_0) - g_3(x_0) \ , \quad f(x_0) = g_{123}(x_0) - g_4(x_0)$$

und erhalten mit Blick auf (**??**) für die Stabilität σ_{12} von g_{12}, σ_{123} von g_{123} und schließlich σ_g von (4.79) die Abschätzungen

$$\begin{aligned} \sigma_{12} &\leq \max\{\sigma_1, \sigma_2\} = 1 \ , \\ \sigma_{123} &\leq \frac{|g_{12}(x_0)| + |g_3(x_0)|}{|g_{12}(x_0) - g_3(x_0)|} \max\{\sigma_{12}, \sigma_3\} \approx 7 \ , \\ \sigma_g &\leq \frac{|g_{123}(x_0)| + |g_4(x_0)|}{|g_{123}(x_0) - g_4(x_0)|} \max\{\sigma_{123}, \sigma_4\} \approx 10^{21} \ . \end{aligned}$$

Dementsprechend müssen wir im Ergebnis mit einem relativen Fehler der Größenordnung

$$10^{21} eps = 10^{21} \cdot 10^{-16} = 10^5$$

rechnen. Also können die ersten 6 Stellen unbrauchbar sein. Genau das ist passiert.

Die äquivalente Umformulierung

$$f(x) = (x - 2a)^3$$

von (4.79) führt auf den Algorithmus

$$f(x_0) = h_2 \circ h_1(x_0) \ , \qquad h_1(x_0) = x_0 - 2a \ , \quad h_2(y_1) = y_1^3 \ . \tag{4.80}$$

Man beachte, daß die *unvermeidliche Auslöschung* nun in h_1 *am Anfang des Algorithmus* untergebracht ist. Direkte Anwendung von Satz 4.7 liefert für die Stabilität σ_h von (4.80) die Abschätzung

$$\sigma_h \leq 1 + \kappa_2 = 4 \ .$$

Das ist eine echte Verbesserung.

4.3 Drei-Term-Rekursionen

4.3.1 Rekursive Auswertung

Wir betrachten die Drei-Term-Rekursion (3.55) aus Abschnitt 3.3 mit vorgegebenen Anfangswerten x_{-1} und x_0. Ausnutzung der rekursiven Struktur führt unmittelbar auf die folgende Berechnungsvorschrift zur Auswertung von $f_k(x_0) = x_k$.

Algorithmus 4.9 (Rekursive Auswertung einer Drei-Term-Rekursion)

```
function f = REKDT(x0,K)

  x(-1) = xM;
   x(0) = x0;

  for k = 0:K-1
    x(k+1) = -a*x(k)-b*x(k-1);
  end

  f=x(K);

return
```

Gegenüber der geschlossenen Darstellung aus Lemma 3.8 hat Algorithmus 4.9 den Vorteil, daß er auch auf *inhomogene Drei-Term-Rekursionen mit variablen Koeffizienten* der Form (3.53) angewandt werden kann.

Wir wollen die Stabilitätseigenschaften von Algorithmus 4.9 untersuchen. Dabei berücksichtigen wir der Einfachheit halber nur die Rundung

$$\tilde{x}_{k+1} = \mathrm{rd}(x_{k+1}) = (-a\tilde{x}_k - b\tilde{x}_{k-1})(1+\varepsilon_k) \;, \qquad |\varepsilon_k| \le eps \;. \tag{4.81}$$

Mögliche Rundungsfehler oder gar Auslöschung bei der Berechnung von $-a\tilde{x}_k - b\tilde{x}_{k-1}$ werden ignoriert. Die gestörte Drei-Term-Rekursion (4.81) hat somit die variablen Koeffizienten $a_k = a(1+\varepsilon_k)$, $b_k = b(1+\varepsilon_k)$. Um den technischen Aufwand im Rahmen zu halten, nehmen wir vereinfachend an, daß in jedem Schritt derselbe Rundungsfehler ε gemacht wird, also

$$\varepsilon = \varepsilon_k \;, \quad |\varepsilon| \le eps \qquad \forall k = 0, 1, \ldots \;.$$

Dann wird aus (4.81) die Drei-Term-Rekursion

$$\tilde{x}_{k+1} = -a(1+\varepsilon)\tilde{x}_k - b(1+\varepsilon)\tilde{x}_{k-1} \;, \qquad \tilde{x}_1 = x_{-1} \;, \;\; \tilde{x}_0 = x_0 \;, \tag{4.82}$$

mit *konstanten Koeffizienten*. Das zugehörige *gestörte* charakteristische Polynom

$$\lambda^2 + \lambda a(1+\varepsilon) + b(1+\varepsilon)$$

hat die ε-abhängigen Nullstellen $\lambda_1(\varepsilon)$, $\lambda_2(\varepsilon)$. Offenbar sind dabei $\lambda_1(0)$, $\lambda_2(0)$ gerade die Nullstellen des exakten charakteristischen Polynoms (3.56). Unter der Voraussetzung $|\lambda_2(0)| > |\lambda_1(0)|$ aus Abschnitt 3.3 gilt ebenfalls $|\lambda_2(\varepsilon)| > |\lambda_1(\varepsilon)|$, falls $|\varepsilon|$ genügend klein ist. Wir nehmen an, daß

$$|\lambda_2(\varepsilon)| > |\lambda_1(\varepsilon)| \;, \qquad |\varepsilon| \le eps \;, \tag{4.83}$$

vorliegt. Dann ist nach Lemma 3.8 die gestörte Lösung $\tilde{x}_k$ gegeben durch

$$\tilde{x}_k = x_k(\varepsilon) = \alpha(\varepsilon)(\lambda_1(\varepsilon))^{k+1} + \beta(\varepsilon)(\lambda_2(\varepsilon))^{k+1} \;, \qquad \alpha(\varepsilon) = \frac{\lambda_2(\varepsilon)x_{-1} - x_0}{\lambda_2(\varepsilon) - \lambda_1(\varepsilon)} \;, \quad \beta(\varepsilon) = x_{-1} - \alpha(\varepsilon) \;. \tag{4.84}$$

Die so definierte Funktion

$$x_k : [-eps, eps] \ni \varepsilon \mapsto x_k(\varepsilon) \in \mathbb{R}$$

liefert für $\varepsilon = 0$ die exakte Lösung $x_k(0)$ der Drei-Term-Rekursion (3.55). In der Notation der vorausgegangenen Abschnitte 3.3 und 4.2 bedeutet das

$$f_k(x_0) = x_k(0) \ , \qquad \tilde{f}_k(\varepsilon, x_0) = x_k(\varepsilon) \ .$$

Offenbar ist $x_k(\varepsilon)$ differenzierbar in $\varepsilon = 0$. Damit können wir die Stabilität σ_k von Algorithmus 4.9 direkt angeben. Nach Definition der Ableitung gilt nämlich

$$\lim_{\varepsilon \to 0} \frac{x_k(\varepsilon) - x_k(0)}{\varepsilon} = x_k'(0)$$

und gleichbedeutend

$$x_k(\varepsilon) - x_k(0) = x_k'(0)\varepsilon + o(\varepsilon) \ .$$

Einsetzen ergibt

$$\frac{|f_k(x_0) - \tilde{f}_k(\varepsilon, x_0)|}{|f_k(x_0)|} = \frac{|x_k'(0)|}{|x_k(0)|}|\varepsilon| + o(\varepsilon) \ .$$

In direkter Analogie zu Satz 3.2 erhalten wir also

$$\sigma_k = \frac{|x_k'(0)|}{|x_k(0)|} \ .$$

Nun können wir das Hauptresultat dieses Abschnitts formulieren.

Satz 4.10 *Es sei $x_0 \neq 0$, $f_k(x_0) = x_k(0) \neq 0$ und es sei (4.83) erfüllt. Unter der Voraussetzung $x_0 \neq \lambda_1 x_{-1}$ gilt dann für genügend große k die Abschätzung*

$$\sigma_k \leq C \ k$$

mit einer Konstanten C, die nicht von k abhängt.
Im Falle $x_0 = \lambda_1 x_{-1}$ gilt für genügend große k

$$\sigma_k \geq c \left| \frac{\lambda_2}{\lambda_1} \right|^k$$

mit einer k-unabhängigen Konstanten c.

Beweis: Wir schreiben kurz $\lambda_i = \lambda_i(0)$ und $\lambda_i' = \lambda_i'(0)$, $i = 1, 2$, sowie $\alpha = \alpha(0)$, $\alpha' = \alpha'(0)$ und $\beta = \beta(0)$, $\beta' = \beta'(0)$. Außerdem setzen wir

$$q = \frac{\lambda_1}{\lambda_2} \ .$$

Wir beginnen mit dem Fall $x_0 \neq \lambda_1 x_{-1}$, nehmen also an, daß $\beta = (x_0 - \lambda_1 x_{-1})/(\lambda_2 - \lambda_1) \neq 0$ vorliegt. Mit Produkt- und Kettenregel folgt dann

$$\begin{aligned} x_k'(0) &= \alpha' \lambda_1^{k+1} + (k+1)\alpha \lambda_1^k \lambda_1' + \beta' \lambda_2^{k+1} + (k+1)\beta \lambda_2^k \lambda_2' \\ &= \lambda_2^k \left((\alpha' \lambda_1 + (k+1)\alpha \lambda_1') q^k + \beta' \lambda_2 + (k+1)\beta \lambda_2' \right) \ . \end{aligned}$$

Berücksichtigt man nun $|q| < 1$ und $k|q|^k \to 0$ für $k \to \infty$, so erhält man für genügend großes k die Abschätzung

$$|x_k'(0)| \leq c_1 k |\lambda_2|^k$$

mit einer geeigneten k-unabhängigen Konstanten c_1. Außerdem erhalten wir etwa aus (4.84)

$$x_k(0) = \alpha \lambda_1^{k+1} + \beta \lambda_2^{k+1} = \lambda_2^k (\alpha \lambda_1 q^k + \beta \lambda_2) \ .$$

Aus $\lambda_2 \neq 0$, $\beta \neq 0$ und $|q| < 1$ folgt damit für genügend großes k die Abschätzung

$$|x_k(0)| = |\lambda_2|^k \left| |\beta\lambda_2| - |\alpha\lambda_1||q|^k \right| \geq c_2 > 0$$

mit einer geeigneten k-unabhängigen Konstanten c_2. Einsetzen ergibt

$$\frac{|x_k'(0)|}{|x_k(0)|} \leq \frac{c_1 k}{c_2} = Ck$$

und damit die Behauptung.

Nun betrachten wir den Fall $\beta = 0$. Aus (4.84) erhält man unmittelbar $\alpha = x_{-1}$ und wegen $x_0 \neq 0$ ergibt sich daraus $\alpha \neq 0$ und $\lambda_1 \neq 0$. Etwas langwieriger ist der Nachweis von $\lambda_1' \neq 0$. Insgesamt erhält man schließlich $\beta' \neq 0$. Damit können wir $|x_k'(0)|$ nach unten abschätzen. Für genügend große k gilt

$$|x_k'(0)| = |\lambda_2|^k \left| (\alpha'\lambda_1 + (k+1)\alpha\lambda_1')q^k + \beta'\lambda_2 \right| \geq |\lambda_2|^k \left| |\beta'\lambda_2| - |\alpha'\lambda_1 + (k+1)\alpha\lambda_1'||q|^k \right| \geq c_3|\lambda_2|^k > 0$$

mit einer geeigneten k-unabhängigen Zahl $c_3 > 0$. Zur Abschätzung von $|x_k(0)|$ verwenden wir die Darstellung (3.57) aus dem Beweis von Satz 3.9. Mit elementaren Umformungen folgt

$$|x_k(0)| = |\lambda_2|^k \left| \frac{1-q^{k+1}}{1-q} \right| |x_0||q|^k \left| \frac{1-q}{1-q^{k+1}} \right| = |x_0||\lambda_2|^k|q|^k \ .$$

Zusammengenommen führen diese beiden Abschätzungen auf

$$\sigma_k = \frac{|x_k'(0)|}{|x_k(0)|} \geq \frac{c_3}{|x_0|}|q|^{-k} = c \left| \frac{\lambda_2}{\lambda_1} \right|^k \ .$$

Damit ist alles bewiesen. ■

Im *generischen Fall* $x_0 \neq \lambda_1 x_{-1}$ oder, gleichbedeutend, $\beta \neq 0$ wächst also die Auswirkung der Störung in jedem Schritt proportional zur Anzahl k der Schritte. Auch wenn die Konstante C groß sein kann, so ist das ein zumindest qualitativ akzeptables Verhalten. Wir nennen Algorithmus 4.9 daher stabil. Diese Einschätzung wollen wir anhand eines numerischen Beispiels überprüfen. Dazu betrachten wir wieder die Drei-Term-Rekursion (3.58)

$$4x_{k+1} - 4x_k - 3x_{k-1} = 0 \ , \qquad x_{-1} = 1$$

mit dem Anfangswert $x_0 = 1$ aus Abschnitt 3.3. Die relativen Fehler bleiben durchweg im Bereich

k	10	100	1000	1500
rel. Fehler	0	0	$0.1 \cdot 10^{-16}$	$0.1 \cdot 10^{-16}$

Tabelle 7: Rekursive Auswertung der Drei-Term-Rekursion im generischen Fall

der Maschinengenauigkeit und bestätigen damit die Stabilität von Algorithmus 4.9 im Falle $\beta \neq 0$.[29]

Ist $\beta = 0$, so haben wir es mit der Berechnung einer schlecht konditionierten, rezessiven Lösung zu tun (vgl. Abschnitt 3.3). Nach Satz 4.10 wächst dann die Stabilität selbst im besten Fall exponentiell! Algorithmus 4.9 ist instabil. Was das bedeutet, wollen wir anhand des numerischen Beispiels aus Abschnitt 3.3 illustrieren. In Abbildung 13 sieht man auf dem linken Bild die berechneten Folgenglieder $\tilde{x}_k$ über dem Index k. Bis etwa $k = 2950$ zeigt die Lösung zwar das qualitativ richtige Verhalten. Dann aber setzt sich der dominante Lösungszweig durch und macht die Ergebnisse völlig unbrauchbar. Daß es nicht so lange dauern muß, zeigt das Beispiel

$$x_{k+1} + 2x_k - x_{k-1} = 0 \ , \qquad x_{-1} = 1 \ , \quad x_0 = \sqrt{2} - 1 \ ,$$

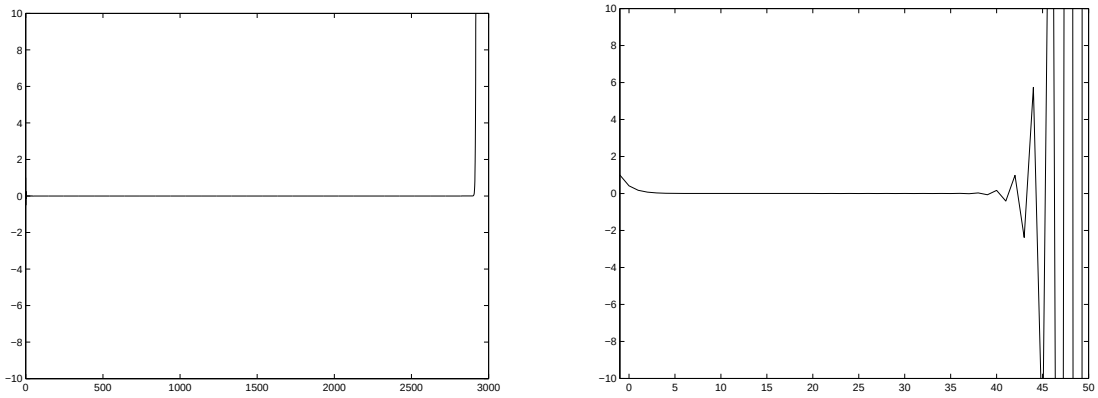


Abbildung 13: Instabilitäten von Algorithmus 4.9 bei rezessiven Lösungen

mit der rezessiven Lösung $x_k = x_0\lambda_1^k$, $\lambda_1 = \sqrt{2}-1$. Wie rechts in Abbildung 13 zu sehen ist, macht sich die Instabilität schon nach etwa 40 Schritten verheerend bemerkbar. Algorithmus 4.9 ist zur Approximation rezessiver Lösungen nicht zu gebrauchen! Das zeigt unsere theoretische Analyse, und das bestätigen auch die numerischen Rechnungen.

Was tun? Die geschlossene Darstellung aus Lemma 3.8 macht zwar keinerlei Stabilitätsprobleme, ist aber auf konstante Koeffizienten beschränkt. Im Falle variabler Koeffizienten müssen zur Berechnung rezessiver Lösungen spezielle, aufwendigere Algorithmen verwendet werden. Darum geht es im nächsten Abschnitt.

4.3.2 Der Algorithmus von Miller

Wir betrachten nach wie vor die Drei-Term-Rekursion (3.55) aus Abschnitt 3.3, konzentrieren uns dabei aber ausschließlich auf die Berechnung rezessiver Lösungen, also auf den Fall

$$x_0 = \lambda_1 x_{-1} \,, \qquad |\lambda_2| > |\lambda_1| > 0 \,. \tag{4.85}$$

Dabei sind x_{-1}, x_0 die gegebenen Anfangswerte und λ_1, λ_2 die Nullstellen des zugehörigen charakteristischen Polynoms 1.12. Mit Hilfe von Lemma 3.8 können wir dann zwar zu jedem x_0 die exakte Lösung

$$x_k = x_0\lambda_1^k \,, \qquad k = 1, \ldots, \tag{4.86}$$

direkt angeben, allerdings nur für konstante Koeffizienten. Auf der anderen Seite versagt der allgemein anwendbare rekursive Algorithmus 4.9 bei rezessiven Lösungen (3.59) wegen mangelnder Stabilität. Das haben wir im vorigen Abschnitt gesehen.

Einen Ausweg aus diesem Dilemma bietet die zu (3.55) gehörige Rückwärtsrekursion

$$y_{k-1} = -\frac{1}{b}\left(a y_k + y_{k+1}\right) \,, \quad k = n, \ldots, 0 \,, \tag{4.87}$$

mit gegebenen Anfangswerten y_n und y_{n+1}. Man beachte, daß sich $b \neq 0$ aus $\lambda_1 \neq 0$ ergibt.[30] Die

[29] Angesichts von nur 15 verfügbaren Stellen wächst der absolute Fehler allerdings auf über 10^{160} an.

[30] Selbstverständlich ließe sich (4.87) durch die Indextransformation $k' = n - k$ auch in der üblichen Form (3.55) schreiben.

Nullstellen des zur Rückwärtsrekursion (4.87) gehörigen charakteristischen Polynoms

$$\mu^2 + \frac{a}{b}\mu + \frac{1}{b} = 0$$

sind gerade

$$\mu_1 = \frac{1}{\lambda_1} , \qquad \mu_2 = \frac{1}{\lambda_2} ,$$

und aus (4.85) folgt

$$|\mu_1| > |\mu_2| .$$

Im Vergleich mit der Vorwärtsiteration dreht sich also das Wachstum der zu λ_1 bzw. λ_2 gehörenden Beiträge zur Lösung gerade um. Bei der Rückwärtsrekursion ist der zu $\mu_1 = \lambda_1^{-1}$ gehörige Lösungsanteil dominant! Wären x_n und x_{n+1} bekannt, so könnte man daher alle anderen Folgenglieder x_k, $k < n$, mit dem rekursiven Algorithmus 4.9 in stabiler Weise berechnen. Leider ist das gerade nicht der Fall.

Trotzdem können wir die stabile Rückwärtsrekursion (4.87) verwenden, um für jedes feste k_0 zumindest Näherungen $x_k^{(n)}$, $k = 1, \ldots, k_0$, zu konstruieren, welche mit wachsendem n die exakten Werte x_k immer genauer approximieren[31]. Dazu wählen wir die Anfangswerte

$$y_n = 1 , \qquad y_{n+1} = 0$$

und berechnen aus der Rückwärtsrekursion (4.87) die zugehörige Lösung

$$y_k = \alpha\mu_1^{n-k+1} + \beta\mu_2^{n-k+1} = \alpha\lambda_1^{-(n-k+1)} + \beta\lambda_2^{-(n-k+1)} , \qquad k = n-1, \ldots, 0 . \tag{4.88}$$

Das ist nach Satz 4.10 mit Algorithmus 4.9 *in stabiler Weise* möglich. Wir brauchen dazu weder λ_1, λ_2 noch α, β explizit zu kennen. Für die gewählten Anfangswerte $y_n = 1$ und $y_{n+1} = 0$ folgt aus der Lösungsdarstellung in Lemma 3.8 unmittelbar $\alpha \neq 0$. Zusammen mit $|\mu_1| > |\mu_2|$ ergibt sich daraus $y_0 \neq 0$ für genügend großes n. Wir setzen

$$x_k^{(n)} = x_0 \frac{y_k}{y_0} , \qquad k = 1, \ldots, k_0 .$$

Die Rechtfertigung dafür liefert der folgende Satz.

Satz 4.11 *Unter der Voraussetzung $x_k \neq 0$, $k = 1, \ldots, k_0$, gilt*

$$\lim_{n\to\infty} \frac{|x_k - x_k^{(n)}|}{|x_k|} = 0 , \qquad k = 1, \ldots, k_0 .$$

Beweis: Unter Verwendung von (4.86) erhalten wir

$$x_k - x_k^{(n)} = x_k - x_0\frac{y_k}{y_0} = x_k\left(1 - \lambda_1^{-k}\frac{y_k}{y_0}\right) .$$

Einsetzen von (4.88) ergibt wegen $\alpha \neq 0$

$$\lambda_1^{-k}\frac{y_k}{y_0} = \lambda_1^{-k}\frac{\alpha\lambda_1^{-(n-k+1)} + \beta\lambda_2^{-(n-k+1)}}{\alpha\lambda_1^{-(n+1)} + \beta\lambda_2^{-(n+1)}} = \frac{1 + \frac{\beta}{\alpha}\left(\frac{\lambda_1}{\lambda_2}\right)^{n-k+1}}{1 + \frac{\beta}{\alpha}\left(\frac{\lambda_1}{\lambda_2}\right)^{n+1}} \to 1 , \quad \text{für} \quad n \to \infty$$

[31] Wir wissen bereits, daß numerische Verfahren im allgemeinen nur (rundungs-) fehlerbehaftete Lösungen liefern. Daher ist es durchaus kein Nachteil, wenn ein Verfahren nicht die exakte Lösung, sondern beliebig genaue Approximationen liefert.

und damit die Behauptung. ■

Für genügend große n wird also x_k durch $x_k^{(n)}$ beliebig genau approximiert. Andererseits bedeutet ein großes n aber auch einen großen Rechenaufwand. Die Frage nach einem „genügend großen, aber nicht zu großen “ Genauigkeitsparameter n, führt auf die Frage nach möglichst genauen Schätzungen des Approximationsfehlers $|x_k - x_k^{(n)}|/|x_k|$ in Abhängigkeit von n. Wir wollen auf dieses Thema hier nicht eingehen und überlassen einfach dem Anwender die Wahl.

Der folgende Algorithmus berechnet eine Näherung des Folgenglieds x_k einer rezessiven Lösung der Drei-Term-Rekursion (3.55).

Algorithmus 4.12 (Miller, 1952)

```
function f = MILLER(x0,K,n)

  y(n) = 1;  y(n+1) = 0;

  for k = n:-1:0
    y(k-1) = -(a*y(k)+y(k+1))/b;
  end

  f = x0*y(K)/y(0);

return
```

Wir wollen Algorithmus 4.12 ausprobieren und betrachten dazu das Beispiel

$$x_{k+1} + 2x_k - x_{k-1} = 0\,, \qquad x_{-1} = 1\,, \quad x_0 = \lambda_1\,, \quad \lambda_1 = \sqrt{2} - 1\,,$$

an dem der rekursive Algorithmus 4.9 im vorigen Abschnitt so kläglich gescheitert ist. Zum Vergleich verwenden wir jeweils die geschlossene Lösungsdarstellung $x_k = \lambda_1^k$. Bei Wahl von $n = 510$ liefert Algorithmus 4.12 die folgenden Ergebnisse.

k_0	10	50	100	500
rel. Fehler	$0.2 \cdot 10^{-16}$	$0.1 \cdot 10^{-15}$	$0.2 \cdot 10^{-15}$	$0.3 \cdot 10^{-10}$

Tabelle 8: Approximation rezessiver Lösungen mit dem Miller-Algorithmus

Für $k_0 = 10, 50, 100$ erhalten wir also bis auf Maschinengenauigkeit das richtige Ergebnis. Für $k_0 = 500$ macht sich die Wahl von $n = 510$ durch nachlassende Genauigkeit bemerkbar. Für größere k_0 hat man auch n weiter zu vergrößern. Dabei stößt man auf eine neue Schwierigkeit: Bei Wahl von $n > 805$ liegt der zugehörige Wert von $y(0)$ außerhalb des darstellbaren Bereichs von Gleitkommazahlen. Jede Implementierung von Algorithmus 4.12, die nicht nur Testzwecken dienen, sondern anderweitige Verwendung finden soll, muß dem Rechnung tragen!

Im Gegensatz zu der Lösungsdarstellung aus Lemma 3.8 lässt sich der Algorithmus von Miller auf Drei-Term-Rekursionen mit variablen Koeffizienten verallgemeinern, wird dann allerdings etwas aufwendiger. Für Einzelheiten verweisen wir auf Kapitel 6 im Lehrbuch von Deuflhard und Hohmann [**?**].

4.4 Ausblick

Wir haben die Stabilität von Zerlegungen einer Funktion $f : I \to \mathbb{R}$ bezüglich Störungen der elementaren Funktionen g_i untersucht. Dabei haben wir uns auf Rundungsfehler konzentriert und deshalb das relative Fehlermaß zugrunde gelegt. In völliger Analogie hätten wir aber auch die absolute Stabilität definieren können.

Zerlegt man komplexe Simulationsprozesse in Teilschritte, so operieren die elementaren Funktionen $g_i : Y_{i-1} \to Y_i$ auf beliebigen Mengen, typischerweise auf Funktionenräumen. In den Teilschritten spielen dann nicht nur Rundungsfehler, sondern Messfehler, Modellfehler, algebraische Fehler oder Diskretisierungsfehler eine Rolle. Zu deren Quantifizierung hat man jeweils geeignete Fehlermaße zu entwickeln und zu verwenden.

Das alles ändert nichts an der grundsätzlichen Gültigkeit der Resultate dieses Abschnitts: Die Stabilität beschreibt die Akkumulation von Fehlern bei der rekursiven Auswertung der gestörten Elementarfunktionen. Maßgebend für die Stabilität ist Kondition und Reihenfolge der Elementarfunktionen.

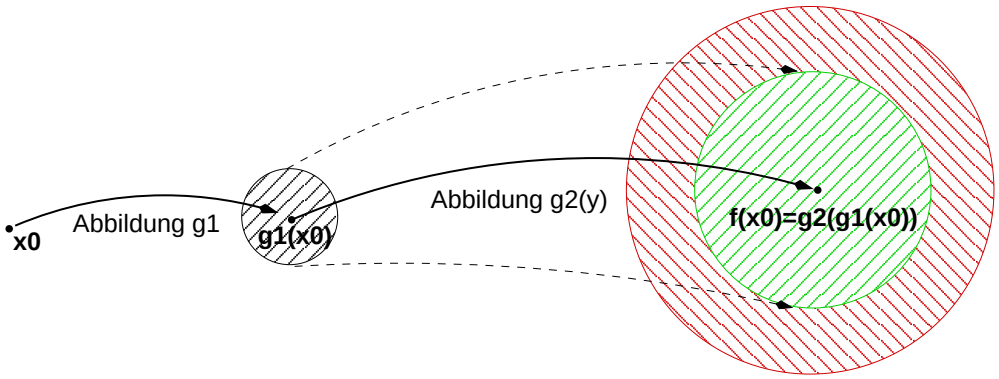


Abbildung 14: Die Stabilität der Auswertung von $f = g_1 \circ g_2 : X \to Y$ an der Stelle x_0

Dieser Sachverhalt ist in Abbildung 14 illustriert.

Teil II

Komplexität und Effizienz

In Teil I haben wir gesehen, daß reelle Zahlen durch Ziffernsysteme nicht exakt dargestellt werden können. Wir müssen sie auf die eine oder andere Weise durch rationale Zahlen approximieren. Dies führte uns auf konzeptionelle Probleme im Zusammenhang mit dem Runden, die wir durch Begriffe wie Kondition und Stabilität beschreiben und analysieren konnten. Wir haben uns entschieden, reelle Zahlen durch (generalisierte) Gleitkommazahlen mit fester Mantissenlänge zu approximieren, nicht nur da diese Zahlen effizient zu handhaben und die Operationen auf ihnen leicht zu analysieren sind, sondern auch, weil Gleitkommazahlen in der praktischen Realisierung in Computerprozessoren verwendet werden. Die Probleme, die beim Rechnen mit gerundeten Zahlen auftreten, liegen jedoch in der Natur der reellen Zahlen selbst und treten unabhängig von der konkreten Realisierung der gerundeten Zahlen auf, haben aber dennoch auch Einfluß auf die Darstellung dieser Zahlen im Computer.

So können Gleitkommazahlen mit einer endlichen Mantisse – so lang diese auch sein mag – nicht einmal jede rationale Zahl darstellen (vgl. das Patriot-Desaster). Periodische Brüche scheiden genauso aus wie zuvor irrationale Zahlen. Die Fesseln, die uns von den reellen Zahlen von selbst auferlegt wurden, geben wir an die rationalen Zahlen weiter. Dabei gibt es durchaus mathematische Probleme, bei denen keine irrationalen Zahlen auftauchen. Die Verwendung von Gleitkommazahlen bei ihrer Beschreibung führt nun zu den selben Stabilitätsproblemen, die bei der Approximation reeller Zahlen auftreten, obwohl dies in diesem Falle nicht „in der Natur“ des Problems liegt. Ein Beispiel für eine solche Fragestellung ist die Bestimmung der konvexen Hülle einer Punktmenge in der Ebene, deren Lösung wir in Abschnitt 7 behandeln werden. Bei der Ausführung der dort vorgestellten Algorithmen treten an keiner Stelle irrationale Werte auf, so lange die Koordinaten der gegebenen Punkte selbst ganzzahlige oder rationale Koordinaten besitzen. Dennoch ist die Bestimmung der konvexen Hülle einer Punktmenge der Ebene bei Verwendung von Gleitkommazahlen schlecht konditioniert, was zu verschiedenensten Rechenfehlern führt. Es stellt sich also die Frage, warum in der Praxis trotz der numerischen Probleme, die durch mangelnde Rechengenauigkeiten bei Gleitkommaarithmetik auftreten, oftmals Gleitkommazahlen an Stelle von Bruchzahlen der Form $\frac{a}{b}$ verwendet werden. Denn läßt man für a und b ganze Zahlen beliebiger Größe zu, so wie sie in Abschnitt 1.2.3 beschrieben wurden, so kann jede rationale Zahl exakt dargestellt werden.

Mit Einführung solcher variablen Zahlenkodierungen verliert man jedoch die Eigenschaft des Computers, elementare arithmetische Operationen wie Addition und Multiplikation in konstanter Rechenzeit erledigen zu können. Die Zeit zum Ausführen einer Rechenoperation nimmt stattdessen zu, wenn die Zahlen „länger“ werden. Die Algorithmen werden langsamer.

In diesem Kapitel soll es darum gehen, ein Maß für die Geschwindigkeit von Algorithmen zu entwickeln, das es erlaubt, verschiedene Algorithmen für ein Problem bezüglich ihrer Laufzeit zu vergleichen. Es kommt uns dabei nicht so sehr auf die exakte Zeit an, die der Algorithmus vom Start der Berechnung an benötigt, bis er das Ergebnis liefert. Diese mit einer Stoppuhr messbare Zeit hängt von vielen äußeren Umständen ab, z.B. dem Typ des Prozessors, der gewählten Programmiersprache, anderen Prozeßen, die zur selben Zeit auf der Maschine laufen, und so weiter. Viel interessanter ist es jedoch, zu analysieren, wie sich die Laufzeit eines Algorithmus mit der Problemgröße der gegebenen Probleminstanz verändert.

5 Der Euklidische Algorithmus

5.1 Darstellung von Bruchzahlen

Die Frage, wie man rationale Zahlen im Computer repräsentiert, so daß eine exakte Arithmetik auf der Menge $\mathbb{Q}$ ausgeführt werden kann, ist ein wichtiger Aspekt der Computeralgebra. Dies in allen Details darzustellen, würde an dieser Stelle zu weit führen. Wir wollen jedoch eine der auftretenden Schwierigkeiten etwas genauer untersuchen.

Rationale Zahlen lassen sich wie in Abschnitt1.3 beschrieben allgemein als Bruchzahlen $\frac{a}{b}$ darstellen, wobei a und b durch zwei beliebige ganze Zahlen gegeben sind. Diese Darstellung ist jedoch nicht eindeutig. Zunächst einmal stellt sich die Frage, wie mit Vorzeichen umzugehen ist, denn es gilt:

$$\frac{-1}{2} = \frac{1}{-2}, \qquad \frac{1}{2} = \frac{-1}{-2}.$$

Hier können wir eine Eindeutigkeit der Darstellung erzwingen, indem wir fordern, daß der Nenner b stets positiv sein muss.

Eine zweite Hürde stellt sich durch die Gleichheit von Bruchzahlen, die durch Erweitern und Kürzen durch verschiedene Brüche dargestellt werden können:

$$\frac{1}{2} = \frac{2}{4} = \frac{731}{1462}.$$

Brüche, deren Zähler und Nenner gekürzt werden können, treten insbesondere nach den Ausführen der arithmetischen Standardoperationen Addition und Multiplikation auf. Wir erinnern uns an die Rechenregeln (1.9) für Brüche:

$$\frac{a}{b} + \frac{a'}{b'} = \frac{ab' + a'b}{bb'}\ , \qquad \frac{a}{b} \cdot \frac{a'}{b'} = \frac{aa'}{bb'}\ .$$

Führt man nun Additionen und Multilikationen von Bruchzahlen mehrfach hintereinander aus, so erhält man immer größere Nenner (und im allgemeinen auch Zähler). Bedenkt man das eingangs Gesagte, daß Rechenoperationen wie Addition und Multiplikation um so länger dauern, je größer die zu verknüpfenden Zahlen sind, so wird klar, daß diese Operationen also nach jeder ausgeführten Operation langsamer werden. Eine quantitave Analyse dieses Effekts beschreiben wir in Abschnitt 6.3.

Neben der Verschlechterung der Laufzeit elemnentarer Operationen erschweren ungekürzte Brüche wegen ihrer nicht-eindeutigen Darstellung das Erkennen von Gleichheit verschiedener Brüche enorm.[32] So läßt sich erst durch kürzen erkennen, daß die Brüche $\frac{1557}{1813}$ und $\frac{2898}{3381}$ den gleichen Wert haben. Beide sind gleich der Bruchzahl $\frac{6}{7}$.

[32] Für das Vergleichen zweier beliebiger Bruchzahlen ist es hingegen hilfreich, sie auf den Hauptnenner zu bringen. Dazu erweitert man beide Brüche mit dem kleinsten gemeinsamen Vielfachen kgV beider Nenner. Zur Bestimmung des kleinsten gemeinsamen Vielfachen zweier Zahlen ist allerdings wiederum die Kenntnis des größten gemeinsamen Vielfachen hilfreich, denn es gilt der Zusammenhang

$$kgV(m,n) = \frac{m \cdot n}{\mathrm{ggT}(m,n)}.$$

Bevor zwei Brüche auf den Hauptnenner gebracht werden, ist es wieder aus den genannten Effizienzgründen empfehlenswert, sie beide vorher zu kürzen.

Um nun einen Bruch $\frac{a}{b}$ zu kürzen, teilen wir seinen Zähler a und seinen Nenner b jeweils durch den größten gemeinsamen Teiler $\mathrm{ggT}(a,b)$ von a und b. Wir erhalten eine eindeutige Darstellung für den gekürzten Bruch.

5.2 Der größte gemeinsame Teiler

Wir wollen nun die Berechnung des größten gemeinsamen Teilers für zwei ganze Zahlen a und b untersuchen. Dabei interessieren wir uns für die Anzahl der benötigten Rechenoperationen, unabhängig davon, welche Zeit diese einzelnen Operationen benötigen. Der Einfluß, den die Kodierungslänge der Zahlen dabei hat, wird von uns also zunächst vernachlässigt und erst später nachgeliefert. Außerdem nehmen wir im Folgenden an, daß $a \geq b$, was uns einige umständliche Formulierungen ersparrt. In allen Algorithmen, in denen dies eine Rolle spielt, kann diese Eigenschaft erreicht werden, indem als erstes a und b vertauscht werden, falls $b > a$ gilt.

Definition 5.1 (größter gemeinsamer Teiler – ggT) *Eine Zahl d, die zwei ganze Zahlen $a, b \in \mathbb{N}$ teilt ($d|a$ und $d|b$), heißt gemeinsamer Teiler von a und b. Die größte positive Zahl d, die gemeinsamer Teiler von a und b ist, heißt größter gemeinsamer Teiler von a und b oder kurz* $\mathrm{ggT}(a,b)$.

Sind die Zahlen a und b beide ungleich Null, so liegt der $\mathrm{ggT}(a,b)$ zwischen eins und $|b|$. Wir können ihn also durch stupides Ausprobieren herausfinden, indem wir alle Zahlen i von 1 bis b daraufhin untersuchen, ob sie a und b teilen. die größte dieser Zahlen ist dann unser gesuchter $\mathrm{ggT}(a,b)$.

Algorithmus 5.2 (ggT-Berechnung naiv)

```
Input: positive Zahlen a >= b>0
Output: ggT(a,b)

ggT := 1

for i = 2...b
     if i|a and i|b
          ggT := i
     endif
endfor

return ggT
```

Bei diesem trivialem Algorithmus führen wir insgesamt b Schleifendurchläufe aus, wobei in jedem Durchlauf zwei Teilbarkeitstests in Form von Divisionen durchgeführt werden. Ausserdem finden wir damit alle Teiler von a und b, obwohl wir uns ja nur für den größten interessieren. Deshalb erscheint es schlauer zu sein, die Schleife rückwärts zu durchlaufen und beim ersten (also größten) gefundenen gemeinsamen Teiler abzubrechen.

Algorithmus 5.3 (ggT-Berechnung rückwärts)

```
Input: positive Zahlen a >= b>0
```

```
Output: ggT(a,b)

for i := b...2
     if i|a and i|b
          return i
     endif
endfor

return 1
```

Ist z.B. b ein Teiler von a, so wird diese Variante des Algorithmus bereits nach dem ersten Schleifendurchlauf mit dem korrekten Ergebnis abbrechen. Also scheint dieser Algorithmus viel besser zu sein. Wenn jedoch $\mathrm{ggT}(a, b) = 1$ gilt, dann sind beide Algorithmen gleich schnell (oder besser gesagt: gleich langsam). Wir können also nicht garantieren, daß der zweite Algorithmus schneller ist als der erste. Und leider tritt genau dieser ungünstige Fall sehr häufig ein, öfter als man glauben mag. Somit sind für uns also beide bisher kennengelernten Algorithmen zur Bestimmung des ggT gleich schlecht. Ihre Laufzeiten wachsen im schlimmsten Fall jeweils linear mit dem Wert der Zahl b.

5.3 Der Euklidische Algorithmus

Es stellt sich die Frage, ob es vielleicht noch bessere Algorithmen gibt, die dieses Problem mit weniger Aufwand lösen können. Der beste bekannte Algorithmus ist der Euklidische Algorithmus, der auf ein bereits um 300 v.Chr. von Euklid in seinem Werk „Die Elemente" beschriebenen Verfahren beruht[33]:

> *Wenn CD aber AB nicht misst, und man nimmt bei AB, CD abwechselnd immer das kleinere vom größeren weg, dann muss (schließlich) eine Zahl übrig bleiben, die die vorangehende misst.*

Dieser Subtraktionsalgorithmus führt ebenfalls zu schlechten Laufzeiten, falls b im Vergleich zu a sehr klein ist oder beide Zahlen fast gleich groß sind. Deshalb verwendet man den Euklidischen Algorithmus heutzutage in der folgenden Divisions-Variante:

Algorithmus 5.4 (Euklidischer Algorithmus)

```
Input: positive Zahlen a >= b>0
Output: ggT(a,b)

m=a;
n=b;
while n>0
    r= m modulo n
```

[33] Das meiste in Eulids Elementen zusammengefasste mathematische Wissen hat einen weitaus älteren Ursprung. Euklids Leistung bestand jedoch unzweifelhaft darin, dieses Wissen zu sammeln und in einem umfassenden Werk systematisiert zu axiomatisieren. So ist auch das hier erwähnte Verfahren der gegenseitigen Wechselwegnahme („antepheiresis") bereits vor Euklid bekannt gewesen.

```
    m=n
    n=r
endwhile

return m
```

Die Korrektheit des Euklidischen Algorithmus beruht auf der Eigenschaft

$$\mathrm{ggT}(a, 0) = a \tag{5.1}$$

sowie der mehrfachen Anwendung des folgenden Satzes.

Satz 5.5 (Rekursionsatz für den ggT) *Für beliebige positive Zahlen $a, b \in \mathbb{N}$ gilt*

$$\mathrm{ggT}(a, b) = \mathrm{ggT}(b, a \bmod b).$$

Beweis der Korrektheit des Euklidischen Algorithmus: In jedem Schritt des Euklidischen Algorithmus werden nun die Zahlen m und n durch die Zahlen n und $m \bmod n$ ersetzt. Nach Satz 5.5 bleibt der ggT dabei unverändert. Da die Variable n in jedem Schritt echt kleiner aber niemals negativ wird, muss sie nach endlich vielen Schritten Null werden, womit der Algorithmus abbricht und den aktuellen Wert von m zurückliefert. Gleichung (5.1) sowie die wiederholte Anwendung von Satz 5.5 sagen aus, daß dieser Wert gleich dem ggT von a und b ist. ■

Für die Analyse des Euklidischen Algorithmus bedienen wir uns des Satzes von Lamé. Wir bezeichnen dabei die k-te Fibonacci-Zahl mit F_k. Die Fibonacci-Zahlen sind die Folgenglieder der durch

$$\begin{aligned} F_0 = 1, \qquad F_1 = 1, \\ F_{n+1} = F_{n-1} + F_n \quad \forall n = 1, 2, 3, ... \end{aligned} \tag{5.2}$$

bestimmten Drei-Term-Rekursion.

Satz 5.6 (Lamé, 1848) *Falls $a > b \geq 1$ und $b < F_{k+1}$ für eine natürliche Zahl k gilt, so benötigt der Euklidische Algorithmus höchstens k Schritte zur Bestimmung des* $\mathrm{ggT}(a, b)$.

Dieser Satz ist eine direkte Folgerung aus einer etwas genaueren Aussage:

Lemma 5.7 *Seien $a > b \geq 1$ zwei Zahlen, für die der Euklidische Algorithmus genau k Schritte (mit $k \geq 1$) zur Berechnung von* $\mathrm{ggT}(a, b)$ *benötigt. Dann gilt $a \geq F_{k+1}$ und $b \geq F_k$.*

Beweis: Wir führen eine vollständige Induktion über k. Als Induktionsanfang betrachten wir $k = 1$. Da nach Voraussetzung $1 \leq b < a$ gilt, ist die Behauptung $b \geq 1 = F_1$ trivialerweise erfüllt. Mit $a > b$ folgt direkt $a \geq 2 = F_2$, womit die Behauptung für $k = 1$ bewiesen ist.

Sei nun $k > 1$ und die Aussage für $k - 1$ bereits bewiesen. Im erstenSchritt werden nun $m = b$ und $n = a \bmod b$ gesetzt. Da der Algorithmus noch mindestens einen weiteren Schritt durchzuführen hat, gilt $n > 0$. Außerdem wissen wir aus der Definition des mod-Operators, daß $b > a \bmod b$, also $m > n$. Somit berechnen nun die weiteren Schritte den größten gemeinsamen Teiler $\mathrm{ggT}(m, n)$ von m und n in $k - 1$ Schritten. Nach Induktionsvoraussetzung gilt also $b = m \geq F_k$ und $(a \bmod b) = n \geq F_{k-1}$.

Da wir nach Voraussetzung $a > b > 0$ wissen, können wir aus den Eigenschaften der Modulo-Rechnung nun wegen $\lfloor a/b \rfloor \geq 1$ die Ungleichung

$$b + (a \bmod b) = b + (a - \lfloor a/b \rfloor b) \leq a$$

folgern. Setzen wir unsere Kenntnis aus der Induktionsvoraussetzung ein, so erhalten wir

$$a \geq b + (a \bmod b) \geq F_k + F_{k-1} = F_{k+1},$$

was den letzten Teil unserer Behauptung beweist. ■

Der Satz von Lamé sagt aus, daß der Euklidische Algorithmus höchstens soviele Schritte zur Bestimmung des $\mathrm{ggT}(a, b)$ benötigt, wie es Fibonacci-Zahlen kleiner oder gleich b gibt. Diese Schranke ist auch tatsächlich scharf, d.h. wir können Zahlenbeispiele finden, in denen auch exakt so viele Schritte benötigt werden.

Lemma 5.8 *Für jedes $k \geq 1$ benötigt der Euklidische Algorithmus exakt k Schritte zur Bestimmung von* $\mathrm{ggT}(F_{k+1}, F_k) = 1$.

Beweis: Auch dieses Lemma läßt sich am besten durch vollständige Induktion über k beweisen. Für $k = 1$ haben wir den $\mathrm{ggT}(2, 1)$ zu bestimmen. Dies erfolgt mit dem Euklidischen Algorithmus in genau einem Schritt.

Sei also $k > 1$ und die Behauptung für $k-1$ bereits bewiesen. Aufgrund der Definitionsgleichung (5.2) der Fibonacci-Zahlen gilt,

$$F_{k+1} \bmod F_k = F_{k-1} \tag{5.3}$$

und somit erfolgt die Berechnung des $\mathrm{ggT}(F_{k+1}, F_k)$ mittels des Euklidischen Algorithmus in dem im ersten Schritt mittels

$$\mathrm{ggT}(F_{k+1}, F_k) = \mathrm{ggT}(F_k, (F_{k+1} \bmod F_k) = \mathrm{ggT}(F_k, F_{k-1}) \tag{5.4}$$

auf den letzteren ggT zurückgeführt wird, der dann nach Induktionsvoraussetzung in $k-1$ Schritten berechnet wird.

Da $\mathrm{ggT}(2, 1) = 1$, liefert uns die Rekursionsgleichung (5.4) zusätzich, daß $\mathrm{ggT}(F_{k+1}, F_k) = 1$ für alle $k \in \mathbb{N}$. ■

Abbildung 15 zeigt die Werte des in den einzelnen Schritten des Euklidischen Algorithmus berechneten Rests r bei der Division für zwei verschiedene Eingabe-Paare a und b. Die rechte Abbildung zeigt dabei das Verhalten für die Eingaben $a = F_{11} = 144$ und $b = F_{10} = 89$. Es werden 10 Schritte benötigt, wobei der errechnete Rest in jedem Schritt jeweils die nächst kleinere Fibonacci-Zahl darstellt.

Satz 5.9 *Der Euklidische Algorithmus benötigt zur Bestimmung des* $\mathrm{ggT}(a, b)$ *für $a \geq b \geq 1$ höchstens* $\log_\Phi(b) + 1$ *Schritte, wobei* $\Phi = \frac{1+\sqrt{5}}{2}$.

Beweis: Betrachten wir zunächst den Fall $a = b \geq 1$. In diesem Fall wird genau ein Schritt ausgeführt. Wegen

$$1 = 0 + 1 = \log_\Phi 1 + 1 \leq \log_\Phi b + 1$$

stimmt die Aussage wohl. Sei also im folgenden $a > b$.

Da die Folge der Fibonacci-Zahlen monoton wächst, gibt es ein $k \geq 1$ mit der Eigenschaft $F_{k+1} > b \geq F_k$. Lemma 3.8 liefert uns als Lösung der bestimmenden Drei-Term-Rekursion für die Fibonacci-Zahlen die geschlossene Lösung

$$F_k = \frac{1}{\sqrt{5}} \left(\left(\frac{1+\sqrt{5}}{2} \right)^{k+1} - \left(\frac{1-\sqrt{5}}{2} \right)^{k+1} \right). \tag{5.5}$$

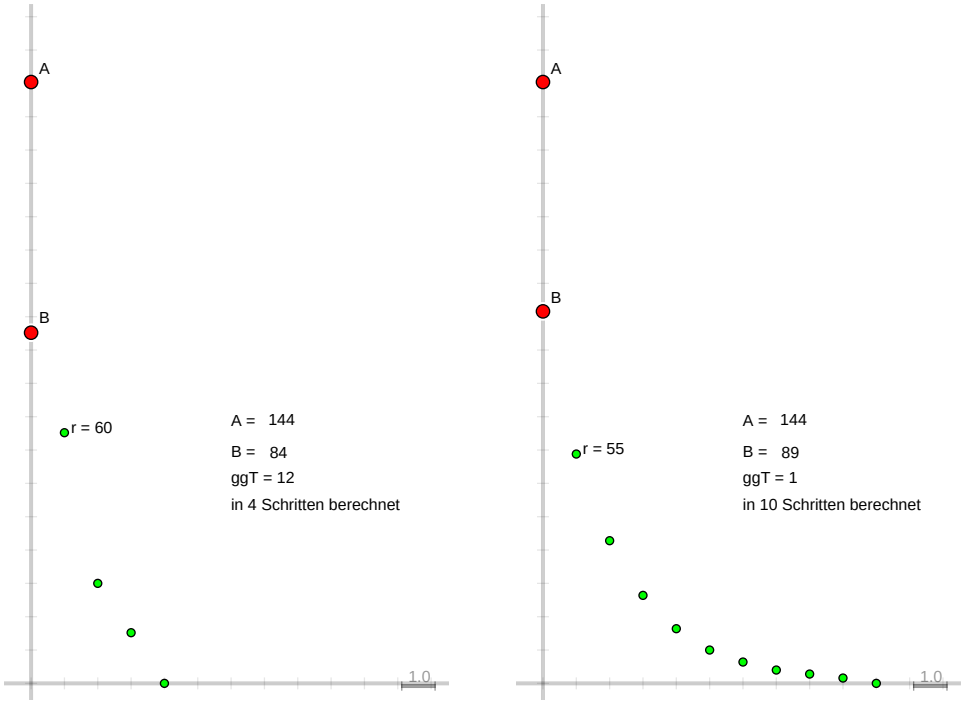


Abbildung 15: **Verhalten des Euklidischen Algorithmus.**

Durch geschicktes Umformen erhalten wir

$$
\begin{aligned}
F_k =& \frac{1+\sqrt{5}}{2\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^k - \frac{1-\sqrt{5}}{2\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^k \\
=& \frac{1+\sqrt{5}}{2\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^k - \frac{1-\sqrt{5}}{2\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^k \left(\frac{2}{1+\sqrt{5}}\right)^k \left(\frac{1+\sqrt{5}}{2}\right)^k \\
=& \left(\frac{1+\sqrt{5}}{2\sqrt{5}} - \frac{1-\sqrt{5}}{2\sqrt{5}}\left(\frac{1-\sqrt{5}}{1+\sqrt{5}}\right)^k\right)\left(\frac{1+\sqrt{5}}{2}\right)^k \\
\overset{(*)}{>}& \left(\frac{1+\sqrt{5}}{2\sqrt{5}} - \left|\frac{1-\sqrt{5}}{2\sqrt{5}}\left(\frac{1-\sqrt{5}}{1+\sqrt{5}}\right)^k\right|\right)\left(\frac{1+\sqrt{5}}{2}\right)^k \\
=& \left(\frac{1+\sqrt{5}}{2\sqrt{5}} - \frac{\sqrt{5}-1}{2\sqrt{5}}\left(\frac{\sqrt{5}-1}{1+\sqrt{5}}\right)^k\right)\left(\frac{1+\sqrt{5}}{2}\right)^k \\
=& \frac{1}{2\sqrt{5}}\left((1+\sqrt{5}) - (1-\sqrt{5})\left(\frac{1-\sqrt{5}}{1+\sqrt{5}}\right)^k\right)\left(\frac{1+\sqrt{5}}{2}\right)^k \\
\overset{(**)}{\geq}& \frac{1}{2\sqrt{5}}\left((1+\sqrt{5}) - (1-\sqrt{5})\frac{1-\sqrt{5}}{1+\sqrt{5}}\right)\left(\frac{1+\sqrt{5}}{2}\right)^k \\
=& \frac{1}{2\sqrt{5}}\left(\frac{(1+\sqrt{5})^2 - (1-\sqrt{5})^2}{1+\sqrt{5}}\right)\left(\frac{1+\sqrt{5}}{2}\right)^k \\
=& \frac{1}{2\sqrt{5}}\cdot\frac{4\sqrt{5}}{1+\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^k = \frac{2}{1+\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^k = \left(\frac{1+\sqrt{5}}{2}\right)^{k-1}.
\end{aligned}
$$

Die Ungleichung $(*)$ gilt, da für beliebige Zahlen x und y stets $x-y > x-|y|$. Die Korrektheit der zweiten Abschätzung $(**)$ sieht man ein, wenn man berücksichtigt, daß die Folge $\left(\left(\frac{1-\sqrt{5}}{1+\sqrt{5}}\right)^k\right)_{k\in\mathbb{N}}$ monoton fallend ist.

Zusammengefaßt ergibt sich also mit der Zahl $\Phi := \frac{1+\sqrt{5}}{2}$ des Goldenen Schnitts[34]

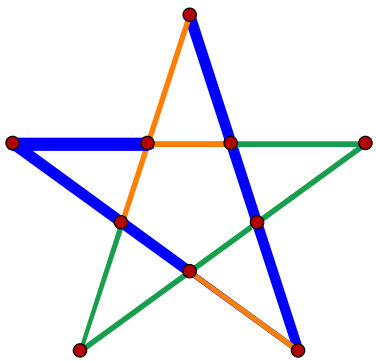
$$F_k > \Phi^{k-1} \qquad \forall k \geq 1. \tag{5.6}$$

Nach dem Satz von Lamé ist die Zahl $I(b)$ der Schritte zur Berechnung des $\mathrm{ggT}(a,b)$ aber höchstens k, also gilt

$$\begin{aligned} I(b) &\leq k = \log_\Phi(\Phi^k) = \log_\Phi(\Phi^{k-1} \cdot \Phi) = \log_\Phi(\Phi^{k-1}) + \log_\Phi(\Phi) \\ &< \log_\Phi(F_k) + 1 \leq \log_\Phi(b) + 1. \end{aligned}$$

■

Durch eine genauere Analyse läßt sich sogar zeigen, daß die Zahl der Schritte des Euklidischen Algorithmus durch $1 + \log_\Phi(b/\,\mathrm{ggT}(a,b))$ beschränkt ist. Es bleibt schließlich noch zu bemerken, daß der Euklidische Algorithmus für Eingaben mit $b = 0$ keinen Schritt ausführt.



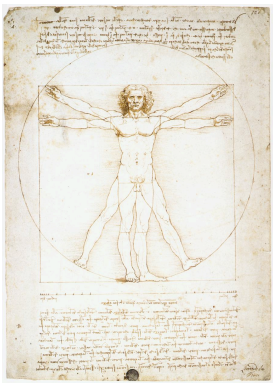


Abbildung 16: **Der Goldene Schnitt im Pentagramm und bei Leonardo da Vinci**

6 Die Asymptotische Notation

Viele Algorithmen sind so kompliziert, daß eine exakte Berechnung ihrer Laufzeit nicht oder nur schwer möglich ist, zumal diese von äußeren Rahmenbedingungen wie der verwendeten Programmiersprache oder dem Prozessor, auf dem das Programm ausgeführt wird, abhängt. Im allgemeinen ist es aber die Arbeit nicht wert, die genaue Laufzeit eines Algorithmus zu bestimmen, wenn multiplikative Faktoren oder Terme niederer Ordnung durch die Effekte, die durch wachsende Größe der Eingabedaten auftreten, dominiert werden. So haben wir bei der Aufwandsabschätzung des Euklidischen Algorithmus durch ausschweifende Umformungen und Abschätzungen den Einfluss des Terms $\left(\frac{1-\sqrt{5}}{2}\right)^k$ gegenüber dem des Terms $\left(\frac{1+\sqrt{5}}{2}\right)^k$ vernachlässigen können. Diese Analysen haben im Ergebnis dazu geführt, daß wir für die Anzahl der Schritte am Ende neben dem bestimmenden Term $\log_\Phi(b)$ eine additive Konstante von 1 erhalten haben. Letztendlich ist es aber bei

[34] Als Goldenen Schnitt bezeichnet man ein bestimmtes Zahlenverhältnis (meist im Zusammenhang mit Längenmaßen), dem in der Kunst eine besondere Ästhetik zugeschrieben wird. Dieses Verhältnis taucht in vielen geometrischen Figuren wie z.B. dem Pentagramm ebenso auf wie in der Natur. Die Irrationalität dieses Zahlenverhältnis wurde bereits um 450 v.Chr. von Hippasos von Metapont entdeckt. Der Goldene Schnitt, der seinen Namen 1835 von Martin Ohm erhielt, beschäftigt bis heute viele große Mathematiker und übte auch stets starken Einfluß auf die Kunst, die Architektur und die Musik aus.

ausreichend großen Zahlen a un b unerheblich, ob wir einen Schritt mehr oder weniger benötigen. Der Term $\log_\Phi(b)$ bestimmt letztendlich das Laufzeitverhalten unseres Algorithmus.

Wir interessieren uns folglich bei der Laufzeitanalyse von Algorithmen nur noch um die Größenordnungen, um die die Laufzeit mit der Größe des Inputs wächst. Dazu führen wir einen neuen Formalismus, die asymptotische Notation, ein.

6.1 Von kleinen ohs und großen Omegas

Definition 6.1 (Asymptotische Notation) *Seien $f : \mathbb{R}^+ \to \mathbb{R}^+$ und $g : \mathbb{R}^+ \to \mathbb{R}^+$ zwei Funktionen.*

a) *Wir sagen $f(x)$ wächst höchstens so schnell wie $g(x)$, falls*

$$\exists c > 0 \text{ und } \exists x_0 \in \mathbb{R}^+ \text{ mit } f(x) \leq c \cdot g(x) \text{ für alle } x \geq x_0 \tag{6.7}$$

und schreiben dafür kurz $f(x) = O(g(x))$ (sprich: „$f(x)$ ist groß Oh von $g(x)$ für $x \to \infty$“).

b) *Wir sagen, $g(x)$ wächst schneller als $f(x)$, falls es für jedes $c > 0$ ein $x_0 \in \mathbb{R}^+$ mit $f(x) < c \cdot g(x)$ für alle $x \geq x_0$ gibt, also*

$$\forall c > 0 \ \exists x_0 \in \mathbb{R}^+ \text{ mit } f(x) < c \cdot g(x) \text{ für alle } x \geq x_0, \tag{6.8}$$

und schreiben dafür kurz $f(x) = o(g(x))$ (sprich: „$f(x)$ ist klein oh von $g(x)$ für $x \to \infty$“).

c) *Entsprechend können wir auch untere Schranken für Funktionen angeben: Falls $g(x) = O(f(x))$, so schreiben wir $f(x) = \Omega(g(x))$ (sprich: „$f(x)$ ist groß Omega von $g(x)$ für $x \to \infty$“), und falls $g(x) = o(f(x))$, so schreiben wir $f(x) = \omega(g(x))$ („$f(x)$ ist klein omega von $g(x)$ für $x \to \infty$“).*

d) *Falls sowohl $f(x) = O(g(x))$ als auch $g(x) = O(g(x))$ gilt, also falls es zwei Konstanten $c_1 > 0$ und $c_2 > 0$ und ein $x_0 \in \mathbb{R}^+$ gibt, so daß $c_1 \cdot g(x) \leq f(x) \leq c_2 \cdot g(x)$ für alle $x \geq x_0$ gilt, so schreiben wir $f(x) = \Theta(g(x))$ (sprich: „$f(x)$ ist Theta von $g(x)$ für $x \to \infty$“). Die Funktionen $f(x)$ und $g(x)$ wachsen dann in der gleichen Größenordnung.*

Zur Bestimmung von Größenordnungen sind die Definitionen allein oft nicht hilfreich. Wir können uns jedoch ein Hilfsmittel aus der Analysis zu Nutze machen, das auf dem Grenzwert von Funktionen basiert.

Satz 6.2 *Seien $f, g : \mathbb{R}^+ \to \mathbb{R}^+$ zwei Funktionen. Dann gilt:*

a) *$f(x) = o(g(x))$ genau dann, wenn*

$$\lim_{x \to \infty} \frac{f(x)}{g(x)} = 0. \tag{6.9}$$

b) *$f(x) = O(g(x))$ genau dann, wenn*

$$\exists c > 0 \text{ mit } \lim_{x \to \infty} \frac{f(x)}{g(x)} \leq c. \tag{6.10}$$

c) *$f(x) = \Omega(g(x))$ genau dann, wenn*

$$\exists c > 0 \text{ mit } \lim_{x\to\infty} \frac{f(x)}{g(x)} \geq c. \tag{6.11}$$

d) *$f(x) = \Theta(g(x))$ genau dann, wenn*

$$\exists c_1, c_2 > 0 \text{ mit } c_1 \leq \lim_{x\to\infty} \frac{f(x)}{g(x)} \leq c_2. \tag{6.12}$$

e) *Insbesondere folgt:*

Falls es ein $c > 0$ gibt, so daß $lim_{x\to\infty}\frac{f(x)}{g(x)} = c$, so ist $f(x) = \Theta(g(x))$.

Beweis:

a) Wir verwenden die Definitionen von o und des Grenzwertes.

$$\begin{aligned}
& f(x) = o(g(x)) \\
\Leftrightarrow \quad & \forall c > 0\ \exists x_0 \in \mathbb{R}^+ \text{ mit } f(x) < c \cdot g(x) \text{ für alle } x \geq x_0 \\
\Leftrightarrow \quad & \forall c > 0\ \exists x_0 \in \mathbb{R}^+ \text{ mit } \frac{f(x)}{g(x)} < c \text{ für alle } x \geq x_0, \text{ da } g(x) > 0 \\
\Leftrightarrow \quad & \lim_{x\to\infty} \frac{f(x)}{g(x)} = 0.
\end{aligned}$$

b)-d) Die Aussagen folgen wiederum dierkt aus den Definitionen von O, Ω und Θ. Wir beweisen die Aussage speziell für Θ. Die anderen beiden Beweise erfolgen entsprechend durch geeignete Weglassung.
Betrachten wir also die Definition von Θ:

$$\begin{aligned}
& f(x) = \Theta(g(x)) \\
\Leftrightarrow \quad & \exists c_1, c_2 > 0 \exists x_0 > 0 : c_1 g(x) \leq f(x) \leq c_2 g(x) \text{ für alle } x \geq x_0 \\
\Leftrightarrow \quad & \exists c_1, c_2 > 0 \exists x_0 > 0 : c_1 = \frac{c_1 g(x)}{g(x)} \leq \frac{f(x)}{g(x)} \leq \frac{c_2 g(x)}{g(x)} = c_2 \text{ für alle } x \geq x_0 \\
\Leftrightarrow \quad & \exists c_1, c_2 > 0 : c_1 = \frac{c_1 g(x)}{g(x)} \leq \lim_{x\to\infty} \frac{f(x)}{g(x)} \leq \frac{c_2 g(x)}{g(x)} = c_2.
\end{aligned}$$

e) Für den Beweis der letzten Behauptung verwenden wir abermals die Definition des Grenzwertes

$$\begin{aligned}
& \lim_{x\to\infty} \frac{f(x)}{g(x)} = c \\
\Leftrightarrow \quad & \forall \epsilon > 0 \exists x_0 > 0 : \left| c - \frac{f(x)}{g(x)} \right| \leq \epsilon \text{ für alle } x \geq x_0 \\
\Leftrightarrow \quad & \forall \epsilon > 0 \exists x_0 > 0 : c - \epsilon \leq \frac{f(x)}{g(x)} \leq c - \epsilon \text{ für alle } x \geq x_0
\end{aligned}$$

Wählen wir nun speziell $\epsilon := \frac{c}{2} > 0$ und setzen $c_1 := c - \epsilon = \frac{c}{2} > 0$ und $c_2 := c + \epsilon = \frac{3}{2}c > 0$, so folgt also

$$\exists x_0 > 0 : c_1 \leq \frac{f(x)}{g(x)} \leq c_2 \text{ für alle } x \geq x_0$$

was nach dem zuvor Gezeigten $f(x) = \Theta(g(x))$ bedeutet.

■

Die erste Aussage von Satz 6.2 zeigt den Zusammenhang zwischen Definition 6.1b) und unserer früheren Definition 2.1 des Landau-Symbols o. Dazu wollen wir beide Definitionen verallgemeinern:

Definition 6.3 (Allgemeine Definition des Landau-Symbols) *Sei $I \subseteq \mathbb{R}$ ein Intervall und $A \in I$. Weiterhin seien $f : I \to \mathbb{R}$ und $g : I \to \mathbb{R}$ zwei Funktionen. Wir sagen, f wird in einer Umgebung von A durch g dominiert, falls*

$$\lim_{x \to A} \frac{f(x)}{g(x)} = 0 \tag{6.13}$$

und schreiben dafür $f(x) = o(g(x))$.

Satz 6.2a) sagt nun, daß für positive Funktionen diese Definition der aus Definition 6.1b) mit $A = \infty$ entspricht. Mit $A = 0$ und $g(x) = x$ erhalten wir andererseits unsere Definition 2.1.

Als Beispiel zum Umgang mit dieser Notation wollen wir uns nun Polynom-Funktionen ansehen. Wir zeigen zunächst, daß jeder Polynom $p(x) = a_n x^n + a_{n-1}x^{n-1} + ... + a_1 x^1 + a_0$ vom Grade n die gleiche Größenordnung wie das entsprechende Monom x^n vom gleichen Grade hat. Als erstes zeigen wir exemplarisch unter dierekter Verwendung von Definition 6.1a), daß $p(x)$ höchstens so schnell wächst wie x^n. Wir verwenden dazu mit der Dreiecksungleichung eine grundlegende Eigenschaft der Betragsfunktion. Außerdem wollen wir benutzen, daß positive Zahlen kleiner werden, wenn man sie durch eine Zahl, die größer als 1 ist, teilt. Deshalb setzten wir $x_0 = 1$. Dann gilt für alle $x \geq x_0$:

$$\begin{aligned} p(x) \leq |p(x)| &\leq |a_n|x^n + |a_{n-1}|x^{n-1} + ... + |a_1|x^1 + |a_0| \\ &\left(|a_n| + \frac{|a_{n-1}|}{x} + ... + \frac{|a_1|}{x^{n-1}} + \frac{|a_0|}{x^n}\right) x^n \\ &\leq (|a_n| + |a_{n-1}| + ... + |a_1| + |a_0|)x^n. \end{aligned}$$

Mit $x_0 = 1$ und $c_1 = |a_n| + |a_{n-1}| + ... + |a_1| + |a_0|$ folgt also $p(x) = O(x^n)$.

Für Polynome mit der zusätzlichen Eigenschaft $p(x) > 0$ für alle $x > 0$ zeigen wir nun unter Anwendung unseres gerade bewiesenen Grenzwert-Kriteriums sogar, daß beide Funktionen gleich schnell wachsen, daß also $p(x) = \Theta(x^n)$ ist.[35] Mit der Aussage aus Satz 6.2e) folgt unsere Behauptung aus der Tatsache, daß wegen $p(x) > 0$ für alle $x > 0$ insbesondere der Leitkoeffizient a_n positiv sein muss und somit

$$\begin{aligned} \lim_{x \to \infty} \frac{p(x)}{x^n} &= \lim_{x \to \infty} \frac{a_n x^n + a_{n-1}x^{n-1} + ... + a_1 x + a_0}{x^n} \\ &= \lim_{x \to \infty} \left(a_n + \frac{a_{n-1}}{x} + ... + \frac{a_1}{x^{n-1}} + \frac{a_0}{x^n}\right) \\ &= a_n > 0 \end{aligned}$$

gilt.

Wir sehen also, daß alle Polynome gleichen Grades in der gleichen Größenordnung wachsen. Auf der anderen Seite wird jedes Polynom von Polynomen mit größerem Grad dominiert, denn falls $p(x) = a_n x^n + a_{n-1}x^{n-1} + ... + a_1 x^1 + a_0$ ein Polynom vom Grad n und $q(x) = b_m x^m + b_{m-1}x^{m-1} +$

[35] Diese Aussage gilt für beliebige Polynome, die Einschränkung auf $p(x) > 0$ für $x > 0$ ist nicht notwendig. Allerdings haben wir unsere Definitionen und Aussagen bisher nur für positive Funktionen getroffen. Die Verallgemeinerung ist nicht besonders schwierig, erfordert jedoch einige technische Details, um Probleme mit negativen Koeffizienten und Nullstellen des Polynoms zu umgehen. Dies wollen wir uns und dem Leser an dieser Stelle ersparen.

$... + b_1 x^1 + b_0$ ein Polynom vom Grad $m > n$ ist, so gilt:

$$\begin{aligned}\lim_{x\to\infty} \frac{p(x)}{q(x)} &= \lim_{x\to\infty} \frac{a_n x^n + a_{n-1}x^{n-1} + ... + a_1 x^1 + a_0}{b_m x^m + b_{m-1}x^{m-1} + ... + b_1 x^1 + b_0} \\ &= \lim_{x\to\infty} \frac{\frac{a_n}{x^{m-n}} + \frac{a_{n-1}}{x^{m-n+1}} + ... + \frac{a_1}{x^{m-1}} + \frac{a_0}{x^m}}{b_m + \frac{b_{m-1}}{x^1} + ... + \frac{b_1}{x^{m-1}} + \frac{b_0}{x^m}} \cdot \frac{x^m}{x^m} \\ &= \lim_{x\to\infty} \frac{\frac{a_n}{x^{m-n}} + \frac{a_{n-1}}{x^{m-n+1}} + ... + \frac{a_1}{x^{m-1}} + \frac{a_0}{x^m}}{b_m + \frac{b_{m-1}}{x^1} + ... + \frac{b_1}{x^{m-1}} + \frac{b_0}{x^m}} \\ &= \frac{0}{b_m} = 0,\end{aligned} \tag{6.14}$$

also ist $p(x) = o(q(x))$. Insgesamt haben wir gezeigt:

Lemma 6.4 *Seien $p(x)$ und $q(x)$ zwei Polynome. Dann gilt:*

- $p(x) = \Theta(x^n)$, *wenn* $\operatorname{grad} p(x) = n$ *ist.*
- $p(x) = \Theta(q(x))$, *wenn* $\operatorname{grad} p(x) = \operatorname{grad} q(x)$ *ist.*
- $p(x) = o(q(x))$, *wenn* $\operatorname{grad} p(x) < \operatorname{grad} q(x)$ *ist.*

Das Landau-Symbol o läßt uns, wie wir gesehen haben, Größenordnungen unterscheiden. Wir können für zwei beliebige Funktionen $f(x)$, $g(x)$ entscheiden, ob sie gleich schnell wachsen, oder welche Funktion schneller wächst, also eine größere Größenordnung hat. Desshalb führen wir eine Ordnung der Größenordnungen ein, in dem wir $O(f(x)) < O(g(x))$ schreiben, falls $f(x) = o(g(x))$ ist. Für einige wichtige Funktionen, die bei der Laufzeitanalyse des öfteren eine Rolle spielen, ist diese Ordnung durch

$$O(1) < O(\log x) < O(x) < O(x \log x) < O(x^2) < O(x^n) < O(x^{\log x}) < O(2^x) \tag{6.15}$$

gegeben. Wie empfehlen, den Beweis dieser Abschätzungen als Übungen selbst durchzuführen.

6.2 Laufzeitanalysen

Die Laufzeit eines Computerprogramms, das ein (mathematisches) Problem löst, hängt von einer Vielzahl von Faktoren ab. Einige dieser Faktoren sind

- der Algorithmus, also das Verfahren, mit dem das Problem gelöst wird;
- die Eingabedaten, für die das Problem gelöst werden soll;
- die Programmiersprache und die tatsächliche Implementation des Algorithmus auf dem Computer;
- der Prozessor, auf dem das Programm ausgeführt wird.

Die letzten beiden dieser Faktoren sind nicht von mathematischer Natur sondern rein technisch und sind deshalb nicht theoretisch zu analysieren. Streichen wir diese äußeren Faktoren aus der Laufzeitanalyse, so bleibt noch immer die Untersuchung des Laufzeitverhaltens von Algorithmen in Abhängigkeit von den Eingabedaten. Dabei muss man sich nun auf ein geeignetes Aufwandsmaß einigen. Im allgemeinen betrachtet man dafür jeweils die Anzahl der elementaren Operationen wie Addition, Multiplikation oder Vergleich zweier Werte. Dabei ist zu berücksichtigen, daß diese verschiedenen Typen von Operationen selbst unterschiedlich aufwendig auszuführen sind. Da wir aber nur an der Größenordnung des zeitlichen Aufwands interessiert sind, betrachten wir bei der Aufwandsanalyse nur die jeweils relevanten Operationen. Dies können die jeweils zeitaufwendigsten Operationen[36] oder die mit einer Häufigkeit von größter Größenordnung sein. Bei der Bestimmung des größten gemeinsamen Teilers mittels der Algorithmen 5.2 bis 5.4 sind dies z.B. die Divisionen mit Rest, die bei der Überpfung der Teilbarkeitseigenschaft bzw. bei der Modulo-Rechnung verwendet werden.

Wir können nun jedem Algorithmus $\mathcal{A}$ eine Laufzeitfunktion $L_{\mathcal{A}}$ zuordnen, die für jede Instanz $\mathcal{I}$ von Eingabedaten die Laufzeit $L_{\mathcal{A}}(\mathcal{I})$ (also die Anzahl der relevanten Operationen) angibt. In der Praxis haben jedoch nicht alle Eingabedaten den gleichen Einfluß auf die Laufzeit, meistens hängt diese nur von ganz bestimmten, meist positiven Werten, die diese Instanz beschreiben. Dies kann z.B. die Anzahl der Ein- und Ausgabewerte, ihre Summe oder ihr größter Wert sein. Beim Euklidischen Algorithmus 5.4 zur Bestimmung des $\mathrm{ggT}(a,b)$ war es z.B. die kleinere der beiden positiven Zahlen a und b, wie wir in Satz 5.9 gesehen haben.

Das Ziel der Laufzeitanalyse ist es, die Größenordnung der Laufzeitfunktion $L_{\mathcal{A}}(\mathcal{I})$ möglichst genau zu bestimmen. Wir suchen also eine Funktion $f(I)$, so daß $L_{\mathcal{A}}(\mathcal{I}) = \Theta(f(I))$. Ist es uns nicht möglich, so ein $f(I)$ exakt zu finden, so geben wir uns damit zufrieden, ein möglichst kleines $f(x)$ zu finden, für das wenigstens $L_{\mathcal{A}}(\mathcal{I}) = O(f(x))$ gilt.

Beschränkt man sich jedoch bei der Analyse nur auf ganz bestimmte Parameter der Eingabedaten, so gibt es sicherlich eine ganze Menge verschiedener Eingabeinstanzen, die durch die gleichen Parameter beschrieben werden. Analysieren wir z.B. den Euklidischen Algorithmus in Abhängigkeit des Wertes b (also des kleineren Wertes der beiden Zahlen a und b), so ist dieser Parameter b bei der Berechnung von $\mathrm{ggT}(24,12)$ und von $\mathrm{ggT}(19,12)$ jeweils gleich 12, dennoch benötigt der Euklidische Algorithmus im ersten Fall nur einen Schritt zur Berechnung des ggT, im zweiten Fall jedoch fünf Schritte.

In der Algorithmischen Mathematik haben sich deshalb verschiedene Methoden zur Analyse von Algorithmen entwickelt. Am weitesten verbreitet ist die sogenannte ,,Worst-Case-Analyse". Dabei untersucht man für jeden Algorithmus die Laufzeit des ungünstigsten Falls, der für die verschiedenen Parameter gewählt werden kann. So erhält man Laufzeiten, die man keines Falls überschreitet. Im Beweis zu Satz 5.9 haben wir eine solche Worst-Case-Analyse für die Anzahl der Schritte des Euklidischen Algorithmus durchgeführt, wobei sich der ungünstigste Fall ergeben hat, wenn als Eingabewerte zwei aufeinanderfolgende Fibonacci-Zahlen gewählt wurden.

Satz 6.5 *Der Euklidische Algorithmus zur Bestimmung des* $\mathrm{ggT}(a,b)$ *für* $a \geq b \geq 1$ *hat eine Worst-Case-Laufzeit* $L_{euklid}(a,b)$ *der Größenordnung* $\Theta(log(b))$.

[36] Grundsätzlich sind Punkt-Operationen wie Multiplikation $\cdot$ und Division : als zeitauwendiger einzuschätzen als die Strich-Operationen Addition + und Subtraktion −.

Beweis: Nach Satz 5.9 benötigt der Euklidische Algorithmus maximal $\log_\Phi(b)+1$ Schritte. In jedem Schritt wird eine Division mit Rest ausgeführt. Somit benötigt der Algorithmus also maximal $\log_\Phi(b)+1$ Dvisionen. Die Laufzeit kann also durch $L_{\text{euklid}}(a,b) \leq c \cdot (\log_\Phi(b))$ abgeschätzt werden. Somit ist $L_{\text{euklid}}(a,b) = O(\log(b))$.

Da diese Abschätzung nach Lemma 5.8 scharf ist, benötigt der Euklidische Algorithmus im Worst-Case auch $\Omega(\log(b))$ Divisionen. Also ist insgesamt $L_{\text{euklid}}(a,b) = \Theta(log(b))$. ■

Neben der Worst-Case-Analyse ist auch die Average-Case-Analyse von Interesse, bei der die erwartete Laufzeit für eine zufällig gewählte Instanz bestimmt wird, die aber unter Umständen beliebig lange überschritten werden kann. Es wird sozusagen über alle möglichen Instanzen zu einer fest gewählten Parametermenge gemittelt. Die Durchführung einer Average-Case-Analyse ist oftmals um einiges aufwendiger als die Worst-Case-Analyse, liefert dafür jedoch zum Teil die aussagekräftigeren Ergebnisse. Allerdings benötigt man für sie exakte Kenntnisse über die Verteilung der möglichen Eingabe-Instanzen.

6.3 Zur Laufzeit von Addition und Multiplikation von Dualzahlen beliebiger Länge

Wir wollen an dieser Stelle noch einmal auf die in Abschnitt 1.3.4 besprochene Möglichkeit eingehen, rationale Zahlen im Rechner exakt durch Paare von Dualzahlen beliebiger Länge entsprechend Abschnitt 1.2.3 darzustellen. Dabei wollen wir nun untersuchen, welche Rechenzeit für die beiden arithmetischen Grundoperationen Addition und Multiplikation benötigt wird.

Gegeben seien die beiden Zahlen $x = \frac{a}{b}$ und $y = \frac{c}{d}$, wobei a, b, c, d vier ganze Zahlen in der Darstellung aus 1.2.3 und b sowie d positiv sind. Der Einfachheit halber beschränken wir uns auf den Fall, daß auch a und c positiv und alle vier Zahlen a, b, c, d als Binärzahlen mit jeweils $n = \lceil \log_2(\max a, b, c, d) \rceil$ Ziffern gegeben sind.

Entsprechend der Rechenregeln (1.9) für Brüche führen wir also die Addition und die Multiplikation von x und y auf Additionen und Multiplikationen von a, b, c, d zurück:

$$x + y = \frac{a}{b} + \frac{c}{d} = \frac{ad + cb}{bd} \,, \qquad x \cdot y = \frac{a}{b} \cdot \frac{c}{d} = \frac{ac}{bd} \,.$$

Das heißt, für eine Addition von zwei Bruchzahlen werden drei Multiplikationen und eine Addition ganzer Zahlen benötigt, für eine Multiplikation zweier Bruchzahlen zwei Multiplikationen ganzer Zahlen. Anschließend ist jeweils einmal der Euklidische Algorithmus zum Kürzen der Zahlen durchzuführen.

Wir untersuchen deshalb zu erst, welchen Aufwand die elementaren Rechenoperation für zwei positive ganze Zahlen a und b mit einer Länge von n Bit haben. Wenn wir die schriftlichen Additions- und Multiplikationsverfahren wie in Abschnitt 1.1.4 anwenden, so müssen für die Addition alle n Stellen der beiden Zahlen sowie eventuell auftretende Überträge mit einander addiert werden. Es werden also insgesamt $2n$ Bit-Additionen vorgenommen, um das Ergebnis zu erhalten. Die Laufzeit der Addition ist folglich $\Theta(n)$ und die resultierende Summe hat eine Länge von maximal $n+1$ Bit. Bei der Multiplikation ist sogar jede Stelle der ersten mit jeder Stelle der zweiten Zahl zu multiplizieren und die einzelnen Ergebnisse geeignet zu addieren. Es ergibt sich eine Laufzeit des schriftlichen Multiplikationsalgorithmus von $\Theta(n^2)$, wobei das Produkt eine Länge von $2n$ Bit erreichen kann.

Mit einem Trick kann man die Multiplikation zweier n-stelliger Dualzahlen jedoch noch beschleu-

nigen. Seien dazu a und b zwei Dualzahlen, deren Stellenzahl n eine Zweierpotenz sei. Dies können wir durch Auffüllen mit führenden Nullen stets erreichen. Wir wollen die Zahlen nun so in jeweils zwei $\frac{n}{2}$-stellige Zahlen aufspalten, daß

$$a = \boxed{p \mid q}$$

$$b = \boxed{r \mid s}.$$

Das heißt, $a = p2^{\frac{n}{2}} + q$ und $b = r2^{\frac{n}{2}} + s$. Setzen wir nun

$$\begin{aligned} u &:= (p+q)(r+s), \\ v &:= pr, \\ w &:= qs, \end{aligned}$$

so können wir das gesuchte Produkt auch als

$$a \cdot b = v2^n + (u - v - w)2^{\frac{n}{2}} + w$$

darstellen. Wir haben die Multiplikation zweier n-stelliger Zahlen auf drei Multiplikationen zweier $\frac{n}{2}$-stelliger Zahlen[37] sowie zweier Multiplikationen mit Zweierpotenzen und einiger Additionen zurückgeführt. Die Additionen haben dabei einen Aufwand von $\Theta(n)$. Da das Multiplizieren einer Dualzahl mit einer Zweierpotenz nur dem Anhängen von entsprechend vielen Nullen entspricht, haben diese Operationen ebenfalls einen Aufwand von $\Theta(n)$. Die Laufzeit der Multiplikation zweier n-stelliger Zahlen hat also die dreifache Laufzeit einer Multiplikation von $\frac{n}{2}$-stelligen Zahlen gefolgt von einer Laufzeit von $\Theta(n)$ für das Zusammenfügen der Teilergebnisse. Wir können für die Laufzeit also eine Rekursionsgleichung der Form

$$L_{mult}(n) = 3L_{mult}(\frac{n}{2}) + \Theta(n) \tag{6.16}$$

angeben. Die bei dieser Rechnung auftrenden drei kleineren Produkte können wir entsprechend zerlegen und wiederum auf noch kleinere Produkte zurückführen, usw.

Zur Lösung einer solchen Rekursionsgleichung ziehen wir einen Satz zu Hilfe, der uns sagt, wie sich die Laufzeit eines Algorithmus verhält, wenn wir ein großes Problem in mehrere Teilprobleme aufteilen können, die zunächst einzeln gelöst und am Ende zur Gesamtlösung zusammengesetzt werden können.

Satz 6.6 (Aufteilungs- und Beschleunigungssatz) *Seien $a > 0, b, c$ natürliche Zahlen und sei folgende Rekursionsgleichung gegeben:*

$$\begin{aligned} f(1) &\leq \frac{c}{a} \\ f(a \cdot n) &\leq b \cdot f(n) + c \cdot n \quad \textit{für } n > 1 \end{aligned}$$

Dann gilt:

$$f(n) = \begin{cases} O(n), & \textit{falls } a > b \\ O(n \cdot \log_2(n)), & \textit{falls } a = b \\ O(n^{\log_a(b)}), & \textit{falls } a < b \end{cases}$$

[37] Zur Berechnung der Zahl u ist unter Umständen ein Produkt zweier $n+1$-stelliger Zahlen notwendig, falls bei der Summenbildung der beiden Klammern ein Übertrag-Bit entsteht. Dies macht eine etwas genauere Analyse notwendig, die jedoch am Ende auf das gleiche Ergebnis führt. Die unumgänglichen Deitails bei einer detaillierteren Analyse trragen jedoch nichts zum grundlegenden Verständnis des Sachverhalts bei und können gegebenenfalls z.B. in [8] nachgelesen werden.

Für den Beweis dieses Satzen verweisen wir den interessierten Leser auf weiterführende Literatur über die Analyse von Algorithmen, z.B. auf [3] oder auf [8].

Das Multiplizieren zweier 1-Bit-Dualzahlen benötigt genau einen Schritt, also ist $L_{mult}(1) = 1$. Da wir unseren Term $\Theta(n)$ durch $c \cdot n$ für ein genügend großes c angeben können, erfüllt unsere Laufzeitfunktion die Voraussetzungen des Aufteilungs- und Beschleunigungssatzes und wir erhalten eine Laufzeit für das Multiplizieren zweier n-Bit-Dualzahlen von

$$L_{mult}(n) = O(n^{\log_2(3)}) < O(n^{1,59}).$$

Wir konnten die Laufzeit der Multiplikation also durch Aufteilung in kleinere Probleme von $O(n^2)$ auf $O(n^{1,59})$ beschleunigen.

Zusammenfassend können wir nun die Laufzeiten der Addition und Multiplikation zweier Bruchzahlen bestimmen.

Satz 6.7 *Seien $x = \frac{a}{b}$ und $y = \frac{c}{d}$ zwei positive Brüche, die durch n-stellige Dualzahlen a, b, c, d gegeben sind. Dann gilt für die Laufzeiten $L_{add,\mathbb{Q}}$ der Addition und $L_{mult,\mathbb{Q}}$ der Multiplikation: $L_{add,\mathbb{Q}}(n) = O(n^{\log_2(3)})$ und $L_{mult,\mathbb{Q}}(n) = O(n^{\log_2(3)})$.*

Beweis: Für die Berechnung der Summe und des Produkts von x und y sind zunächst drei bzw. zwei Multiplikationen zweier n-stelliger Dualzahlen durchzuführen. Die Laufzeit dieser Multiplikationen beträgt $O(n^{\log_2(3)})$. Die anschließende Summenbildung bei der Addition hat mit $O(n)$ eine Laufzeit von geringerer Größenordnung und spielt deshalb keine Rolle. Abschließend ist zum Kürzen des Ergebnisses einmal der Euklidische Algorithmus auf Zähler und Nenner auszuführen. Als Eingabe für den Algorithmus sind eine $2n + 1$-Bit und eine $2n$-Bit lange Dualzahl gegeben. Nach Satz 6.5 hat der Euklidische Algorithmus somit eine Laufzeit von $\Theta(n)$. Seine Laufzeit fällt folglich ebenfalls nicht ins Gewicht. ■

6.4 Übungen

Aufgabe 6.1 *Man beweise $\log(n) = o(n)$ und $n^k = o(2^n)$ für jedes feste $k \in \mathbb{N}$.*

Man verwende als Hilfsmittel die „Regel von L'Hopital“:

$$\lim_{x\to\infty} \frac{f(x)}{g(x)} = \lim_{x\to\infty} \frac{\frac{d}{dx} f(x)}{\frac{d}{dx} g(x)}$$

7 Konvexe Hülle von Punkten in der Ebene

In Abschnitt 5 haben wir verschiedene Algorithmen zur Berechnung des größten gemeinsamen Teilers zweier natürlicher Zahlen betrachtet und auf ihre Laufzeit untersucht. Anschließend haben wir in Abschnitt 6 der Laufzeitanalyse von Algorithmen ein theoretisches Fundament basierend auf einem geeigneten Formalismus gegeben. Nun wollen wir als weitere Anwendung ein Problem aus der algorithmischen Geometrie betrachten, das eine breite Anwendung in der Computergrafik und somit insbesondere in der Entwicklung von Computerspielen findet.

Eine Möglichkeit, komplexe Objekte wie z.B. in Computerspielen häufig vorkommende Raumschiffe oder Kreaturen darzustellen, besteht darin, sie durch Polygone anzunähern. Je genauer die Darstellung sein soll (und je realistischer die Objekte damit aussehen sollen), umso feiner muss diese

Annäherung sein, also umso mehr kleine Polygone werden benötigt. Die realistische Darstellung dieser vielen kleinen Polygone auf dem Bildschirm erfordert jedoch eine enorm große Rechenzeit, so daß animierte Grafiken umso langsamer werden, je realistischer eine Computergrafik sein soll. Vor allem bei Computerspielen, bei denen die Grafik in ,,Echtzeit" berechnet werden muss, stellt das selbst moderne PCs vor eine fast unlösbare Aufgabe. Um den Computer bei dieser rechenintensiven Arbeit zu entlasten, möchte man die nötigen Berechnungen nur für jene Objekte durchführen, die auch auf dem Bildschirm sichtbar sind, also nicht durch andere Objekte verdeckt werden.

Nun ist aber auch die Entscheidung, ob ein kompliziertes Objekt durch ein anderes verdeckt wird, eine sehr aufwendige Aufgabe, die wiederum viel Rechenzeit erfordert. Um diese Frage schneller entscheiden zu können, bedient man sich eines Tricks. Statt die Frage zu klären, ob ein kompliziertess Objekt A durch ein zweites Objekt B verdeckt wird, vereinfacht man das Objekt A zu einem veränderten Objekt $\bar{A}$, das das Objekt A vollständig enthält aber in seiner Struktur möglichst einfach ist. Wird bereits das Objekt $\bar{A}$ durch B verdeckt, so gilt das erst recht für das originale Objekt A.

Das gesuchte Objekt $\bar{A}$ ist nun gerade durch die sogenannte konvexe Hülle von A gegeben, also der kleinsten konvexen Menge, die A enthält.

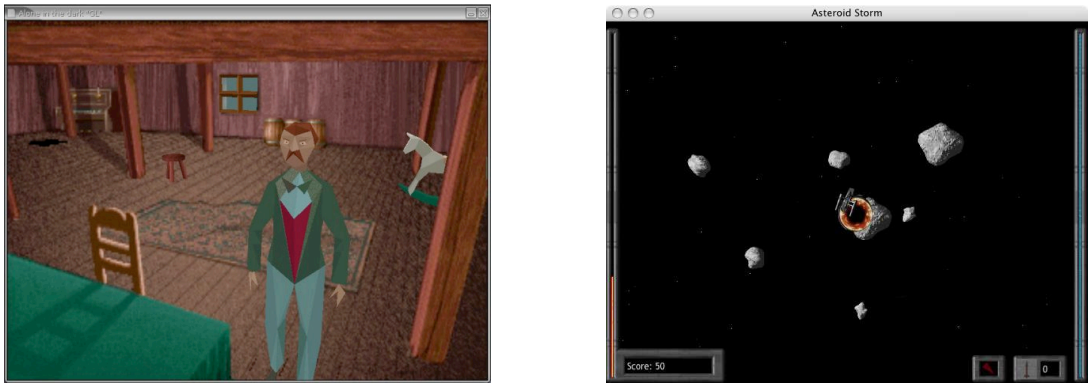


Abbildung 17: **Konvexe Polygone in der Computer**
Konvexe Polygone werden zur Darstellung komplexer Objekte sowie zur Erkennung von Kollisionen verwendet. (Screenshots aus den Computerspielen *Free in the Dark*, einem *Alone in the Dark*-Klon, sowie aus *Asteroid Storm*).

Eine weitere, sehr ähnliche Anwendung ist die Kollisionserkennung, die z.B. bei Flusimulationen eine Rolle spielt. Hierbei gilt es zu überprüfen, ob zwei komplexe Objekte wie z.B. Flugzeuge, Raketen oder Raumschiffe sich berühren oder sogar ineinander dringen. Auch hier wird die i.A. rechenintensive Bestimmung durch die Verwendung der konvexen Hüllen der Objekte vorentschieden. Sind bereits die konvexen Hüllen berührungsfrei, so gilt dies erst recht für die Objekte selbst.

Wir wollen uns deshalb nun mit der Bestimmung der konvexen Hülle einer Punktmenge beschäftigen, wobei wir uns auf den einfach zu untersuchenden Fall einer Punktmenge in der zweidimensionalen Ebene beschränken wollen.

Gegeben sei also im weiteren eine endliche Menge $\mathcal{M} \subset \mathbb{R}^2$ von Punkten der Ebene. Gesucht ist die konvexe Hülle $\bar{\mathcal{M}} = conv(\mathcal{M})$ von $\mathcal{M}$, also die kleinste konvexe Mange, die alle Punkte von $\mathcal{M}$ enthält.

Anschaulich können wir uns vorstellen, die Punkte $P \in \mathcal{M}$ seien Nägel in einem Brett. Die konvexe Hülle erhalten wir nun, indem wir ein Gummiband über die Nägel ziehen, daß alle Nägel im Inneren des Bandes liegen. Lassen wir nun das Gummiband los, so zieht es sich so weit zusammen, bis es von einigen Nägeln aufgehalten wird. Diese Nägel sind die Eckpunkte der konvexen Hülle, die anderen Punkte, die das Gummiband nicht berühren, sind innere Punkte. Zwischen den Eckpunkten verläuft das Gummiband entlang von geraden Segmenten.

Zur Bestimmung der konvexen Hülle einer Punktmenge reicht es uns deshalb, die Eckpunkte oder die begrenzenden Segmente zu bestimmen.

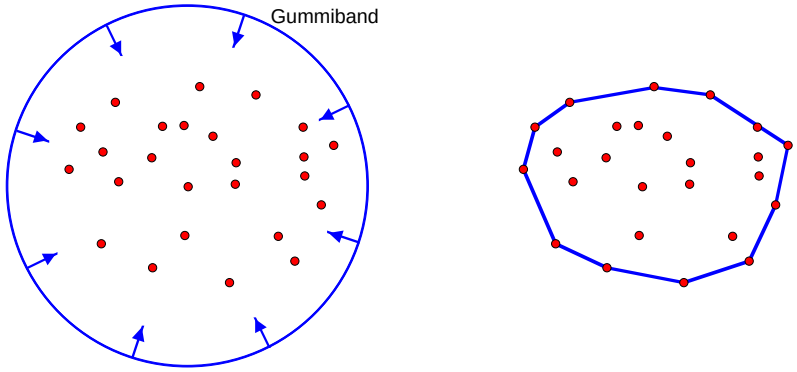


Abbildung 18: **Konvexe Hülle eine Punktmenge mittels Gummiband**

7.1 Ein einfacher Algorithmus

Unser Ziel ist es nun, einen Algorithmus zu entwerfen, der genau die Eckpunkte der konvexen Hülle findet und diese im mathematisch positiven Sinn (also gegen den Uhrzeigersinn) ausgibt. Wir wollen zunächst überlegen, was die Randsegmente der konvexen Hülle von den anderen unterscheidet. Dazu betrachten wir einmal die Trägergeraden der Randsegmente. Es fällt auf, daß alle Punkte der Punktmenge $\mathcal{M}$ auf der selben Seite der Trägergeraden liegen. Betrachten wir hingegen eine Gerade, die durch einen der inneren Punkte verläuft, so gibt es stets auf beiden Seiten der Geraden Punkte der Punktmenge. Die Ursache dafür liegt in der Tatsache, daß jede Gerade die euklidische Ebene in zwei konvexe Halbebenen zerlegt. Dies ist eine elementare Eigenschaft der Geometrie der euklidischen Ebene und hat ihre Ursache im Axiom von Pasch[38]. Für tiefergehende geometrische Studien der euklidischen Ebene verweisen wir auf [7].

Wir können unser Problem also lösen, in dem wir für jedes Verbindungsgerade $g = PQ$ durch zwei Punkte P und G aus $\mathcal{M}$ überprüfen, ob alle Punkte von $\mathcal{M}$ in der selben Halbebene von g liegen. Da wir die Eckpunkte später im mathematisch positiven Sinn ausgeben wollen, ist es sinnvoll die Geraden PQ zwischen zwei Punkten P und Q zu orientieren. Für eine orientierte Gerade schreiben wir $\vec{PQ}$ und stellen uns vor, wir stünden in P und würden in Richtung Q gucken. Ist $\vec{PQ}$ nun die Trägergrade eines Randsegments der konvexen Hülle von $\mathcal{M}$, so müssen alle anderen Punkte R aus $\mathcal{M}$ links von $\vec{PQ}$ liegen, damit wir den Rand der konvexen Hülle gegen den Uhrzeigersinn

[38] Max Pasch (1843-1930) wurde nach seinem Studium in Berlin Mathematik-Professor in Gießen. Er war ein Wegbereiter der Axiomatik in der Geometrie, und legte somit den Grundstein für David Hilberts späteres Axiomensystem der Geometrie.

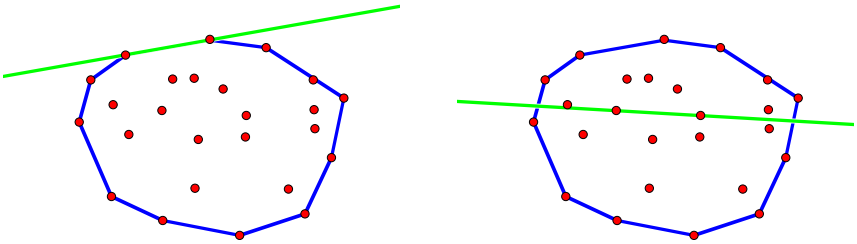


Abbildung 19: **Randgeraden und innere Geraden.** Geraden durch innere Punkte separieren die Punktmenge.

durchlaufen. Eine einfache Möglichkeit die Lage eines Punktes R bezüglich einer Geraden $\vec{PQ}$ zu bestimmen, ist durch die Berechnung des Winkels $\angle PQR$ gegeben. Hat dieser Winkel einen Wert zwischen 0 und π, so liegt R rechts von $\vec{PQ}$, ist der Wert des Winkels zwischen π und 2π, so liegt R links von $\vec{PQ}$. Bei einem Winkel von 0 oder π sind P, Q und R kollinear, also auf einer gemeinsamen Geraden gelegen. Lemma 7.1 zeigt uns eine weitere Möglichkeit, die Lage von R zu bestimmen, die auf die rechnerisch aufwendige Winkelberechnung verzichtet. Dazu nehmen wir an, die Punkte $P, Q, R \in \mathbb{R}^2$ seien im üblich Karthesisischen Koordinatensystem durch $P = (x_P, y_P)$, $Q = (x_Q, y_Q)$ und $R = (x_R, y_R)$ gegeben.

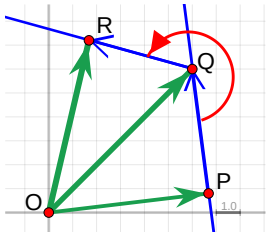


Abbildung 20: **Linksdrehung**

Lemma 7.1 *Seien $P, Q, R \in \mathbb{R}^2$ drei Punkte der Ebene und sei*

$$\omega := \det \begin{pmatrix} x_P & x_Q & x_R \\ y_P & y_Q & y_R \\ 1 & 1 & 1 \end{pmatrix} = x_P y_Q + x_Q y_R + x_R y_P - x_P y_R - x_R y_Q - x_Q y_P. \tag{7.17}$$

Dann gilt:

$$R \text{ liegt genau dann } \begin{cases} \text{links von } \vec{PQ}, & \text{wenn } \omega > 0; \\ \text{auf } \vec{PQ}, & \text{wenn } \omega = 0; \\ \text{rechts von } \vec{PQ}, & \text{wenn } \omega < 0 \end{cases}$$

Unser – zugegebener Maßen sehr einfacher – Konvexe-Hülle-Algorithmus sieht folglich in etwa so aus:

Algorithmus 7.2 (Konvexe Hülle)

procedure `Konvexe Hülle(())`
Input : (endliche) Punktmenge $\mathcal{M}$ der Ebene.
Output: Menge H der Eckpunkte der konvexen Hülle von $\mathcal{M}$ im mathematisch positiven Sinn sortiert.

begin
  $K := \emptyset$;
  for all $((P,Q) \in \mathcal{M}$ *mit* $P \neq Q)$ **do**
    if (*kein Punkt* $R \in \mathcal{M}$ *liegt rechts von* $\vec{PQ}$) **then**
      $K := K \cup \{\overline{PQ}\}$;
    end
  end
  $H :=$ eine mathematisch positiv sortierte Liste aus den in K vorkommenden Eckpunkten.
  return H;
end

Satz 7.3 *Sei $\mathcal{M}$ eine n-elementige Punktmenge. Dann berechnet Algorithmus 7.2 die konvexe Hülle von $\mathcal{M}$ in einer Laufzeit von $\Theta(n^3)$.*

Beweis: Es gibt $n \cdot (n-1)$ Punktpaare (P,Q) mit Punkten aus $\mathcal{M}$. Für jedes Punktepaar muss die Lage von jedem der n Punkte aus $\mathcal{M}$ geprüft werden. Somit sind insgesamt $n \cdot n \cdot (n-1) = \Theta(n^3)$ Lageprüfungen notwendig, die entsprechend Lemma 7.1 in konstanter Zeit durchgeführt werden können.

Schließlich muss noch aus den in K enthaltenen Kanten die Liste H der Eckpunkte erzeugt werden. Dies kann durch geeignete, hier nicht näher erleuterte Sortierverfahren in einer Laufzeit von $O(n^2)$ oder je nach verwendetem Algorithmus sogar in $O(n\log(n))$ erreicht werden (siehe z.B. [4]). Insgesamt wird die Laufzeit dieses Schrittes jedoch durch die Laufzeit der vorherigen Kanten-Suche dominiert. Die gesamt-Laufzeit des Algorithmus ist also $\Theta(n^3)$. ■

Wir haben also einen $\Theta(n^3)$-Algorithmus zur Berechnung der konvexen Hülle von $\mathcal{M}$ gefunden. Dieser Algorithmus ist jedoch durch das kubische Wachstum seiner Laufzeit mit der Anzahl der zu untersuchenden Punkte für praktische Anwendungen viel zu langsam. In dem Konvexe-Hülle-Applet kann man spüren, wie sich das Wachstum der Laufzeit bereits für wenige Punkte bemerkbar macht, indem das Applet mit zunehmender Anzahl der Punkte immer träger wird. Das bedeutet, daß wir für praktische Anwendungen einen besseren Algorithmus brauchen.

7.2 Der Graham-Algorithmus

Der Trick zur Verbesserung der Laufzeit ist es nun, bereits am Anfang des Algorithmus eine geeignete Sortierung der Punkte vorzunehmen und anschließend alle Punkte nacheinander in dieser Reihenfolge in die Betrachtung einzubeziehen.

Wir suchen zunächst einen Punkt ganz links in der Ebene, also den Punkt S mit der kleinsten x-Koordinate. Falls es mehrere linkeste Punkte gibt, nehmen wir einfach den ,,obersten" davon, also den mit der größten y-Koordinate.

Anschließend sortieren wir die Punkte $P \in \mathcal{M}\backslash\{S\}$ so, daß der Winkel, der zwischen einer gedachten

senkrechten Geraden durch S und der Geraden $\vec{SP}$ immer größer wird (siehe Abbildung 21).[39] Zur Bestimmung der Reihenfolge der Punkte in L können wir uns wiederum der Rechts-Links-Beziehung aus Lemma 7.1 bedienen, denn ein Punkt P muss genau dann vor einem Punkt Q in der Liste stehen, wenn Q links von $\vec{SP}$ liegt. Wir erhalten so eine sortierte Liste $L = (S = P_1, P_2, P_3, ..., P_n)$.

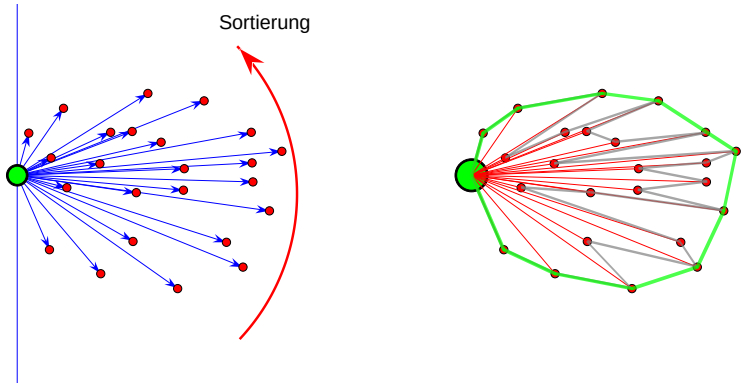


Abbildung 21: **Graham Algorithmus.** Sortierung der Punkte und Graham-Scan.

Mit dieser sortierten Liste wird nun der sogenannte Graham-Scan durchgeführt. Wir benötigen dafür eine weitere Liste H, die am Ende die Eckpunkte der konvexen Hülle enthalten wird. Zunächst werden in diese Liste H die ersten drei Punkte aus L eingefügt.

Iterativ wird nun immer der nächste Punkt R aus L betrachtet. Außerdem schauen wir auf den zuletzt in H eingefügten Punkt Q sowie den davor in H eingefügten Punkt P. Liegt nun R links von $\vec{PQ}$, so wird R in H eingefügt. Andernfalls wird Q aus H entfernt und es werden wiederum die beiden zuletzt in H eingefügten Punkte betrachtet. Dies wird solange wiederholt, bis alle Punkte aus L in H eingefügt wurden. Algorithmus 7.4 zeigt das Vorgehen noch einmal zusammenfassend.

[39] Sollte es mehrere Punkte geben, die im gleichen Winkel zu dieser Senkrechten liegen, so brauchen wir von diesen im Folgenden nur noch den am weitesten von S entfernten Punkt betrachten. Die anderen können keine Eckpunkte der konvexen Hülle sein. Allerdings ist dieses Hinauswerfen nichtbenötigter Punkte nicht zwingend erforderlich.

Algorithmus 7.4 (Graham-Algorithmus)

procedure `Konvexe Hülle(())`
Input : (endliche) Punktmenge $\mathcal{M}$ der Ebene.
Output: Menge H der Eckpunkte der konvexen Hülle von $\mathcal{M}$ im mathematisch positiven Sinn sortiert.

```
begin
    S := der linkeste (oberste) Punkt aus M;
    L := aufsteigend nach Winkeln sortierte Liste(S = P_1, P_2, ..., P_n);
    H := (P_1, P_2, P_3);
    for (i = 4 to n) do
        R := P_i aus S;
        repeat
            Q := zu letzt in H eingefügter Punkt;
            P := davor in H eingefügter Punkt;
            if (R liegt nicht links von PQ) then
                entferne Q aus H;
            end
        until (R liegt links von PQ);
        Füge R in H ein.
    end
    return H;
end
```

Wir wollen nun die Laufzeit von Algorithmus 7.4 untersuchen. wie wir feststellen werden, wird seine Laufzeit von dem Sortieren der Punkte am Anfang des Algorithmus dominiert. Deshalb müssen wir zunächst wissen, mit welcher Laufzeit wir die n Punkte sortieren können.

Lemma 7.5 *Seien L eine Liste mit n Elementen und $\preceq$ eine Ordnung auf der Menge der Elemente von L. Dann gibt es Sortieralgorithmen, die die Liste L in $\Theta(n \log(n))$ Vergleichen zweier Elemente bzgl. $\preceq$ sortieren.*

Ein Beispiel für einen solchen Sortieralgorithmus ist Merge-Sort, bei dem die zu sortierende Liste zunächst in zwei gleichlange Teillisten aufgeteilt wird. Anschließend werden beide nun halb so langen Teillisten ebenfalls mittels Merge-Sort sortiert und anschließend geeignet gemischt. Der Aufteilungs- und Beschleunigungssatz 6.6 liefert dann die gewünschte Laufzeit von $\Theta(n \log(n))$. Dieser und andere Sortieralgorithmen werden z.B. in [3] und [8] ausführlich besprochen.

Nun steht einer Laufzeitanalyse des Graham-Algorithmus nichts mehr im Wege.

Satz 7.6 *Der Graham-Algorithmus 7.4 bestimmt die Eckpunkte der konvexen Hülle einer Punktmenge $\mathcal{M}$ des $\mathbb{R}^2$ mit n Punkten in einer Laufzeit von $\Theta(n \log(n))$.*

Beweis: Zunächst wird der linkeste Punkt gesucht. Dazu muss jeder Punkt einmal auf seine Koordinaten untersucht werden, was durch insgesamt $\Theta(n)$ Koordinatenvergleiche erreicht wird. Das anschließende Sortieren aller Punkte kann nach Lemma 7.5 in einer Zeit von $\Theta(n \log(n))$ erfolgen, da jeder einzelne Vergleich nach Lemma 7.1 in $\Theta(1)$ Zeit durchgeführt werden kann.

Im weiteren Verlauf wird jeder der n Punkt der Liste L genau einmal in H eingefügt, da im weiteren Verlauf des Algorithmus aus H entfernte Punkte nicht wieder betrachtet werden. Das heißt, jeder Punkt wird

- einmal in H eingefügt und
- höchstens einmal aus H entfernt.

Da nach jedem Durchlauf der repeat-Schleife aber entweder ein Punkt gelöscht oder eingefügt wird, kann diese Schleife höchsten $2n$-mal durchlaufen werden. Jede einzelne Operation in der Schleife benötigt wiederum nur $O(1)$ Zeit. Somit hat der Graham-Scan eine Laufzeit von $O(n)$.

Insgesamt ist also das Sortieren am Anfang der für die Laufzeit entscheidende Teil des Algorithmus, der somit eine Laufzeit von $\Theta(n \log(n))$ besitzt. ■

In Abschnitt 8.2 werden wir sehen, daß es keine anderen Algorithmen geben kann, deren Laufzeit eine bessere Größenordnung besitzt.

8 Komplexität und Effizienz

Mit der Suche nach dem größten gemeinsamen Teiler zweier natürlicher Zahlen und der Bestimmung der konvexen Hülle einer Punktmenge in der Ebene haben wir zwei mathematische Probleme betrachtet, für die wir verschiedene Algorithmen mit unterschiedlicher Laufzeit kennen gelernt haben. So hatte Algorithmus 5.2 zur naiven Berechnung des größten gemeinsamen Teilers der Zahlen a und b eine Laufzeit von $\Theta(\min(a, b))$, wohingegen die Laufzeit des Euklidischen Algorithmus mit $\Theta(\log(\min(a, b)))$ von kleinerer Größenordnung war. Bei der Bestimmung der konvexen Hülle einer Punktmenge mit n Punkten konnten wir die Laufzeit sogar von $\Theta(n^3)$ für Algorithmus 7.2 auf $\Theta(n \log(n))$ durch Verwendung des Graham-Algorithmus 7.4 reduzieren.

Wir interessieren uns nun dafür, ob es noch schnellere Algorithmen für diese Probleme gibt, oder ob wir die Fahnenstange bereits erreicht haben.

8.1 Komplexe Probleme, effiziente Algorithmen

Definition 8.1 (Komplexität) *Sei (P) ein mathematisches Problem und sei $\mathbf{A}$ die Menge aller Algorithmen, die das Problem lösen. Die Laufzeitfunktion eines Algorithmus $\mathcal{A}$ aus $\mathbf{A}$ sei $L_{\mathcal{A}}$. Dann bezeichnen wir*

$$\inf_{\mathcal{A} \in \mathbf{A}} \{L_{\mathcal{A}}\}$$

als die Komplexität des Problems (P).

Die Komplexität eines Problems gibt also die kleinstmögliche Größenordnung für die Laufzeit eines Algorithmus an, der dieses Problem löst.

Jeder bekannte Algorithmus zur Lösung eines Problems liefert somit eine obere Schranke für die Komplexität des Problems. So wissen wir zum Beispiel bereits, daß die Komplexität der Berechnung des größten gemeinsamen Teilers zweier positiver Zahlen a und b höchstens $O(\log(\min(a, b)))$ ist. Da wir niemals alle möglichen Algorithmen zur Lösung eines Problem kennen werden, erfordert die Bestimmung einer unteren Schranke mehr theoretische Überlegungen. Findet man jedoch eine

solche untere Schranke für die Komplexität und einen Algorithmus zur Lösung des Problems mit einer Laufzeit in der gleichen Größenordnung, so hat man die Komplexität des Problems bestimmt.

So besagt z.B. Lemma 7.5, daß das Problem, eine Liste mit n Elementen bezüglich einer Ordnung $\preceq$ zu sortieren in $\Theta(n \log(n))$ gelöst werden kann. Ein Algorithmus, der diese obere Komplexitätsschranke für das Sortieren liefert, ist der auf Seite 85 kurz beschriebene Algorithmus MergeSort. Daß es keinen besseren Algorithmus geben kann, liefert das Resultat des folgenden Satzes 8.2 über die untere Komplexitätsschranke für das Sortieren.

Satz 8.2 (Untere Komplexitätsschranke für das Sortieren mit Vergleichen) *Jeder auf paarweisen Vergleichen bezüglich einer Ordnung $\preceq$ basierende Sortieralgorithmus benötigt zum Sortieren einer n-elementigen Liste im Worst-Case sowie im Mittel $\Omega(n \log(n))$ Vergleiche.*

Somit ist die Komplexität für das Sortieren einer Liste tatsächlich durch $\Theta(n \log(n))$ gegeben.

Allen bisher betrachteten Problemen (ggT, konvexe Hülle, Sortieren) ist gemein, daß zu ihrer Lösung Algorithmen bekannt sind, deren Laufzeit durch ein Polynom abgeschätzt werden kann. Dies ist jedoch nicht für alle Probleme der Fall.

Der Vergleich von Größenordnungen (6.15) hatte gezeigt, daß das Wachstum eines beliebigen Polynoms von kleinerer Größenordnung ist als von Funktionen, in denen das Argument im Exponenten auftaucht. Es gibt also einen Qualitätssprung zwischen Polynomen und sogenannten superpolynomialen Funktionen.

Ein Problem, für das kein Algorithmus existiert, dessen Laufzeit durch ein Polynom abgeschätzt werden kann, ist meist selbst für kleine Eingabewerte nicht in vertretbarer Zeit exakt lösbar, d.h. die Rechenzeit wächst sehr schnell über ein in der praktischen Anwendung sinnvolles Maß hinaus. Man sagt deshalb, ein solches Problem sei nicht effizient lösbar.

Dem entsprechend nennen wir einen Algorithmus dessen Laufzeit durch ein polynom abgeschätzt werden kann effizient oder einen polynomialen Algorithmus. Ein Problem, zu dessen Lösung ein polynomieller Algorithmus existiert, heißt selbst polynomial. Die Menge aller (mathematischen) Probleme, die polynomial lösbar sind wird mit $\mathcal{P}$ bezeichnet.

8.2 So schwer wie Sortieren

Der Graham-Algorithmus 7.4 führte das Problem, die konvexe Hülle einer Punktmenge $\mathcal{M}$ mit n Punkten zu finden, auf das Sorieren der Punkte zurück. Wir haben damit eine obere Komplexitätsschranke für die Bestimmung der konvexen Hülle von $O(n \log(n))$ erhalten. Nun soll gezeigt werden, das es nicht besser geht, d.h. daß $\Omega(n \log(n))$ auch eine untere Komplexitätsschranke für die konvexe Hülle darstellt.

Ein häufig verwendeter Trick bei der Bestimmung einer unteren Komplexitätsschranke $\Omega(f_1)$ für ein Problem (P_1) ist es, ein anderes Problem (P_2), dessen Komplexität $\Theta(f_2)$ bereits bekannt ist, auf das Problem P_1 zurückzuführen. Gäbe es dann einen Algorithmus $\mathcal{A}_1$, der das Problem (P_1) in einer Laufzeit von $L_{\mathcal{A}_1} < O(f_2)$ löst, so könnte das Problem (P_2) ebenfalls in $O(L_{\mathcal{A}_1})$ gelöst werden, was ein Widerspruch zur unteren Komplexitätsschranke $\Theta(f_2)$ von (P_2) wäre.

Wir zeigen nun, daß man eine Liste von n Zahlen $(x_1, x_2, ..., x_n)$ sortieren kann, indem man die

konvexe Hülle einer geeigneten Punktmenge bestimmt. Dazu definieren wir uns die Punktmenge $\mathcal{M} = \{P_1, P_2, ..., P_n\}$ mittels $P_i := (x_i, x_i^2)$. Alle so definierten Punkte liegen auf einer Parabel, also einem konvexen Funktionsgrafen. Somit sind alle Punkte in $\mathcal{M}$ Eckpunkte der konvexen Hülle von $\mathcal{M}$. Ein Algorithmus $\mathcal{A}_{konv}$, der die Eckpunkte der konvexen Hülle in einer mathematischen positiven Durchlaufrichtung bestimmt, liefert also eine sortierte Liste der Zahlen, in dem man vom linkesten Punkt aus jeweils die x-Koordinate der Punkte ausgibt.

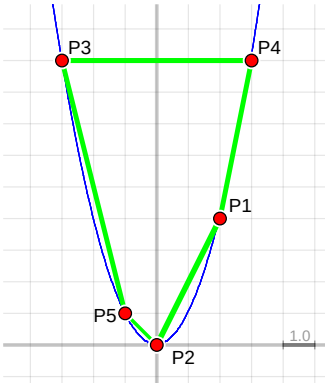


Abbildung 22: **Sortieren mittels konvexer Hülle.** In diesem Beispiel sollte die Liste $(2, 0, -3, 3, -1)$ sortiert werden.

Angenommen, es gäbe nun einen Konvexe-Hülle-Algorithmus $\mathcal{A}_{konv}$, dessen Laufzeit $L(\mathcal{A}_{konv})$ von kleinerer Größenordnung als $\Theta(n \log(n))$ wäre, dann könnte eine solche Liste von Zahlen folglich auch schneller sortiert werden. Dies widerspräche jedoch Satz 8.2. Folglich haben wir bewiesen:

Satz 8.3 *Jeder Algorithmus, der zu einer n-elementigen Punktmenge $\mathcal{M}$ der Ebene die Eckpunkte der konvexen Hülle von $\mathcal{M}$ bestimmt und in mathematisch positiver Orientierung ausgibt, hat eine Laufzeit von $\Omega(n \log(n))$.*

Somit ist der Graham-Algorithmus 7.4 ein bezüglich der Laufzeit optimaler Algorithmus zur Bestimmung der konvexen Hülle. Außerdem haben wir festgestellt:

> *Wenn man schneller sortieren könnte, könnte man auch schneller konvexe Hüllen berechnen.*
>
> *Wenn man schneller konvexe Hüllen berechnen könnte, könnte man auch schneller sortieren.*

In diesem Sinne sind das Sortieren und das Konvexe-Hülle-Bestimmen zwei äquivalente Problemstellungen.

8.3 Ein Problem mit nichtpolynomieller Komplexität?

Es gibt sehr viele für die Praxis relevante Problemstellungen, deren Komplexität bisher noch offen ist, d.h. für die noch nicht bekannt ist, ob sie effizient lösbar sind oder nicht. Das bekannteste

Beispiel für ein solches Problem ist das sogenannte Problem des Handlungsreisenden (Travelling Salesman Problem, kurz TSP).

Gegeben seien n Städte $S_1, ..., S_n$ sowie die jeweiligen Entfernungen d_{ij} zwischen den Stadten S_i und S_j. Gesucht ist eine kürzeste Tour oder Rundreise durch alle Städte, also eine Permutation $\pi = (\pi_1, ..., \pi_n)$ der Städte, deren Gesamtlänge

$$\text{cost}(\pi) = \sum_{i=1}^{n-1} d_{\pi_i, \pi_{i+1}} + d_{\pi n, \pi_1}$$

unter allen Permutationen der Städte minimal wird.

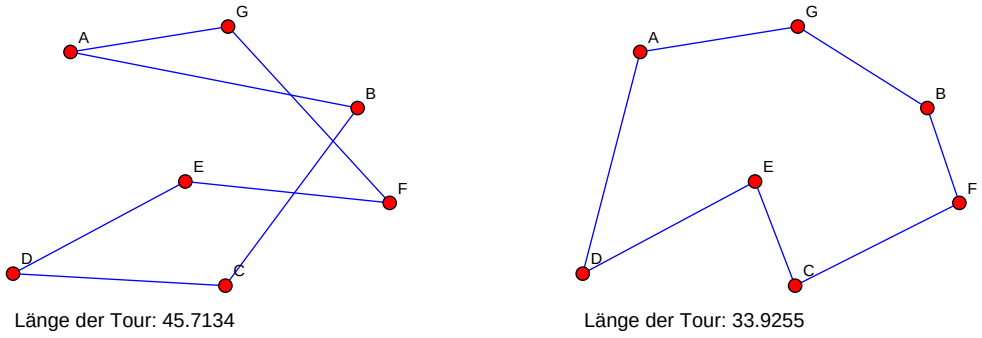


Abbildung 23: **Traveling salesman Problem.** Zwei verschiedene Touren.

Auf den ersten Blick scheint das Problem gar nicht so viel anders zu sein als die Suche nach der konvexen Hülle einer Punktmenge. In beiden Fällen ist eine n-elementige Punktmenge gegeben und es soll ein geeigneter geschlossener Kantenzug ermittelt werden. Beim Konvexe-Hülle-Problem durchläuft dieser Kantenzug jedoch nur die Eckpunkte der konvexen Hülle, beim Traveling Salesman Problem, werden alle Punkte so durchlaufen, daß der Kantenzug möglichst kurz wird.

Dennoch ist das TSP ein offenes Problem, welches seit langer Zeit im Mittelpunkt der Komplexitätstheorie steht. Ein Grund dafür ist, daß das TSP ein sogenanntes $\mathcal{NP}$-schweres Problem ist. Das bedeutet, ein polynomieller Algorithmus für das TSP würde sofort polynomielle Algorithmen für eine Vielzahl anderer wichtiger Probleme aus der Komplexitätsklasse $\mathcal{NP}$ liefern. $\mathcal{NP}$ ist hierbei die Abkürzung für ,,nicht-deterministisch polinomiell" – nicht zu verwechseln mit ,,nicht-polynomiell" – und bezeichnet die Klasse von Problemen, die auf einer nicht-deterministischen Turingmaschine (ein einfaches, idealisiertes Computermodell) in polynomieller Zeit gelöst werden können.

Die Frage, ob es einen polynomiellen Algorithmus für das TSP gibt ist eng mit der Frage verbunden, ob $\mathcal{P} = \mathcal{NP}$. Dies ist eines der sieben Millienium Probleme, für deren Lösung das Clay Mathematics Institute of Cambridge jeweils einen Preis in Höhe von einer Million Dollar ausgesetzt hat [9, 2]. Die vorherrschende Meinung besteht übrigens derzeit darin, daß es keinen polynomiellen TSP-Algorithmus gibt.

Für eine genauere Beschäftigung mit der Komplexitätstheorie verweisen wir auf die einschlägige Fachliteratur. Eine Einführung in diese Materie liefert z.B. [10]. Als weiterführende Lektüre über die Komplexität verschiedener Promblestellungen empfiehlt sich [5]. Unter [1] findet sich eine Auflistung von zur Zeit knapp 450 verschiedenen Komplexitätsklassen.

Auch zur Lösung des Traveling Salesman Problems können wir sehr leicht einen naiven Algorithmus finden:

Algorithmus 8.4 (Ausprobieren) Probiere alle möglichen Permutationen durch und nimm die Permutation π mit der geringsten Länge $\text{cost}(\pi)$.

Es gibt jedoch $n!$ verschiedene Permutationen der n Städte. Somit muss die Kostenfunktion $n!$ mal ausgewertet werden. Dieser naive Algorithmus hat somit eine Laufzeit von $\Theta(n \cdot n!)$. Durch verschiedene Tricks lassen sich bei der Untersuchung bereits vorab viele der möglichen Touren ausschließen, was zu besseren Algorithmen führt. Keiner der bisher bekannten Algorithmen kann jedoch das Traveling Salesman Problem in polynomieller Zeit exakt lösen. Da solche Lösungsalgorithmen somit nur für sehr kleine n praktisch durchführbar sind, gibt man sich in der Praxis mit suboptimalen Touren zufrieden. Die Entwicklung von Verfahren, welche möglichst gute Touren liefern, ist unter anderem Gegenstand der Diskreten Optimierung. Einen interessanten Überblick über das Traveling Salesman Problem findet man unter anderem in dem Artikel von Grötschel und Padberg [6].

Literatur

[1] S. Aaronson and G. Kuperberg. Complexity zoo. http://qwiki.caltech.edu/wiki/Complexity_Zoo.

[2] M. Atiyah. The Millennium Prize Problems. Springer, 2002.

[3] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. Introduction to Algorithms, Second Edition. MIT Press, Cambridge, 2001.

[4] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. Computational Geometry – Algorithms and Applications, Second, Revised Edition. Springer Verlag, Berlin Heidelberg, 2000.

[5] M.R. Garey and D.S. Johnson. Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman & Co., New York, 1979. Das Standardwerk, wenn es um $\mathcal{NP}$-Vollständigkeit geht.

[6] M. Groetschel and M. Padberg. Die optimierte Odysee. Spektrum der Wissenschaft, 4:76–85, 1999. Oder: http://www.spektrum.de/odyssee.html.

[7] D. Hilbert. Grundlagen der Geometrie, 14. Auflage. Teubner Verlag, Stuttgart Leipzig, 1999.

[8] R.H. Möhring. Computerorientiere Mathematik I mit Java. 2005. Vorlesungsskript zur Computerorientiereten Mathematik an der TU Berlin. Die TU-CoMa behandelt hauptsächlich Algorithmen und Datenstrukturen. http://www.math.tu-berlin.de/coga/teaching/coma/Skript-I-Java/.

[9] Clay Mathematics Institute of Cambridge. Millienium problems. http://www.claymath.org/millennium/.

[10] I. Wegener. Theoretische informatik. Teubner Verlag, Stuttgart Leipzig, 1993. Alles zu $\mathcal{P}$, $\mathcal{NP}$, TSP und Turingmaschinen.

Teil III

Lineare Gleichungssyssteme

9 Die Kondition von Matrizen

9.1 Vektor– und Matrixnormen

In diesem vorbereitenden Abschnitt geht es um die Verallgemeinerung des Betrages reeller Zahlen zu Normen von Vektoren und Matrizen.

Definition 9.1 *Es sei V ein linearer Raum über $\mathbb{R}$. Eine Abbildung*

$$\|\cdot\| : \ V \to \mathbb{R}$$

heißt Norm, falls für alle $x, y \in V$ und $\alpha \in \mathbb{R}$ gilt

$$\|x\| \geq 0 \ , \qquad \|x\| = 0 \Leftrightarrow x = 0 \ , \tag{9.1}$$

$$\|\alpha x\| = |\alpha| \ \|x\| \qquad \textit{(Homogenität)} \ , \tag{9.2}$$

$$\|x + y\| \leq \|x\| + \|y\| \qquad \textit{(Dreiecksungleichung)} \ . \tag{9.3}$$

Ein linearer Raum, der mit einer Norm versehen ist, heißt normierter Raum.

Beispiele:

Der Betrag

$$|x| = x \ \mathrm{sgn}(x) \text{ auf } V = \mathbb{R} \ .$$

Die Euklidische Norm

$$\|x\|_2 = (x_1^2 + x_2^2)^{\frac{1}{2}} \text{ auf } V = \mathbb{R}^2 \ .$$

Die Maximumsnorm

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

Die Maximumsnorm

$$\|x\|_\infty = \max_{i=1,\ldots,n} |x_i| \ , \qquad x = (x_i)_{i=1}^n \in \mathbb{R}^n \text{ auf } V = \mathbb{R}^n \ .$$

Die ℓ^p–Norm

$$\|x\|_p = \left(\sum_{i=1}^n |x_i|^p \right)^{\frac{1}{p}} \ , \qquad x = (x_i)_{i=1}^n \in \mathbb{R}^n$$

mit $1 \leq p < \infty$ auf $V = \mathbb{R}^n$.

Bemerkung: Der Nachweis der Eigenschaften (9.1), (9.2), (9.3) für die obigen Beispiele ist einfach. Nur die Dreiecksungleichung für $\|\cdot\|_p$ macht Schwierigkeiten. Sie lautet ausgeschrieben

$$\left(\sum_{i=1}^n |x_i + y_i|^p \right)^{\frac{1}{p}} \leq \left(\sum_{i=1}^n |x_i|^p \right)^{\frac{1}{p}} + \left(\sum_{i=1}^n |y_i|^p \right)^{\frac{1}{p}}$$

und heißt Minkowskische Ungleichung. Einen Beweis findet man z.B. im Funktionalanalysis–Lehrbuch von Werner [1], Seite 12.

Bemerkungen:

- Es gilt für jedes fest gewählte $x \in \mathbb{R}^n$
$$\lim_{p\to\infty} \|x\|_p = \|x\|_\infty \ .$$
- Im Falle von $p = 2$ erhält man die Euklidische Norm
$$\|x\|_2 = \left(\sum_{i=1}^{n} x_i^2 \right)^{\frac{1}{2}} \ .$$
- Mittels der Bijektion $\varphi : \mathbb{R}^{n,n} \to \mathbb{R}^{n^2}$, definiert durch
$$\varphi\left((a_{ij})_{i,j=1}^{n}\right) = (x_i)_{i=1}^{n^2} \ ,$$
$$x_{(i-1)n+j} = a_{ij} \ , \qquad i,j = 1,\ldots,n \ ,$$
läßt sich der lineare Raum der $n \times n$ Matrizen $\mathbb{R}^{n,n}$ mit $\mathbb{R}^{n^2}$ identifizieren. Jede Vektornorm auf $\mathbb{R}^{n^2}$ liefert also eine Matrixnorm auf $\mathbb{R}^{n,n}$. Ein Beispiel ist die sogenannte Frobenius–Norm
$$\|A\|_F = \left(\sum_{i,j=1}^{n} |a_{ij}|^2 \right)^{\frac{1}{2}} \qquad A = (a_{ij})_{i,j=1}^{n} \in \mathbb{R}^{n,n} \ .$$

Wir wollen den Konvergenzbegriff von $\mathbb{R}$ auf normierte Räume verallgemeinern.

Definition 9.2 *Es sei V ein normierter Raum mit der Norm $\|\cdot\|$ und $(x^{(\nu)})_{\nu\in\mathbb{N}}$ eine Folge aus V. Diese Folge heißt Cauchy–Folge, falls es zu jedem $\varepsilon > 0$ ein $\nu_0 \in \mathbb{N}$ gibt, so daß*
$$\|x^{(\nu)} - x^{(\mu)}\| \le \varepsilon \qquad \forall \nu,\mu \ge \nu_0 \ .$$
Die Folge heißt konvergent gegen $x \in V$, also
$$x^{(\nu)} \to x \ , \qquad \nu \to \infty \ ,$$
falls
$$\|x - x^{(\nu)}\| \to 0 \ , \qquad \nu \to \infty \ .$$
V heißt vollständig, falls alle Cauchy–Folgen in V konvergent sind. Ein vollständiger normierter linearer Raum heißt Banach–Raum.

Wir haben schon gesehen, daß man auf $\mathbb{R}^n$ oder $\mathbb{R}^{n,n}$ viele verschiedene Normen definieren kann. Es stellt sich die Frage, ob es Folgen gibt, die bezüglich der einen Norm konvergieren und bezüglich der anderen nicht. Der folgende Satz besagt, daß so etwas nicht sein kann.

Satz 9.3 *Sei V endlichdimensional und $\|\cdot\|$ sowie $|||\cdot|||$ zwei verschiedene Normen auf V. Dann gibt es Konstanten c, C, die nicht von x abhängen mit der Eigenschaft*
$$c\|x\| \le |||x||| \le C\|x\| \qquad \forall x \in V \ .$$

Beweis: Der Beweis beruht auf der Kompaktheit der Einheitskugel in $\mathbb{R}^n$. Wir verweisen wieder auf Werner [1], Seite 26. ■

Bemerkung: Für unendlichdimensionale Räume, wie z.B. $V = C[a, b]$, ist dieser Satz falsch!

Als Folgerung aus Satz 9.3 wollen wir die Vollständigkeit von $\mathbb{R}^n$ und $\mathbb{R}^{n,n}$ zeigen.

Satz 9.4 *Die linearen Räume $\mathbb{R}^n$ und $\mathbb{R}^{n,n}$ sind vollständig bezüglich jeder beliebigen Norm $\|\cdot\|_{\mathbb{R}^n}$ und $\|\cdot\|_{\mathbb{R}^{n,n}}$.*

Beweis: Es sei $(x^{(\nu)})_{\nu\in\mathbb{N}}$ eine Cauchy–Folge in $\mathbb{R}^n$. Nach Satz 9.3 gibt es c, C unabhängig von x, so daß

$$c\|x\|_{\mathbb{R}^n} \le \|x\|_\infty \le C\|x\|_{\mathbb{R}^n} , \qquad \forall x \in \mathbb{R}^n .$$

Also ist $(x^{(\nu)})_{\nu\in\mathbb{N}}$ auch Cauchy–Folge bezüglich $\|\cdot\|_\infty$. Dann ist aber auch die Folge von Komponenten $(x_i^{(\nu)})_{\nu\in\mathbb{N}}$ für jedes fest gewählte $i = 1, \ldots, n$, eine Cauchy–Folge in $\mathbb{R}$. Da $\mathbb{R}$ vollständig ist, gibt es ein $x_i \in \mathbb{R}$ mit

$$x_i^{(\nu)} \to x_i \qquad \nu \to \infty .$$

Führt man diesen Schluß für alle $i = 1, \ldots, n$ durch, erhält man einen Vektor $x = (x_i)_{i=1}^n$ mit der Eigenschaft

$$\|x - x^{(\nu)}\|_\infty \to 0 \qquad \nu \to \infty .$$

Die Konvergenz bezüglich $\|\cdot\|_{\mathbb{R}^n}$ folgt aus

$$\|x - x^{(\nu)}\|_{\mathbb{R}^n} \le \frac{1}{c}\|x - x^{(\nu)}\|_\infty \to 0 \qquad \nu \to \infty .$$

Da sich $\mathbb{R}^{n,n}$ (Matrizen) mit $\mathbb{R}^{n^2}$ (Vektoren) identifizieren lassen, sind wir fertig. ■

Wir wollen uns nun speziell den Matrizen zuwenden. Auf $\mathbb{R}^{n,n}$ ist durch

$$A \cdot B = C , \qquad c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} ,$$

die Multiplikation von Matrizen erklärt. Wir wollen eine Klasse von Matrixnormen betrachten, die mit der Matrix–Multiplikation verträglich ist.

Definition 9.5 *Es sei $\|\cdot\|$ eine Norm auf $\mathbb{R}^n$. Dann ist durch*

$$\|A\| = \sup_{\substack{x\in\mathbb{R}^n \\ x\neq 0}} \frac{\|Ax\|}{\|x\|} , \qquad A \in \mathbb{R}^{n,n} , \tag{9.4}$$

die zugehörige Matrixnorm definiert.

Man überzeuge sich davon, daß durch (9.4) überhaupt eine Norm definiert ist!

Bemerkungen:

- Es gilt
$$\|Ax\| \le \|A\| \, \|x\| .$$
- Es existiert ein $x^* \in \mathbb{R}^n$ mit $\|Ax^*\| = \|A\| \, \|x^*\|$.

- Matrixnormen der Form (9.4) sind mit der Matrixmultiplikation verträglich, d.h. es gilt

$$\|AB\| \leq \|A\| \, \|B\| \qquad \forall A, B \in \mathbb{R}^{n,n} \; , \qquad \text{(Submultiplikativität)} \; .$$

- Die Norm der Einheitsmatrix I ist

$$\|I\| = 1 \; .$$

Satz 9.6 (Zeilensummennorm) *Die Matrixnorm*

$$\|A\|_\infty = \max_{i=1,\ldots,n} \sum_{j=1}^{n} |a_{ij}|$$

gehört zur Maximumsnorm $\|\cdot\|_\infty$ *auf* $\mathbb{R}^n$.

Beweis: Für alle $x \in \mathbb{R}^n$ gilt

$$\begin{aligned} \|Ax\|_\infty &= \max_{i=1,\ldots,n} \left| \sum_{j=1}^{n} a_{ij} x_j \right| \\ &\leq \left(\max_{i=1,\ldots,n} \sum_{j=1}^{n} |a_{ij}| \right) \max_{j=1,\ldots,n} |x_j| = \|A\|_\infty \|x\|_\infty \; . \end{aligned}$$

Es folgt

$$\sup_{x \neq 0} \frac{\|Ax\|_\infty}{\|x\|_\infty} \leq \|A\|_\infty \; .$$

Wir konstruieren nun ein $x^* \in \mathbb{R}^n$ mit der Eigenschaft

$$\|A\|_\infty \|x^*\|_\infty = \|Ax^*\|_\infty \; .$$

Sei dazu i_0 so gewählt, daß

$$\|A\|_\infty = \sum_{j=1}^{n} |a_{i_0 j}| \; .$$

Dann sei

$$x_j^* = \; \operatorname{sgn}(a_{i_0 j}) \qquad j = 1, ..., n \; .$$

Es folgt offenbar $\|x^*\|_\infty = 1$ und

$$\|A\|_\infty \|x^*\|_\infty \geq \|Ax^*\|_\infty \geq \left| \sum_{j=1}^{n} a_{i_0 j} x_j^* \right| = \sum_{j=1}^{n} |a_{i_0 j}| = \|A\|_\infty \|x^*\|_\infty \; .$$

■

9.2 Störung von Matrix und rechter Seite

Wir betrachten das lineare Gleichungssystem

$$Ax = b \; . \tag{9.5}$$

Dabei seien die Koeffizientenmatrix $A = (a_{ij})_{i,j=1}^n \in \mathbb{R}^{n,n}$ sowie die rechte Seite $b = (b_i)_{i=1}^n \in \mathbb{R}^n$ gegeben und die Lösung $x = (x_i)_{i=1}^n \in \mathbb{R}^n$ gesucht.

Satz 9.7 *Die Koeffizientenmatrix A sei regulär, d.h.*

$$Ax \neq 0 \qquad \forall x \in \mathbb{R}^n \; , \quad x \neq 0 \; .$$

Dann ist (9.5) *eindeutig lösbar mit der Lösung* $x = A^{-1}b$.

Im folgenden sei $A \in \mathbb{R}^{n,n}$ regulär und $b \neq 0$, so daß eine eindeutig bestimmte Lösung $x \neq 0$ existiert. Außerdem wird mit $\|\cdot\|$ durchweg eine Vektornorm und die zugehörige Matrixnorm bezeichnet.

Wir wollen nun die Auswirkungen von Eingabefehlern in den Daten A, b auf die Lösung x untersuchen. Dabei spielt die sogenannte Kondition von A eine zentrale Rolle.

Definition 9.8 *Der Ausdruck*

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

heißt Kondition von A.

Bemerkung: Es gibt eine MATLAB–Funktion `cond`.

Es gilt

$$\kappa(A) = \|A\|\|A^{-1}\| \geq \|A^{-1}A\| = \|I\| = 1 \; .$$

Insbesondere ist $\kappa(I) = 1$. Eine weitere wichtige Eigenschaft ist die Submultiplikativität der Kondition. Sind A und B regulär, so folgt aus

$$\|AB\| \leq \|A\|\|B\| \; , \qquad \|(AB)^{-1}\| = \|B^{-1}A^{-1}\| \leq \|A^{-1}\|\|B^{-1}\|$$

unmittelbar

$$\kappa(AB) \leq \kappa(A)\kappa(B) \; .$$

Als erstes betrachten wir Störungen der rechten Seite b.

Satz 9.9 *Sei x die Lösung von (9.5) und $\tilde{x}$ die Lösung des gestörten Systems*

$$A\tilde{x} = \tilde{b} \tag{9.6}$$

mit beliebigem $\tilde{b} \in \mathbb{R}^n$. Dann gilt

$$\frac{\|x - \tilde{x}\|}{\|x\|} \leq \kappa(A) \; \frac{\|b - \tilde{b}\|}{\|b\|} \; . \tag{9.7}$$

Es existieren rechte Seiten b, $\tilde{b} \in \mathbb{R}^n$, so daß in (9.7) Gleichheit vorliegt.

Beweis: Die Submultiplikativität der Matrixnorm liefert

$$\|x - \tilde{x}\| = \|A^{-1}b - A^{-1}\tilde{b}\| \leq \|A^{-1}\| \; \|b - \tilde{b}\| \; , \qquad \|Ax\| \leq \|A\|\|x\|. \tag{9.8}$$

Einsetzen ergibt die Abschätzung

$$\frac{\|x - \tilde{x}\|}{\|x\|} \leq \frac{\|Ax\|}{\|x\|}\|A^{-1}\|\frac{\|b - \tilde{b}\|}{\|b\|} \leq \|A\| \; \|A^{-1}\|\frac{\|b - \tilde{b}\|}{\|b\|}$$

und damit (9.7).

Nach Definition der Matrixnorm existieren x^* und b^* mit den Eigenschaften $\|x^*\| = \|b^*\| = 1$, $\|Ax^*\| = \|A\|$ und $\|A^{-1}b^*\| = \|A^{-1}\|$. Damit gilt bei Wahl von $b = Ax^*$, $\tilde{b} = b - \varepsilon b^*$ und $\varepsilon > 0$ das Gleichheitszeichen in den beiden Abschätzungen (9.8). Einsetzen liefert das Gleichheitszeichen in (9.7). ■

Die Kondition $\kappa(A)$ ist also eine obere Schranke für die Verstärkung des relativen Fehlers in der rechten Seite b. Eine kleinere Schranke, die *gleichmäßig für alle* $b \in \mathbb{R}^n$ besteht, gibt es nicht.

In welchen Zusammenhang steht dieses Resultat mit unserem Konditionsbegriff aus Abschnitt 3? Um diese Frage zu beantworten, bemerken wir zunächst, daß sich die Definition 3.6 der relativen Kondition $\kappa_{\rm rel}(\xi_0)$ der Auswertung einer Funktion f an einer Stelle ξ_0 direkt auf Funktionen $f : \mathbb{R}^n \mapsto \mathbb{R}^m$ erweitern lässt[40]. Dazu hat man in (3.51) nur die Beträge durch die jeweiligen Vektornormen zu ersetzen. Nun betrachten wir die Funktion $f : \mathbb{R}^n \mapsto \mathbb{R}^n$, definiert durch

$$f(\xi) = A^{-1}\xi \ .$$

Die Auswertung von f an der Stelle $b \in \mathbb{R}^n$ ist offenbar gleichbedeutend mit der Lösung des linearen Gleichungssystems (9.5). Satz 9.9 besagt nun nichts weiter als

$$\kappa_{\rm rel}(b) \leq \kappa(A) \qquad\qquad \forall\, b \in \mathbb{R}^n \ .$$

Wir betrachten als nächstes die Auswirkung von Störungen der Koeffizientenmatrix A auf die Lösung x von (9.5). Diesmal müssen wir einen längeren Anlauf nehmen. Der Grund liegt darin, daß der entsprechende Lösungsoperator $f : \{X \in \mathbb{R}^{n,n} \mid X \text{ regulär}\} \mapsto \mathbb{R}^n$, definiert durch

$$x = f(X) = X^{-1}b \ , \qquad X \in \mathbb{R}^{n,n} \ ,$$

nun nichtlinear ist. Insbesondere ist nicht zu erwarten, daß ein gestörtes System

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

mit *beliebig* gestörter Matrix $\tilde{A} \in \mathbb{R}^{n,n}$ überhaupt eine Lösung $\tilde{x}$ hat.

Lemma 9.10 *Es sei $C \in I\!R^{n,n}$ und $\|C\| < 1$. Dann ist $I - C$ regulär, und es gilt*

$$(I - C)^{-1} = I + \sum_{k=1}^{\infty} C^k \qquad \textit{(Neumannsche Reihe).}$$

Beweis: Erinnerung: $\mathbb{R}^{n,n}$ versehen mit $\|\cdot\|$ ist vollständig! Wir zeigen zunächst, daß die Partialsummen

$$S_n = \sum_{k=0}^{n} C^k$$

eine Cauchy–Folge in $\mathbb{R}^{n,n}$ bilden. Dies folgt aus

$$\|S_n - S_m\| \leq \left\| \sum_{k=n}^{m} C^k \right\| \leq \sum_{k=n}^{m} \|C^k\| \leq \sum_{k=n}^{m} \|C\|^k \to 0 \qquad n, m \to \infty \ ,$$

denn die geometrische Reihe $\sum_{k=0}^{n} \|C\|^k$ konvergiert in $\mathbb{R}$. Aus der Vollständigkeit von $\mathbb{R}^{n,n}$ folgt die Konvergenz von S_n. Es gibt also ein $S \in \mathbb{R}^{n,n}$, so daß

$$\|S_n - S\| \to 0 \qquad n \to \infty \ .$$

Nun folgt

$$S_n(I - C) = S_n - S_nC = I + S_n - S_{n+1} \to I \qquad n \to \infty \ .$$

Andererseits gilt

$$S_n(I - C) \to S(I - C) \qquad n \to \infty \ .$$

Eindeutigkeit des Grenzwertes liefert

$$S(I - C) = I \ .$$

■

[40] Wir bezeichnen die unabhängige Variable an hier mit ξ, um Konflikte mit der Lösung x von (9.5) zu vermeiden.

Satz 9.11 *Sei x die Lösung von (9.5) und $\tilde{A} \in \mathbb{R}^{n,n}$ eine gestörte Koeffizientenmatrix mit der Eigenschaft*

$$\|A - \tilde{A}\| < \|A^{-1}\|^{-1} .$$

Dann existierte eine eindeutig bestimmte Lösung $\tilde{x}$ des gestörten Systems

$$\tilde{A}\tilde{x} = b \tag{9.9}$$

und es gilt

$$\frac{\|x - \tilde{x}\|}{\|x\|} \leq \kappa(A) \, \frac{\|A - \tilde{A}\|}{\|A\|} \; + \; o(\|A - \tilde{A}\|) . \tag{9.10}$$

Es existieren Koeffizientenmatrizen A, $\tilde{A} \in \mathbb{R}^{n,n}$, so daß in (9.10) Gleichheit vorliegt.

Beweis: Für $C = A^{-1}(A - \tilde{A})$ gilt nach Voraussetzung

$$\|C\| \leq \|A^{-1}\| \, \|A - \tilde{A}\| < 1 . \tag{9.11}$$

Auf Grund von Lemma 9.10 ist daher $I - C$ regulär. Wegen

$$\tilde{A} = A(I - C)$$

ist auch $\tilde{A}$ als Produkt zweier regulärer Matrizen regulär.

Wir kommen zum Nachweis von (9.10). Es gilt

$$\tilde{A}^{-1} = (A(I - C))^{-1} = (I - C)^{-1}A^{-1} = \left(I + C + \sum_{k=2}^{\infty} C^k \right) A^{-1} = A^{-1} + CA^{-1} + C^2(I - C)^{-1}A^{-1} . \tag{9.12}$$

Wegen (9.9) ist

$$\|(I - C)^{-1}\| \leq \sum_{k=0}^{\infty} \|C\|^k = \frac{1}{1 - \|C\|} \leq \frac{1}{1 - \|A^{-1}\| \, \|A - \tilde{A}\|} . \tag{9.13}$$

Durch Einsetzen der Beziehung (9.12) und der Abschätzungen (9.11) und (9.13) folgt

$$\begin{aligned}
\|x - \tilde{x}\| &= \|A^{-1}b - \tilde{A}^{-1}b\| \\
&= \|A^{-1}b - A^{-1}b - CA^{-1}b - C^2(I - C)^{-1}A^{-1}b\| \\
&\leq \left(\|C\| + \|C\|^2 \|(I - C)^{-1}\| \right) \|x\| \\
&\leq \left(\|A^{-1}\| + \frac{\|A^{-1}\|^2 \|A - \tilde{A}\|}{1 - \|A^{-1}\| \, \|A - \tilde{A}\|} \right) \|A - \tilde{A}\| \, \|x\| \\
&= \left(\|A\| \, \|A^{-1}\| + \frac{\|A\| \, \|A^{-1}\|^2 \, \|A - \tilde{A}\|}{1 - \|A^{-1}\| \, \|A - \tilde{A}\|} \right) \frac{\|A - \tilde{A}\|}{\|A\|} \|x\| .
\end{aligned}$$

Damit ist (9.10) beweisen.

Bei Wahl von $A = I$ und $\tilde{A} = (1 + \varepsilon)I$ ist offenbar $\kappa(A) = 1$ und $\|A - \tilde{A}\|/\|A\| = \varepsilon$. Für beliebiges $b \in \mathbb{R}^n$, $b \neq 0$, ist dann $x = b$ und $\tilde{x} = (1 + \varepsilon)^{-1}b$. Einsetzen liefert

$$\frac{\|x - \tilde{x}\|}{\|x\|} = \frac{\varepsilon}{1 + \varepsilon} = \varepsilon + o(\varepsilon) = \kappa(A) \, \frac{\|A - \tilde{A}\|}{\|A\|} + o(\|A - \tilde{A}\|) .$$

■

Schließlich notieren wir noch ohne Beweis ein Resultat zur Auswirkung von Störungen von Koeffizientenmatrix *und* rechter Seite auf die Lösung x von (9.5).

Satz 9.12 *Sei x die Lösung von (9.5), $\tilde{b} \in \mathbb{R}^n$ und $\tilde{A} \in \mathbb{R}^{n,n}$ eine gestörte Koeffizientenmatrix mit der Eigenschaft*

$$\|A - \tilde{A}\| < \|A^{-1}\|^{-1} .$$

Dann existierte eine eindeutig bestimmte Lösung $\tilde{x}$ des gestörten Systems

$$\tilde{A}\tilde{x} = \tilde{b} \tag{9.14}$$

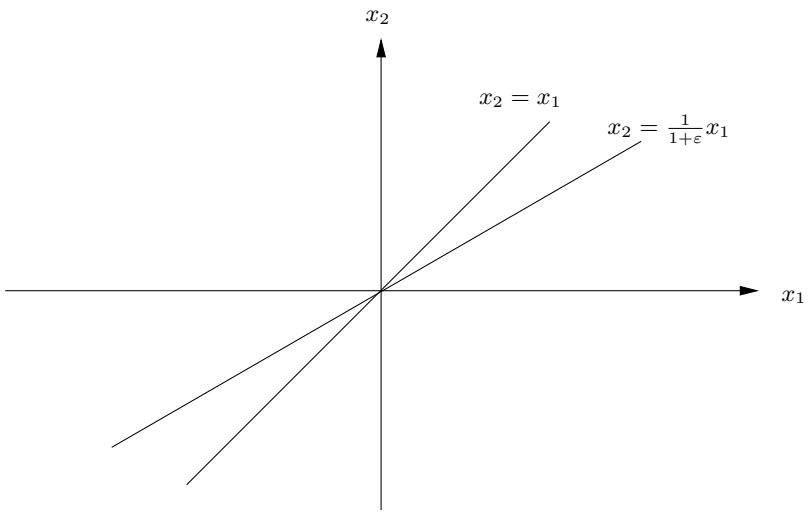
und es gilt

$$\frac{\|x - \tilde{x}\|}{\|x\|} \leq \kappa(A) \left(\frac{\|b - \tilde{b}\|}{\|b\|} + \frac{\|A - \tilde{A}\|}{\|A\|} \right) + o(\|A - \tilde{A}\|) . \tag{9.15}$$

Es existieren rechte Seiten b, $\tilde{b} \in \mathbb{R}^n$ und Koeffizientenmatrizen A, $\tilde{A} \in \mathbb{R}^{n,n}$, so daß in (9.15) Gleichheit vorliegt.

Es kann sein, daß zwar A regulär ist, aber $\tilde{A} = \mathrm{rd}(A)$ nicht.

Beispiel: (Schleifender Schnitt)



Den eindeutig bestimmten Schnittpunkt $x = 0$ berechnet man aus $Ax = 0$ mit

$$A = \begin{pmatrix} -1 & 1 \\ -1 & 1+\varepsilon \end{pmatrix} .$$

Die Inverse von A ist

$$A^{-1} = (-\varepsilon)^{-1} \begin{pmatrix} 1+\varepsilon & -1 \\ 1 & -1 \end{pmatrix} .$$

Bei der Berechnung der Kondition verwenden wir die Zeilensummennorm und erhalten

$$\kappa_\infty(A) = \|A\|_\infty \|A^{-1}\|_\infty = \frac{(2+\varepsilon)^2}{\varepsilon} \to \infty , \qquad \varepsilon \to 0 .$$

Ist $\varepsilon <$ eps (Maschinengenauigkeit), so folgt

$$\mathrm{rd}(A) = \tilde{A} , \qquad \tilde{A} = \begin{pmatrix} -1 & 1 \\ -1 & 1 \end{pmatrix} .$$

Die Regularitätsbedingung an $\|A - \tilde{A}\| < \|A^{-1}\|^{-1}$ aus Satz 9.11 und Satz 9.12 ist dann nicht erfüllt. Offenbar ist $\tilde{A}$ singulär. Die Gleitkomma-Approximation

$$\tilde{A}\tilde{x} = 0\,, \qquad \tilde{A} = \mathrm{rd}(A)\,,$$

hat damit *keine* eindeutig bestimmte Lösung.

Die Kondition $\kappa(A)$ lässt sich als Quantifizierung der Regularität von A interpretieren.

Satz 9.13 *Für alle regulären Matrizen A gilt*

$$\inf\left\{ \frac{\|A-B\|}{\|A\|} \;\middle|\; B \text{ singulär} \right\} \geq \frac{1}{\kappa(A)}\,.$$

Beweis: Ist B singulär, so existiert ein Vektor $x^* \neq 0$ mit $Bx^* = 0$. Für diesen Vektor gilt

$$\frac{\|A-B\|}{\|A\|} \geq \frac{\|(A-B)x\|}{\|A\|\|x\|} = \frac{\|Ax\|}{\|A\|\|x\|} = \frac{\|A^{-1}\|\|Ax\|}{\|A\|\|A^{-1}\|\|x\|} \geq \frac{\|A^{-1}Ax\|}{\kappa(A)\|x\|} = \frac{1}{\kappa(A)}.$$

■

Im Falle der Zeilensummennorm $\|\cdot\|_\infty$ kann man sogar zeigen, daß es eine singuläre Matrix B mit der Eigenschaft

$$\frac{\|A-B\|_\infty}{\|A\|_\infty} = \frac{1}{\kappa(A)}$$

gibt! In nächster Umgebung einer schlecht konditionierten Matrix befinden sich also singuläre Matrizen. In diesem Sinne sind schlecht konditionierte Matrizen sind „fast singulär“.

Beispiel: (Schleifender Schnitt)
Wir betrachten wieder die Matrix

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

mit der Kondition $\kappa_\infty(A) = \|A\|_\infty \|A^{-1}\|_\infty = \frac{(2+\varepsilon)^2}{\varepsilon}$. Für die singuläre Matrix B,

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

erhält man gerade

$$\frac{\|A-B\|_\infty}{\|A\|_\infty} = \frac{\varepsilon}{(2+\varepsilon)^2} = \frac{1}{\kappa(A)}\,.$$

Abschließend untersuchen wir noch einige Koeffizientenmatrizen, die bei der Diskretisierung von Integralgleichungen entstehen (siehe Anhang).

Beispiel: (Fredholmsche Integralgleichung 1. Art)

Wir betrachten die in (2.13) definierte Koeffizientenmatrix A, die wir aus unserem Galerkin–Verfahren erhalten haben. Als Parameter wählen wir $|\vec{\sigma}| = 1$, $L = 1$ und $n = 1$. Der MATLAB-Befehl `cond` liefert `cond(A) = Inf`. Damit ist A nicht von einer singulären Matrix zu unterscheiden. Unser Galerkin–Verfahren liefert, je nach rechter Seite, keine oder unendlich viele Lösungen. Daran ändert sich nichts, wenn man n vergrößert. Das deutet darauf hin, daß mit unserem kontinuierlichen Modell (2.5) etwas nicht stimmt. Das ist tatsächlich so! Im Falle einer Fredholmschen

Integralgleichung mit stetigem Kern $K(x,\xi)$ liegen Existenz und Eindeutigkeit einer Lösung i.a. nicht vor. Außerdem läßt sich der Einfluß von Störungen in den Raten u (gemessen in der Maximumsnorm) auf die Lösung f nicht beschränken.

Beispiel: (Fredholmsche Integralgleichung 2. Art)

Als nächstes betrachten wir unser Populationsmodell (2.7). Wir wählen als Parameter $T = \beta = 1$, $K(\tau) = \tau(1-\tau)$ und stellen in Abbildung 24 die Kondition der Koeffizientenmatrix (2.16), die wir aus unserem Galerkin–Ansatz erhalten haben, über n dar. Die Resultate deuten darauf hin, daß eine Fredholmsche Integralgleichung 2. Art mit stetigem Kern $K(x,\xi)$ ein korrekt gestelltes Problem sein könnte. Auch diese Vermutung erweist sich in der mathematischen Analyse als richtig.

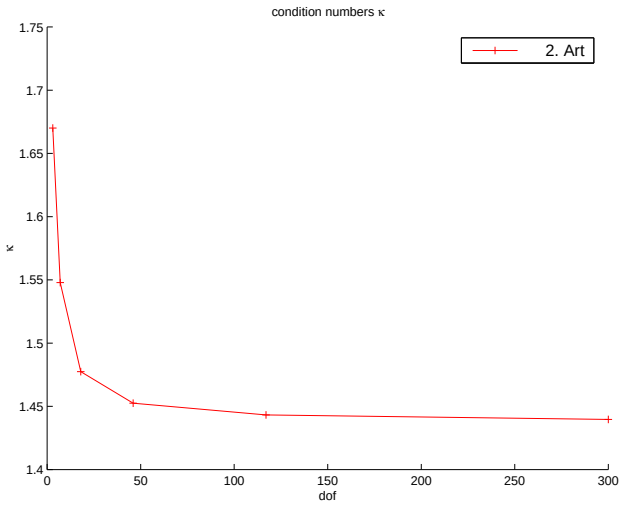


Abbildung 24: Kondition von unterschiedlich feinen Diskretisierungen einer Fredholmschen Integralgleichung 2. Art.

10 Gaußscher Algorithmus

10.1 Motivation

Grundidee des Gaußschen Algorithmus (nach Gauß: Theoria Motus etc. 1809) ist die sukzessive Elimination der Unbekannten.

Beispiel:

$$\begin{aligned} x_1 + 4x_2 + 7x_3 &= 5, \\ 2x_1 + 5x_2 + 8x_3 &= -1, \\ 3x_1 + 6x_2 + 10x_3 &= 0. \end{aligned}$$

Auflösen nach x_1:

$$x_1 = 5 - 4x_2 - 7x_3 \ . \tag{10.16}$$

Elimination von x_1 durch Einsetzen in die verbliebenen Gleichungen:

$$\begin{aligned} -3x_2 - \ \ 6x_3 &= -11, \\ -6x_2 - 11x_3 &= -15. \end{aligned}$$

Auflösen nach x_2:

$$-3x_2 = -11 + 6x_3 \tag{10.17}$$

und Elimination von x_2 aus der zweiten Gleichung ergibt

$$x_3 = 7 \ . \tag{10.18}$$

Einsetzen in (10.17):

$$x_2 = -\frac{31}{3} \ .$$

Einsetzen in (10.16):

$$x_1 = -\frac{8}{3} \ .$$

Einen anderen Blickwinkel erlaubt die folgende Matrixschreibweise des Gaußschen Algorithmus

$$\begin{pmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 10 \end{pmatrix} \quad \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \quad = \quad \begin{pmatrix} 5 \\ -1 \\ 0 \end{pmatrix}$$

$$A \qquad\qquad x \qquad = \qquad b \ .$$

Sukzessive Elimination lässt sich auf der erweiterten Matrix

$$\left(\begin{array}{ccc|c} 1 & 4 & 7 & 5 \\ 2 & 5 & 8 & -1 \\ 3 & 6 & 10 & 0 \end{array}\right)$$

durchführen: Eliminieren von x_1:

$$\left(\begin{array}{ccc|c} 1 & 4 & 7 & 5 \\ 2 & 5 & 8 & -1 \\ 3 & 6 & 10 & 0 \end{array}\right) \begin{array}{l} \\ -2 * 1. \text{ Zeile} \\ -3 * 1. \text{ Zeile} \end{array} \rightarrow \left(\begin{array}{ccc|c} 1 & 4 & 7 & 5 \\ 0 & -3 & -6 & -11 \\ 0 & -6 & -11 & -15 \end{array}\right) \ .$$

Eliminieren von x_2:

$$\left(\begin{array}{ccc|c} 1 & 4 & 7 & 5 \\ 0 & -3 & -6 & -11 \\ 0 & -6 & -11 & -15 \end{array}\right) \begin{array}{c} \\ \\ -2 * 2. \text{ Zeile} \end{array} \rightarrow \left(\begin{array}{ccc|c} 1 & 4 & 7 & 5 \\ 0 & -3 & -6 & -11 \\ 0 & 0 & 1 & 7 \end{array}\right) \ .$$

Als Ergebnis erhalten wir das gestaffelte Gleichungssystem:

$$\begin{pmatrix} 1 & 4 & 7 \\ 0 & -3 & -6 \\ 0 & 0 & 1 \end{pmatrix} \quad \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \quad = \quad \begin{pmatrix} 5 \\ -11 \\ 7 \end{pmatrix}$$

$$R \qquad\qquad x \qquad = \qquad z \ .$$

R obere Dreiecksmatrix; sukzessives Einsetzen liefert x

Beobachtung: Aus den im Eliminationsprozess auftretenden Faktoren bilde man die untere Dreiecksmatrix

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 3 & 2 & 1 \end{pmatrix} .$$

Dann gilt (nachrechnen!)

$$A = LR.$$

Zufall? Wir werden sehen.

10.2 Gaußscher Algorithmus und LR–Zerlegung

Wir betrachten das lineare Gleichungssystem

$$Ax = b \tag{10.19}$$

mit

$$A = (a_{ij})_{i,j=1}^{n} \in \mathbb{R}^{n,n} , \qquad \text{regulär} ,$$
$$b = (b_i)_{i=1}^{n} \in \mathbb{R}^{n} .$$

Der Koeffizient a_{11} heißt Pivotelement (frz. Drehpunkt). Unter der Voraussetzung

$$a_{11} \neq 0$$

erhalten wir nach dem 1. Eliminationsschritt die erweiterte Matrix

$$(A^{(1)}|b^{(1)}) = \left(\begin{array}{cccc|c} a_{11}^{(1)} & a_{12}^{(1)} & \cdots & a_{1n}^{(1)} & b_1^{(1)} \\ 0 & a_{22}^{(1)} & \cdots & a_{2n}^{(1)} & \vdots \\ \vdots & \vdots & & \vdots & \vdots \\ 0 & a_{n2}^{(1)} & \cdots & a_{nn}^{(1)} & b_n^{(1)} \end{array}\right) .$$

Dabei ist

$$a_{1j}^{(1)} = a_{1j} , \quad j = 1, \ldots, n , \quad b_1^{(1)} = b_1 ,$$
$$a_{ij}^{(1)} = a_{ij} - \frac{a_{i1}}{a_{11}} a_{1j} , \quad i, j = 2, \ldots, n ,$$
$$b_i^{(1)} = b_i - \frac{a_{i1}}{a_{11}} b_1 , \quad i = 2, \ldots, n .$$

Das Pivotelement für den 2. Eliminationsschritt ist $a_{22}^{(1)}$. Wir haben

$$a_{22}^{(1)} \neq 0$$

vorauszusetzen, um die Unbekannte x_2 aus den Gleichungen 3 bis n zu eliminieren. Unter der Voraussetzung nichtverschwindender Pivotelemente

$$a_{kk}^{(k-1)} \neq 0 \ , \qquad k = 2, \ldots, n-1 \ ,$$

werden durch weitere Eliminationsschritte die Gleichungssysteme

$$A^{(k)} x = b^{(k)} \ , \qquad k = 1, \ldots, n-1 \ ,$$

erzeugt. Die Berechnung erfolgt analog zum 1. Eliminationsschritt. Das führt auf folgenden Algorithmus:

Algorithmus 10.1 (Gaußsche Elimination)

Für $k = 1, ..., n-1$ berechne
{
Für $i = k+1, .., n$ berechne
{
$\ell_{ik} = \dfrac{a_{ik}^{(k-1)}}{a_{kk}^{(k-1)}}$
$b_i^{(k)} = b_i^{(k-1)} - \ell_{ik} b_k^{(k-1)}$
$a_{ik}^{(k)} = 0$
Für $j = k+1, ..., n$ berechne
{
$a_{ij}^{(k)} = a_{ij}^{(k-1)} - \ell_{ik} a_{kj}^{(k-1)}$
}
}
}

Dabei ist $a_{ij}^{(0)} = a_{ij}$ und $b_i^{(0)} = b_i$ gesetzt.

Ergebnis ist das Dreieckssystem

$$\begin{pmatrix} a_{11}^{(n-1)} & a_{12}^{(n-1)} & \cdots & a_{1n}^{(n-1)} \\ 0 & a_{22}^{(n-1)} & \cdots & a_{2n}^{(n-1)} \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & a_{nn}^{(n-1)} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \begin{pmatrix} b_1^{(n-1)} \\ b_2^{(n-1)} \\ \vdots \\ b_n^{(n-1)} \end{pmatrix}$$

Dieses System läßt sich einfach auflösen.

Algorithmus 10.2 (Rückwärtssubstitution)

$$x_n = \frac{1}{a_{nn}^{(n-1)}} b_n^{(n-1)}.$$

Für $i = n-1, \ldots, 1$ berechne

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

(Warum ist $a_{nn}^{(n-1)} \neq 0$?)

Bemerkungen zur Implementierung:

- Man kann die alten Elemente
$$a_{ij}^{(k-1)} , \qquad i,j = k+1, \ldots, n$$
mit den neuen Elementen
$$a_{ij}^{(k)} , \qquad i,j = k+1, \ldots, n$$
überschreiben.
- Man kann statt der erzeugten Nullen in der k-ten Spalte die Eliminationsfaktoren
$$\ell_{ik} = \frac{a_{ik}^{(k-1)}}{a_{kk}^{(k-1)}} , \qquad k = k+1, \ldots, n ,$$
abspeichern.

Wir wollen nun den Aufwand zur Lösung des linearen Gleichungssystems (10.19) mit dem Gaußschen Algorithmus abschätzen. Wir wählen

Aufwandsmaß = Anzahl der Multiplikationen.

Bemerkungen:

- Die Wahl der Anzahl der Multiplikationen als Aufwandsmaß hat historische Gründe. Frühere FPUs (floating-point processing units) konnten viel schneller addieren als multiplizieren. Bei modernen FPUs ist der Unterschied dagegen zu vernachlässigen, so daß eigentlich die Anzahl der Elementaroperationen ein geeigneteres Aufwandsmaß wäre. An der asymptotischen Ordnung ändert das aber nichts.
- Auch hier ist der ermittelte Aufwand bei weitem nicht proportional zur Rechenzeit und gestattet nur qualitative Aussagen über das Verhalten für große n. Die eigentlich viel interessantere Rechenzeit kann selbst bei gleicher Anzahl von Elementaroperationen je nach Implementierung um einen Faktor 100 oder mehr unterschiedlich sein!

Wir zählen zunächst die Multiplikationen von Algorithmus 10.1 (Gaußsche Elimination). Dann wird aus jeder for–Schleife eine Summation und wir erhalten

$$\begin{aligned}
&\sum_{k=1}^{n-1} \sum_{i=k+1}^{n} \left(1 + \sum_{j=k+1}^{n} 1 \right) \\
&= \sum_{k=1}^{n-1} \sum_{i=k+1}^{n} (n-k+1) = \sum_{k=1}^{n-1} (n-k+1)(n-k) \qquad \text{(setze } j = n-k\text{)} \\
&= \sum_{j=1}^{n-1} (j+1)j = \frac{1}{3}(n^3 - n) \ .
\end{aligned}$$

Die letzte Identität bestätigt man durch vollständige Induktion. Der zusätzliche Aufwand für die Berechnung von $b^{(n-1)}$ ist

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

Der Aufwand für Algorithmus 10.2 (Rückwärtssubstitution) ist

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

Damit erhalten wir als Gesamtaufwand für die Lösung von (10.19) mit dem Gaußschen Algorithmus

$$\begin{aligned} \frac{1}{3}(n^3 - n) + \frac{1}{2}(n^2 - n) + \frac{1}{2}(n^2 + n) &= \frac{1}{3}n^3 + n^2 - \frac{1}{3}n \\ &= \frac{1}{3}n^3 + \mathcal{O}(n^2) \ . \end{aligned}$$

Bemerkung: Man vergleiche den polynomiellen Aufwand des Gaußschen Algorithmus mit dem exponentiellen Aufwand der Cramerschen Regel (Lineare Algebra II).

Der Aufwand läßt sich reduzieren, wenn A gewisse Struktureigenschaften hat, etwa tridiagonal oder symmetrisch und positiv definit ist. Wir verweisen wieder auf weiterführende Vorlesungen oder z.B. auf das Lehrbuch von Deuflhard und Hohmann [**?**], Kapitel 1.

Als nächstes wollen wir unserer Vermutung aus dem einführenden Abschnitt 10 nachgehen.

Wir definieren $G_k \in \mathbb{R}^{n,n}$ durch

$$(G_k)_{ij} = \begin{cases} \ell_{ik} = \frac{a_{ik}^{(k-1)}}{a_{kk}^{(k-1)}} & i = k+1, \ldots, n, \ j = k \\ 0 & \text{sonst} \end{cases}$$

und $I \in \mathbb{R}^{n,n}$ sei die Einheitsmatrix.

Lemma 10.3 *Es gilt*

$$\begin{aligned} A^{(k)} &= (I - G_k)A^{(k-1)} \\ b^{(k)} &= (I - G_k)b^{(k-1)} \qquad k = 1, \ldots, n-1 \ , \end{aligned}$$

wobei wieder $A^{(0)} = A$ und $b^{(0)} = b$ gesetzt ist.

Beweis: Ausrechnen liefert mit der Definition von G_k

$$\begin{aligned} \left(G_k A^{(k-1)} \right)_{ij} &= \sum_{l=1}^{n} (G_k)_{il} \left(A^{(k-1)} \right)_{lj} \\ &= (G_k)_{ik} \left(A^{(k-1)} \right)_{kj} \\ &= \begin{cases} 0 & i \le k \ , \ j = 1, \ldots, n \\ \ell_{ik} a_{kj}^{(k-1)} & i \ge k+1 \ , \ j = 1, \ldots, n \end{cases} . \end{aligned}$$

Also ist

$$\left((I-G_k)A^{(k-1)}\right)_{ij} = \begin{cases} a_{ij}^{(k-1)} & i \le k\ ,\ j=1,\dots,n \\ a_{ij}^{(k-1)} - \ell_{ik}a_{kj}^{(k-1)} & i \ge k+1\ ,\ j=1,\dots,n \end{cases}$$

■

Satz 10.4 *Ist der Gaußsche Algorithmus für $A \in \mathbb{R}^{n,n}$ durchführbar (d.h. erhält man Pivotelemente $a_{kk}^{(k-1)} \neq 0$) und ergeben sich dabei die Eliminationsmatrizen $G_1, \dots, G_{n-1}$, so gilt*

$$A = LR$$

mit

$$L = I + \sum_{k=1}^{n-1} G_k = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ \ell_{21} & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ \ell_{n1} & \cdots & \ell_{n,n-1} & 1 \end{pmatrix}, \quad R = \prod_{k=1}^{n-1}(I-G_{n-k})A = \begin{pmatrix} a_{11}^{(n-1)} & \cdots & \cdots & a_{1n}^{(n-1)} \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & a_{nn}^{(n-1)} \end{pmatrix}.$$

Beweis: Nach Lemma 10.3 ist

$$\begin{aligned} R = A^{(n-1)} &= (I-G_{n-1})A^{(n-2)} \\ &= (I-G_{n-1})(I-G_{n-2})A^{(n-3)}\ . \\ &= (I-G_{n-1})\cdots(I-G_1)A\ . \end{aligned} \tag{10.20}$$

Weiter gilt

$$\begin{aligned} (G_kG_k)_{ij} &= \sum_{l=1}^{n}(G_k)_{il}(G_k)_{lj} \\ &= (G_k)_{ik}(G_k)_{kj} = (G_k)_{ik}\cdot 0 = 0 \qquad \forall i,j=1,\dots,n\ , \end{aligned} \tag{10.21}$$

also $(I+G_k)(I-G_k) = I - G_kG_k = I$ und somit

$$(I-G_k)^{-1} = I + G_k\ .$$

Aus (10.20) ergibt sich daher

$$A = (I+G_1)\cdots(I+G_{n-1})R\ .$$

ähnlich wie (10.21) folgt

$$G_k \cdot G_l = 0 \qquad \forall l \ge k\ .$$

Daher ist

$$\begin{aligned} (I+G_1)\cdots(I+G_{n-1}) &= (I+G_2)\cdots(I+G_{n-1}) + (G_1 + \underbrace{G_1G_2}_{=0})(I+G_3)\cdots(I+G_{n-1}) \\ &= (I+G_2)\cdots(I+G_{n-1}) + G_1 \\ &= I + G_1 + G_2 + \cdots + G_{n-1} = L\ . \end{aligned} \tag{10.22}$$

■

Der Gaußsche Algorithmus liefert also (falls durchführbar!) eine Faktorisierung

$$A = LR$$

in Dreiecksmatrizen $L, R \in \mathbb{R}^{n,n}$. Kennt man eine solche LR–Zerlegung von A, so kann man das Gleichungssystem $Ax = b$ in zwei Schritten lösen, nämlich

$$\begin{aligned} \text{Vorwärtssubstitution:} &\quad Lz = b \\ \text{Rückwärtssubstitution:} &\quad Rx = z\ . \end{aligned}$$

Dieses Vorgehen ist besonders empfehlenswert, wenn mehrere Gleichungssysteme mit verschiedenen rechten Seiten zu lösen sind. Der Aufwand ist dann jeweils nur $\mathcal{O}(n^2)$.

10.3 Die Stabilität des Gaußschen Algorithmus

Nach Satz 10.4 lässt sich der Gaußsche Algorithmus wie folgt formulieren

$$A^{(k)} = (I - G_k)A^{(k-1)} , \quad A^{(0)} = A , \qquad b^{(k)} = (I - G_k)b^{(k-1)} , \quad b^{(0)} = b , \tag{10.23}$$

$$x = R^{-1}z , \qquad R = A^{(n-1)} , \quad z = b^{(n-1)} . \tag{10.24}$$

Bei der praktischen Durchführung in Gleitkommaarithmetik treten Rundungsfehler auf. Vereinfachend nehmen wir an, daß die exakten Eliminationsmatrizen verwendet werden und nur nach jedem Eliminationsschritt gerundet wird. Dies führt auf den folgenden approximativen Algorithmus

$$\tilde{A}^{(k)} = \mathrm{rd}\left((I - G_k)\tilde{A}^{(k-1)}\right) , \quad \tilde{A}^{(0)} = A , \qquad \tilde{b}^{(k)} = \mathrm{rd}\left((I - G_k)\tilde{b}^{(k-1)}\right) , \quad \tilde{b}^{(0)} = b , \tag{10.25}$$

$$\tilde{x} = \tilde{R}^{-1}\tilde{z} , \qquad \tilde{R} = \tilde{A}^{(n-1)} , \quad \tilde{z} = \tilde{b}^{(n-1)} . \tag{10.26}$$

Dabei ist die Rundung komponentenweise zu verstehen, also

$$\mathrm{rd}(B) = (\mathrm{rd}(b_{ij}))_{i,j=1}^{n} , \qquad \mathrm{rd}(b) = (\mathrm{rd}(b_i))_{i=1}^{n} .$$

Man bestätigt leicht

$$\frac{\|B - \mathrm{rd}(B)\|_\infty}{\|B\|_\infty} \le eps , \quad \forall B \in \mathbb{R}^{n,n} , \qquad \frac{\|b - \mathrm{rd}(b)\|_\infty}{\|b\|_\infty} \le eps , \quad \forall b \in \mathbb{R}^n . \tag{10.27}$$

Wir betrachten daher von nun an die Maximums-Norm $\|\cdot\| = \|\cdot\|_\infty$.

Zunächst untersuchen wir die Fehlerfortpflanzung im Eliminationsprozess.

Lemma 10.5 *Es gilt*

$$\frac{\|R - \tilde{R}\|_\infty}{\|R\|_\infty} \le \sigma_E \; eps + o(eps) , \qquad \frac{\|z - \tilde{z}\|_\infty}{\|z\|_\infty} \le \sigma_E \; eps + o(eps) ,$$

wobei

$$\sigma_E = \sum_{j=1}^{n-1} \prod_{k=j+1}^{n-1} \kappa_k = 1 + \kappa_{n-1}(1 + \kappa_{n-2}(1 + \ldots \kappa_3(1 + \kappa_2)) \ldots)) , \qquad \kappa_k = \kappa_\infty(I - G_k).$$

gesetzt ist.

Beweis: Wir zeigen nur die erste der beiden Abschätzungen. Der Nachweis der zweiten erfolgt analog. Die Grundidee besteht darin, eine Rekursion für die Fehler

$$e_k = \frac{\|A^{(k)} - \tilde{A}^{(k)}\|_\infty}{\|A^{(k)}\|_\infty} , \quad k = 1, \ldots, n-1 , \qquad e_0 = 0 ,$$

herzuleiten. Zunächst bemerken wir

$$\|A^{(k-1)}\|_\infty = \|(I - G_k)^{-1}A^{(k)}\|_\infty \le \|(I - G_k)^{-1}\|_\infty \|A^{(k)}\|_\infty .$$

Mit der Dreiecksungleichung folgt nun aus (10.27) und dieser Abschätzung folgt nun

$$\begin{aligned} e_k &\leq \frac{\|A^{(k)} - (I - G_k)\tilde{A}^{(k-1)}\|_\infty}{\|A^{(k)}\|_\infty} + eps\ \frac{\|(I - G_k)\tilde{A}^{(k-1)}\|_\infty}{\|A^{(k)}\|_\infty} \\ &\leq \frac{\|I - G_k\|_\infty \|A^{(k-1)} - \tilde{A}^{(k-1)}\|_\infty}{\|A^{(k)}\|_\infty} + eps\ \frac{\|(I - G_k)(A^{(k-1)} - \tilde{A}^{(k-1)})\|_\infty}{\|A^{(k)}\|_\infty} + eps \\ &\leq \kappa_k(1 + eps)e_{k-1} + eps\ . \end{aligned}$$

Die Anwendung von Lemma 4.6 mit $\alpha_k = \kappa_k(1 + eps)$ und $\beta_k = eps$ liefert nun

$$\frac{\|R - \tilde{R}\|_\infty}{\|R\|_\infty} = e_{n-1} \leq eps \sum_{j=1}^{n-1} \prod_{k=j+1}^{n-1} \kappa_k(1 + eps) \leq eps \sum_{j=1}^{n-1} \prod_{k=j+1}^{n-1} \kappa_k + o(eps)$$

und damit die Behauptung. ■

Nun können wir die angestrebte Fehlerabschätzung formulieren.

Satz 10.6 *Es gilt*

$$\frac{\|x - \tilde{x}\|}{\|x\|} \leq \sigma_G\ eps + o(eps)\ ,$$

wobei

$$\sigma_G = 2\kappa(A)\sigma_K\ \sigma_E\ , \qquad \sigma_K = \prod_{k=1}^{n-1} \kappa_k\ ,$$

gesetzt ist.

Beweis: Aus Satz 9.12 folgt unmittelbar

$$\frac{\|x - \tilde{x}\|}{\|x\|} \leq \kappa(R) \left(\frac{\|R - \tilde{R}\|}{\|R\|} + \frac{\|z - \tilde{z}\|}{\|z\|} \right)$$

Nun gilt

$$\frac{\|R - \tilde{R}\|}{\|R\|} + \frac{\|z - \tilde{z}\|}{\|z\|} \leq 2\sigma_E\ eps + o(eps)$$

wegen Lemma 10.5. Außerdem ist

$$\kappa(R) = \kappa((I - G_{n-1}) \cdots (I - G_1)A) = \kappa(A) \prod_{k=1}^{n-1} \kappa_k\ .$$

Damit ist alles gezeigt. ■

Nach Satz 10.6 ist die Kondition κ_k der Eliminationsmatrizen $I - G_k$ entscheidend für die Stabilität des Gaußschen Algorithmus. Am besten wäre $\kappa_k = 1$. Der folgende Satz dämpft unsere Hoffnungen.

Satz 10.7 *Es gilt*

$$\kappa_\infty(I - G_k) = \max_{i=k+1,\ldots,n} (1 + |\ell_{ik}|)^2\ , \qquad \ell_{ik} = \frac{a_{ik}^{(k-1)}}{a_{kk}^{(k-1)}}\ .$$

Insbesondere ist $\kappa_\infty(I - G_k) = 1$ genau dann, wenn $a_{ik}^{(k-1)} = 0\ \forall i = k + 1, \ldots, n$ vorliegt.

Beweis: Offenbar ist

$$\|I - G_k\|_\infty = \max_{i=k+1,\ldots,n}(1 + |\ell_{ik}|) \ .$$

Aus $(I - G_k)^{-1} = I + G_k$ folgt weiter

$$\|(I - G_k)^{-1}\|_\infty = \|I + G_k\|_\infty = \max_{i=k+1,\ldots,n}(1 + |\ell_{ik}|) \ .$$

und damit die Behauptung. ■

Nach Satz 10.7 liegt $\kappa_k = 1$ genau dann vor, wenn es nichts zu elimineren gibt. Im nicht-trivialen Fall ist $\kappa_k > 1$. Zumindest sollte man aber $\kappa_k \gg 1$ vermeiden, wie folgendes Beispiel zeigt.

Beispiel: (2×2–Matrix)
Wir betrachten das lineare Gleichungssystem

$$\begin{aligned} 10^{-4}x_1 + x_2 &= 1 \\ x_1 + x_2 &= 2 \end{aligned}$$

mit der exakten Lösung

$$x_1 = 1 + \frac{1}{9999} \ , \qquad x_2 = 1 - \frac{1}{9999} \ .$$

Es ist

$$A^{-1} = (10^{-4} - 1)^{-1} \begin{pmatrix} 1 & -1 \\ -1 & 10^{-4} \end{pmatrix}$$

die Inverse der Koeffizientenmatrix

$$A = \begin{pmatrix} 10^{-4} & 1 \\ 1 & 1 \end{pmatrix} \ .$$

Wir können daraus die Kondition

$$\kappa_\infty(A) = \|A\|_\infty \|A^{-1}\|_\infty = 2 \cdot 2(1 - 10^{-4})^{-1} \approx 4$$

ablesen. Unser Problem ist also gut konditioniert!

Wir nehmen an, daß uns 3 gültige Stellen zur Verfügung stehen. Dann ist

$$\mathrm{rd}(x_1) = 1 \ , \qquad \mathrm{rd}(x_2) = 1$$

die gerundete, exakte Lösung. Der Gaußsche Algorithmus liefert

$$\left(\begin{array}{cc|c} 10^{-4} & 1 & 1 \\ 1 & 1 & 2 \end{array}\right) \to \left(\begin{array}{cc|c} 10^{-4} & 1 & 1 \\ 0 & -\mathrm{rd}(9999) & -\mathrm{rd}(9998) \end{array}\right)$$

$$= \left(\begin{array}{cc|c} 10^{-4} & 1 & 1 \\ 0 & -10000 & -10000 \end{array}\right)$$

und damit

$$\tilde{x}_1 = 0 \ , \qquad \tilde{x}_2 = 1 \ .$$

Nach Zeilentausch ist $a_{11} = 1 \gg 10^{-4}$. Man erhält

$$\left(\begin{array}{cc|c} 1 & 1 & 2 \\ 10^{-4} & 1 & 1 \end{array}\right) \to \left(\begin{array}{cc|c} 1 & 1 & 2 \\ 0 & \mathrm{rd}(0.9999) & \mathrm{rd}(0.9998) \end{array}\right)$$

$$= \left(\begin{array}{cc|c} 1 & 1 & 2 \\ 0 & 1 & 1 \end{array}\right)$$

und damit die gerundete exakte Lösung

$$\tilde{x}_1 = 1 \ , \qquad \tilde{x}_2 = 1 \ !$$

Beispiel: (Wilkinson–Matrix)

Als zweites Beispiel betrachten wir die Matrix

$$A = (a_{ij})_{i,j=1}^{n}\ , \qquad a_{ij} = \begin{cases} 1 & \text{falls } i = j \text{ oder } j = n \\ -1 & \text{falls } i > j \\ 0 & \text{sonst} \end{cases}$$

Die Abbildung 25 zeigt, daß zwar die Kondition von A nur moderat mit n wächst, die Kondition von R aber explodiert: Für $n = 50$ gilt beispielsweise $\kappa(A) = 22.3$, aber $\kappa(R) = 7.5\ 10^{14}$! Der Gaußsche Algorithmus 10.1 erweist sich als instabil.

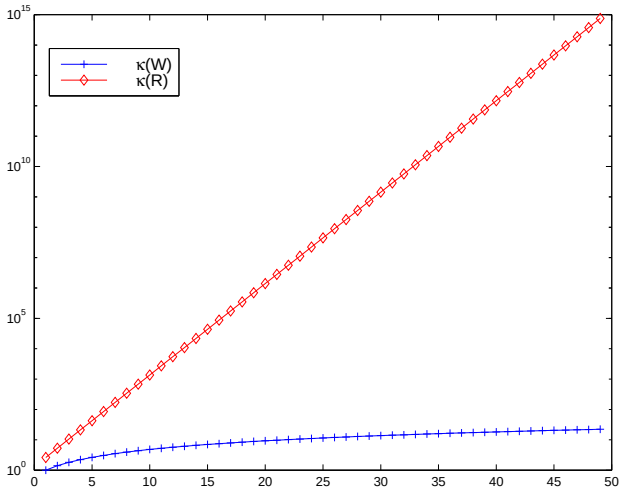


Abbildung 25: Kondition der Wilkinson-Matrix und ihres Faktors R.

10.4 Gaußscher Algorithmus mit Spaltenpivotsuche

Um die Durchführung des Gaußschen Algorithmus zu sichern, wollen wir nun vor jedem Eliminationsschritt durch Zeilentausch sichern, daß das (eventuell neue) Pivotelement $a_{kk}^{(k-1)} \neq 0$ ist. (Ist das im Falle regulärer Matrizen immer möglich?) Aus Stabilitätsgründen wählen wir $a_{kk}^{(k-1)}$ betragsmäßig möglichst groß, um entsprechend Satz 10.7 für möglichst gut konditionierte Eliminationsmatrizen $I - G_k$ zu sorgen.

Diese Überlegungen führen auf die folgende praktikable Variante des „naiven“ Verfahrens 10.1.

Algorithmus 10.8 (Gauß-Elimination mit Spaltenpivoting)

Für $k = 1, \ldots, n-1$ berechne
{
$k_0 = k$
Für $i = k+1, \ldots, n$ berechne
{
Falls $\left|a_{ik}^{(k-1)}\right| > \left|a_{k_0,k}^{(k-1)}\right|$, setze $k_0 := i$
}

Vertausche die k–te Zeile mit der k_0–ten Zeile

k–ter Eliminationsschritt wie in Algorithmus 10.1.

}

Bemerkungen:

- Der Zusatzaufwand für Spaltenpivoting besteht in $\mathcal{O}(n^2)$ Vergleichen und Vertauschungen.
- Aufgrund unserer Stabilitätsanalyse sollte $\max_{i=k+1,\ldots,n} |\ell_{ik}|$ möglichst klein sein. Daher sollte man besser k_0 so wählen, daß gilt

$$\max_{i=k,..,n} \frac{\left|a_{ik}^{(k-1)}\right|}{\left|a_{k_0k}^{(k-1)}\right|} \leq \max_{i=k,..,n} \frac{\left|a_{ik}^{(k-1)}\right|}{\left|a_{jk}^{(k-1)}\right|} \qquad \forall j = k, \ldots, n \ .$$

  Dieses Vorgehen ist jedoch aufwendiger. Meist verwendet man stattdessen eine geschickte Skalierung des Gleichungssystems. Wir verweisen auf Deuflhard und Hohmann [**?**], S. 13.

Die Wahl der Pivot-Elemente sichert

$$|\ell_{ik}| = \left|\frac{a_{ik}^{(k-1)}}{a_{kk}^{(k-1)}}\right| \leq 1 \ , \qquad i = k+1, \ldots, n \ , \quad k = 1, \ldots, n-1 \ .$$

Daraus folgt

$$\sigma_E \leq 2^{n-1} - 1 \ , \qquad \sigma_K \leq 2^{n-1} \ ,$$

und man gewinnt aus Satz 10.6 die Stabilitätsabschätzung

$$\frac{\|x - \tilde{x}\|_\infty}{\|x\|_\infty} \leq \kappa(A) 2^{n-1}(2^{n-1} - 1)\ eps + o(eps) \ .$$

Also ist für große n damit zu rechnen, daß der Gauß-Algorithmus, auch mit Spaltenpivoting, instabil ist. Für Matrizen mit Spezialstruktur (symmetrisch positiv definite, diagonaldominante, zufällige, Bandmatrizen, etc.) können deutlich bessere Resultate gezeigt werden.

Eine allgemein anwendbare Methode zur Verbesserung der Stabilität ist die Nachiteration (siehe [**?**]):

1. zerlege $A = LR$
2. löse $Lz = b$
3. löse $R\tilde{x} = z$
4. berechne $r = b - A\tilde{x}$
5. löse $L\tilde{z_r} = r$
6. löse $R\tilde{d} = \tilde{z_r}$
7. setze $x = \tilde{x} + \tilde{d}$

Das Analogon zu Satz 10.4 ist nun

Satz 10.9 *Die Gaußsche Elimination mit Spaltenpivotsuche liefert eine Zerlegung*

$$LR = PA$$

mit unterer Dreiecksmatrix L, oberer Dreiecksmatrix R und einer Permutationsmatrix P. PA unterscheidet sich von A also nur durch Vertauschung der Zeilen.

Beweis: Die Vertauschung von k–ter Zeile mit k_0–ter Zeile wird durch die Permutationsmatrix P_k repräsentiert. So bewirkt $P_k A^{(k-1)}$ gerade die Vertauschung von k–ter und k_0–ter Zeile. Dann liefert der Eliminationsprozess Matrizen G_k mit

$$(I - G_{n-1})P_{n-1} \cdots (I - G_1)P_1 A = R \ .$$

Wegen $P_k^{-1} = P_k$ und $(I - G_k)^{-1} = (I + G_k)$ ist

$$A = P_1(I + G_1) \dots P_{n-1}(I + G_{n-1})R \ . \tag{10.28}$$

Wir setzen

$$Q_k = P_{n-1} \cdots P_k \ , \quad k = 1, \dots, n-1 \ , \quad Q_n = I \ .$$

Wegen $P_k P_k = I$ gilt dann

$$\begin{aligned} Q_k P_k (I + G_k) &= Q_{k+1} P_k P_k (I + G_k) \\ &= (I + Q_{k+1} G_k Q_{k+1}^{-1}) Q_{k+1} \\ &= (I + Q_{k+1} G_k) Q_{k+1} \ , \end{aligned}$$

denn

$$G_k Q_{k+1}^{-1} = G_k P_{k+1} \cdots P_{n-1} = G_k \ .$$

Multiplikation von rechts mit P_l bewirkt nämlich die Vertauschung der Spalten l und $l_0 \geq l$ und die sind beide Null für $l > k$.

Setzt man $P = Q_1 = P_{n-1} \cdots P_1$, so folgt aus (10.28) schließlich

$$\begin{aligned} P P_1 (I + G_1) \cdots P_{n-1}(I + G_{n-1}) &= (I + Q_2 G_1) Q_2 P_2 (I + G_2) \cdots P_{n-1}(I + G_{n-1}) \\ &= (I + Q_2 G_1)(I + Q_3 G_2) \cdots (I + G_{n-1}) \\ &= I + \sum_{m=1}^{n-1} Q_{k+1} G_k = L \ , \end{aligned} \tag{10.29}$$

denn die Matrizen $Q_{k+1} G_k$ sind von der gleichen Bauart wie G_k und man kann wie in (10.22) schließen. Aus (10.28) und (10.29) folgt die Behauptung. ■

Übrigens liefert der MATLAB–Befehl

```
[L,R,P]=lu(A)
```

eine LR–Zerlegung von A mit Pivoting. Ausprobieren!

Literatur

[1] D. Werner. Funktionalanalysis. Springer, 2. Auflage, 1997. Ein gefeiertes Lehrbuch zur Einführung in die Funktionalanalysis. Anwendungen auf Integralgleichungen finden sich in Abschnitt VI. 4.

A Elementares zum Rechnen mit Vektoren und Matrizen

Wir beschränken uns auf das Elementarste und verweisen für alle schönen mathematischen Ausführungen zu diesem Thema auf die Vorlesungen zur linearen Algebra.

A.1 Vektoren

Rein rechentechnisch ist ein Vektor x eine Spalte oder Zeile von Zahlen $x_1, \ldots, x_n$

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \quad \text{oder} \quad x = \left(x_1, x_2, \cdots, x_n\right).$$

Dabei ist n eine beliebige positive ganze Zahl n, die Länge des Vektors. Man bezeichnet den ersten Fall als Spaltenvektor und den zweiten als Zeilenvektor. Wir beschränken uns hier auf reelle Zahlen $x_k \in \mathbb{R}$; die x_k heissen Komponenten oder Einträge des Vektors und wir schreiben der Einfachheit halber oft $x = (x_k)_{k=1,\ldots,n}$ sowohl für Spalten- als auch Zeilen-Vektoren.

Für Vektoren gelten folgende Basisoperationen:

- Multiplikation mit einem Skalar: Für ein beliebiges $\alpha \in \mathbb{R}$ und einen Spaltenvektor $x = (x_k)_{k=1,\ldots,n}$ ist

  $$\alpha x = \begin{pmatrix} \alpha x_1 \\ \alpha x_2 \\ \vdots \\ \alpha x_n \end{pmatrix},$$

  d.h. alle Komponenten des Vektors werden mit dem Skalar multipliziert. Gleiches gilt für Zeilenvektoren.

- Addition von Vektoren: Für zwei Spaltenvektoren $x = (x_k)_{k=1,\ldots,n}$ und $y = (y_k)_{k=1,\ldots,n}$ ist

  $$x + y = \begin{pmatrix} x_1 + y_1 \\ x_2 + y_2 \\ \vdots \\ x_n + y_n \end{pmatrix}.$$

  Wiederum gilt gleiches für Zeilenvektoren. Es können nur Vektoren gleicher Länge addiert werden. Die Addition von einem Zeilen- mit einem Spaltenvektor ist nicht definiert.

Beispiel:
Sei $x = (1,\ 2,\ 3)$ und $y = (0,\ 7.5,\ -2)$. Dann ist

$$x + y = (1,\ 9.5,\ 1) \quad \text{und} \quad x - \frac{1}{2}y = (1,\ -1.75,\ 4).$$

A.2 Matrizen

Matrizen sind rechentechnisch nichts weiter als rechteckige Felder von Zahlen oder anders ausgedrückt: eine $n \times m$ Matrix A ist eine Zeile aus m Spaltenvektoren der Länge n, wobei n und m positive ganze Zahlen sind. Für $n = 2$ und $m = 3$ sieht das z.B. so aus:

$$A = \begin{pmatrix} 1 & 6 & -1 \\ 2 & \sqrt{2} & 0 \end{pmatrix}.$$

Oder allgemein:

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{pmatrix}.$$

Natürlich könnten wir auch sagen, dass eine $n \times m$-Matrix eine Spalte der Länge n aus Zeilenvektoren der Länge m ist! Eine 1×1-Matrix ist nichts weiter als eine einzelne Zahl, ist also identisch mit einem Skalar.

Bemerkung: Offensichtlich ist eine $1 \times m$-Matrix nichts anderes als ein Zeilenvektor der Länge m und eine $n \times 1$-Matrix ein Spaltenvektor der Länge n.

Wiederum beschränken wir uns hier auf reelle Einträge a_{kl} und schreiben für unsere $n \times m$-Matrix auch vereinfacht $A = (a_{kl})_{k=1,\ldots,n;l=1,\ldots,m}$. Die Menge aller möglichen $n \times m$-Matrizen mit reellen Einträgen bezeichnen wir mit

$$\mathbb{R}^{n,m}$$

so dass wir als einfachste Schreibweise für den Satz "Die $n \times m$-Matrix A mit Einträgen a_{kl}" haben: $A = (a_{kl}) \in \mathbb{R}^{n,m}$. Im Falle $n = m$, also gleichen Spalten- und Zeilenlänge, spricht man von quadratischen Matrizen.

Für Matrizen gelten folgende Basisoperationen:

- Multiplikation mit einem Skalar: Für ein beliebiges $\alpha \in \mathbb{R}$ und eine Matrix $A = (a_{kl}) \in \mathbb{R}^{n,m}$ ist

  $$\alpha A = (\alpha a_{kl}) \in \mathbb{R}^{n,m},$$

  d.h. alle Einträge der Matrix werden mit dem Skalar multipliziert.

- Addition von Matrizen: Für zwei Matrizen $A = (a_{kl}) \in \mathbb{R}^{n,m}$ und $B = (b_{kl}) \in \mathbb{R}^{n,m}$ ist

$$A + B \;=\; (a_{kl} + b_{kl}) \in \mathbb{R}^{n,m}.$$

  Es können nur Matrizen gleicher Grösse addiert werden.

- Multiplikation von Matrizen: Für zwei Matrizen $A = (a_{kl}) \in \mathbb{R}^{n,m}$ und $B = (b_{kl}) \in \mathbb{R}^{m,k}$ ist

$$A \cdot B \;=\; (c_{jl}) \in \mathbb{R}^{n,k} \quad \text{mit} \quad c_{jl} = \sum_{i=1}^{m} a_{ji} b_{il}, \quad j = 1, \ldots, n,\, l = 1, \ldots, k.$$

  Also ergibt die Multiplikation einer $n \times m$-Matrix und einer $m \times k$-Matrix eine $n \times k$-Matrix. Andere Produkte sind nicht definiert.

Die Multiplikation von Matrizen soll etwas genauer kommentiert werden: Wenn wir zuerst einmal den Fall $n = k = 1$ betrachten, so multiplizieren wir also einen Zeilenvektor $A \in \mathbb{R}^{1,m}$ mit einem gleichlangen Spaltenvektor $B \in \mathbb{R}^{m,1}$. Das Ergebnis ist eine 1×1-Matrix, also ein Skalar, und zwar die Zahl, die sich als Summe der Produkte der Einträge ergibt:

$$A \cdot B \;=\; \big(a_{11},\, a_{12},\, \cdots,\, a_{1m}\big) \cdot \begin{pmatrix} b_{11} \\ b_{21} \\ \\ \vdots \\ \\ b_{m1} \end{pmatrix} = \sum_{i=1}^{m} a_{1i} b_{i1}.$$

Beispiel:
Für $m = 3$ ergibt sich das Produkt von Zeilen- und Spaltenvektor zu:

$$\big(1,\, 2,\, \sqrt{2}\big) \cdot \begin{pmatrix} -1 \\ 1 \\ -\sqrt{2} \end{pmatrix} \;=\; -1 + 2 + (-2) \;=\; -1.$$

Das Produkt einer $n \times m$-Matrix A und einer $m \times k$-Matrix B lässt sich dann verstehen als die Matrix C, deren Einträge c_{jl} sich aus dem Produkt des j-ten Zeilenvektors von A mit dem l-ten Spaltenvektor aus B ergibt!

Beispiel:
Für $n = 2$, $m = k = 3$ ergibt sich das Produkt zu:

$$A \cdot B = \begin{pmatrix} 1 & 2 & \sqrt{2} \\ 0 & 3 & 1/\sqrt{2} \end{pmatrix} \cdot \begin{pmatrix} -1 & 0 & 1 \\ 1 & 7 & 1 \\ -\sqrt{2} & 0 & \sqrt{2} \end{pmatrix} = \begin{pmatrix} -1 & 14 & 5 \\ 2 & 21 & 4 \end{pmatrix},$$

wobei sich der erste Eintrag in der ersten Zeile ($= -1$) aus dem Produkt der ersten Zeile von A und der ersten Spalte von B ergibt, was genau das Produkt aus dem vorstehenden Beispiel ist.

Diese Definitionen implizieren, dass die üblichen Kommutativitäts- und Assoziativitätsgesetze für die Addition und das Assoziativitätsgesetz für die Multiplikation gelten, d.h. wir haben für alle Matrizen A, B, C passender Grösse:

$$\begin{aligned} A + B &= B + A \\ A + (B + C) &= (A + B) + C \\ A \cdot (B \cdot C) &= (A \cdot B) \cdot C, \end{aligned}$$

während das Kommutativitätsgesetz für die Matrizenmultiplikation nicht gilt, d.h. i.A. ist:

$$AB \neq BA.$$

Finde zur übung zwei 2×2-Matrizen, deren beiden Produkte nicht gleich sind!

Diagonalmatrizen sind spezielle quadratische Matrizen $A = (a_{kl}) \in \mathbb{R}^{n,n}$ mit der Eigenschaft, dass nur die Diagonaleinträge a_{kk} von Null verschieden sein können, während alle Nicht-Diagonaleinträge verschwinden: $a_{kl} = 0$, falls $k \neq l$. Die Einheitsmatrix oder Identität $I \in \mathbb{R}^{n,n}$ ist die spezielle Diagonalmatrix, deren Einträge allesamt gleich Eins sind:

$$I = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & 1 \end{pmatrix}.$$

Multiplikation einer beliebigen Matrix $A = (a_{kl}) \in \mathbb{R}^{n,m}$ mit der Identität $I \in \mathbb{R}^{m,m}$ ergibt daher immer

$$A \cdot I = A.$$

Eine weitere wichtige Basisoperationen auf Matrizen ist die Transposition: Transposition einer Matrix $A = (a_{kl}) \in \mathbb{R}^{n,m}$ ergibt die $m \times n$-Matrix, bezeichnet mit A^T, die dadurch entsteht, dass man die Zeilen von A als Spalten von A^T nimmt:

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{pmatrix} \quad \text{liefert} \quad A^T = \begin{pmatrix} a_{11} & a_{21} & \cdots & a_{n1} \\ a_{12} & a_{22} & \cdots & a_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1m} & a_{2m} & \cdots & a_{nm} \end{pmatrix}.$$

Beispiel:

$$A = \begin{pmatrix} 1 & 2 & \sqrt{2} \\ 0 & 3 & 1/\sqrt{2} \end{pmatrix} \quad \text{liefert } A^T = \begin{pmatrix} 1 & 0 \\ 2 & 3 \\ \sqrt{2} & 1/\sqrt{2} \end{pmatrix}.$$

Offensichtlich ergeben sich zwei einfache Beobachtungen:

- Das Transponieren eines Spaltenvektors ergibt einen Zeilenvektor und umgekehrt.
- Doppeltes Transponieren hat keinen Effekt: $(A^T)^T = A$.

Quadratische Matrizen mit der Eigenschaft $A^T = A$ heissen symmetrisch, bei ihnen ändert also das Vertauschen von Spalten und Zeilen nichts.

Beispiel:

$$A = \begin{pmatrix} 1 & 2 \\ 2 & 3 \end{pmatrix} \quad \text{ist symmetrisch, denn } A^T = \begin{pmatrix} 1 & 2 \\ 2 & 3 \end{pmatrix}.$$

A.3 Anwendungen

Wir wollen zwei sehr einfach Fälle betrachten: die Matrixdarstellung linearer Gleichungssysteme und die Drehung eines Vektors in der Ebene.

Beispiel: Matrixdarstellung linearer Gleichungssysteme

Ein lineares Gleichungssystem aus n Gleichungen für die n Unbekannten $x_1, \ldots, x_n$ hat die allgemeine Form

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

Unter Ausnutzung der obigen Regeln für die Matrizenmultiplikation können wir das umschreiben in die Matrixform des linearen Gleichungssystems:

$$\boxed{Ax = b\ .}$$

Dabei sind $A = (a_{ij}) \in \mathbb{R}^{n,n}$ sowie $b = (b_i) \in \mathbb{R}^{n,1}$ gegeben und $x = (x_i) \in \mathbb{R}^{n,1}$ gesucht.

Das Gleichungssystem könnte z.B. für $n = 2$ lauten

$$\begin{aligned} x_1 + 2x_2 &= 0 \\ -x_1 + 3x_2 &= 1. \end{aligned}$$

Das ergäbe dann

$$A = \begin{pmatrix} 1 & 2 \\ -1 & 3 \end{pmatrix}, \quad x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}, \quad \text{und} \quad b = \begin{pmatrix} 0 \\ 1 \end{pmatrix}.$$

Beispiel: Drehung eines Vektors in der Ebene

In Abbildung 26 ist der Spaltenvektor

$$v = \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$

als Vektor in der Zeichenebene aufgespannt durch ein x_1-x_2-Koordinatensystem dargestellt. Mit

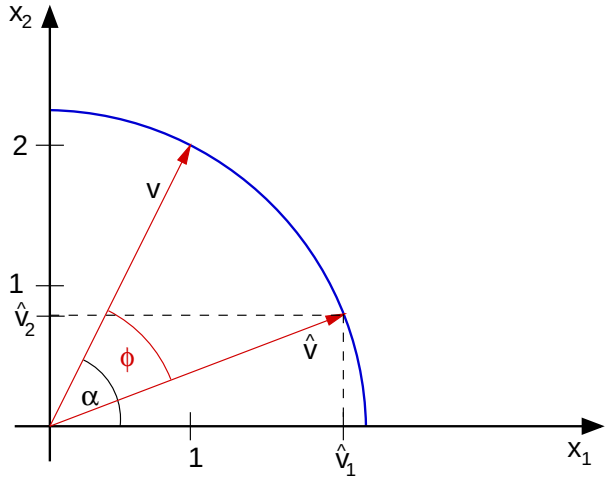


Abbildung 26: Drehung eines Vektors v um den Winkel ϕ im Uhrzeigersinn.

dem Winkel α zwischen v und der x_1-Achse erhalten wir nach den elementaren trigonometrischen Regeln:

$$v_1 = \sqrt{5}\,\cos\alpha$$
$$v_2 = \sqrt{5}\,\sin\alpha$$

Wir wollen diesen Vektor um einen Winkel ϕ im Uhrzeigersinn drehen. Wiederum nach den elementaren trigonometrischen Regeln hat dann der Ergebnisvektor

$$\hat{v} = \begin{pmatrix} \hat{v}_1 \\ \hat{v}_2 \end{pmatrix}$$

die Einträge

$$\hat{v}_1 = \sqrt{5}\,\cos(\alpha - \phi) = \sqrt{5}\,(\cos\alpha \cdot \cos\phi + \sin\alpha \cdot \sin\phi)$$
$$\hat{v}_2 = \sqrt{5}\,\sin(\alpha - \phi) = \sqrt{5}\,(\sin\alpha \cdot \cos\phi - \sin\phi \cdot \cos\alpha)\,,$$

oder

$$\hat{v}_1 = \cos\phi \cdot v_1 \;+\; \sin\phi \cdot v_2$$
$$\hat{v}_2 = -\sin\phi \cdot v_1 \;+\; \cos\phi \cdot v_2,$$

was wir vereinfacht ausdrücken können als

$$\hat{v} = C \cdot v, \qquad \text{mit} \qquad C = \begin{pmatrix} \cos\phi & \sin\phi \\ -\sin\phi & \cos\phi \end{pmatrix}.$$

Die Transformationsmatrix C hat eine interessante Eigenschaft, es gilt nämlich:

$$C^T C \;=\; C\,C^T \;=\; I.$$

Matrizen mit dieser Eigenschaft nennt man orthonormal oder unitär.

B Beispiele zur Modellierung

Lineare Gleichungssysteme mit 1000, 10 000, 100 000 oder mehr Unbekannten fallen im allgemeinen nicht vom Himmel, sonderen ergeben sich aus praktischen Anwendungen. Am Anfang steht dabei ein mathematisches Modell des betreffenden Sachverhalts. Oft handelt es sich dabei um Differential- oder Integralgleichungen. Untersuchungen zur Existenz und Eindeutigkeit von Lösungen ist Gegenstand der Analysis. Mit endlichem Aufwand können wir nur endlichdimensionale Näherungen der unbekannten Lösungsfunktion berechnen. Dazu muß das kontinuierliche Problem diskretisiert werden. Untersuchungen der Genauigkeit der resultierenden Approximationen sind Gegenstand der numerischen Analysis. Am Schluß der Kette steht dann die algebraische Lösung des diskreten Problems, beispielsweise eines linearen Gleichungssystems.

In diesem Anhang wollen wir den Modellierungs- und Diskretisierungsschritt anhand von drei anscheinend völlig verschiedener Beispiele illustrieren. Mathematisch besteht jedoch eine enge Verwandtschaft: Es handelt sich jeweils um sogennante Fredholmsche Integralgleichungen und die Diskretisierung durch sogenannte Galerkin-Verfahren für auf lineare Gleichungssysteme.

Das dabei im folgenden präsentierte Bonus–Material dient zur Einordnung von Kapitel III in einen größeren Zusammenhang. Es sei allen ans Herz gelegt, denen mit zunehmender Nähe zum Tellerrand nicht schwindelig wird. Alle anderen seien gewarnt. Die Kenntnis der folgenden Zeilen ist zur erfolgreichen Teilnahme an der Vorlesung Computerorientierte Mathematik I nicht erforderlich.

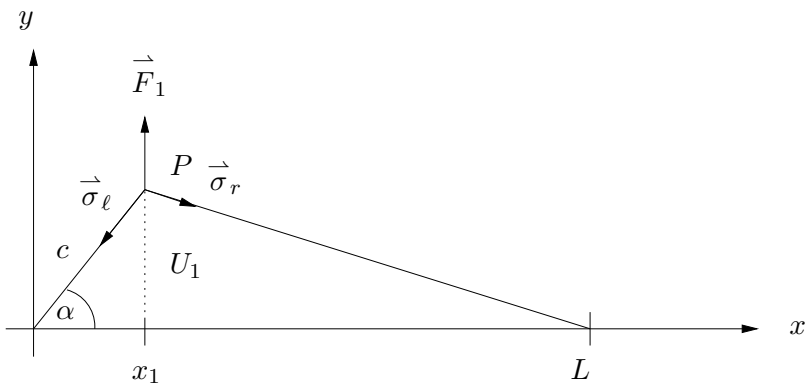
B.1 Auslenkung einer Saite

Wir betrachten eine beidseitig eingespannte Saite der Länge $L > 0$. Unser Ziel ist es, ein mathematisches Modell herzuleiten, welches es erlaubt, die Auslenkung $u : [a, b] \to \mathbb{R}$ (Einheit: Meter) durch eine gegebene vertikale Kraftdichte $f : [a, b] \to \mathbb{R}$ (Einheit: $\frac{\text{Newton}}{\text{Meter}}$) zu berechnen.

Wir betrachten als erstes die Ruhelage.

In diesem Fall heben sich in jedem Punkt P auf der Saite die angreifenden Käfte $\vec{\sigma}$ und $-\vec{\sigma}$ auf. Wir gehen im folgenden davon aus, daß die Spannung in Ruhelage $|\vec{\sigma}|$ bekannt ist.

Als zweites Teilproblem betrachten wir die Auslenkung durch eine vertikale Punktkraft $\vec{F_1} = \binom{0}{F_1}$ in $x_1 \in (0, L)$



Kräftegleichgewicht in P bedeutet: $\vec{\sigma_l} + \vec{\sigma_r} + \vec{F_1} = \vec{0}$ (Kräftebilanz).

Wir berechnen die y–Komponente von $\vec{\sigma}_l = \binom{\sigma_{l,x}}{\sigma_{l,y}}$. Es gilt

$$\sigma_{l,y} = -|\vec{\sigma_l}| \sin\alpha \ .$$

Für kleine Auslenkungen $U_1 \approx 0$ gilt

$$|\vec{\sigma_l}| \approx |\vec{\sigma}| \qquad \text{(Spannung in Ruhelage)}$$

und

$$\alpha \approx 0 \Rightarrow c \approx x_1 \Rightarrow \sin\alpha = \frac{U_1}{c} \approx \frac{U_1}{x_1} \ ,$$

also insgesamt

$$\sigma_{l,y} \approx -|\vec{\sigma}| \frac{U_1}{x_1} \ .$$

Analog erhält man

$$\sigma_{r,y} \approx -|\vec{\sigma}| \frac{U_1}{L - x_1} \ .$$

Einsetzen in die Kräftebilanz liefert

$$0 = \sigma_{l,y} + \sigma_{r,y} + F_1 \approx -|\vec{\sigma}| U_1 \left(\frac{1}{x_1} + \frac{1}{L - x_1} \right) + F_1 \ .$$

Jetzt kommt ein naheliegender Schritt. Wir setzen einfach

$$-|\vec{\sigma}| U_1 \left(\frac{1}{x_1} + \frac{1}{L - x_1} \right) + F_1 \stackrel{!}{=} 0 \ . \tag{2.1}$$

Dadurch machen wir einen Fehler, den sogenannten Modellfehler. Aus unseren überlegungen folgt, daß der Modellfehler „klein“ ist, solange U_1 „klein“ ist. Eine quantitative Kontrolle haben wir nicht.

Aus (2.1) können wir U_1 ausrechnen.

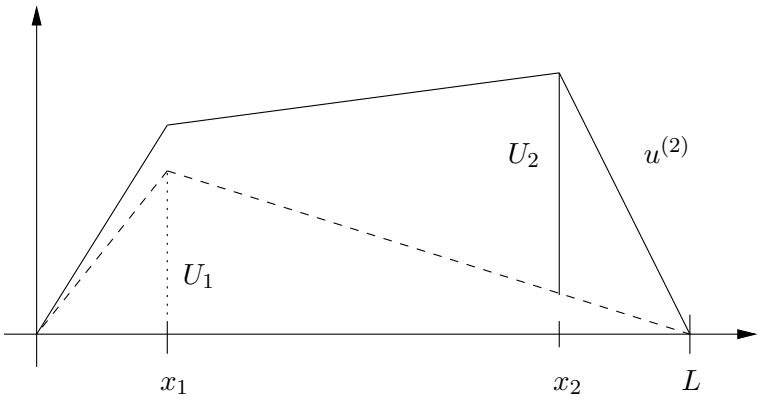
$$U_1 = \frac{(L - x_1)x_1}{|\vec{\sigma}| L} F_1 \ .$$

Durch eine weitere Kräftebilanz bestätigt man, daß die Auslenkung $U_1(x)$ auf $[0, x_1]$ und $[x_1, L]$ linear sein muß (übung). Das bedeutet

$$U_1(x) = K(x, x_1)F_1 \ ,$$

$$K(x, x_1) = \begin{cases} \frac{(L-x_1)x}{|\vec{\sigma}|L} F_1 & 0 \le x \le x_1 \\ \frac{(L-x)x_1}{|\vec{\sigma}|L} F_1 & x_1 \le x \le L \ . \end{cases}$$

Als drittes Teilproblem betrachten wir nun die Auslenkung durch zwei vertikale Punktkräfte, nämlich $\vec{F}_1 = \binom{0}{F_1}$ in $x_1 \in (0, L)$ und $\vec{F}_2 = \binom{0}{F_2}$ in $x_2 \in (0, L)$.



Im Falle $U_1 \approx 0$ und $U_2 \approx 0$ gilt für das resultierende Inkrement $U_2(x)$

$$U_2(x) \approx K(x, x_2)F_2$$

und daher für die gesamte Auslenkung $u^{(2)}(x)$

$$u^{(2)}(x) \approx U_1(x) + U_2(x) = K(x, x_1)F_1 + K(x, x_2)F_2 \ .$$

Wir machen einen zweiten Modellfehler und setzen

$$u^{(2)}(x) \stackrel{!}{=} U_1(x) + U_2(x) = K(x, x_1)F_1 + K(x, x_2)F_2.$$

Jetzt schließen wir von 2 auf n Punktquellen. Dazu sei

$$0 = x_0 < x_1 < \cdots < x_{n-1} < x_n = L$$

eine Zerlegung von $[0, L]$ und es seien $\vec{F}_0, \ldots, \vec{F}_{n-1}$ vertikale Punktkräfte in $x_0, \ldots, x_{n-1}$. Induktive Anwendung unserer Argumentation aus dem vorigen Schritt liefert für kleine Auslenkungen $u^{(n)}(x)$ die Darstellung

$$u^{(n)}(x) = \sum_{i=0}^{n-1} U_i(x) = \sum_{i=0}^{n-1} K(x, x_i)F_i.$$

Wir kommen zum letzten Schritt unserer Herleitung. Gegeben sei eine

$$\text{vertikale Kraftdichte } f : [a, b] \to \mathbb{R}.$$

Wir approximieren f durch

$$F_i = f(x_i)(x_{i+1} - x_i) \qquad i = 0, \ldots, n-1 \ .$$

Die Auslenkung durch die entsprechenden vertikalen Punktkräfte ist

$$u^{(n)}(x) = \sum_{i=0}^{n-1} K(x, x_i) f(x_i)(x_{i+1} - x_i) \ . \tag{2.2}$$

Als Modellierer gehen wir davon aus, daß diese Summe für

$$\max_{i=0,\ldots,n-1} (x_{i+1} - x_i) \to 0$$

und jedes feste $x \in [a, b]$ gegen

$$\int_0^L K(x, \xi) f(\xi) d\xi = u(x) \tag{2.3}$$

konvergiert.

Als mathematisches Modell haben wir damit die explizite Darstellung

$$u(x) = \int_0^L K(x, \xi) f(\xi) d\xi \tag{2.4}$$

mit

$$K(x, \xi) = \begin{cases} \frac{(L-\xi)x}{|\vec{\sigma}|L} & 0 \le x \le \xi \\ \frac{(L-x)\xi}{|\vec{\sigma}|L} & \xi \le x \le L \end{cases}$$

der Auslenkung $u \in C[0, L]$ einer in $u(0) = u(L) = 0$ eingespannten Saite der Länge L mit der Spannung $|\vec{\sigma}|$ durch die einwirkenden Kraftdichte $f \in C[0, L]$ hergeleitet. Ist f bekannt, so können wir u durch (numerische) Integration bestimmen.

Bemerkungen:

- Als Mathematiker wissen wir aus der Analysis I, daß für stetige f die Riemannsche Summe (2.2) gegen das Integral (2.3) konvergiert.
- Ist f nicht Riemann-integrierbar, so ist das Integral in (2.4) nicht definiert. In diesem Fall ist unser Modell mathematisch sinnlos.
- Unser Modell (2.4) liefert für alle stetigen Kraftdichten f eine Auslenkung u, ist also für alle stetigen f mathematisch sinnvoll.
- Aufgrund der Herleitung wissen wir, daß das Modell nur *für „genügend kleine“ f* physikalisch sinnvoll ist. Eine genaue Quantifizierung von „klein“ haben wir nicht hergeleitet.
- Die Herleitung von (2.4) über (2.2) liefert ein numerisches Verfahren zur Approximation von u gleich mit. Das ist typisch. Dieses Verfahren ist jedoch meist nicht besonders effizient, d.h. Aufwand und Genauigkeit der Approximation stehen in keinem guten Verhältnis. Auch das ist typisch. Bevor wir anspruchsvollere Verfahren entwickeln und verstehen können, haben wir allerdings ein paar Vorbereitungen zu treffen. Das ist allerdings Inhalt der Vorlesung Computerorientierte Mathematik II.

Ist umgekehrt u bekannt, so kann man daran denken, (2.4) als Berechnungsvorschrift für f zu verwenden. Dieser Fall ist komplizierter, denn die Werte von f sind durch (2.4) nicht explizit gegeben.

Definition B.1 *Es seien $g : [a,b] \to \mathbb{R}$ und der Kern $K : [a,b]^2 \to \mathbb{R}$ gegeben. Dann heißt*

$$\int_a^b K(x,\xi)w(\xi)\,d\xi = g(x) \qquad \forall x \in [a,b] \tag{2.5}$$

Fredholmsche Integralgleichung erster Art für die unbekannte Funktion $w : [a,b] \to \mathbb{R}$.

Die inverse Berechnung der Kraftdichte f aus der Auslenkung u erfordert also die Lösung einer Fredholmschen Integralgleichung erster Art.

B.2 Populationsdynamik

Wir betrachten die zeitliche Entwicklung einer Population, z.B. von Bakterien. Es sei

$$\text{Anzahl der Individuen zum Zeitpunkt } t: \ u(t) \ .$$

Natürlich ist $u(t) \in \mathbb{N}$. Wir gehen jedoch im folgenden davon aus, daß $u(t) \gg 1$. Dann ist die (vereinfachende!) Betrachtungsweise

$$u : \mathbb{R} \to \mathbb{R}$$

sinnvoll. Vor allem in der Ingenieurliteratur nennt man diese Annahme Kontinuumshypothese. (Es gibt keinen Zusammenhang mit der Cantorschen Kontinuumshypothese!)

Wachstum oder Rückgang der Population beschreibt die

$$\text{Wachstumsrate: } w(t) = u'(t) \ .$$

Veränderungen der Population werden durch äußere Einflüsse, also durch die

$$\text{Einwanderungsrate: } g(t) \ ,$$

durch Tod oder durch Fortpflanzung (hier: Teilung) bewirkt. Der Einfachheit halber nehmen wir an, daß unsere Bakterien unsterblich sind. Es sei

$$\text{Anteil der Individuen im Alter } \tau, \text{ die sich im Zeitraum } \Delta t \text{ teilen: } K(\tau)\Delta t \ .$$

Wir nehmen an, daß das teilungsfähige Alter τ durch $\alpha \le \tau \le \beta$ begrenzt ist.

Wir wollen nun eine Gleichung für die Wachstumsrate w ermitteln. Dazu wählen wir zunächst $\Delta t = (\beta - \alpha)/n$, n groß, und setzen

$$\tau_k = \alpha + k\Delta t \ , \qquad k = 0, \ldots, n \ .$$

Dann bilanzieren wir den Zuwachs im Zeitraum t bis $t + \Delta t$ wie folgt

$$\begin{aligned} w(t) \approx & \frac{u(t) - u(t - \Delta t)}{\Delta t} \\ \approx & g(t) + \underbrace{\frac{u(t - \tau_0) - u(t - \tau_1)}{\Delta t}}_{\text{Alter: } \tau_0 \text{ bis } \tau_1} K(\tau_0)\Delta t \\ & + \underbrace{\frac{u(t - \tau_1) - u(t - \tau_2)}{\Delta t}}_{\text{Alter: } \tau_1 \text{ bis } \tau_2} K(\tau_1)\Delta t \\ & + \cdots \\ & + \underbrace{\frac{u(t - \tau_{n-1}) - u(t - \tau_n)}{\Delta t}}_{\text{Alter: } \tau_{n-1} \text{ bis } \tau_n} K(\tau_{n-1})\Delta t \\ \approx & g(t) + \sum_{k=0}^{n-1} w(t - \tau_k) K(\tau_k) \Delta t \ . \end{aligned}$$

Für $\Delta t \to 0$ erwartet der Modellierer die Konvergenz

$$\sum_{k=0}^{n-1} w(t - \tau_k) K(\tau_k) \Delta t \to \int_{\alpha}^{\beta} w(t - \tau) K(\tau) \, d\tau \ .$$

Welche Bedingungen sind hinreichend?

Insgesamt erhalten wir als Populationsmodell die sogenannte Lotkasche Integralgleichung

$$w(t) = g(t) + \int_{\alpha}^{\beta} w(t - \tau) K(\tau) \, d\tau \ . \tag{2.6}$$

Es sei $T > 0$ fest gewählt. Um eine Integralgleichung herzuleiten, aus der sich $w(t)$ $\forall t \in [0, T]$ bestimmen lässt, machen wir zwei Annahmen. Erstens gehen wir davon aus, daß die Population erst ab $t = 0$ einwandert, daß also

$$u(t) = 0 \qquad \forall t \leq 0$$

gilt. Daraus folgt für jedes feste $t \in [0, T]$ sofort

$$w(t - \tau) = 0 \qquad \forall \tau > t \ .$$

Zweitens nehmen wir $\alpha = 0$ an. Damit erhält (2.6) die Gestalt

$$w(t) = g(t) + \int_{0}^{\min\{t, \beta\}} w(t - \tau) K(\tau) \, d\tau \qquad \forall t \in [0, T] \ .$$

Wir nehmen noch zwei äquivalente Umformungen vor. Unter Verwendung von

$$K(\tau) = 0 \qquad \forall \tau \notin [0, \beta]$$

Abbildung 27: Bilder des Weltraumteleskops Hubble vor und nach dem Einbau der Korrekturoptik. Hier die Spiralgalaxie M100.

erhält man

$$w(t) = g(t) + \int_{t-T}^{t} w(t-\tau)K(\tau)\, d\tau \qquad \forall t \in [0,T]$$

und Variablensubstitution

$$t - \tau = \eta$$

liefert

$$w(t) = g(t) + \int_{0}^{T} w(\eta)K(t-\eta)\, d\eta \qquad \forall t \in [0,T]\ . \tag{2.7}$$

Das ist unser Populationsmodell.

Definition B.2 *Es seien $g : [a,b] \to \mathbb{R}$ und der Kern $K : [a,b]^2 \to \mathbb{R}$ gegeben. Dann heißt*

$$w(x) - \int_{a}^{b} K(x,\xi)w(\xi)\, d\xi = g(x) \qquad \forall x \in [a,b] \tag{2.8}$$

Fredholmsche Integralgleichung 2. Art für die unbekannte Funktion $w : [a,b] \to I\!R$.

B.3 Bildverarbeitung

Wir untersuchen das „Hubble-Problem“. Die NASA brachte 1990 ein Spiegelteleskop in eine erdnahe Umlaufbahn, von dem man sich Aufnahmen bisher ungesehener Qualität versprach, da der störende Einfluß der Erdatmosphäre entfällt. Aufgrund eines Fertigungsfehlers (der Hauptspiegel war um wenige Mikrometer falsch geschliffen) lieferte das Teleskop jedoch nur enttäuschend verwaschene Bilder. Erst 1993 konnte eine Korrekturoptik eingebaut werden (siehe Abbildung 27). In der Zwischenzeit bemühte man sich, den Abbildungsfehler in den Daten mathematisch zu korrigieren.

Zur Vereinfachung betrachten wir als eindimensionales Problem die Abbildung eines schmalen Streifens des Firmaments auf einen schmalen Bildstreifen. Die Position am Himmel werde mit der Variablen $\xi \in [\xi_l, \xi_r]$ und die Bildposition mit der Variablen $x \in [x_l, x_r]$ bezeichnet. Dann können wir die Helligkeit des Firmaments entlang des Streifens als Funktion $w : [\xi_l, \xi_r] \to \mathbb{R}$ und die Bildhelligkeit als Funktion $b : [x_l, x_r] \to \mathbb{R}$ beschreiben.

Die Abbildungsleistung einer perfekten Optik ist dann durch

$$b(x) = w\Big((x - x_l)\frac{\xi_r - \xi_l}{x_r - x_l} + \xi_l\Big)$$

gegeben. Allerdings ist keine Optik perfekt, insbesondere eine fehlerhaft geschliffene nicht, so daß das Licht eines Punktes am Himmel auf mehrere Bildpunkte verteilt wird, oder umgekehrt ein Bildpunkt Licht von mehreren Punkten am Firmament aufsammelt:

$$b(x) = \sum_i \alpha_i w(\xi_i)$$

Eine kontinuierliche Beschreibung ist allerdings angemessener:

$$b(x) = \int_{\xi_l}^{\xi_r} K(x, \xi) w(\xi)\, d\xi \qquad \forall x \in [x_l, x_r] \tag{2.9}$$

Dabei beschreibt die Übertragungsfunktion $K(x, \xi)$, welcher Anteil des Lichts vom Himmelspunkt ξ auf den Bildpunkt x abgebildet wird. In Form von Bilddaten verfügbar ist also die Funktion b, die sich von der eigentlich interessierenden Helligkeitsverteilung w deutlich unterscheiden kann. Das „Hubble-Problem“ besteht also darin, zu der gemessenen Helligkeitsverteilung b aus (2.9) eine korrigierte Helligkeitsverteilung w zu ermitteln. Offenbar ist (2.9) eine Fredholmsche Integralgleichung erster Art.

Die Rekonstruktion des gewünschten Bildes kann sich durchaus sehen lassen (Abbildung 28).

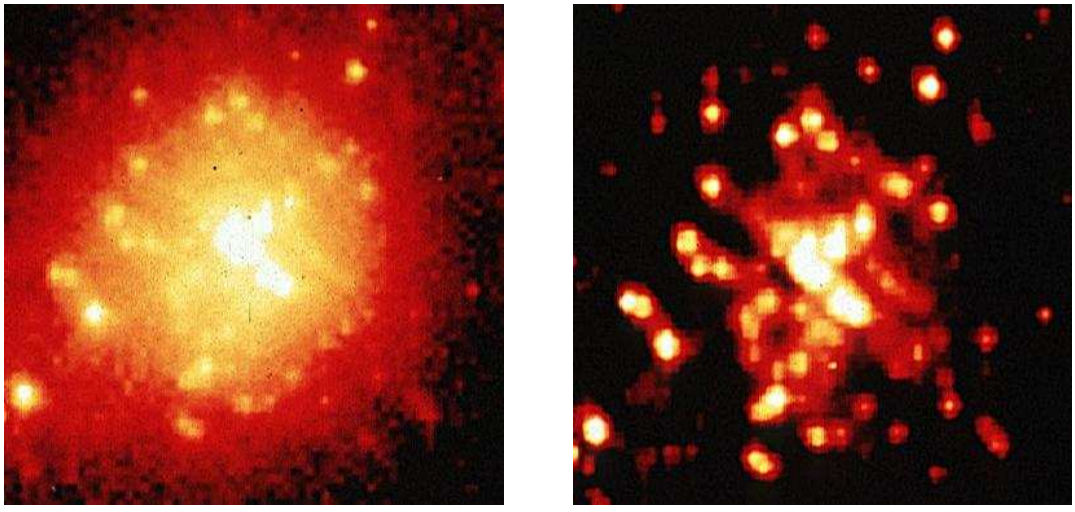


Abbildung 28: Bilder des Weltraumteleskops Hubble vor und nach der mathematischen Korrektur.

Die Untersuchung von Existenz und Eindeutigkeit von Lösungen der Integralgleichungen (2.5), (2.8) und (2.9) sprengt den Rahmen dieser Darstellung. Unter dem Stichwort „Fredholm–Operatoren“

wird dieses Thema in höchst eleganter Weise in der Funktionalanalysis behandelt. Ungeduldigen sei verraten, daß es sich bei der Fredholmsche Integralgleichung erster Art mit stetiger Kernfunktion um ein klassisches Beispiel für ein *schlecht gestelltes Problem* handelt: Bei Wahl der natürlichen Maximums-Norm auf $C[0, L]$ ist die absolute Kondition des Problems $\kappa = \infty$. Demgegenüber sind Fredholmsche Integralgleichung zweiter Art mit stetiger Kernfunktion korrekt gestellt im Sinne von Hadamard (Existenz, Eindeutigkeit und endliche Kondition). Wer es genauer wissen will sei bespielsweise auf das Lehrbuch von Werner [2] verwiesen. Wer sich speziell für Integralgleichungen interessiert, findet im Werk von Drabek und Kufner [1] eine brauchbare Einführung.

B.3.1 Galerkin–Verfahren

Wir betrachten nur die Integralgleichung erster Art (2.5). Die vorstellte Idee lässt sich direkt auf das Populationsproblem (2.8) übertragen.

Abgesehen von Spezialfällen ist eine geschlossene Lösung nicht möglich. Zur approximativen Lösung wählen wir ein Gitter

$$a = x_0 < x_1 < \cdots < x_n = b$$

und die stückweise konstanten Ansatzfunktionen

$$\varphi_k(x) = \begin{cases} 1 & x \in [x_k, x_{k+1}) \\ 0 & \text{sonst} \end{cases} , \quad k = 0, \ldots, n-1.$$

Zur Approximation von w machen wir nun den Ansatz

$$w_n = \sum_{k=0}^{n-1} W_k \varphi_k \ . \tag{2.10}$$

Um die unbekannten Koeffizienten $W_k, k = 0, \ldots, n-1$, zu bestimmen, setzen wir w_n in die Integralgleichung (2.5) ein[41] und erhalten

$$\sum_{k=0}^{n-1} W_k \int_a^b K(x, \xi)\varphi_k(\xi)\, d\xi = g(x) \ . \tag{2.11}$$

Wir können nicht erwarten, daß wir die W_k so bestimmen können, daß (2.11) für alle $x \in [a, b]$ gilt. Also fordern wir nur

$$\sum_{k=0}^{n-1} W_k \int_a^b K(x_j, \xi)\varphi_k(\xi)\, d\xi = g(x_j) =: g_j \qquad j = 0, \ldots, n-1 \ . \tag{2.12}$$

Schließlich nutzen wir die spezielle Gestalt der Ansatzfunktionen φ_k aus und erhalten

$$\int_a^b K(x_j, \xi)\varphi_k(\xi)\, d\xi = \int_{x_k}^{x_{k+1}} K(x_j, \xi)\, d\xi =: a_{jk} \ . \tag{2.13}$$

[41] Zur Approximation von (2.8) hätten wir in (2.8) einzusetzen. Das ist der einzige Unterschied.

Damit lassen sich die n Bestimmungsgleichungen (2.12) in der Form

$$\sum_{k=0}^{n-1} a_{jk} W_k = g_j \ , \qquad j = 0, \ldots, n-1 \ , \tag{2.14}$$

schreiben. Setzt man

$$A = (a_{jk})_{j,k=0}^{n-1} \in \mathbb{R}^{n,n}, \quad W = (W_k)_{k=0}^{n-1}, \quad G = (g_j)_{j=0}^{n-1} \in \mathbb{R}^n, \tag{2.15}$$

so ist (2.14) gleichbedeutend mit dem linearen Gleichungssystem

$$AW = G \ .$$

Analog erhält man zur Approximation von (2.8) die Koeffizienten

$$a_{jk} = \begin{cases} -\int\limits_{x_k}^{x_{k+1}} K(x_j, \xi)\, d\xi & j \neq k \\ 1 - \int\limits_{x_j}^{x_{j+1}} K(x_j, \xi)\, d\xi & j = k \end{cases} , \qquad j, k = 0, \ldots, n-1 \ . \tag{2.16}$$

Näherungsverfahren, die auf einem Ansatz der Form (2.10) beruhen, werden als Galerkin–Verfahren bezeichnet.

Es stellt sich die Frage, ob und unter welchen Bedingungen wir Konvergenz

$$\|w_n - w\|_\infty \to 0 \ , \qquad n \to \infty$$

erhalten. Auch diese Frage geht über unsere gegenwärtigen Möglichkeiten hinaus, und wir verweisen auf Vorlesungen über Funktionalanalysis und/oder [1, 2].

Literatur

[1] P. Drabek and A. Kufner. Integralgleichungen. Mathematik für Ingenieure und Naturwissenschaftler. Teubner, 1996. Eine anwendungsorientierte Einführung, die eine guten Hintergrund für theoretische Betrachtungen liefert.

[2] D. Werner. Funktionalanalysis. Springer, 2. Auflage, 1997. Ein gefeiertes Lehrbuch zur Einführung in die Funktionalanalysis. Anwendungen auf Integralgleichungen finden sich in Abschnitt VI. 4.