Freie Universität Berlin

INSTITUT FÜR MATHEMATIK II

Computerorientierte Mathematik I

Ralf Kornhuber, Christof Schütte und Martin Weiser

Bearbeitet von Rolf Krause, Susanna Gebauer, Alexander Fischer und Gunter Carqué

3. Auflage: Wintersemester 2003/2004

Typeset und Layout: Sabrina Nordt und Martin Weiser

Für aufmerksames Durchsehen danken wir:
Stefan Preyer

Inhaltsverzeichnis

1 Vorbemerkungen

Dieses Skriptum soll die Vorlesung "Computerorientierte Mathematik I" begleiten. Diese Veranstaltung wird gemeinsam für Studierende der Mathematik im 1. Semester und für Studierende der Bioinformatik im 3. Semester angeboten. Zusammen mit der Nachfolgeveranstaltung "Computerorientierte Mathematik II" soll sie eine möglichst elementare Einführung in den Bereich der Mathematik bieten, der sich mit den Problemen und Möglichkeiten der Computernutzung auseinandersetzt. Der Bogen der Computernutzung spannt sich heute von der Simulation realer Anwendungen — wie z.B. der Wettervorhersage mittels Computersimulation — bis hin zum automatischen Beweisen.

In einer zweistündigen Vorlesung müssen wir uns also auf Teilgebiete einschränken, in denen die Probleme und Möglichkeiten der Computernutzung in besonderem Maße zentral sind und in denen sich prägnante Einsichten mit einer Darstellung zentraler Begriffe verbinden lassen.

1.1 Worum geht es?

Die vielleicht wichtigsten Begriffe sind dabei die Folgenden:

1. Verläßlichkeit, Sicherheit und Genauigkeit:

   In wie weit kann man computergenerierten Ergebnissen trauen? Welche computer-spezifischen Fehlerquellen lassen sich ausmachen, lassen sich diese eliminieren und wenn nicht, wie kann man sie kontrollieren?

2. Effizienz:

   Wenn ich ein Problem mit dem Rechner verläßlich lösen kann, kann ich es dann auch schnell und verläßlich lösen? Lösungen werden nur dann von praktischem Interesse sein, wenn sie in der Zeit, die ich zur Verfügung habe, auch berechnet werden können.

Um diese zentralen Begriffe wollen wir die Vorlesung also aufbauen; die mathematischen Teilgebiete werden also nach ihnen ausgewählt. Die Teilgebiete, die nach der Meinung der Autoren in einem Skriptum zur Computerorientierte Mathematik vor diesem Hintergrund vor allen anderen angesprochen werden sollten, sind:

- Numerische Mathematik,
- Diskrete Mathematik,
- Computeralgebra,

natürlich nicht notwendig in dieser Reihenfolge.

In der Tat bilden Probleme aus der numerischen Mathematik in diesem Skriptum den Schwerpunkt. Wir werden uns zentral mit den Konsequenzen der Tatsache auseinandersetzen, dass heutige Rechner die Zahlen nur bis zu einer bestimmten Länge (Größe, Genauigkeit) korrekt darzustellen erlauben. Die Konsequenz der daraus resultierenden Rundungs- oder Abschneidefehler wird detailliert zu diskutieren sein und bestimmen wesentliche Teile der Kapitel 2 und 4.

Im Vergleich zur numerischen werden wir die diskrete Mathematik nur "streifen". Das ist hauptsächlich der zur Verfügung stehenden Zeit geschuldet. In Kapitel 3 werden wir Aspekte der diskreten Mathematik berühren, wenn wir die Begriffe "Komplexität" und "Effizienz" zu erarbeiten versuchen.

Leider wird die Computeralgebra in diesem Skriptum nicht vorkommen. Diese Tatsache ist wiederum der knappen Zeit geschuldet, aber auch dem Unvermögen der Verfasser eine in die zeitlichen Vorgaben passende Kurzdarstellung eines Aspektes zu erstellen. Nichtsdestoweniger ist sie zu beklagen und wir hoffen in zukünftigen Versionen Abhilfe schaffen zu können.

1.2 Programmieren in Matlab

Ein weiterer wesentlicher Teil einer Veranstaltung "Computerorientierte Mathematik" ist die Programmierung: Jede(r) Teilnehmer(in) muss grundlegende Kenntnisse der Programmierung in MATLAB erwerben, einmal um die Veranstaltung formal erfolgreich abschliessen zu können, zum anderen (und viel wichtiger) um die in der Vorlesung vermittelten Einsichten selbst am Rechner nachempfinden zu können.

Wir werden uns daher die gesamten ersten Wochen des Semesters mit der Einführung in grundlegende Elemente von MATLAB beschäftigen. Wenn Sie sich als Neuling im Bereich "UNIX-Nutzung" und/oder "Programmierung" bezeichnen würden, dann sind Sie eingeladen, ja aufgefordert, die angebotenen betreuten Rechnerübungen zu nutzen.

Dieses Skriptum enthält keinen Abschnitt über MATLAB; Grundlage der Einführung in die Programmierung in MATLAB unter UNIX ist das Skriptum

> J. Behrens und A. Iske. *MATLAB. Eine freundliche Einführung.* Version 1.1. TU München, siehe auch http://www.mathematik.tu-muenchen.de/m3/

das wir entweder gedruckt oder elektronisch für Sie bereithalten.

1.3 Wiederholung: Matrizen und Vektoren

Die grundlegende Datenstruktur in MATLAB ist die der Matrix. Im Allgemeinen sollten die zum Verständnis der Vorlesung und der MATLAB-Einführung notwendigen Grundlagen der Matrizen- und Vektorrechnung aus der Schule bekannt und geläufig sein. Da das im Speziellen leider nicht immer der Fall ist, wird die Vorlesung diesen Aspekt zu Beginn kurz wiederholen. Eine kurze Zusammenfassung findet man in Anhang A.

2 Kondition und Stabilität

Zum Aufwärmen beschäftigen wir uns mit einem (scheinbar) einfachen Problem.

Problem 2.1 (Polynomauswertung)
Gegeben seien das Polynom

$$f(x,y) = x^3 + 12xy^2 - 8y^3 - 6x^2y \tag{2.1}$$

und die Argumente

$$x = 10\,000\,000, \quad y = 4\,999\,999\ .$$

Gesucht ist der Funktionswert $f(x,y)$.

Eine Berechnungsvorschrift liefert die direkte Umsetzung des Problems in ein MATLAB–Programm.

Algorithmus 2.2 (Auswertung von $f = x^3 + 12xy^2 - 8y^3 - 6x^2y$)

```
clear all;
x = 10000000
y = 4999999
format long e;
f = x^3 + 12*x*y^2 - 8*y^3 - 6*x^2*y
```

Algorithmus 2.2 liefert das Ergebnis:

```
x =
    10000000
y =
    4999999
f =
    524288
```

also $f(x,y) = 524288$.

Unter Verwendung der binomischen Formel

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

können wir die Darstellung (2.1) von $f(x,y)$ umschreiben in

$$f(x,y) = (x-2y)^3\ . \tag{2.2}$$

Diese Darstellung führt auf das folgende MATLAB–Programm.

Algorithmus 2.3 (Auswertung von $f = (x-2y)^3$)

```
clear all;
x=10000000
y= 4999999
format long e;
f = (x - 2*y)^3}
```

Dieses Programm liefert

```
x =
    10000000
y =
    4999999
f =
    8
```

Offenbar ist für $x = 10\,000\,000$ und $y = 4\,999\,999$ gerade $y = \frac{1}{2}x - 1$ und damit das Ergebnis von Algorithmus 2.3 richtig.

Wir wollen verstehen, weshalb Algorithmus 2.2 ein so dramatisch falsches Ergebnis liefert. Beide MATLAB–Programme kommen mit

- Darstellung von Zahlen im Rechner
- Auswertung der Grundrechenarten im Rechner

aus. Hierbei muss es allem Anschein nach bei Algorithmus 2.2 ernste Probleme geben. Wir wenden uns zunächst dem ersten Punkt zu.

2.1 Zahlendarstellung und Rundungsfehler

2.1.1 Ganze Zahlen (integer)

Zur Darstellung einer ganzen Zahl z sollen N Bits

$$\boxed{d_{N-1}d_{N-2}\ \cdots\ d_0}\,, \quad d_i = 0, 1$$

verwendet werden. Man spricht von einem *Bitmuster*.

Wir beginnen mit der Darstellung *natürlicher Zahlen* als *unsigned integer*. Bei dieser Darstellung entspricht jedes Bitmuster in folgender Weise einer Zahl

$$\underbrace{\boxed{d_{N-1}d_{N-2}\ \cdots\ d_0}}_{\text{Bitmuster}} \hat{=} \underbrace{\sum_{i=0}^{N-1} d_i\, 2^i}_{\text{dargestellte Zahl}} \quad .$$

Bemerkungen:

- Das Bitmuster läßt sich als Dualzahl lesen.
- Bei gegebenem N umfasst der *darstellbare Bereich* alle ganzen Zahlen z mit der Eigenschaft

$$z_{\min} = 0 \leq z \leq z_{\max} = 2^N - 1 \ .$$

- Alle ganzen Zahlen im darstellbaren Bereich werden exakt dargestellt.

Beispiel:
Wir betrachten den Fall $N = 4$.

$$\begin{aligned} 0000 &\mathrel{\hat{=}} 0 \\ 0001 &\mathrel{\hat{=}} 1 \\ 0010 &\mathrel{\hat{=}} 2 \\ &\vdots \\ 1111 &\mathrel{\hat{=}} 15 = 2^4 - 1 \end{aligned}$$

Als nächstes beschreiben wir die Rechnerdarstellung *ganzer Zahlen* als *integer*. Mit Blick auf die obige Darstellung nichtnegativer Zahlen liegt zunächst folgende Darstellung nahe

$$\boxed{s \mid d_{N-2}d_{N-3} \ \cdots \ d_0} \mathrel{\hat{=}} (-1)^s \sum_{i=0}^{N-2} d_i \ 2^i \ .$$

Ein *Vorzeichenbit* s entscheidet, ob die dargestellte Zahl negativ ist oder nicht. Mit den übrigen Bits wird verfahren wie bisher. Wieder werden alle ganzen Zahlen im darstellbaren Bereich

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

exakt dargestellt.

Beispiel:
Wieder sei $N = 4$.

$$\begin{aligned} 0000 &\mathrel{\hat{=}} 0 & 1000 &\mathrel{\hat{=}} 0 \\ 0001 &\mathrel{\hat{=}} 1 & 1001 &\mathrel{\hat{=}} -1 \\ &\vdots & &\vdots \\ 0111 &\mathrel{\hat{=}} 7 & 1111 &\mathrel{\hat{=}} -7 = -(2^3 - 1) \end{aligned}$$

Die obige Darstellung hat folgende Nachteile

– Die Darstellung der 0 ist nicht eindeutig.
– Die Subtraktion ist nicht ohne weiteres auf Addition zurückzuführen.

Aus diesen Gründen verwendet man in der Praxis nicht die oben beschriebene Darstellung nichtpositiver Zahlen, sondern die folgende *Darstellung negativer Zahlen im Zweierkomplement*:

$$\boxed{1 \,\big|\, d_{N-2} \ \cdots \ d_0} \;\hat{=}\; -\left(1 + \sum_{i=0}^{N-2} (1 - d_i)\, 2^i\right) \ .$$

Beispiel:
Es sei wieder $N = 4$.

$$\begin{aligned}
1111 \quad &\hat{=} \; -\left(1 + \sum_{i=0}^{2} 0 \cdot 2^i\right) \quad = -1 \\
1110 \quad &\hat{=} \; -(1 \quad + \quad 1 \qquad) \quad = -2 \\
&\ \ \vdots \\
1000 \quad &\hat{=} \; -\left(1 + \sum_{i=0}^{2} 2^i\right) \qquad = -2^3
\end{aligned}$$

Eine gegebene, negative Dezimalzahl codiert man im Zweierkomplement durch Dualdarstellung, Umklappen der Bits und Addition von 1:

$$\underbrace{-3}_{\text{Dezimalzahl}} \quad = \quad \underbrace{-0011}_{\text{Dualdarstellung}} \quad \hat{=} \quad \underbrace{1101}_{\text{Bitmuster}} \ .$$

Der darstellbare Bereich ist bei gegebenem N

$$z_{\min} = -2^{N-1} = -\left(1 + \sum_{i=0}^{N-2} 1 \cdot 2^i\right) \le z \le \sum_{i=0}^{N-2} 1 \cdot 2^i = 2^{N-1} - 1 = z_{\max} \ .$$

Praxis: Intel–Prozessoren verwenden das Zweierkomplement mit $N = 16,\ 32,\ 64$ Bits (short, word, long integer). Im Falle von word–integer ist

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

Der Versuch einer Integer–Darstellung ganzer Zahlen mit der Eigenschaft

$$z < z_{\min} \quad \text{bzw.} \quad z > z_{\max}$$

führt zu *underflow* bzw. *overflow*. Es gibt verschiedene Möglichkeiten, mit underflow bzw. overflow umzugehen:

a) Fehlermeldung
b) Weiterrechnen mit

$$\tilde{z} = z \bmod m \qquad \text{(Rest bei Division von } z \text{ durch } m\text{)}.$$

Dabei ist $m = z_{\max} - z_{\min} + 1$.

Bei MATLAB wird nicht mit integer–Darstellungen gerechnet. Ganze Zahlen werden wie reelle Zahlen behandelt.

2.1.2 Reelle Zahlen (real)

Die integer–Darstellung ganzer Zahlen, die im darstellbaren Bereich liegen, ist exakt. Programme zur *symbolischen Formelauswertung*, wie z. B. MAPLE, ermöglichen exakte Darstellungen auch für

rationale Zahlen (Paare von ganzen Zahlen), algebraische Zahlen (Tupel von ganzen Zahlen) und einige transzendente Zahlen (über entsprechende Funktionen). Dieser Weg führt in die Computer–Algebra und wir wollen ihn hier nicht weiter verfolgen.

Unser Ziel ist es, mit N Bits eine möglichst genaue, aber im allgemeinen *nicht exakte*, Darstellung reeller Zahlen zu ermöglichen. Eine solche Darstellung bildet die Grundlage von *numerischem Rechnen*.

Definition 2.4 *Gegeben seien*

Basis:	$q \in \mathbb{N} \qquad q \geq 2$
Bereich darstellbarer Exponenten:	$[e_{\min}, e_{\max}] \cap \mathbb{Z}$
Mantissenlänge:	$\ell \in \mathbb{N}$

Lässt sich dann eine Zahl $x \in \mathbb{R}$ mit

Exponent:	$[e_{\min}, e_{\max}] \cap \mathbb{Z}$
Mantisse:	$a = \sum_{i=1}^{\ell} a_i q^{-i}, \quad a_i = 0, ..., q-1, \quad a_1 \neq 0$
Vorzeichenbit:	$s \in \{0, 1\}$

in der Form

$$\tilde{x} = (-1)^s a q^e \tag{2.3}$$

schreiben, so heißt $\tilde{x}$ Gleitkommazahl. Die Darstellung (2.3) heißt normalisierte Gleitkommadarstellung von $\tilde{x}$. Die Menge aller Gleitkommazahlen nennen wir G, genauer $G(q, \ell, [e_{\min}, e_{\max}])$.

Bemerkungen:

- Die Codierung einer Gleitkommazahl im Rechner erfolgt über die normalisierte Darstellung. Man hat dazu das Vorzeichenbit s, die Mantisse a, also $a_1, \ldots, a_\ell$, und den Exponent e, also eine ganze Zahl, zu speichern.
- Ist $\tilde{x} = aq^e \in G$, so gilt offenbar
$$\tilde{x} = aq^e = Aq^E$$
mit $A = a\, q^j$, $E = e - j$ und beliebiges $j \in \mathbb{Z}$. Die Normalisierung $q^{-1} \leq a < 1$ sichert die *Eindeutigkeit* der Codierung im Rechner.

Beispiel:

Wir betrachten den Fall $q = 2$. Dann kann wegen $a_i \in \{0, 1\}$, $i = 1, \ldots, \ell$ die Mantisse direkt als Bitmuster abgespeichert werden. Wegen $a_1 \neq 0$ ist notwendigerweise a_1 stets 1 und muß nicht mit abgespeichert werden. Dazu kommen ein Vorzeichenbit und die integer–Darstellung des Exponenten e als Bitmuster der Länge M.

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

Ist zum Beispiel $\ell = 5$ und $M = 3$, so wird die rationale Dualzahl -110.11 mit $N = 1 + (\ell - 1) + M$ Bits wie folgt im Rechner dargestellt

$$-110.11 = -0.11011\ 2^3 \hat{=} \boxed{1 \,\big|\, 1011 \,\big|\, 011}\ .$$

Praxis:

	q	ℓ	$e_{\min}$	$e_{\max}$
IEEE-Arithmetik (float)	2	24	-127	128
IEEE-Arithmetik (double)	2	53	-1023	1024
Cray–Rechner	2	48,96	-16384	8191
Wiss. Rechner	10	10	-98	100
IBM System/309	16			

Offenbar gilt für jede Gleitkommazahl $\tilde{x}$

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

Im Falle von $|\tilde{x}| < x_{\min}$ bzw. $|\tilde{x}| > x_{\max}$ gibt es *underflow* bzw. *overflow*. Tritt underflow auf, so wird bei den meisten Rechnern mit $\tilde{x} = 0$ weitergerechnet, overflow führt auf $\tilde{\mathtt{x}} = \mathtt{inf}$ (infinity) und damit auf mehr oder weniger sinnlose Ergebnisse.

Beachte, 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 damit erkauft, daß nun natürlich nicht mehr alle ganzen Zahlen x mit $x_{\min} \le x \le x_{\max}$ *exakt* darstellbar sind.

Beispiel:
Wir betrachten diesmal $q = 2$, $\ell = 2$ und $M = 5$. Dann ist die ganze Dualzahl

$$x = 101 = 0.101\ 2^3$$

nicht in G, da die Mantisse $a = 0.101$ nicht mit $\ell = 2$ Bits darstellbar ist. Es liegt nahe, in solchen Fällen zu runden und genau das wird auch gemacht.

Satz 2.5 *Es sei q ein Vielfaches von zwei. Zu jeder Zahl $x \in [x_{\min}, x_{\max}]$, $|x| \ge q^{e_{\min}}$, existiert ein* $\mathrm{rd}(x) \in G$ *mit relativem Fehler*

$$\frac{|x - \mathrm{rd}(x)|}{|x|} \le \frac{1}{2} q^{-(\ell-1)} =: \mathrm{eps}.$$

Die Zahl eps *heißt Maschinengenauigkeit.*

Beweis: Sei $x \in [x_{\min}, x_{\max}]$ und ohne Beschränkung der Allgemeinheit (o.B.d.A.) $x \ge q^{e_{\min}}$, dann existieren

$$a^* = \sum_{i=1}^{\infty} a_i q^{-i}\ , \qquad a_i \in \{0, ..., q-1\}\ , \qquad a_1 \neq 0\ ,$$

und $e \in [e_{\min}, e_{\max}]$, so daß

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

Beachte insbesondere, daß aus $q^e > a^* q^e = x \ge q^{e_{\min}}$ die Abschätzung $e \ge e_{\min}$ folgt.

Runden von a^* auf ℓ Stellen

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

liefert

$$|a - a^*| \leq \frac{1}{2} q^{-\ell} \ . \tag{2.4}$$

Zum Beweis von (2.4) betrachten wir zwei Fälle.

Fall 1: $a_{\ell+1} < \frac{1}{2}q$ (abrunden).

Wegen $\frac{1}{2}q \in \mathbb{N}$ ist dann $a_{\ell+1} + 1 \leq \frac{1}{2}q$. Daher gilt

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

Fall 2: $a_{\ell+1} \geq \frac{1}{2}q$ (aufrunden).

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

Damit ist (2.4) bewiesen.

Wir setzen

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

Nun müssen wir zeigen, daß

$$\mathrm{rd}(x) \in G \tag{2.5}$$

gilt. Im Falle $q^{-1} < a < 1$ ist das klar.

Beim Aufrunden kann aber noch der Fall $a = 1$ auftreten, falls $a_1 = a_2 = ... = a_\ell = q - 1$ vorliegt. Dann ist $\mathrm{rd}(x) = 1\ q^e = 0.1\ q^{e+1} \in G$, falls $e < e_{\max}$ ist. Der Fall $e = e_{\max}$ steht aber im Widerspruch zu $x \in [x_{\min}, x_{\max}]$. Damit ist auch (2.5) erledigt.

Nun folgt aus (2.4) sofort

$$\begin{aligned} |x - \mathrm{rd}(x)| &= |a^* - a| q^e \\ &\leq \frac{1}{2} q^{e-\ell} \ . \end{aligned}$$

Weiter gilt wegen $a_1 \geq 1$

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

und insgesamt

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

■

Bemerkung: Die Abbildung

$$\mathrm{rd} \ : \ [x_{\min}, x_{\max}] \to G$$

ordnet jedem $x \in \mathbb{R}$ genau eine Gleitkommazahl $\tilde{x} = \mathrm{rd}(x) \in G \subset [x_{\min}, x_{\max}]$ zu.

Offenbar ist

$$\tilde{x} = \mathrm{rd}(x) \qquad \forall x \in G$$

und daher $\mathrm{rd} \circ \mathrm{rd} = \mathrm{rd}$ ($\circ$ bedeutet die Hintereinanderausführung). Eine Abbildung mit dieser Eigenschaft heißt *Projektion*.

Beim Auswerten von $\mathrm{rd}(x)$ wird auf ℓ Stellen gerundet. $\mathrm{rd}(x)$ hat also ℓ *gültige Stellen* (im q–System).

Bemerkung: Vor allem in der Ingenieurliteratur wird der relative Fehler oft in Prozent angegeben. 1% Fehler bedeuten zum Beispiel

$$\frac{|x - \tilde{x}|}{|x|} \leq \frac{1}{100} \ .$$

Es existiert ein $\varepsilon \in \mathbb{R}$ mit

$$\mathrm{rd}(x) = (1 + \varepsilon)x \qquad |\varepsilon| \geq \mathrm{eps}.$$

Achtung: Der *absolute Fehler*

$$|x - \mathrm{rd}(x)| \leq |x|\mathrm{eps}$$

wächst mit $|x|$.

Offenbar kann es vorkommen, daß verschiedene Zahlen x, y auf dieselbe Gleitkommazahl $\tilde{x} = \mathrm{rd}(x) = \mathrm{rd}(y)$ gerundet werden: Die Abbildung rd ist nicht injektiv!

Satz 2.6 *Es sei*

$$\tilde{x} = aq^e \in G \ .$$

Dann gilt $\tilde{x} = \mathrm{rd}(x)$ *für alle* x *mit der Eigenschaft*

$$-r \leq x - \tilde{x} < r \ , \qquad r = \frac{1}{2} q^{e-\ell} \ .$$

Beweis: Es sei

$$\tilde{x} = aq^e = q^e \sum_{i=1}^{\ell} a_i q^{-i}$$

und

$$x = q^e \left(\sum_{i=1}^{\ell} a_i q^{-i} + w \right) = a^* q^e \ .$$

Ist $\tilde{x} = \mathrm{rd}(x)$, so folgt aus (2.4) sofort $|a - a^*| = |w| \leq \frac{1}{2} q^{-\ell}$. Beim Abrunden gilt sogar das Kleinerzeichen, also

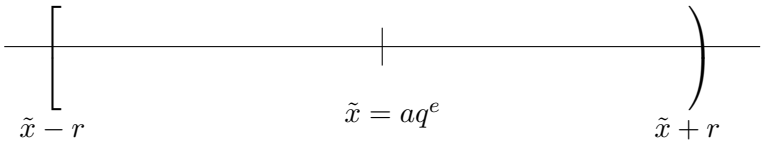
$$-\frac{1}{2} q^{-\ell} \leq w < \frac{1}{2} q^{-\ell} \ .$$

Also ist

$$-\frac{1}{2} q^{e-\ell} \leq x - \tilde{x} = q^e w < \frac{1}{2} q^{e-\ell}.$$

■

Bemerkung: Die Gleitkommazahl $\tilde{x}$ repräsentiert ein ganzes Intervall reeller Zahlen $E(\tilde{x}) = [\tilde{x} - r, \tilde{x} + r)$, $r = \frac{1}{2} q^{e-\ell}$.



Im Sinne der Äquivalenzrelation (symmetrisch, reflexiv, transitiv)

$$x \cong y \overset{\text{Def.}}{\Leftrightarrow} \mathrm{rd}(x) = \mathrm{rd}(y)$$

sind alle Zahlen $x \in E(\tilde{x})$ äquivalent.

Gleichheit von x und $\tilde{x}$ im Rahmen der Gleitkommadarstellung bedeutet also $x \in E(\tilde{x})$, d.h. x ist gleich $\tilde{x}$, falls $x \in [\tilde{x} - r, \tilde{x} + r]$. Deshalb sind Abfragen der Form `x == x̃` kritisch und sogar falsch und sollten durch einen Ausdruck der Gestalt $|\mathtt{x} - \tilde{\mathtt{x}}| \leq \mathtt{r}$ ersetzt werden.

Merke: Rechnen mit Gleitkommazahlen $\tilde{x} \in G$ bedeutet Rechnen mit Intervallen $E(\tilde{x}) \subset \mathbb{R}$.

2.2 Kondition

2.2.1 Relative Kondition

Wir wollen untersuchen, wie sich Rundungsfehler in den Eingabedaten auf die Ergebnisse der Elementaroperationen $+, -, \cdot, /$ auswirken.

Dabei betrachten wir nur den relativen Fehler und setzen $\tilde{x} = \mathrm{rd}(x)$.

Addition: Es seien $x, y > 0$

$$\begin{aligned}
\frac{|(x+y) - (\tilde{x}+\tilde{y})|}{|x+y|} &= \frac{|(x-\tilde{x}) + (y-\tilde{y})|}{|x+y|} \leq \frac{|x-\tilde{x}| + |y-\tilde{y}|}{|x+y|} \\
&= \frac{|x|}{|x+y|}\frac{|x-\tilde{x}|}{|x|} + \frac{|y|}{|x+y|}\frac{|y-\tilde{y}|}{|y|} \\
&\leq \frac{|x|+|y|}{|x+y|} \max\left\{\frac{|x-\tilde{x}|}{|x|}, \frac{|y-\tilde{y}|}{|y|}\right\} \\
&= \max\left\{\frac{|x-\tilde{x}|}{|x|}, \frac{|y-\tilde{y}|}{|y|}\right\} .
\end{aligned}$$

Der relative Fehler wird nicht vergrößert!

Subtraktion: Sei $x - y \neq 0$ und $x, y > 0$

$$\begin{aligned}
\frac{|(x-y) - (\tilde{x}-\tilde{y})|}{|x-y|} &\leq \frac{|x|}{|x-y|}\frac{|x-\tilde{x}|}{|x|} + \frac{|y|}{|x-y|}\frac{|y-\tilde{y}|}{|y|} \\
&\leq \frac{|x|+|y|}{|x-y|} \max\left\{\frac{|x-\tilde{x}|}{|x|}, \frac{|y-\tilde{y}|}{|y|}\right\} .
\end{aligned}$$

Offenbar gilt

$$x \approx y \Rightarrow \frac{|x|+|y|}{|x-y|} \gg 1.$$

Folgerung: Bei Subtraktion nahezu gleich großer Zahlen kann der Verstärkungsfaktor über alle Grenzen wachsen. Man spricht von *Auslöschung*.

Beispiel:
Wir gehen aus von $\ell = 4$ gültigen Stellen im Dezimalsystem. Also ist die Maschinendarstellung von $x = \frac{1}{3}$ gerade $\tilde{x} = 0.3333$. Weiter sei $y = \tilde{y} = 0.3332$. Wir erhalten

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

also statt vorher vier nur noch eine gültige Stelle! Der Rechner gibt im allgemeinen statt „?“ irgendeine Zahl aus. Es gibt keinen Grund für deren Richtigkeit! Man kann nachrechnen, daß in unserem Beispiel die Verstärkung des relativen Fehlers bei $\frac{1}{4}10^4$ liegt.

Multiplikation: Es gilt für $x, y \neq 0$

$$\frac{|xy - \tilde{x}\tilde{y}|}{|xy|} \leq \left(2 + \max\left\{\frac{|x-\tilde{x}|}{|x|}, \ \frac{|y-\tilde{y}|}{|y|}\right\}\right) \max\left\{\frac{|x-\tilde{x}|}{|x|}, \ \frac{|y-\tilde{y}|}{|y|}\right\}$$

(übung). Offenbar ist

$$\lim_{\tilde{x}\to x, \tilde{y}\to y} \left(2 + \max\left\{\frac{|x-\tilde{x}|}{|x|}, \frac{|y-\tilde{y}|}{|y|}\right\}\right) = 2 \ ,$$

d.h. kleine relative Fehler in x, y werden bei Multiplikation nur wenig vergrößert.

Division: Für $x, y \neq 0$ und $\tilde{y} \neq 0$ berechnet man

$$\frac{\left|\frac{x}{y} - \frac{\tilde{x}}{\tilde{y}}\right|}{\left|\frac{x}{y}\right|} \leq \left|\frac{y}{x}\right| \left(\frac{|x-\tilde{x}|}{|y|} + \frac{|\tilde{x}||\tilde{y}-y|}{|y||\tilde{y}|}\right) \leq \left(1 + \frac{|y||\tilde{x}|}{|x||\tilde{y}|}\right) \max\left\{\frac{|x-\tilde{x}|}{|x|}, \ \frac{|y-\tilde{y}|}{|y|}\right\}.$$

und es gilt

$$\lim_{\tilde{x}\to x, \tilde{y}\to y} \left(1 + \frac{|y||\tilde{x}|}{|x||\tilde{y}|}\right) = 2.$$

Kleine relative Fehler in x, y werden also bei Division nur wenig vergrößert.

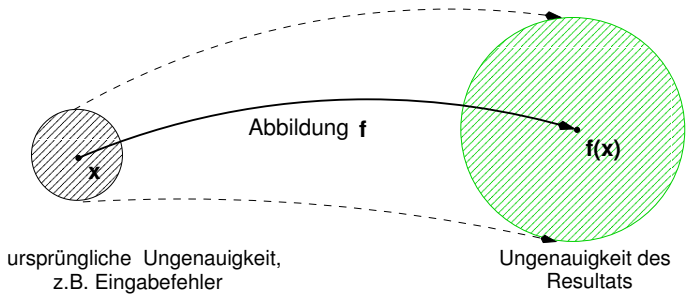


Abbildung 1: Die Kondition mißt die Verstärkung der Ungenauigkeit durch die Operation f.

Aus den obigen Beobachtungen destillieren wir nun den zentralen Begriff dieses Abschnitts.

Definition 2.7 *Gegeben sei eine Funktion $f : \mathbb{R}^n \to \mathbb{R}$ und das Problem*

$$\textit{Auswertung von } f(x) \neq 0 \textit{ für die gegebenen Daten } x = (x_1, \ldots, x_n). \qquad \text{(P)}$$

Zu jedem $\varepsilon > 0$ gebe es eine Zahl $\kappa_{\text{rel}}(\varepsilon)$, so daß

$$\frac{|f(x) - f(\tilde{x})|}{|f(x)|} \leq \kappa_{\text{rel}}(\tilde{x}) \max\left\{ \frac{|x_1 - \tilde{x}_1|}{|x_1|}, \ldots, \frac{|x_n - \tilde{x}_n|}{|x_n|} \right\} . \tag{2.6}$$

für alle $\tilde{x} = (\tilde{x}_1, \ldots, \tilde{x}_n) \in \mathbb{R}^n$ mit

$$\max\left\{ \frac{|x_1 - \tilde{x}_1|}{|x_1|}, \ldots, \frac{|x_n - \tilde{x}_n|}{|x_n|} \right\} \leq \varepsilon.$$

Zudem gebe es zu jedem $\varepsilon > 0$ ein solches $\tilde{x}$, so daß in (2.6) Gleichheit gilt. Dann ist

$$\kappa_{\text{rel}} := \lim_{\varepsilon \to 0} \kappa_{\text{rel}}(\varepsilon)$$

die relative Kondition von (P). Im Falle $\kappa_{\text{rel}} \gg 1$ heißt (P) schlecht konditioniert (bezüglich des relativen Fehlermaßes), sonst gut konditioniert.

Bemerkung: Ist (P) schlecht konditioniert, so liegt eine *unvermeidbare Fehlerverstärkung* vor. Es bewirken *kleine Störungen der Daten große Störungen des Resultates.*

Bemerkungen:

- Die Kondition ist abhängig von der Aufgabenstellung (hier: Auswertung einer Funktion f) *und* den Daten (hier: $x_1, \ldots, x_n$).
- Lässt man Bedingung (ii) in Definition 2.7 weg, so erhält man nur eine obere Schranke für die relative Kondition κ_{rel}.

Beispiel:
Der skalare Fall $n = 1$
Sei $f : \mathbb{R} \to \mathbb{R}$ differenzierbar in $x \in \mathbb{R}$ und $f(x) \neq 0$. Dann gilt für beliebiges $\tilde{x} \in \mathbb{R}$, $\tilde{x} \neq x$,

$$\frac{|f(x) - f(\tilde{x})|}{|f(x)|} = \kappa_{\text{rel}}(\tilde{x}) \frac{|x - \tilde{x}|}{|x|}$$

mit

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

Nach Definition der Ableitung gilt

$$\lim_{\tilde{x} \to x} \kappa_{\text{rel}}(\tilde{x}) = |f'(x)| \frac{|x|}{|f(x)|} = \kappa_{\text{rel}} .$$

Wie lässt sich dieses Resultat interpretieren?

Beispiel: Addition
Im Falle der *Addition* positiver Zahlen, also $f(x_1, x_2) = x_1 + x_2$ mit $x_1, x_2 > 0$, haben wir

$$\frac{|(x_1 + x_2) - (\tilde{x}_1 + \tilde{x}_2)|}{|x_1 + x_2|} \leq \kappa_{\text{rel}}(\tilde{x}_1, \tilde{x}_2) \max\left\{ \frac{|x_1 - \tilde{x}_1|}{|x_1|}, \frac{|x_2 - \tilde{x}_2|}{|x_2|} \right\}$$

mit $\kappa_{\text{rel}}(\tilde{x}_1, \tilde{x}_2) = 1$ gezeigt. Im Falle $x_1 > \tilde{x}_1$, $x_2 > \tilde{x}_2$ und $|x_1 - \tilde{x}_1|/|x_1| = |x_2 - \tilde{x}_2|/|x_2|$ gilt Gleichheit. Also ist nach Definition 2.7 $\kappa_{\text{rel}} = \kappa_{\text{rel}}(\tilde{x}_1, \tilde{x}_2) = 1$.

Beispiel: Subtraktion

Bei der *Subtraktion* haben wir $f(x_1, x_2) = x_1 - x_2$ mit $x_1, x_2, x_1 - x_2 > 0$. Wir haben gesehen, daß

$$\frac{|(x_1 - x_2) - (\tilde{x}_1 - \tilde{x}_2)|}{|x_1 - x_2|} \leq \kappa_{\text{rel}}(\tilde{x}_1, \tilde{x}_2) \max\left\{ \frac{|x_1 - \tilde{x}_1|}{|x_1|}, \frac{|x_2 - \tilde{x}_2|}{|x_2|} \right\} \qquad \forall \tilde{x}_1, \tilde{x}_2 \in \mathbb{R} \tag{2.7}$$

mit

$$\kappa_{\text{rel}}(\tilde{x}_1, \tilde{x}_2) = \frac{|x_1| + |x_2|}{|x_1 - x_2|}$$

gilt. Analog zur Addition findet man für jedes $\varepsilon > 0$ gestörte Daten $\tilde{x}_1$, $\tilde{x}_2$ mit $|x_1 - \tilde{x}_1| \leq \varepsilon$ und $|x_2 - \tilde{x}_2| \leq \varepsilon$, für welche in (2.7) Gleichheit gilt: Die Abschätzung ist für beliebig kleine Störungen scharf. Damit ist nach Definition 2.7

$$\kappa_{\text{rel}} = \kappa_{\text{rel}}(\tilde{x}_1, \tilde{x}_2) = \frac{|x_1| + |x_2|}{|x_1 - x_2|} \ .$$

Für $x \approx y$, ist die Subtraktion schlecht konditioniert, sonst aber gut konditioniert (Datenabhängigkeit!).

Beispiel: Multiplikation

Im Falle der *Multiplikation*, also $f(x_1, x_2) = x_1 x_2$ mit $x_1, x_2 \neq 0$, haben wir

$$\frac{|x_1 x_2 - \tilde{x}_1 \tilde{x}_2|}{|x_1 x_2|} \leq \kappa_{\text{rel}}(\tilde{x}_1, \tilde{x}_2) \max\left\{ \frac{|x_1 - \tilde{x}_1|}{|x_1|}, \frac{|x_2 - \tilde{x}_2|}{|x_2|} \right\} \qquad \forall \tilde{x}_1, \tilde{x}_2 \in \mathbb{R}$$

mit

$$\kappa_{\text{rel}}(\tilde{x}_1, \tilde{x}_2) = 2 + \max\left\{ \frac{|x - \tilde{x}|}{|x|}, \frac{|y - \tilde{y}|}{|y|} \right\}$$

erhalten. Auch diese Abschätzung ist für beliebig kleine Störungen scharf. Weiter folgt aus $\tilde{x}_1^\nu \to x_1$ und $\tilde{x}_2^\nu \to x_2$ für $\nu \to \infty$ sofort

$$\lim_{\nu \to \infty} \kappa_{\text{rel}}(\tilde{x}_1^\nu, \tilde{x}_2^\nu) = 2 = \kappa_{\text{rel}} \ .$$

Beispiel: Division

Auf die *Division*, also $f(x_1, x_2) = \frac{x_1}{x_2}$ mit $x_1, x_2 \neq 0$, ist Definition 2.7 nicht anwendbar (warum nicht?), sondern muß leicht modifiziert werden (wie?). Für $\tilde{x}_2 \neq 0$ erhält man

$$\kappa_{\text{rel}}(\tilde{x}_1, \tilde{x}_2) = 1 + \frac{|x_2||\tilde{x}_1|}{|x_1||\tilde{x}_2|} \to 2 = \kappa_{\text{rel}} \ .$$

Beispiel: Polynomauswertung

Wir betrachten unser Modellproblem (2.1), die Auswertung von

$$f(x, y) = x^3 - 6x^2 y + 12xy^2 - 8y^3 = (x - 2y)^3$$

in $x = 10\ 000\ 000$, $y = 4\ 999\ 999$. Da wir den Mittelwertsatz der Differentialrechnung noch nicht kennen, müssen wir etwas rechnen. Es gilt zunächst

$$\begin{aligned} (x - 2y)^3 - (\tilde{x} - 2\tilde{y})^3 &= (x - 2y)^3 - (x - 2y + (\tilde{x} - x - 2(\tilde{y} - y)))^3 \\ &= -3(x - 2y)^2(\tilde{x} - x - 2(\tilde{y} - y)) \\ &\quad -3(x - 2y)(\tilde{x} - x - 2(\tilde{y} - y))^2 \\ &\quad -(\tilde{x} - x - 2(\tilde{y} - y))^3. \end{aligned}$$

Einsetzen liefert

$$\frac{|(x - 2y)^3 - (\tilde{x} - 2\tilde{y})^3|}{|(x - 2y)^3|} \leq \kappa_{\text{rel}}(\tilde{x}, \tilde{y}) \max\left\{ \frac{|x - \tilde{x}|}{|x|}, \frac{|y - \tilde{y}|}{|y|} \right\}$$

mit

$$\kappa_{\text{rel}}(\tilde{x},\tilde{y}) = \frac{3|x|+6|y|}{|x-2y|}\left(1+\frac{3(|x|+2|y|)^2}{|x-2y|}\max\left\{\frac{|x-\tilde{x}|}{|x|},\frac{|y-\tilde{y}|}{|y|}\right\}+\frac{(|x|+2|y|)^3}{|x-2y|^2}\max\left\{\frac{|x-\tilde{x}|}{|x|},\frac{|y-\tilde{y}|}{|y|}\right\}^2\right).$$

Auch diese Abschätzung ist scharf für beliebig kleine Störungen, und es gilt

$$\lim_{\tilde{x}\to x,\tilde{y}\to y}\kappa_{\text{rel}}(\tilde{x},\tilde{y}) = \frac{3|x|+6|y|}{|x-2y|} = \kappa_{\text{rel}}\ .$$

Einsetzen von $x = 10\,000\,000$ und $y = 4\,999\,999$ liefert

$$\kappa_{\text{rel}} = \frac{30\,000\,000 + 29\,999\,994}{2} = 29\,999\,997 \approx 3\cdot 10^7 \gg 1.$$

Unsere scheinbar harmlose Polynomauswertung ist also ziemlich schlecht konditioniert!

2.2.2 Absolute Kondition

Es mag unter gewissen Umständen sinnvoll sein, sich für die Auswirkungen des *absoluten Fehlers* in den Daten auf den *absoluten Fehler* des Funktionswertes zu interessieren. Aufschluß darüber gibt die *absolute Kondition*, die wir in vollständiger Analogie zur relativen Kondition definieren (vgl. Definition 2.7).

Definition 2.8 *Zu jedem $\varepsilon > 0$ gebe es eine Zahl $\kappa_{\text{abs}}(\varepsilon)$, so daß*

$$|f(x) - f(\tilde{x})| \le \kappa_{\text{abs}}(\tilde{x})\max\{|x_1-\tilde{x}_1|,\ldots,|x_n-\tilde{x}_n|\}\ . \tag{2.8}$$

für alle $\tilde{x} = (\tilde{x}_1,\ldots,\tilde{x}_n) \in \mathbb{R}^n$ mit

$$\max\{|x_1-\tilde{x}_1|,\ldots,|x_n-\tilde{x}_n|\} \le \varepsilon.$$

Zudem gebe es zu jedem $\varepsilon > 0$ ein solches $\tilde{x}$, so daß in (2.8) *Gleichheit gilt. Dann ist*

$$\kappa_{\text{abs}} := \lim_{\varepsilon\to 0}\kappa_{\text{abs}}(\varepsilon)$$

die absolute Kondition von (P)*. Im Falle $\kappa_{\text{abs}} \gg 1$ heißt* (P) *schlecht konditioniert (bezüglich des absoluten Fehlermaßes), sonst gut konditioniert.*

Es gibt Probleme, deren relative Kondition schlecht, deren absolute Kondition aber gut ist und umgekehrt!

Zusammen mit einer Konditionszahl sollte man also immer angeben, auf welches Fehlermaß (relativ oder absolut) sich diese bezieht!

Beispiel: Der skalare Fall $n = 1$
Sei $f : \mathbb{R} \to \mathbb{R}$ differenzierbar in $x \in \mathbb{R}$. Dann gilt für beliebiges $\tilde{x} \in \mathbb{R}$, $\tilde{x} \ne x$,

$$|f(x) - f(\tilde{x})| = \kappa_{\text{abs}}(\tilde{x})\ |x-\tilde{x}|$$

mit

$$\kappa_{\text{abs}}(\tilde{x}) = \frac{|f(x)-f(\tilde{x})|}{|x-\tilde{x}|}\ .$$

Nach Definition der Ableitung gilt

$$\lim_{\tilde{x}\to x} \kappa_{\text{abs}}(\tilde{x}) = |f'(x)| = \kappa_{\text{abs}}.$$

Wie lässt sich dieses Resultat interpretieren?

Beispiel: Multiplikation
Es gilt

$$|x \cdot y - \tilde{x} \cdot \tilde{y}| \leq \kappa_{\text{abs}}(\tilde{x}) \max\{|x - \tilde{x}|, |y - \tilde{y}|\}$$

mit $\kappa_{abs}(\tilde{x}) = |\tilde{x}| + |y|$. Offenbar konvergiert

$$\lim_{\tilde{x}\to x} \kappa_{\text{abs}}(\tilde{x}) = |x| + |y| = \kappa_{\text{abs}}$$

und damit ist $\kappa_{\text{abs}} = |x| + |y| \gg \kappa_{\text{rel}} = 2$, falls $|x| \gg 1$ oder $|y| \gg 1$.

2.3 3–Term–Rekursion

Schlechte Kondition tritt nicht nur in Zusammenhang mit Auslöschung auf. Als Beispiel betrachten die folgende Aufgabe:
Finde das Folgenglied x_{50} der Folge x_k, $k = 0, 1, ...$ mit den Eigenschaften

$$\begin{aligned} & x_0 = 1\ , \qquad x_1 = \sqrt{2} - 1 && \text{(a)} \\ & x_{k+1} + 2x_k - x_{k-1} = 0\ , \qquad k = 1, 2, ...\ . && \text{(b)} \end{aligned} \tag{2.9}$$

(2.9) heißt *Anfangswertproblem für die homogene Differenzengleichung 2–ter Ordnung* (2.9b), kurz 3–Term–Rekursion.

Bemerkung: x_k beschreibt z.B. die zeitliche Entwicklung einer Population.

Bemerkung: Problem (2.9) hat eine eindeutig bestimmte Lösung.

Es liegt nahe, ausgehend von den Anfangswerten, die Folgenglieder x_{k+1} mittels der Differenzengleichung (2.9b) aus x_k und x_{k-1} zu bestimmen. Diese Idee führt sofort auf

Algorithmus 2.9 (Rekursion)

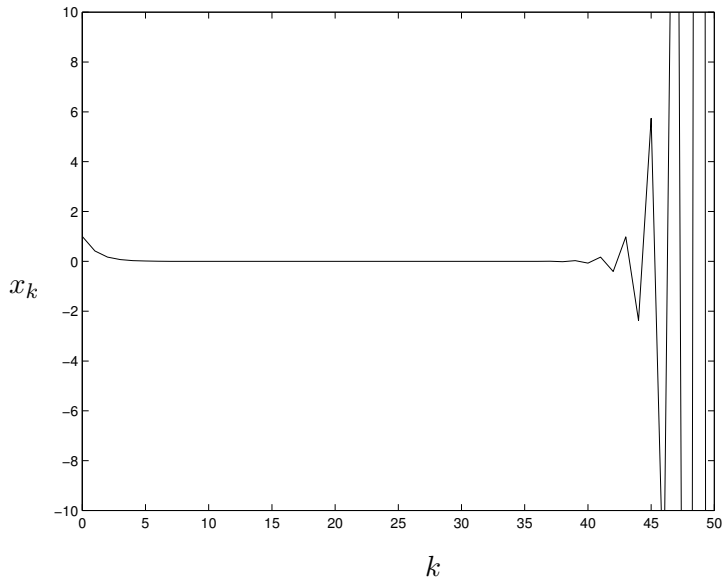
```
function []=rekursion1

a = 2;
b = -1;

x0 = 1;
x1 = sqrt(2) - 1;

x(1) = x0;
x(2) = x1;

K(1) = 0;
K(2) = 1;
```

```
for k = 2:50

   x(k+1) = - a*x(k) - b*x(k-1);
   K(k+1) = k;

end

disp 'Rekursion liefert:'

x_50 = x(51)

plot(K,x,'-r');

axis([0,50,-10,10]);
```

Das Ergebnis ist

```
x_50=
   -4.704832528316996e+02
```

und folgende Grafik, welche x_k über $k = 0, ..., 50$ zeigt.



Nach unseren Erfahrungen mit der Polynomauswertung sind wir mißtrauisch geworden und wollen das Problem genauer analysieren. Etwas allgemeiner betrachten wir die Differenzengleichung

$$x_{k+1} + ax_k + bx_{k-1} = 0 \tag{2.10}$$

für gegebene Koeffizienten $a, b \in \mathbb{R}$ und Anfangswerte $x_0, x_1 \in \mathbb{R}$.

Satz 2.10 *Seien λ_1, λ_2 Nullstellen des charakteristischen Polynoms*

$$\lambda^2 + a\lambda + b = 0$$

und α, β Lösung von

$$\begin{aligned} \alpha + \beta &= x_0 \\ \alpha\lambda_1 + \beta\lambda_2 &= x_1 \ . \end{aligned}$$

Dann gilt:

$$\boxed{x_k = \alpha\lambda_1^k + \beta\lambda_2^k \ .}$$

Beweis: Vollständige Induktion:
Die Fälle $k = 0, 1$ sind klar. Die Behauptung gelte nun für $k, k-1$ mit $k \geq 1$. Dann folgt

$$\begin{aligned} &\alpha\lambda_1^{k+1} + \beta\lambda_2^{k+1} + a(\alpha\lambda_1^k + \beta\lambda_2^k) + b(\alpha\lambda_1^{k-1} + \beta\lambda_2^{k-1}) \\ &= \alpha\lambda_1^{k-1}(\lambda_1^2 + a\lambda_1 + b) + \beta\lambda_2^{k-1}(\lambda_2^2 + a\lambda_2 + b) = 0 \ . \end{aligned}$$

■

Wir wollen Satz 2.10 auf unser Beispiel anwenden. Das zugehörige charakteristische Polynom

$$\lambda^2 + 2\lambda - 1 = 0$$

hat die Nullstellen $\lambda_1 = \sqrt{2} - 1$, $\lambda_2 = \sqrt{2} + 1$. Die Koeffizienten $\alpha = 1$, $\beta = 0$ erhält man aus

$$\begin{aligned} \alpha + \beta &= 1 \\ \alpha\lambda_1 + \beta\lambda_2 &= \sqrt{2} - 1 \ . \end{aligned}$$

Die Lösung ist also

$$\boxed{x_k = \lambda_1^k > 0 \ , \qquad k = 0, 1, \ldots \ .}$$

Allgemeiner liefert Satz 2.10 folgenden Algorithmus:

Algorithmus 2.11 (Geschlossene Lösung)

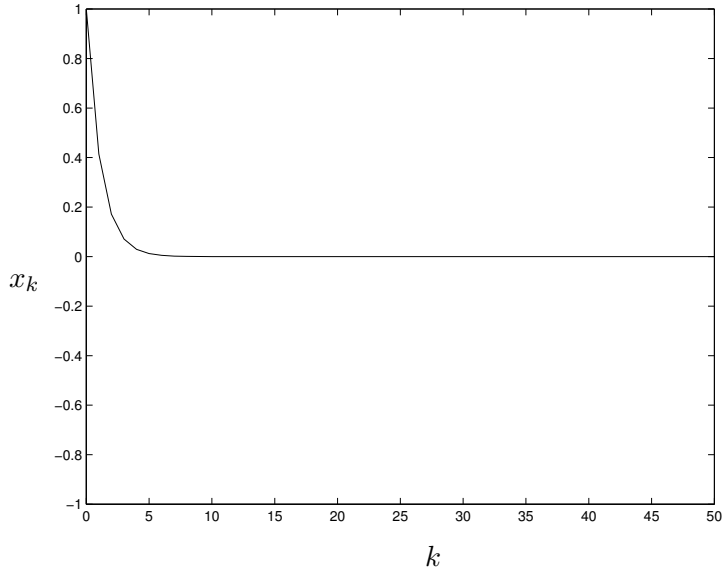
```
function []=rekursion2
a = 2;
b = -1;

x0 = 1;
x1 = sqrt(2) - 1;

lambda1 = -a/2 + sqrt(a^2/4-b);
```

```
lambda2 = -a/2 - sqrt(a^2/4-b);

format long e

beta = (x1 - lambda1*x0)/(lambda2 - lambda1)
alpha = 1 - beta

for k = 1:51

   y(k) = alpha*lambda1^(k-1)+beta*lambda2^(k-1);
   K(k) = k-1;

end

disp 'geschlossene Loesung liefert:'
x_50 = y(k)

plot(K,y,'-g');
axis([0,50,-1,1]);
```

Das Ergebnis ist

```
x_50=
   7.264667356934908e-20
```

Das starke Abklingen der Folge x_k zeigt auch die folgende Abbildung



Mit Blick auf die geschlossene Lösung aus Satz 2.10 und das entsprechende Ergebnis von Algorithmus 2.11 erweist sich das Resultat von Algorithmus 2.9 als völlig falsch!

Um zu verstehen wie das passieren konnte, untersuchen wir zunächst die *Kondition* unseres Problems (2.9). Offenbar wird durch $f(x_0, x_1) := x_{50}$ eine Abbildung $f : \mathbb{R}^2 \to \mathbb{R}$ definiert. Entsprechend Definition 2.7 berechnen wir die Lösung $\tilde{x}_{50} = f(\tilde{x}_0, \tilde{x}_1)$ zu den gestörten Anfangsdaten

$$\tilde{x}_0 = 1 + \varepsilon_0 \ , \qquad \tilde{x}_1 = (1 + \varepsilon_1)\lambda_1$$

mit relativem Fehler $\varepsilon_0, \varepsilon_1 \leq \text{eps}$. Wir erhalten die gestörten Koeffizienten

$$\tilde{\alpha} = 1 + \varepsilon_0 \left(1 + \frac{\lambda_1}{2}\right) - \varepsilon_1 \frac{\lambda_1}{2} \ , \qquad \tilde{\beta} = \frac{\lambda_1}{2}(\varepsilon_1 - \varepsilon_0)$$

und schließlich die gestörte Lösung

$$\tilde{x}_k = \left(1 + \varepsilon_0 \left(1 + \frac{\lambda_1}{2}\right) - \varepsilon_1 \frac{\lambda_1}{2}\right) \lambda_1^k + \frac{\lambda_1}{2}(\varepsilon_1 - \varepsilon_0)\lambda_2^k \ .$$

Offenbar gilt

$$\begin{aligned} \frac{|\tilde{x}_k - x_k|}{|x_k|} &= \frac{\left|\left(\varepsilon_0 \left(1 + \frac{\lambda_1}{2}\right) - \varepsilon_1 \frac{\lambda_1}{2}\right) \lambda_1^k + \frac{\lambda_1}{2} (\varepsilon_1 - \varepsilon_0) \lambda_2^k\right|}{\lambda_1^k} \\ &\leq \left(1 + \lambda_1 + \lambda_1 \left(\frac{\lambda_2}{\lambda_1}\right)^k\right) \max\{\varepsilon_1, \varepsilon_2\} \end{aligned}$$

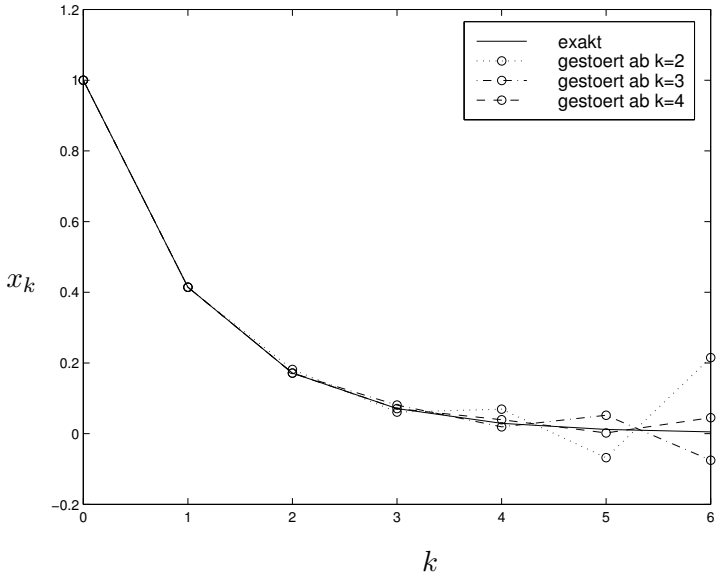
Wegen $\frac{\lambda_2}{\lambda_1} = \frac{\sqrt{2}+1}{\sqrt{2}-1} > 1$ wächst der Verstärkungsfaktor mit wachsendem k exponentiell an! Für $k = 50$ gilt insbesondere

$$\frac{|x_{50} - \tilde{x}_{50}|}{|x_{50}|} \leq \left(1 + \lambda_1 + \lambda_1 \left(\frac{\lambda_2}{\lambda_1}\right)^{50}\right) \max\{\varepsilon_1, \varepsilon_2\}.$$

Die Kondition von unserer 3–Term–Rekursion ist daher

$$\kappa_{\text{rel}} = 1 + \lambda_1 + \lambda_1 \left(\frac{\lambda_2}{\lambda_1}\right)^{50} \approx 1.8 \cdot 10^{38},$$

also unglaublich schlecht! Der Grund liegt im parasitären Lösungszweig λ_2^k, der exponentiell wächst, während die Lösung exponentiell fällt. Dieser Sachverhalt ist in der folgenden Abbildung illustriert. Trotz exakter Anfangsdaten führen Störungen von $\varepsilon = 0.01$ jeweils ab $k = 2, 3$ und 4 schnell zu falschen Ergebnissen. Im realistischen Fall von $\varepsilon \approx 10^{-16}$ tritt dieser Effekt nur später ein.



Im Anhang C werden wir überlegen, wie man das Problem so umformulieren kann, daß (natürlich aus anderen Eingabedaten!) der gesuchte Wert λ_{50} recht genau berechnet werden kann.

2.4 Stabilität

In diesem Abschnitt wollen wir endlich klären, was bei unserem Modellproblem (2.1) schiefgegangen ist. Zunächst kommen wir dazu wieder auf die Elementaroperationen zurück.

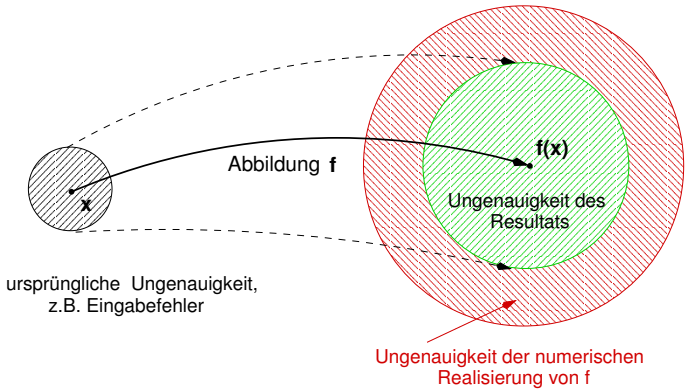


Abbildung 2: Während die Kondition die unvermeidbare Verstärkung der Ungenauigkeit durch die Abbildung f misst, bezieht sich der Stabilitätsbegriff auf die zusätzliche Ungenauigkeit durch die numerische Realisierung von f.

Nach Durchführung jeder Elementaroperationen muß das Ergebnis wieder gerundet werden. Anstelle von $x + y$ erhält man also beispielsweise $\mathrm{rd}(\tilde{x} + \tilde{y})$ mit $\tilde{x} = \mathrm{rd}(x)$, $\tilde{y} = \mathrm{rd}(y)$. Zu dem (verstärkten) Eingangsfehler kommt also nach Durchführung einer Elementaroperation noch ein *zusätzlicher* Rundungsfehler!

Die Auswertung eines Polynoms erfordert die geschachtelte Auswertung von Elementaroperationenen. In jedem Schritt wird dabei der Eingangsfehler verstärkt und *es tritt ein neuer Rundungsfehler hinzu*. Diese zusätzlichen Fehler haben wir bei der Konditionsanalyse nicht betrachtet.

Es gibt verschiedene Möglichkeiten, ein und dasselbe Polynom in Elementaroperationen zu zerlegen, beispielsweise

$$f(x, y) = x^3 - 6x^2y + 12xy^2 - 8y^3 = (x - 2y)^3.$$

Je nachdem, welche Elementaroperationen vorkommen und in welcher Reihenfolge sie vorkommen, können sich all die im Laufe der Rechnung auftretenden, zusätzlichen Rundungsfehler mehr oder weniger stark *aufschaukeln*. Je weniger, desto *stabiler* ist der entsprechende Algorithmus. Wir wollen diesen Sachverhalt an zwei Beispielen studieren.

Beispiel: Distributivgesetz

Wir gehen von $\ell = 4$ gültigen Stellen im Dezimalsystem $q = 10$ aus. Ziel ist die Auswertung von

$$f(x, y, z) = x(y - z) = xy - xz \qquad x = 1111,\ y = 1234,\ z = 1233.$$

Die Kondition des Problems ist

$$\kappa_{\mathrm{rel}} = 1 + \frac{|y| + |z|}{|y - z|} = 2468,$$

also ziemlich schlecht. Offenbar ist aber $\mathrm{rd}(x) = x$, $\mathrm{rd}(y) = y$ und $\mathrm{rd}(z) = z$. Es treten also *keine* Eingangsfehler auf, die *unvermeidlicherweise* durch die schlechte Kondition verstärkt werden könnten.
Der erste Algorithmus beruht auf der Auswertung von $x(y - z)$. Wir erhalten

$$\begin{aligned} \mathrm{rd}(\mathrm{rd}(y) - \mathrm{rd}(z)) &= \mathrm{rd}(y - z) = \mathrm{rd}(1234 - 1233) = \mathrm{rd}(1) = y - z \\ \mathrm{rd}(\mathrm{rd}(x) \cdot \mathrm{rd}(\mathrm{rd}(y) - \mathrm{rd}(z))) &= \mathrm{rd}(1111 \cdot 1) = \mathrm{rd}(1111) = 1111 \end{aligned}$$

und damit die exakte Lösung!
Der zweite Algorithmus beruht auf der Auswertung von $xy - xz$. Wir erhalten

$$\begin{aligned}\mathrm{rd}(\mathrm{rd}(x)\cdot\mathrm{rd}(y)) &= \mathrm{rd}(x\cdot y) = \mathrm{rd}(1111\cdot 1234) = \mathrm{rd}(1370974) = 1371000\\ \mathrm{rd}(\mathrm{rd}(x)\cdot\mathrm{rd}(z)) &= \mathrm{rd}(x\cdot y) = \mathrm{rd}(1111\cdot 1233) = \mathrm{rd}(1369863) = 1370000\\ \mathrm{rd}(\mathrm{rd}(x)\cdot\mathrm{rd}(y) - \mathrm{rd}(x)\cdot\mathrm{rd}(z))) &= \mathrm{rd}(1371000 - 1370000) = \mathrm{rd}(1000) = 1000\end{aligned}$$

und damit ein ziemlich falsches Ergebnis!

Beispiel: Polynomauswertung

Wir betrachten wieder unser Modellproblem (2.1), also die Auswertung von

$$f(x,y) = x^3 - 6x^2y + 12xy^2 - 8y^3 = (x-2y)^3$$

in $x = 10\ 000\ 000$, $y = 4\ 999\ 999$. Wie wir schon wissen, ist die relative Kondition dieses Problems $\kappa_{\mathrm{rel}} \approx 3\cdot 10^7$. Andererseits werden die Daten $x = 10\ 000\ 000$ und $y = 4\ 999\ 999$ exakt im Rechner dargestellt. Es treten also keine Eingangsfehler auf. Dennoch lieferte die Auswertung von $f(x,y) = x^3 - 6x^2y + 12xy^2 - 8y^3$ ein völlig falsches Ergebnis, im Gegensatz zur Auswertung der Formulierung $f(x,y) = (x-2y)^3$. Um diesen Sachverhalt zu klären, wollen wir nun eine *Stabilitätsanalyse* der beiden entsprechenden Algorithmen durchführen. Dazu werden wir, genau wie der Rechner, die Auswertung jeweils in eine Folge von Elementaroperationen zerlegen und in jedem Schritt über Verstärkung der Eingangsfehler und zusätzliche Rundungsfehler Buch führen. Um dieses etwas mühselige Geschäft zu vereinfachen, sind wir mit einer näherungsweisen Betrachtung zufrieden und vernachlässigen dazu angesichts von $\mathrm{eps}^2 \approx 10^{-32} \ll \mathrm{eps} \approx 10^{-16}$ alle Terme, die eps^2 enthalten (schreiben aber trotzdem "=" oder "$\leq$").

1. Möglichkeit: *Auswertung von* $f(x,y) = (x^3 + 12xy^2) - (6x^2y + 8y^3)$ *(Algorithmus 2.2)*
Eingangsfehler:

$$\frac{|x-\tilde{x}|}{|x|} \leq \mathrm{eps}, \qquad \frac{|y-\tilde{y}|}{|y|} \leq \mathrm{eps},$$

Auswertung von x^2:

$$\frac{|x\cdot x - \tilde{x}\cdot\tilde{x}|}{|x\cdot x|} \leq 2\max\{\mathrm{eps},\mathrm{eps}\} = 2\mathrm{eps}.$$

Rundung:

$$\frac{|\tilde{x}\cdot\tilde{x} - \mathrm{rd}(\tilde{x}\cdot\tilde{x})|}{|\tilde{x}\cdot\tilde{x}|} \leq \mathrm{eps}.$$

Also insgesamt

$$\frac{|x\cdot x - \mathrm{rd}(\tilde{x}\cdot\tilde{x})|}{|x|} \leq 2\mathrm{eps} + \mathrm{eps}\frac{|x|}{|\tilde{x}|} = 3\mathrm{eps} + \mathrm{o}(\mathrm{eps}).$$

Um die Sache etwas abzukürzen, setzen wir

$$f_1(x) = x^3, \qquad \tilde{f}_1(\tilde{x}) = \mathrm{rd}(\tilde{x}\cdot\mathrm{rd}(\tilde{x}\cdot\tilde{x})).$$

Analog zum obigen Vorgehen folgt

$$\frac{|f_1(x) - \tilde{f}_1(\tilde{x})|}{|f_1(x)|} \leq 7\mathrm{eps}.$$

Genauso verfahren wir mit den anderen Produkten und erhalten

$$f_2(x,y) = 12xy^2, \quad \tilde{f}_2(\tilde{x},\tilde{y}) = \mathrm{rd}(12\cdot\tilde{x}\cdot\mathrm{rd}(\tilde{y}\cdot\tilde{y})), \quad \tfrac{|f_2(x,y)-\tilde{f}_2(\tilde{x},\tilde{y})|}{|f_2(x,y)|} \leq 7\mathrm{eps},$$

$$f_3(x,y) = 6x^2y, \quad \tilde{f}_3(\tilde{x},\tilde{y}) = \mathrm{rd}(6\cdot\tilde{y}\cdot\mathrm{rd}(\tilde{x}\cdot\tilde{x})), \quad \tfrac{|f_3(x,y)-\tilde{f}_3(\tilde{x},\tilde{y})|}{|f_2(x,y)|} \leq 7\mathrm{eps},$$

$$f_4(y) = 8y^3, \quad \tilde{f}_4(\tilde{y}) = \mathrm{rd}(8\cdot\tilde{y}\cdot\mathrm{rd}(\tilde{y}\cdot\tilde{y})), \quad \tfrac{|f_4(y)-\tilde{f}_4(\tilde{y})|}{|f_2(x,y)|} \leq 7\mathrm{eps}.$$

Nun kommt die Addition positiver Zahlen

$$f_5(x,y) = f_1(x) + f_2(x,y), \quad \tilde{f}_5(\tilde{x},\tilde{y}) = \mathrm{rd}(\tilde{f}_1(\tilde{x}) + (\tilde{f}_2(\tilde{x},\tilde{y})).$$

Bekanntlich werden dabei Eingangsfehler nicht verstärkt, es kommt nur ein Rundungsfehler hinzu. Insgesamt ist also

$$\frac{|f_5(x,y) - \tilde{f}_5(\tilde{x},\tilde{y})|}{|f_5(x,y)|} \leq 7\mathrm{eps} + \mathrm{eps} = 8\mathrm{eps}.$$

Analog folgt

$$f_6(x,y) = f_3(x,y) + f_4(y), \quad \tilde{f}_6(\tilde{x},\tilde{y}) = \mathrm{rd}(\tilde{f}_3(\tilde{x},\tilde{y}) + \tilde{f}_4(\tilde{y})), \quad \frac{|f_6(x,y) - \tilde{f}_6(\tilde{x},\tilde{y})|}{|f_6(x,y)|} \leq 8\mathrm{eps}.$$

Schließlich kommt die Subtraktion, also

$$f(x,y) = f_5(x,y) - f_6(x,y), \quad \tilde{f}(\tilde{x},\tilde{y}) = \mathrm{rd}(\tilde{f}_5(\tilde{x},\tilde{y}) - \tilde{f}_6(\tilde{x},\tilde{y})).$$

Aus der bekannten Abschätzung der Fehlerverstärkung bei der Subtraktion positiver Zahlem erhalten wir

$$\frac{|f(x,y) - \tilde{f}(\tilde{x},\tilde{y})|}{|f(x,y)|} \leq 8\frac{|f_5(x,y)| + |f_6(x,y)|}{|f(x,y)|} + \mathrm{eps} = \left(8\frac{|x^3 + 12xy^2| + |6x^2y + 8y^3|}{|x-2y|^3} + 1\right)\mathrm{eps}.$$

Das ergibt also insgesamt den Verstärkungsfaktor

$$8\frac{|x^3 + 12xy^2| + |6x^2y + 8y^3|}{|x-2y|^3} + 1 \approx 10^{21} \gg \kappa_{\mathrm{rel}}.$$

Dementsprechend müssen wir im Ergebnis mit einem relativen Fehler der Größenordnung

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

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

2. Möglichkeit: *Auswertung von* $f(x,y) = (x-2y)^3$ *(Algorithmus 2.3).*
Wir beginnen mit

$$f_1(x,y) = x - 2y, \quad \tilde{f}_1(\tilde{x},\tilde{y}) = \mathrm{rd}(f_1(\tilde{x},\tilde{y})).$$

Unsere Fehlerverstärkungsformel für die Subtraktion liefert

$$\frac{|f_1(x,y) - \tilde{f}_1(\tilde{x},\tilde{y})|}{|f_1(x,y)|} \leq \frac{|x| + 2|y|}{|x-2y|}\mathrm{eps} + \mathrm{eps}.$$

Nun kommt die erste Multiplikation, also

$$f_2(x,y) = f_1(x,y)^2, \quad \tilde{f}_2(\tilde{x},\tilde{y}) = \mathrm{rd}(\tilde{f}_1(\tilde{x},\tilde{y})^2).$$

Wir erhalten

$$\frac{|f_2(x,y) - \tilde{f}_2(\tilde{x},\tilde{y})|}{|f_2(x,y)|} \leq 2\frac{|x| + 2|y|}{|x-2y|}\mathrm{eps} + \mathrm{eps}.$$

Zum Abschluß die zweite Multiplikation

$$f(x,y) = f_1(x,y) \cdot f_2(x,y), \quad \tilde{f}(\tilde{x},\tilde{y}) = \mathrm{rd}(\tilde{f}_1(\tilde{x},\tilde{y}) \cdot \tilde{f}_2(\tilde{x},\tilde{y})).$$

Der gesamte Fehler genügt also der Abschätzung

$$\frac{|f(x,y) - \tilde{f}(\tilde{x},\tilde{y})|}{|f(x,y)|} \leq 2\left(1 + 2\frac{|x| + 2|y|}{|x-2y|}\right)\mathrm{eps} + \mathrm{eps} = \left(3 + 4\frac{|x| + 2|y|}{|x-2y|}\right)\mathrm{eps}.$$

Wir erhalten also den Verstärkungsfaktor

$$3 + 4\frac{|x| + 2|y|}{|x-2y|} \approx 10^8 \approx \kappa_{\mathrm{rel}}.$$

Um unsere Stabilitätsanalyse numerisch zu illustrieren, betrachten wir eine leichte Modifikation des MATLAB–Programms in Algorithmus 2.2, nämlich

```
x = 1000 0000;
y =  499 9999;

a = x^3 + 12*x*y^2
b = 8*y^3 + 6*x^2*y

sprintf ('% 20.1f',a)
sprintf ('% 20.1f',b)

f = a - b
```

Das Ergebnis ist

```
a=
   3.9999900000120e+21
b=
   3.9999900000120e+21

ans =
   3.9999900000120258560.0
ans =
   3.9999900000119734272.0

f=
   524288
```

MATLAB liefert mit den obigen Werten $\tilde{a}, \tilde{b}$ gerade die Rundung der exakten Werte a, b auf 16 Stellen genau (Nachrechnen mit Bleistift und Papier)! Intern werden jedoch 20 Stellen verwendet, wie die zweite Ausgabe von $\tilde{a}$ und $\tilde{b}$ in einem anderen Format zeigt. Bei der Subtraktion von $\tilde{a} - \tilde{b}$ tritt *Auslöschung* auf. Der Verstärkungsfaktor ist

$$\kappa_{\text{rel}} = \frac{a+b}{|a-b|} \approx \frac{8 \cdot 10^{21}}{2} \approx 10^{21}.$$

Hier findet sich also unsere oben ermittelte Fehlerverstärkung wieder. Die Tatsache, daß 524388 gerade eine Zweierpotenz ist, deutet darauf hin, daß die ausgelöschten Ziffern intern mit Einsen (im Dualsystem) aufgefüllt werden.

Unsere Stabilitätsanalyse kann man zu einer formalen Definition und einer darauf gründenden Theorie ausbauen (vgl. z.B. Deuflhard und Hohmann [1]). Das wollen wir an dieser Stelle nicht tun. Daher beschränken wir uns auf eine verbale Beschreibung des Stabilitätsbegriffs.

Definition 2.12 *Ein Algorithmus $\tilde{f}(\tilde{x}_1, \ldots, \tilde{x}_n)$ zur Lösung von* (P) *heißt stabil, falls die zusätzlichen Rundungsfehler nach jeder Elementaroperation nur zu einer leichten zusätzlichen Verstärkung der Eingangsfehler führen.*

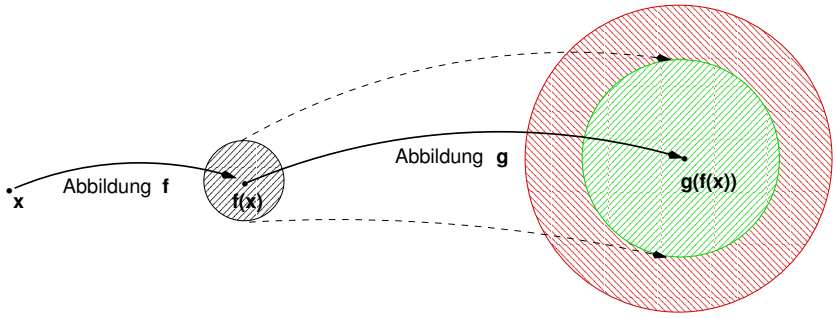


Abbildung 3: Geschachtelte Auswertung: Selbst bei fehlerfreier Eingabe x schaukeln sich Instabilitäten auf, noch verstärkt durch unvermeidbare Fehlerverstärkungen. Vgl. Abb. 4.

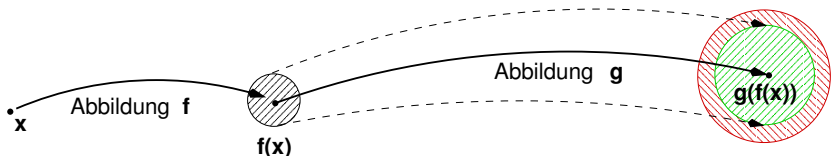


Abbildung 4: Geschachtelte Auswertung: Im stabilen Fall kommt es nur zu einer leichten Fehlerverstärkung durch die Realisierung. Damit ist die gesamte Fehlerverstärkung gering, wenn das Problem gut konditioniert ist. Vgl. Abb. 3.

Bemerkungen:

- Stabilität ist eine Eigenschaft des Algorithmus und *nicht* des zu lösenden Problems.
- Ist das zu lösende Problem schlecht konditioniert (unvermeidbare Fehlerverstärkung), so ist die Verwendung eines stabilen Verfahrens unabdingbar.
- Im Sinne von Definition 2.12 ist Algorithmus 2.3 *stabil* und Algorithmus 2.2 *instabil*. Zwecks stabiler Formulierung von Funktionsauswertungen sollte man folgende Faustregeln beherzigen.
  - Vermeide unnötige Auslöschung!
  - Unvermeidliche Auslöschung an den Anfang des Algorithmus.

Literatur

[1] P. Deuflhard and A. Hohmann. *Numerische Mathematik I.* de Gruyter, 1993. Ein algorithmisch orientiertes Lehrbuch der numerischen Mathematik, von dem in weiterführenden Vorlesungen noch die Rede sein wird. Weiterführendes zu den bisher behandelten Themen findet sich in Kapitel 2 (Fehleranalyse) und Kapitel 6 (3–Term–Rekursionen).

[2] C.W. Ueberhuber. *Numerical Computation 1.* Springer, 1997. Erster Teil eines zweibändigen Werkes über grundlegende Fragen zu numerischem Rechnen. Die Anfangskapitel illustrieren sehr kurz und knapp typische Fragestellungen anhand einfacher Beispiele. Kapitel 4 bietet eine ausführliche Diskussion der Zahlendarstellung im Rechner. Später geht es um geschicktes Programmieren, numerische Software im Internet und typische Modellierungstechniken. Es handelt sich eher um ein Nachschlagwerk als um ein Lehrbuch.

[3] J.H. Wilkinson. *Rundungsfehler.* Springer, 1969. Das klassische Standardwerk. Die erste systematische Untersuchung zu einem damals brandneuen Thema.

3 Komplexität und Effizienz

3.1 Sortieralgorithmen

Wir betrachten folgendes *Sortierproblem*. Gegeben seien

$$z_1, z_2, ..., z_n \in \mathbb{R} \ .$$

Das Problem lautet: Finde eine Permutation $\pi_1, ..., \pi_n$, so daß

$$z_{\pi_1} \leq z_{\pi_2} \leq ... \leq z_{\pi_n} \ .$$

Definition 3.1 *Eine Permutation (Vertauschung) von* $\{1, ..., n\}$ *ist eine bijektive Abbildung von* $\{1, ..., n\}$ *auf sich. Wir schreiben* $\pi(k) = \pi_k$, $k = 1, \ldots, n$.

Bemerkung: Die Zahlen $z_1, \ldots, z_n$ sollen also der Größe nach sortiert werden. Dieses Problem ist lösbar. Die Lösung ist nicht eindeutig bestimmt.

Algorithmus 3.2 (Ausprobieren)
Probiere so lange alle möglichen Permutationen durch, bis die gewünschte Eigenschaft vorliegt.

Dieser Algorithmus macht keinen sonderlich durchdachten Eindruck.

Satz 3.3 *Es gibt* $n! = 1 \cdot 2 \cdot 3 \cdot ... \cdot n$ *Permutationen von* $\{1, ..., n\}$.

Beweis: Vollständige Induktion. ■

Bemerkung: Bei Anwendung von Algorithmus 3.2 müssen wir im schlimmsten Fall

$$(n-1)n! \text{ Vergleiche}$$

durchführen. Achtung: Es ist z. B. `22!=1124000727777607680000` $\approx 10^{21}$.

Der folgende Algorithmus nutzt die transitive Struktur

$$x \leq y \text{ und } y \leq z \Rightarrow x \leq z$$

der Ordnungsrelation „$\leq$“ aus.

Algorithmus 3.4 (bubblesort)

0: Setze $k := n$ und $S_n = \{z_1, ..., z_n\}$

1: Setze $z_{\pi_k} = \max S_k$

2: Setze $S_{k-1} := S_k \setminus \{z_{\pi_k}\}$ und $k := k - 1$

3: Falls $k > 0$, gehe nach 1.

Beispiel:

$$
\begin{aligned}
S_3 = \{4,1,2\}\ ,\ k = 3 & \\
& z_{\pi_3} = \max S_3 = \max\{4,1,2\} = 4 \\
S_2 = S_3 \setminus \{4\} = \{1,2\}\ ,\ k = 2 & \\
& z_{\pi_2} = \max S_2 = \max\{1,2\} = 2 \\
S_1 = S_2 \setminus \{2\} = \{1\}\ ,\ k = 1 & \\
& z_{\pi_1} = \max S_1 = \max\{1\} = 1
\end{aligned}
$$

Das dazugehörige MATLAB-Programm heißt bubblesort.m. Es enthält gleich eine kleine Optimierung: Wenn bei einem inneren Schleifendurchgang keine Vertauschung vorgenommen wurde, ist die Folge offenbar schon richtig sortiert. Dann kann also das Programm abgebrochen werden.

```
function Sz=bubblesort(Uz)
%
% BUBBLESORT - Sortiert einen Vektor der Groesse nach
% Eingabe:     Uz = Eingabevektor
%
% Sz = bubblesort(Uz) gibt den sortierten Vektor Sz aus
%

z = Uz;

n = length(Uz);

for k = n:(-1):2              % Schleife  ueber S_k
  swapped = 0;

  for i=2:k                   % Maximum von S_k ans Ende bringen
     if ( z(i-1) > z(i) )     % Falls z(i) > z(i+1)
        x = z(i-1);
        z(i-1) = z(i);        % Vertauschen von z(i) und z(i+1)
        z(i) = x;
        swapped = swapped+1;  % Anzahl Vertauschungen z\"ahlen
     end
  end

  if swapped > 0              % Falls in einem Durchgang \"uberhaupt
    break;                    % nicht getauscht wurde, ist die
  end                         % Folge sortiert.
end

Sz=z;
```

bubblesort.m kann z.B. von dem folgenden Programm sortlab.m aufgerufen werden.

```
function [] = sortlab(n)
```

```
Uz = rand(n,1);
tic;
z = bubblesort(Uz);
disp 'bubblesort: ';
toc
N = 1:n;
plot(N,z,'-r',N,Uz,'-g');
```

Wir wollen den *Aufwand* von Algorithmus 3.4 untersuchen. Dabei wollen wir uns auf das *asymptotische Verhalten* des Aufwands für große n beschränken. Daher die folgende

Definition 3.5 *Wir schreiben*

$$f(n) = \mathcal{O}(g(n)) \ ,$$

falls $n_0 \in \mathbb{N}$ *und* $c \in I\!R$ *existieren, so daß*

$$f(n) \leq cg(n) \qquad \forall n \geq n_0$$

Bemerkung: Gilt $f(n) = \mathcal{O}(g(n))$, so wächst also f mit wachsendem n nicht schneller als g.

Achtung: $\mathcal{O}(f(n)) = g(n)$ ist sinnlos!

Beispiel:

$$\sin(n) = \mathcal{O}(1)$$
$$x^2 + 2x + 1 - \cos x = \mathcal{O}(x^2).$$

Definition 3.6 *Der Aufwand eines Algorithmus ist die kleinste obere Schranke für das Aufwandsmaß.*

Bei der Wahl des Aufwandsmaßes ignorieren wir den Speicherbedarf und berücksichtigen nur die Rechenzeit. Als einfaches Maß für die Rechenzeit wählen wir

$$\text{Aufwandsmaß} = \text{Anzahl der Vergleiche.}$$

Zur Bestimmung des Aufwandes von bubblesort.m haben wir also die Vergleiche zu zählen:

$k-$Schleife : wird $n-1$ mal durchlaufen

Aufwand pro Durchlauf : $$\sum_{k=2}^{n}(k-1) = \sum_{k=1}^{n-1} k = \frac{k(k-1)}{2} = \mathcal{O}(n^2)$$

Das ist eine dramatische Reduktion verglichen mit dem exponentiellen Aufwand von Algorithmus 3.2. Aber es geht noch besser. Eine tiefergehende Struktureigenschaft steckt in folgendem Lemma.

Lemma 3.7 *Gegeben seien die beiden sortierten Mengen*

$$Sx = \{x_1 \leq x_2 \leq \cdots \leq x_n\}, \qquad Sy = \{y_1 \leq y_2 \leq \cdots \leq y_m\}.$$

Dann läßt sich die Menge $S = Sx \cup Sy$ mit linearem Aufwand, also mit $\mathcal{O}(n+m)$ Vergleichen sortieren.

Wir führen einen konstruktiven Beweis, indem wir einen entsprechenden Algorithmus angeben.

Algorithmus 3.8 (merge)

1. Eingabe:

   $$x_1 \leq x_2 \leq \cdots \leq x_n$$
   $$y_1 \leq y_2 \leq \cdots \leq y_m$$

2. Initialisierung:

   $$k := 1\ , \qquad j := 1$$

3. Einordnen: Falls $x_k \leq y_j$

   {
   x_k vor y_j einordnen.
   Falls $k < n$: $k := k + 1$
   sonst stop.
   }

   sonst

   {
   Falls $j < m$ $\qquad j := j + 1$
   sonst $x_k, x_{k+1}, ..., x_n$
   hinter y_m einordnen und stop.
   }

4. Gehe nach 3.

Beispiel:

$$x = \{1,\ 3,\ 6\}$$
$$y = \{1,\ 2,\ 4,\ 6,\ 8,\ 10\}$$

$k := 1,\ j := 1$	$1 \leq 1:$	1 vor 1 einordnen, $k := 2$
	$3 > 1:$	$j := 2$
	$3 > 2:$	$j := 3$
	$3 \leq 4:$	3 vor 4 einordnen, $k := 3$
	$6 > 4:$	$j := 4$
	$6 \leq 6:$	6 vor 6 einordnen, stop

Lemma 3.9 *Algorithmus 3.8 benötigt maximal $n+m-1$ Vergleiche.*

Beweis: Wir betrachten zunächst den Fall $x_n \leq y_m$. Es sei v_n die Anzahl der benötigten Vergleiche. Wir zeigen durch vollständige Induktion über n, daß dann x_n direkt vor y_{v_n-n+1} eingeordnet wird. Der Fall $n=1$ ist klar, denn nach v_1 Vergleichen ist x_1 gerade vor y_{v_1} eingeordnet. Die Behauptung sei für n richtig. Wird x_{n+1} nach weiteren s Vergleichen eingeordnet, ist die Gesamtanzahl der Vergleiche also $v_{n+1} = v_n + s$, so liegt x_{n+1} direkt vor $y_{v_n-n+1+s-1} = y_{v_{n+1}-(n+1)+1}$.

Wir wissen nun, daß bei v_n Vergleichen x_n vor $y_{v_n-n+1} \leq y_m$ eingeordnet wird. Es muß also insbesondere

$$v_n - n + 1 \leq m$$

sein. Das ist gerade die Behauptung.

Gilt nun $x_{k_0} \leq y_m$ für ein k_0 mit $1 \leq k_0 < n$, so kann man analog mit k_0 anstelle von n schließen. In diesem Fall kommt man also mit $k_0 + m - 1 < n + m - 1$ Vergleichen aus. Ist $y_m < x_1$, so reichen m Vergleiche. ■

Wir nehmen nun der Einfachheit halber an, daß

$$n = 2^m$$

gilt. Die folgenden Überlegungen lassen sich jedoch auch auf beliebige $n \in \mathbb{N}$ übertragen.

Wir zerlegen nun unser großes Problem in viele kleine Probleme (divide and conquer).

Algorithmus 3.10 (mergesort)

1. Ist $n = 1$, so gibt es nichts zu sortieren.
   Andernfalls sortiere $z_1, ..., z_{n/2}$ und $z_{(n/2)+1}, ..., z_n$ (mit Algorithmus 3.10).
2. Verschmelze (merge) die sortierten Teilmengen zur sortierten Gesamtmenge.

Die entsprechende MATLAB–Funktion mergesort.m sieht wie folgt aus.

```
function Sz = mergesort(z)
%
% MERGESORT - Sortiert einen Vektor der Groesse nach
% Eingabe:    z = Eingabevektor
%
% Sz = mergesort(z) gibt den sortierten Vektor Sz aus
%

n = length(z);

if (n==1)  Sz=z;            % Nichts zu sortieren
else
   m = fix(n/2);
   x = z(1:m);              % Halbieren des Vektors
   y = z(m+1:n);
```

```
    Sx = mergesort(x);       % Sortieren der einen Haelfte
    Sy = mergesort(y);       % Sortieren der anderen Haelfte

    Sz = merge(Sx,Sy);       % Verschmelzen
end
```

Bemerkung: Beachte, daß mergesort sich selbst aufruft. Man spricht von einem rekursiven Programm. Im allgemeinen ist die Frage, ob ein rekursives Programm terminiert, höchst kompliziert. Im vorliegenden Fall ist die Sache klar: Nach m Reduktionsschritten ist man bei der Länge $n = 1$ angelangt und beginnt mit dem Verschmelzen.

Die Aufwandsanalyse ist nur geringfügig komplizierter. Der Aufwand für $n = 2^m$ Elemente sei A_n. Er setzt sich offenbar zusammen aus dem Aufwand für das Sortieren der beiden Hälften (je $A_{n/2}$) und dem anschließenden Zusammenfügen (n). Es gilt also

$$\begin{aligned} A_n &= 2A_{n/2} + n \\ &= 2(2A_{n/4} + n/2) + n = 4A_{n/4} + 2n \\ &= \dots \\ &= 2^m A_{n/2^m} + kn = 2^m A_1 + mn \end{aligned}$$

Weil die Sortierung einer Zahl keinen Vergleich erfordert erhalten wir

Satz 3.11 *Der Aufwand von mergesort ist durch*

$$A_n = \mathcal{O}(n \log n)$$

nach oben beschränkt.

Diese Abschätzung ist scharf, d.h. nicht zu verbessern. Für große n ist $\log n \ll n$. Gegenüber bubblesort haben wir also eine signifikante Verbesserung erreicht. Es bleibt die Frage, ob noch effizientere Algorithmen denkbar sind.

Definition 3.12 *Für ein gegebenes Problem sei $\mathcal{A}$ die Menge aller Algorithmen zu dessen Lösung. Dann ist*

$$\inf_{A \in \mathcal{A}} \text{ Aufwand von } A$$

die Komplexität des Problems.

Die Komplexität eines Problems ist also der *unvermeidbare Aufwand* zur Lösung. Bei der Konstruktion *effizienter Algorithmen* geht es darum, vermeidbare Operationen auf der Basis struktureller Einsicht zu vermeiden. Das Ziel ist natürlich, der Komplexität des Problems möglichst nahezukommen.

Aus Satz 3.11 folgt, daß die Komplexität unseres Sortierproblems höchstens $\mathcal{O}(n \log n)$ sein kann.

Satz 3.13 *Die Komplexität des Sortierproblems ist $\mathcal{O}(n \log n)$.*

Beweis: Ein Beweis findet sich z. B. im Lehrbuch von Schöning [7]. ■

Bemerkung: Im Sinne unseres Komplexitätsbegriffes ist also mergesort ein optimaler Algorithmus. Wir haben uns bei der Aufwandsanalyse allerdings immer auf den schlechtestmöglichen Fall konzentriert. In der Praxis, z.B. beim alphabetischen Sortieren, kann man oft von gewissen Wahrscheinlichkeitsverteilungen der Daten ausgehen. Durch geschicktes Ausnutzen dieser a priori Information erhält man dann mit entsprechender Wahrscheinlichkeit bessere Ergebnisse als im schlechtestmöglichen Fall.

3.2 Nichtpolynomielle Komplexität?

Wir betrachten das berühmte *Problem des Handlungsreisenden* (Traveling Salesman Problem), kurz TSP–Problem [4].

Gegeben seien

$$\begin{aligned} &n \text{ Städte}: \quad 1, ..., n \\ &\text{Fahrzeiten}: \ c_{ij} \text{ zwischen } i \text{ und } j \end{aligned}$$

Das Problem lautet: Finde eine Permution $\pi_1, ..., \pi_n$, so daß die *Gesamtfahrzeit*

$$\text{cost } (\pi) = \sum_{i=1}^{n-1} c_{\pi_i, \pi_{i+1}} + c_{\pi_n, \pi_1}$$

unter allen Permutationen von $\{1, ..., n\}$ minimal wird. Eine Permutation heißt in diesem Zusammenhang Tour.

Beispiel:

$$C = \begin{pmatrix} 0 & 212 & 171 & 203 \\ 212 & 0 & 186 & 247 \\ 171 & 186 & 0 & 139 \\ 203 & 247 & 139 & 0 \end{pmatrix}$$

$$\begin{aligned} \text{cost } (1,2,3,4) &= 740 \\ \text{cost } (1,2,4,3) &= 769 \\ \text{cost } (1,3,2,4) &= 807 \end{aligned}$$

Algorithmus 3.14 (Ausprobieren)
Probiere so lange alle möglichen Permutationen durch, bis die gewünschte Eigenschaft vorliegt.

Bemerkung: Wir wissen schon, daß wir bei Algorithmus 3.14 mit $\mathcal{O}(n!)$ Auswertungen des *Kostenfunktionals* $\text{cost}(\pi)$ rechnen müssen. Damit ist dieser Algorithmus nur für sehr kleine n praktisch durchführbar.

Ein Algorithmus hat polynomiellen Aufwand, falls es ein $k \in \mathbb{N}$ gibt, so daß $\mathcal{O}(n^k)$ eine obere Schranke für die benötigten cost–Auswertungen ist. Wie wir gesehen haben, ist Algorithmus 3.14 nicht polynomial.

Gibt es nun einen Algorithmus, der das TSP–Problem mit polynomialem Aufwand löst oder gibt es so einen Algorithmus nicht?

Dies ist ein *offenes Problem*, welches seit langer Zeit im Mittelpunkt der Komplexitätstheorie steht! Ein Grund für das allgemeine Interesse an dieser Frage ist, daß ein polynomialer TSP–Algorithmus sofort polynomiale Algorithmen für eine Vielzahl anderer wichtiger Probleme liefern würde [8]. Die Klasse dieser Probleme heißt *NP–vollständig*. Es gibt tieferliegende Gründe, die hier nicht erörtert werden können (Stichwort P=NP?). NP ist hierbei die Abkürzung für "nicht-deterministisch polynomiell" — nicht zu verwechseln mit "nichtpolynomiell" — und bezeichnet die Klasse von Problemen, die auf einer nicht-deterministischen Turingmaschine (ein einfaches, idealisiertes Computermodell) in polynomieller Zeit lösbar sind. Wir verweisen auf einschlägige Lehrveranstaltungen der Arbeitsgruppe Theoretische Informatik, z.B. „Theoretische Informatik“ oder „Entwurf und Analyse von Algorithmen“. Derzeit herrscht übrigens die Meinung vor, daß es *keinen* polynomiellen TSP-Algorithmus gibt.

Da Algorithmus 3.14 für realistische n praktisch nicht durchführbar ist, 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*. Näheres dazu im Artikel von Grötschel und Padberg [3].

Literatur

[1] M. Aigner. *Diskrete Mathematik*. Vieweg, 1993. Das Buch besteht aus drei Teilen: Abzählung, Graphen und Algorithmen sowie Algebraische Systeme, die weitgehend unabhängig voneinander gelesen werden können.

[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press and McGraw-Hill, 2nd edition, 2001. Das Standardwerk über Algorithmen.

[3] M. Groetschel and M. Padberg. Die optimierte Odysee. *Spektrum der Wissenschaft*, 4:76–85, 1999.

[4] http://www.math.princeton.edu/tsp/. Geschichte, Anwendungen und Weltrekorde zum TSP-Problem.

[5] P.E. Knuth. The art of computer programming. Vol. 3, 1998. Auf absehbare Zeit das Standardwerk zum Thema Sortieren.

[6] P. Morin. http://cg.scs.carleton.ca/∼morin/misc/sortalg/. Pat Morin hat auf seiner homepage eine Reihe mehr oder weniger leistungsfähige Sortieralgorithmen zu einer interaktiven Demo verpackt. Viel Spaß beim Spielen.

[7] U. Schöning. *Algorithmen kurzgefaßt*. Spektrum Akademischer Verlag, 1997. Eine preiswerte (29,80 DM) Einführung in kompakter Form (221 Seiten). Im Mittelpunkt stehen dabei die verschiedensten sequentiellen Algorithmen, deren Komplexitätsanalyse und allgemeine Algorithmen-Paradigmen.

[8] I. Wegener. *Theoretische Informatik*. Teubner, 1993. Alles zu P, NP, TSP und Turingmaschinen.

4 Lineare Gleichungssyssteme

Der erste Abschnitt richtet sich im Rahmen der Vorlesung CoMaI an Interessierte und Bioinformatiker:

4.1 Integralgleichungen

4.1.1 Inverses Problem zur Auslenkung einer Saite

Im Anhang in Abschnitt B.1 wird die Darstellung

$$u(x) = \int_0^L K(x,\xi) f(\xi) d\xi \tag{4.1}$$

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.

Ist umgekehrt u bekannt, so kann man daran denken, (4.1) als Berechnungsvorschrift für f zu verwenden. Dieser Fall ist komplizierter, denn die Werte von f sind durch (4.1) nicht explizit gegeben.

Definition 4.1 *Es seien $g : [a,b] \to \mathbb{R}$ und der Kern $K : [a,b]^2 \to \mathbb{R}$ gegeben. Dann heißt*

$$\int_a^b K(x,\xi) w(\xi)\, d\xi = g(x) \qquad \forall x \in [a,b] \tag{4.2}$$

Fredholmsche Integralgleichung erster Art für die unbekannte Funktion $w : [a,b] \to \mathbb{R}$.

4.1.2 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

Anteil der Individuen im Alter τ, die sich im Zeitraum Δt 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*

$$w(t) = g(t) + \int_\alpha^\beta w(t - \tau) K(\tau) \, d\tau \ . \tag{4.3}$$

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

$$u(t) = 0 \qquad \forall t \le 0$$

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 (4.3) 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

$$K(\tau) = 0 \qquad \forall \tau \notin [0, \beta]$$

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

$$w(t) = g(t) + \int_{0}^{T} w(\eta)K(t-\eta)\,d\eta \qquad \forall t \in [0,T]\ . \tag{4.4}$$

Das ist unser Populationsmodell.

Definition 4.2 *Es seien $g : [a,b] \to \mathbb{R}$ und der Kern $K : [a,b]^2 \to \mathbb{R}$ gegeben. Dann heißt*

$$w(x) - \int_{a}^{b} K(x,\xi)w(\xi)\,d\xi = g(x) \qquad \forall x \in [a,b] \tag{4.5}$$

Fredholmsche Integralgleichung 2. Art für die unbekannte Funktion $w : [a,b] \to I\!R$.

Die Untersuchung von Existenz und Eindeutigkeit von Lösungen der Integralgleichungen (4.2) und (4.5) sprengt den Rahmen dieser Vorlesung. Unter dem Stichwort „Fredholm–Operatoren“ wird dieses Thema in höchst eleganter Weise in der Funktionalanalysis behandelt. Ungeduldige seien auf das Lehrbuch von Werner [4] verwiesen. Wer sich speziell für Integralgleichungen interessiert, findet im Werk von Drabek und Kufner [2] eine brauchbare Einführung.

4.1.3 Galerkin–Verfahren

Wir betrachten zunächst die Integralgleichung (4.2). 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*

$$w_n = \sum_{k=0}^{n-1} W_k \varphi_k\ . \tag{4.6}$$

Um die unbekannten Koeffizienten $W_k, k = 0, \ldots, n-1$, zu bestimmen, setzen wir w_n in die Integralgleichung (4.2) ein und erhalten

$$\sum_{k=0}^{n-1} W_k \int_{a}^{b} K(x,\xi)\varphi_k(\xi)\,d\xi = g(x)\ . \tag{4.7}$$

Wir können nicht erwarten, daß wir die W_k so bestimmen können, daß (4.7) für *alle* $x \in [a,b]$ gilt. Also fordern wir nur

$$\sum_{k=0}^{n-1} W_k \int_{a}^{b} K(x_j,\xi)\varphi_k(\xi)\,d\xi = g(x_j) =: g_j \qquad j = 0, \ldots, n-1\ . \tag{4.8}$$

Schließlich nutzen wir die spezielle Gestalt der Ansatzfunktionen φ_k aus und erhalten

$$\int_{a}^{b} K(x_j,\xi)\varphi_k(\xi)\,d\xi = \int_{x_k}^{x_{k+1}} K(x_j,\xi)\,d\xi =: a_{jk}\ . \tag{4.9}$$

Damit lassen sich die n Bestimmungsgleichungen (4.8) in der Form

$$\sum_{k=0}^{n-1} a_{jk} W_k = g_j \ , \qquad j = 0, \ldots, n-1 \ , \tag{4.10}$$

schreiben. Setzt man

$$A = (a_{jk})_{j,k=0}^{n-1} \in \mathbb{R}^{n,n} \ , \quad W = (W_k)_{k=1}^{n} \ , \quad G = (g_j)_{j=1}^{n} \in \mathbb{R}^n \ , \tag{4.11}$$

so ist (4.10) gleichbedeutend mit dem *linearen Gleichungssystem*

$$AW = G \ .$$

Analog erhält man zur Approximation von (4.5) die Koeffizienten

$$a_{jk} = \begin{cases} -\int\limits_{x_k}^{x_{k+1}} K(x_j, \xi)\, d\xi & j \neq k \\ 1 - \int\limits_{x_j}^{x_{j+1}} K(x_j, \xi)\, d\xi & j = k \end{cases} \ , \qquad j,k = 0, \ldots, n-1 \ . \tag{4.12}$$

Näherungsverfahren, die auf einem Ansatz der Form (4.6) 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 [2, 4].

4.2 Konditionsbetrachtungen

Die Definitionen 2.7 und 2.8 der relativen und absoluten Kondition erfolgte *komponentenweise*. Dies wird ist bei komplexeren Problemen schnell sehr unübersichtlich. Wir werden folgend deshalb die Größen der einzelnen Komponenten geeignet zusammenfassen und eine *normweise* Kondition betrachten.

4.2.1 Matrix– und Vektornormen

Definition 4.3 *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

$$\|x\| \geq 0 \ , \qquad \|x\| = 0 \Leftrightarrow x = 0 \ , \tag{4.13}$$

$$\|\alpha x\| = |\alpha| \ \|x\| \qquad \textit{(Homogenität)} \ , \tag{4.14}$$

$$\|x + y\| \leq \|x\| + \|y\| \qquad \textit{(Dreiecksungleichung)} \ . \tag{4.15}$$

Ein linearer Raum, der mit einer Norm versehen ist, heißt normierter Raum.

Beispiele:

Der Betrag

$$|x| = x \operatorname{sgn}(x) \text{ auf } V = \mathbb{R} .$$

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 $1 \le p < \infty$ auf $V = \mathbb{R}^n$.

Bemerkung: Der Nachweis der Eigenschaften (4.13), (4.14), (4.15) für die obigen Beispiele ist einfach. Nur die Dreiecksungleichung für $\|\cdot\|_p$ macht Schwierigkeiten. Sie lautet ausgeschrieben

$$\left(\sum_{i=1}^n |x_i + y_i|^p \right)^{\frac{1}{p}} \le \left(\sum_{i=1}^n |x_i|^p \right)^{\frac{1}{p}} + \left(\sum_{i=1}^n |y_i|^p \right)^{\frac{1}{p}}$$

und heißt *Minkowskische Ungleichung*. Einen Beweis findet man z.B. im Funktionalanalysis–Lehrbuch von Werner [4], 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 4.4 *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*

$$\|x^{(\nu)} - x^{(\mu)}\| \leq \varepsilon \qquad \forall \nu, \mu \geq \nu_0 \ .$$

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 4.5 *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*

$$c\|x\| \leq |||x||| \leq C\|x\| \qquad \forall x \in V \ .$$

Beweis: Der Beweis beruht auf der Kompaktheit der Einheitskugel in $\mathbb{R}^n$. Wir verweisen wieder auf Werner [4], Seite 26. ■

Bemerkung: Für unendlichdimensionale Räume, wie z.B. $V = C[a, b]$, ist dieser Satz falsch!

Als Folgerung aus Satz 4.5 wollen wir die Vollständigkeit von $\mathbb{R}^n$ und $\mathbb{R}^{n,n}$ zeigen.

Satz 4.6 *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 4.5 gibt es c, C unabhängig von x, so daß

$$c\|x\|_{\mathbb{R}^n} \leq \|x\|_\infty \leq C\|x\|_{\mathbb{R}^n} \ , \qquad \forall x \in \mathbb{R}^n \ .$$

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

$$\|x - x^{(\nu)}\|_{\mathbb{R}^n} \leq \frac{1}{c}\|x - x^{(\nu)}\|_\infty \to 0 \qquad \nu \to \infty \ .$$

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 4.7 *Es sei $\|\cdot\|$ eine Norm auf $\mathbb{R}^n$. Dann ist durch*

$$\|A\| = \sup_{\substack{x \in \mathbb{R}^n \\ x \neq 0}} \frac{\|Ax\|}{\|x\|} \ , \qquad A \in \mathbb{R}^{n,n} \ , \tag{4.16}$$

die zugehörige Matrixnorm definiert.

Man überzeuge sich davon, daß durch (4.16) überhaupt eine Norm definiert ist!

Bemerkungen:

- Es gilt
$$\|Ax\| \leq \|A\| \ \|x\| \ .$$
- Es existiert ein $x^* \in \mathbb{R}^n$ mit $\|Ax^*\| = \|A\| \ \|x^*\|$.
- Matrixnormen der Form (4.16) sind mit der Matrixmultiplikation verträglich. Es gilt nämlich
$$\|AB\| \leq \|A\| \ \|B\| \qquad \forall A, B \in \mathbb{R}^{n,n} \qquad \text{(Submultiplikativität)},$$
- Die Norm der Einheitsmatrix I ist
$$\|I\| = 1 \ .$$

Satz 4.8 *Die Matrixnorm*

$$\|A\|_\infty = \max_{i=1,\ldots,n} \sum_{j=1}^{n} |a_{ij}| \qquad \textit{(Zeilensummennorm)}$$

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

$$\|A\|_\infty \|x^*\|_\infty \geq \|Ax^*\|_\infty \geq \left| \sum_{j=1}^n a_{i_0 j} x_j^* \right| = \sum_{j=1}^n |a_{i_0 j}| = \|A\|_\infty \|x^*\|_\infty \ .$$

■

Schließlich definieren wir wie angekündigt die normweise Kondition.

Definition 4.9 *Seien X und Y normierte lineare Räume, $f : X \to Y$ und $x \neq 0 \in X$. Für jedes $\varepsilon > 0$ gebe es eine Zahl $\kappa_{\text{rel}}(\varepsilon)$, so daß*

$$\frac{\|f(x) - f(\tilde{x})\|}{\|f(x)\|} \leq \kappa_{\text{rel}}(\varepsilon) \frac{\|x - \tilde{x}\|}{\|x\|} \tag{4.17}$$

für alle $\tilde{x} \in X$ mit $\|\tilde{x} - x\| \leq \varepsilon$. Zu jedem $\varepsilon > 0$ existiere mindestens ein $\tilde{x}$ mit $\|x - \tilde{x}\| \leq \varepsilon$, so daß in (4.17) *Gleichheit gilt. Dann heißt*

$$\kappa_{\text{rel}} = \lim_{\varepsilon \to 0} \kappa_{\text{rel}}(\varepsilon)$$

die normweise relative Kondition der Auswertung von $f(x)$.

Ganz analog läßt sich natürlich auch die normweise absolute Kondition definieren.

4.2.2 Störung von Matrix und rechter Seite

Wir betrachten das lineare Gleichungssystem

$$Ax = b \ . \tag{4.18}$$

Dabei seien $A = (a_{ij})_{i,j=1}^n \in \mathbb{R}^{n,n}$ sowie $b = (b_i)_{i=1}^n \in \mathbb{R}^n$ gegeben und $x = (x_i)_{i=1}^n \in \mathbb{R}^n$ gesucht. Existenz und Eindeutigkeit der Lösung von (4.18) sind Gegenstand der Vorlesung *Lineare Algebra I.*

Satz 4.10 *Die Koeffizientenmatrix A sei regulär, d.h.*

$$Ax \neq 0 \qquad \forall x \in \mathbb{R}^n \ , \quad x \neq 0 \ .$$

Dann ist (4.18) *eindeutig lösbar mit der Lösung $x = A^{-1}b$.*

Wir wollen nun die Auswirkungen von Störungen in den Daten A und b auf die Lösung untersuchen. Dabei wird mit $\|\cdot\|$ durchweg eine Vektornorm *und* die zugehörige Matrixnorm bezeichnet.

Wir betrachten das gestörte System

$$A\tilde{x} = \tilde{b} \qquad \text{(Störung der rechten Seite).} \tag{4.19}$$

Die Submultiplikativität der Matrixnorm liefert

$$\|x - \tilde{x}\| = \|A^{-1}b - A^{-1}\tilde{b}\| \leq \|A^{-1}\| \ \|b - \tilde{b}\| \ .$$

Also ist $\kappa_{\text{abs}} = \|A^{-1}\|$ die absolute Kondition von (4.18) bezüglich Störungen der rechten Seite.

Satz 4.11 *Es sei*

$$Ax = b \qquad A\tilde{x} = \tilde{b} \ .$$

Dann gilt

$$\frac{\|x - \tilde{x}\|}{\|x\|} \leq \kappa(A) \frac{\|b - \tilde{b}\|}{\|b\|} \tag{4.20}$$

mit $\kappa(A) = \|A\| \ \|A^{-1}\|$. *Zudem existiert eine rechte Seite* b, *so daß* $\kappa(A)$ *die relative normweise Kondition von* (4.18) *bei Störung der rechten Seite ist.*

Beweis: Wegen (4.19) und $Ax = b$ ist

$$\frac{\|x - \tilde{x}\|}{\|x\|} \leq \frac{\|Ax\|}{\|x\|} \|A^{-1}\| \frac{\|b - \tilde{b}\|}{\|b\|} \leq \|A\| \ \|A^{-1}\| \frac{\|b - \tilde{b}\|}{\|b\|} \ .$$

Nach Definition der Matrixnorm existieren x^* und b^*, so daß $\|x^*\| = \|b^*\| = 1$, $\|Ax^*\| = \|A\|$ und $\|A^{-1}b^*\| = \|A^{-1}\|$. Für die Wahl $b = Ax^*$ und $\tilde{b} = b - \varepsilon b^*$ ist die Abschätzung (4.20) für alle $\varepsilon > 0$ scharf, so daß $\kappa(A)$ die relative normweise Kondition von (4.18) ist. ■

Wir betrachten nun das gestörte Problem

$$\tilde{A}\tilde{x} = b \qquad \text{(Störungen der Matrix)} \ .$$

Diesmal müssen wir einen längeren Anlauf nehmen, denn die Abbildung

$$\mathbb{R}^{n,n} \ni \tilde{A} \to \tilde{x} = \tilde{A}^{-1} b \in \mathbb{R}^n$$

ist *nichtlinear in* $\tilde{A}$. Außerdem ist nicht von vorneherein klar, daß das gestörte Problem überhaupt eine eindeutig bestimmte Lösung $\tilde{x}$ hat!

Lemma 4.12 *Es sei* $C \in \mathbb{R}^{n,n}$ *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

$$S_n(I - C) = S_n - S_n C = I + S_n - S_{n+1} \to I \qquad n \to \infty \ .$$

Andererseits gilt

$$S_n(I - C) \to S(I - C) \qquad n \to \infty \ .$$

Eindeutigkeit des Grenzwertes liefert

$$S(I - C) = I \ .$$

■

Satz 4.13 *Es sei*

$$Ax = b\ , \qquad \tilde{A}\tilde{x} = b.$$

Unter der Bedingung $\|A - \tilde{A}\| < \|A^{-1}\|^{-1}$ *ist* $\tilde{A}$ *regulär und es gilt*

$$\frac{\|x - \tilde{x}\|}{\|x\|} \leq \kappa(A, \tilde{A}) \frac{\|A - \tilde{A}\|}{\|A\|}$$

mit

$$\kappa(A, \tilde{A}) = \|A\|\ \|A^{-1}\| + \frac{\|A\|\ \|A^{-1}\|^2\ \|A - \tilde{A}\|}{1 - \|A^{-1}\|\ \|A - \tilde{A}\|} \to \kappa(A) = \|A\|\ \|A^{-1}\| \qquad \tilde{A} \to A.$$

Damit ist $\kappa(A) = \|A\|\ \|A^{-1}\|$ *eine obere Schranke für die relative Kondition von* (4.18) *bei Störung der Koeffizientenmatrix.*

Beweis: Für $C = A^{-1}(A - \tilde{A})$ gilt nach Voraussetzung

$$\|C\| \leq \|A^{-1}\|\ \|A - \tilde{A}\| < 1\ . \tag{4.21}$$

Auf Grund von Lemma 4.12 ist daher $I - C$ regulär. Wegen

$$\tilde{A} = A(I - C)$$

ist auch $\tilde{A}$ als Produkt zweier regulärer Matrizen regulär. Es gilt

$$\begin{aligned}
\tilde{A}^{-1} &= (A(I - C))^{-1} = (I - C)^{-1} A^{-1} \\
&= \left(I + C + \sum_{k=2}^{\infty} C^k \right) A^{-1} \\
&= A^{-1} + CA^{-1} + C^2 (I - C)^{-1} A^{-1}\ .
\end{aligned}$$

Wegen (4.21) ist

$$\|(I - C)^{-1}\| \leq \sum_{k=0}^{\infty} \|C\|^k = \frac{1}{1 - \|C\|} \leq \frac{1}{1 - \|A^{-1}\|\ \|A - \tilde{A}\|}$$

und es folgt

$$\begin{aligned}
\|x - \tilde{x}\| &= \|A^{-1}b - \tilde{A}^{-1}b\| \\
&= \|A^{-1}b - A^{-1}b - CA^{-1}b - C^2(I - C)^{-1}A^{-1}b\| \\
&\leq \left(\|C\| + \|C\|^2 \|(I - C)^{-1}\| \right) \|x\| \\
&\leq \left(\|A^{-1}\| + \frac{\|A^{-1}\|^2 \|A - \tilde{A}\|}{1 - \|A^{-1}\|\ \|A - \tilde{A}\|} \right) \|A - \tilde{A}\|\ \|x\| \\
&= \left(\|A\|\ \|A^{-1}\| + \frac{\|A\|\ \|A^{-1}\|^2\ \|A - \tilde{A}\|}{1 - \|A^{-1}\|\ \|A - \tilde{A}\|} \right) \frac{\|A - \tilde{A}\|}{\|A\|} \|x\|\ .
\end{aligned}$$

■

Satz 4.14 *Es sei*

$$Ax = b\ , \qquad \tilde{A}\tilde{x} = \tilde{b}\ ,$$

wobei $\tilde{A} \in \mathbb{R}^{n,n}$ *mit* $\|A - \tilde{A}\| < \|A^{-1}\|^{-1}$ *und* $\tilde{b} \in \mathbb{R}^n$ *gewählt sind. Dann gibt es eine Zahl* $\kappa(A, \tilde{A}, \tilde{b})$ *mit der Eigenschaft*

$$\frac{\|x - \tilde{x}\|}{\|x\|} \leq \kappa(A, \tilde{A}, \tilde{b}) \max \left\{ \frac{\|A - \tilde{A}\|}{\|A\|}\ ,\ \frac{\|b - \tilde{b}\|}{\|b\|} \right\}$$

und es gilt

$$\kappa(A, \tilde{A}, \tilde{b}) \to 2\kappa(A) = 2\|A\|\|A^{-1}\| \qquad \tilde{A} \to A \ , \quad \tilde{b} \to b \ .$$

Damit ist $2\kappa(A) = 2\|A\| \ \|A^{-1}\|$ *eine obere Schranke für die relative Kondition von* (4.18) *bei Störung von Koeffizientenmatrix und rechter Seite.*

Beweis: Übung. ■

Bemerkungen:

- Oft nennt man $\kappa(A) = \|A\| \ \|A^{-1}\|$ kurz *Kondition von A.*
- Es gilt $\kappa(A) = \|A\| \ \|A^{-1}\| \geq \|AA^{-1}\| = \|I\| = 1$. Im Falle $\kappa(A) \gg 1$ heißt das lineare Gleichungssystem (4.18) *schlecht konditioniert.*

Bemerkung: Es gibt eine MATLAB–Funktion `cond`.

Die Kondition bietet ein Maß dafür, „wie regulär“ eine Matrix A ist.

Satz 4.15 *Für alle regulären Matrizen A gilt*

$$\inf \left\{ \frac{\|A - B\|}{\|A\|} \mid B \ \textit{singulär} \right\} \geq \frac{1}{\kappa(A)} \ .$$

Beweis: Ist B singulär, so existiert ein Vektor $x^* \neq 0$ mit $Bx^* = 0$. Für diesen Vektor gilt

$$\frac{\|A - B\|}{\|A\|} \geq \frac{\|(A - B)x\|}{\|A\|\|x\|} = \frac{\|Ax\|}{\|A\|\|x\|} = \frac{\|A^{-1}\|\|Ax\|}{\|A\|\|A^{-1}\|\|x\|} \geq \frac{\|A^{-1}Ax\|}{\kappa(A)\|x\|} = \frac{1}{\kappa(A)}.$$

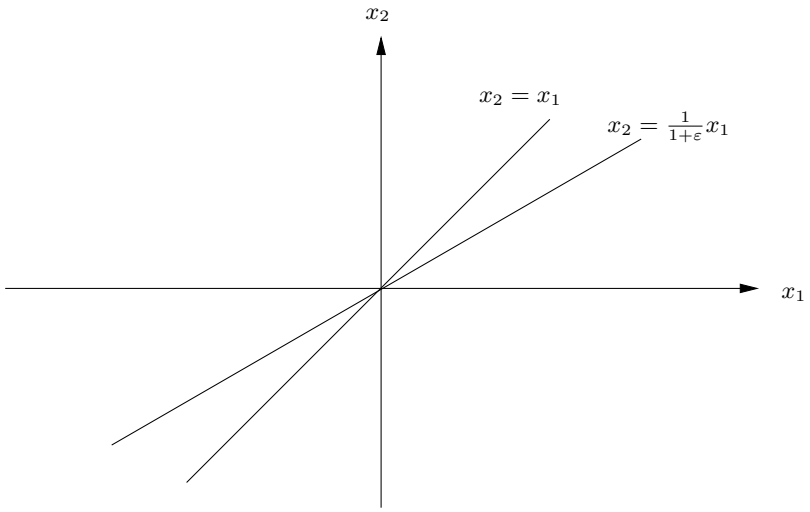
■

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. Grob gesprochen: Schlecht konditionierte Matrizen sind “fast” singulär.

Beispiel: Schleifender Schnitt



Den Schnittpunkt x berechnet man aus $A_\varepsilon x = b$ mit

$$A_\varepsilon = \begin{pmatrix} -1 & 1 \\ -1 & 1+\varepsilon \end{pmatrix} , \qquad b = 0 .$$

Es gilt

$$A_\varepsilon^{-1} = (-\varepsilon)^{-1} \begin{pmatrix} 1+\varepsilon & -1 \\ 1 & -1 \end{pmatrix} .$$

Offenbar gilt für die Zeilensummennorm

$$\kappa_\infty(A_\varepsilon) = \|A_\varepsilon\|_\infty \|A_\varepsilon^{-1}\|_\infty = \frac{(2+\varepsilon)^2}{\varepsilon} \to \infty \qquad \varepsilon \to 0 .$$

Für die singuläre Matrix B_1,

$$B_1 = \begin{pmatrix} -1 + \frac{\varepsilon}{2+\varepsilon} & 1 \\ -1 - \frac{\varepsilon}{2+\varepsilon} & 1+\varepsilon \end{pmatrix} ,$$

erhält man

$$\frac{\|A_\varepsilon - B_1\|_\infty}{\|A_\varepsilon\|_\infty} = \frac{\varepsilon}{(2+\varepsilon)^2} = \frac{1}{\kappa(A)} .$$

Auch die Matrix B_2,

$$B_2 = \begin{pmatrix} -1 & 1 \\ -1 & 1 \end{pmatrix} ,$$

ist singulär und hat einen kleinen Abstand zu A, nämlich

$$\frac{\|A_\varepsilon - B_2\|_\infty}{\|A_\varepsilon\|_\infty} = \frac{\varepsilon}{2+\varepsilon}.$$

Achtung: Ist $\varepsilon <$ eps (Maschinengenauigkeit), so ist A für den Rechner nicht von den singulären Matrizen B_1 und B_2 unterscheidbar!

Beispiel: (Fredholmsche Integralgleichung 1. Art)

Wir betrachten die in (4.9) 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 (4.2) 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 (4.4). Wir wählen als Parameter $T = \beta = 1$, $K(\tau) = \tau(1-\tau)$ und stellen in Abbildung 5 die Kondition der Koeffizientenmatrix (4.12), 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.

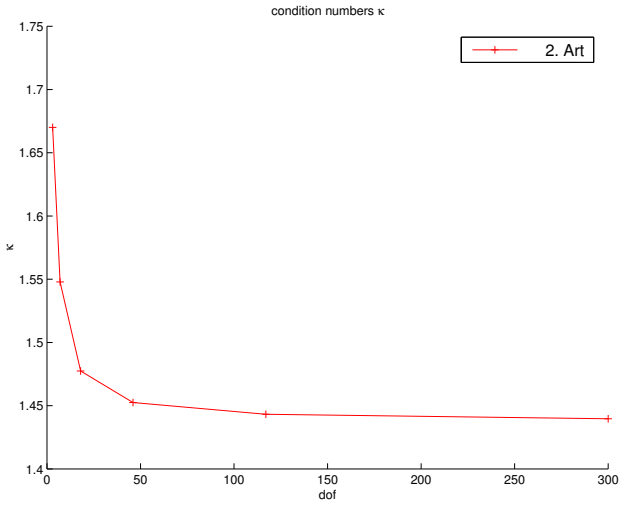


Abbildung 5: Kondition von unterschiedlich feinen Diskretisierungen einer Fredholmschen Integralgleichung 2. Art.

4.3 Gaußscher Algorithmus

4.3.1 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:

$$x_1 = 5 - 4x_2 - 7x_3 \ . \tag{4.22}$$

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:

$$-3x_2 = -11 + 6x_3 \tag{4.23}$$

und Elimination von x_2 aus der zweiten Gleichung ergibt

$$x_3 = 7\ . \tag{4.24}$$

Einsetzen in (4.23):

$$x_2 = -\frac{31}{3}\ .$$

Einsetzen in (4.22):

$$x_1 = -\frac{8}{3}\ .$$

Einen anderen Blickwinkel erlaubt die folgende Matrixschreibweise des Gaußschen Algorithmus

$$\begin{pmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 10 \end{pmatrix} \quad \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \quad = \quad \begin{pmatrix} 5 \\ -1 \\ 0 \end{pmatrix}$$

$$A \qquad\qquad x \quad = \quad b\ .$$

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:

$$\begin{pmatrix} 1 & 4 & 7 \\ 0 & -3 & -6 \\ 0 & 0 & 1 \end{pmatrix} \quad \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \quad = \quad \begin{pmatrix} 5 \\ -11 \\ 7 \end{pmatrix}$$

$$R \qquad\qquad x \quad = \quad z\ .$$

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.

4.3.2 Gaußscher Algorithmus und LR–Zerlegung

Wir betrachten das lineare Gleichungssystem

$$Ax = b \tag{4.25}$$

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

$$\begin{aligned} a_{1j}^{(1)} &= a_{1j} , \quad j = 1, \ldots, n , \quad b_1^{(1)} = b_1 , \\ a_{ij}^{(1)} &= a_{ij} - \frac{a_{i1}}{a_{11}} a_{1j} , \quad i, j = 2, \ldots, n , \\ b_i^{(1)} &= b_i - \frac{a_{i1}}{a_{11}} b_1 , \quad i = 2, \ldots, n . \end{aligned}$$

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 4.16 (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 4.17 (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 (4.25) mit dem Gaußschen Algorithmus abschätzen. Wir wählen

$$\text{Aufwandsmaß} = \text{Anzahl der Multiplikationen.}$$

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 4.16 (Gaußsche Elimination). Dann wird aus jeder Für–Schleife eine Summation und wir erhalten

$$\begin{aligned}
&\sum_{k=1}^{n-1} \sum_{i=k+1}^{n} \left(1 + \sum_{j=k+1}^{n} 1 \right) \\
&= \sum_{k=1}^{n-1} \sum_{i=k+1}^{n} (n-k+1) = \sum_{k=1}^{n-1} (n-k+1)(n-k) \qquad\qquad (\text{setze } j = n-k) \\
&= \sum_{j=1}^{n-1} (j+1)j = \frac{1}{3}(n^3 - n) .
\end{aligned}$$

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 4.17 (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 (4.25) 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 [1], Kapitel 1.

Als nächstes wollen wir unserer Vermutung aus dem einführenden Abschnitt 4.1 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 4.18 *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 4.19 *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

$$L = I + \sum_{k=1}^{n-1} G_k = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ \ell_{21} & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ \ell_{n1} & \cdots & \ell_{n,n-1} & 1 \end{pmatrix} .$$

Beweis: Nach Lemma 4.18 ist

$$\begin{aligned} R = A^{(n-1)} &= (I-G_{n-1})A^{(n-2)} \\ &= (I-G_{n-1})(I-G_{n-2})A^{(n-3)} \\ &= (I-G_{n-1})\cdots(I-G_1)A\ . \end{aligned} \tag{4.26}$$

Weiter gilt

$$\begin{aligned} (G_kG_k)_{ij} &= \sum_{l=1}^{n} (G_k)_{il}(G_k)_{lj} \\ &= (G_k)_{ik}(G_k)_{kj} = (G_k)_{ik}\cdot 0 = 0 \qquad \forall i,j = 1,\dots,n\ , \end{aligned} \tag{4.27}$$

also $(I+G_k)(I-G_k) = I - G_kG_k = I$ und somit

$$(I-G_k)^{-1} = I + G_k\ .$$

Aus (4.26) ergibt sich daher

$$A = (I+G_1)\cdots(I+G_{n-1})R\ .$$

ähnlich wie (4.27) folgt

$$G_k \cdot G_l = 0 \qquad \forall l \ge k\ .$$

Daher ist

$$\begin{aligned} (I+G_1)\cdots(I+G_{n-1}) &= (I+G_2)\cdots(I+G_{n-1}) + (G_1 + \underbrace{G_1G_2}_{=0})(I+G_3)\cdots(I+G_{n-1}) \\ &= (I+G_2)\cdots(I+G_{n-1}) + G_1 \\ &= I + G_1 + G_2 + \cdots + G_{n-1} = L\ . \end{aligned} \tag{4.28}$$

■

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 $\mathcal{O}(n^2)$.

Stabilitätsbetrachtungen

Es ist klar, daß die grundlegende Voraussetzung nichtverschwindender Pivotelemente, also

$$a_{kk}^{(k-1)} \neq 0 \ , \qquad k = 1, \ldots, n-2,$$

im allgemeinen nicht erfüllt ist. Wir werden in diesem Abschnitt sehen, daß auch der Fall

$$a_{kk}^{(k-1)} \approx 0 \qquad \text{für ein } k = 1, \ldots, n-2$$

kritisch ist, weil er zur Instabilität des Gaußschen Algorithmus führt.

Die Lösung des Gleichungssystems $Ax = b$ erfolge in den drei Schritten

1. faktorisiere $A = LR$,
2. löse $Lz = b$,
3. löse $Rx = z$.

Unter der optimistischen (!) Annahme, daß die algorithmische Realisierung der Gauß-Elimination in Gleitkommaarithmetik auf eine Zerlegung

$$A \approx \tilde{L}\tilde{R}$$

mit

$$\frac{\|L - \tilde{L}\|}{\|L\|} \approx \frac{\|R - \tilde{R}\|}{\|R\|} \approx \text{eps}$$

führt, kann der Fehler in der berechneten Lösung $\tilde{x}$ durch die Kondition der Matrizen L und R beschränkt werden. Dabei nehmen wir der Einfachheit halber an, daß die Eingabedaten A und b nur im Rahmen der Maschinengenauigkeit gestört sind:

$$\begin{aligned}
\frac{\|x - \tilde{x}\|}{\|x\|} &\leq \kappa(R) \left(\frac{\|z - \tilde{z}\|}{\|z\|} + \frac{\|R - \tilde{R}\|}{\|R\|} \right) \\
&\leq \kappa(R) \left(\kappa(L) \left(\frac{\|b - \tilde{b}\|}{\|b\|} + \frac{\|L - \tilde{L}\|}{\|L\|} \right) + \text{eps} \right) \\
&\leq \kappa(R)(2\kappa(L) + 1)\text{eps}
\end{aligned}$$

Dem steht ein *unvermeidbarer* Fehler

$$\frac{\|x - \tilde{x}\|}{\|x\|} \leq 2\kappa(A)\text{eps}$$

gegenüber, so daß die Stabilität des Algorithmus von der *zusätzlichen Fehlerverstärkung*

$$\sigma = \frac{\kappa(R)(2\kappa(L) + 1)}{2\kappa(A)}$$

bestimmt wird.

Satz 4.20 *Es gilt* $\kappa(R) \leq \kappa(L)\kappa(A)$.

Beweis:

$$\kappa(R) = \|R\|\|R^{-1}\| = \|L^{-1}A\|\|A^{-1}L\| \leq \|L^{-1}\|\|A\|\|A^{-1}\|\|L\| = \kappa(L)\kappa(A)$$

■

Also vereinfachen wir $\sigma \leq \kappa(L)(\kappa(L) + 1/2) \leq \frac{3}{2}\kappa(L)^2$.

Satz 4.21 *Mit* $L = (I + G_1) \cdot \cdots \cdot (I + G_{n-1})$ *gilt bezüglich der Zeilensummennorm*

$$\kappa(L) \leq (1 + (n-1)\ell_{\max})(1 + \ell_{\max})^{n-1}.$$

mit $\ell_{\max} := \max |\ell_{ik}|$.

Beweis: Offenbar gilt nach Satz 4.19

$$\|L\|_\infty = \max_i \left(\sum_{k=1}^{i-1} |\ell_{ik}| + 1 \right) \leq 1 + (n-1) \max |\ell_{ik}|$$

und

$$\|L^{-1}\|_\infty \leq \prod_{k=1}^{n-1} \|I - G_k\|_\infty \leq \prod_{k=1}^{n-1} \max_{i>k} (1 + |\ell_{ik}|) \leq (1 + \max |l_{ik}|)^{n-1}.$$

■

Damit gilt

$$\sigma \leq \frac{3}{2}(1 + (n-1)\ell_{\max})^2 (1 + \ell_{\max})^{2(n-1)}. \tag{4.29}$$

Falls es überhaupt etwas zu eliminieren gibt, gilt $\ell_{\max} > 0$. Demzufolge *kann* also $n \gg 1$ oder $\max |\ell_{ik}| \gg 1$ *Instabilität* des Gaußschen Algorithmus zur Folge haben.

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 6 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 4.16 erweist sich als instabil.

4.4 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 4.21 für möglichst gute Kondition der Eliminationsmatrix $I - G_k$ zu sorgen.

Diese Überlegungen führen auf die folgende praktikable Variante des „naiven“ Verfahrens 4.16.

Algorithmus 4.22 (Gauß-Elimination mit Spaltenpivoting)

Für $k = 1, \ldots, n-1$ berechne
{
$k_0 = k$

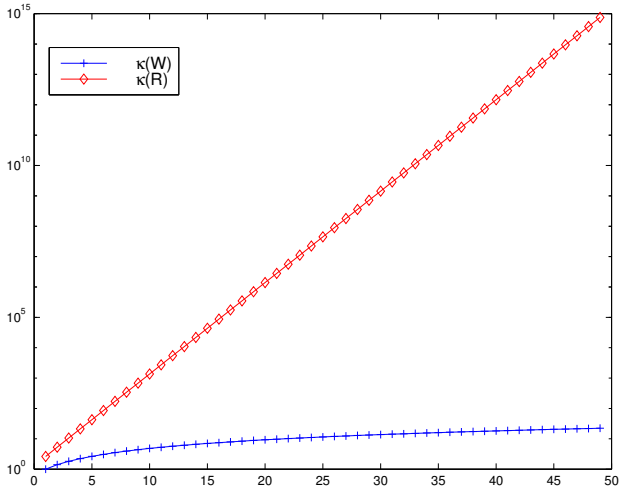


Abbildung 6: Kondition der Wilkinson-Matrix und ihres Faktors R.

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 4.16.
}

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
$$\max_{i=k,..,n} \frac{\left|a_{ik}^{(k-1)}\right|}{\left|a_{k_0 k}^{(k-1)}\right|} \leq \max_{i=k,..,n} \frac{\left|a_{ik}^{(k-1)}\right|}{\left|a_{jk}^{(k-1)}\right|} \qquad \forall j = k, \ldots, n \ .$$
Dieses Vorgehen ist jedoch aufwendiger. Meist verwendet man stattdessen eine geschickte Skalierung des Gleichungssystems. Wir verweisen auf Deuflhard und Hohmann [1], S. 13.

Nun gilt durch die Wahl der Pivot-Elemente $\ell_{\max} \leq 1$, so daß die Stabilität der Gauß-Elimination mit Spaltenpivoting entsprechend (4.29) durch

$$\sigma \leq \frac{3}{2} n^2 2^{2(n-1)}$$

gegeben ist. Für große n kann der Gauß-Algorithmus also auch mit Spaltenpivoting instabil sein. Jedoch ist die Abschätzung natürlich nicht scharf, und für viele Arten von Matrizen (symmetrisch positiv definite, diagonaldominante, zufällige, Bandmatrizen, etc.) können deutlich schärfere Resultate gezeigt werden. Insbesondere führt schon eine *Nachiteration* (siehe [1]) zur Stabilität:

1. zerlege $A = LR$

2. löse $Lz = b$
3. löse $R\tilde{x} = z$
4. berechne $r = b - A\tilde{x}$
5. löse $L\bar{z} = r$
6. löse $R\bar{x} = \bar{z}$
7. berechne $x = \tilde{x} + \bar{x}$

Das Analogon zu Satz 4.19 ist nun

Satz 4.23 *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

$$A = P_1(I + G_1) \ldots P_{n-1}(I + G_{n-1})R \ . \tag{4.30}$$

Wir setzen

$$Q_k = P_{n-1} \cdots P_k \ , \quad k = 1, \ldots, n-1 \ , \quad Q_n = I \ .$$

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 (4.30) schließlich

$$\begin{aligned} P P_1 (I + G_1) \cdots P_{n-1}(I + G_{n-1}) &= (I + Q_2 G_1) Q_2 P_2 (I + G_2) \cdots P_{n-1}(I + G_{n-1}) \\ &= (I + Q_2 G_1)(I + Q_3 G_2) \cdots (I + G_{n-1}) \\ &= I + \sum_{m=1}^{n-1} Q_{k+1} G_k = L \ , \end{aligned} \tag{4.31}$$

denn die Matrizen $Q_{k+1}G_k$ sind von der gleichen Bauart wie G_k und man kann wie in (4.28) schließen. Aus (4.30) und (4.31) folgt die Behauptung. ■

Übrigens liefert der MATLAB–Befehl

```
[L,R,P]=lu(A)
```

eine LR–Zerlegung von A mit Pivoting. Ausprobieren!

Literatur

[1] P. Deuflhard and A. Hohmann. *Numerische Mathematik I.* de Gruyter, 1993. Der Gaußsche Algorithmus wird in Kapitel 1 behandelt. Wer sich für die Lösung schlecht konditionierter Probleme interessiert, sollte in Kapitel 3 nachlesen (oder die Vorlesung Numerik I besuchen).

[2] P. Drabek and A. Kufner. *Integralgleichungen. Mathematik für Ingenieure und Naturwissenschaftler.* Teubner, 1996. Eine anwendungsorientierte Einführung, die einen guten Hintergrund für theoretische Betrachtungen liefert.

[3] N.J. Higham. *Accuracy and Stability of Numerical Algorithms.* SIAM, 1996. Für Spezialisten wird der aktuelle Stand der Forschung auf knapp 700 Seiten zusammengefaßt.

[4] D. Werner. *Funktionalanalysis.* Springer, 2. Auflage, 1997. Ein berühmtes 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 $x_1, \ldots, x_n$

$$x = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \quad \text{oder} \quad x = \big(x_1, x_2, \cdots, x_n\big)\,.$$

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:

$$A = \begin{pmatrix} 1 & 2 \\ 2 & 3 \end{pmatrix} \quad \text{ist symmetrisch, denn } A^T = \quad \begin{pmatrix} 1 & 2 \\ 2 & 3 \end{pmatrix}.$$

A.3 Verwendungsbeispiele

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

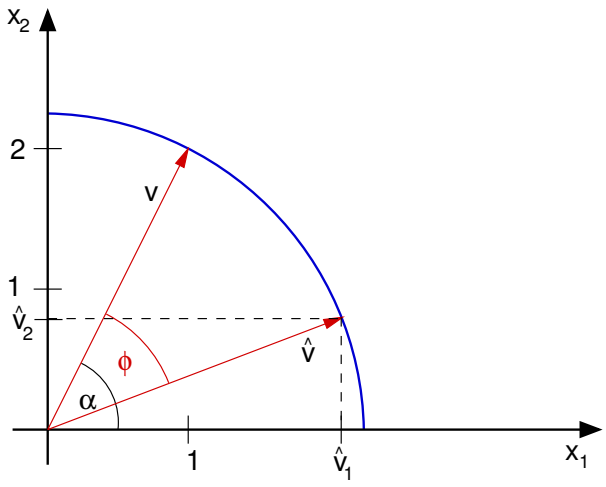


Abbildung 7: Drehung eines Vektors v um den Winkel ϕ im Uhrzeigersinn.

In Abbildung 7 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 dem Winkel α zwischen v und der x_1-Achse erhalten wir nach den elementaren trigonometrischen Regeln:

$$\begin{aligned} v_1 &= \sqrt{5}\,\cos\alpha \\ v_2 &= \sqrt{5}\,\sin\alpha \end{aligned}$$

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

$$\begin{aligned} \hat{v}_1 &= \sqrt{5}\,\cos(\alpha - \phi) = \sqrt{5}\,(\cos\alpha \cdot \cos\phi + \sin\alpha \cdot \sin\phi) \\ \hat{v}_2 &= \sqrt{5}\,\sin(\alpha - \phi) = \sqrt{5}\,(\sin\alpha \cdot \cos\phi - \sin\phi \cdot \cos\alpha)\,, \end{aligned}$$

oder

$$\begin{aligned} \hat{v}_1 &= \cos\phi \cdot v_1 \;+\; \sin\phi \cdot v_2 \\ \hat{v}_2 &= -\sin\phi \cdot v_1 \;+\; \cos\phi \cdot v_2, \end{aligned}$$

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 *Drehmatrizen*. Sie spielen in vielen Bereichen eine wichtige Rolle.

B Beispiele zur Modellierung

Dieser Anhang ist wiederum nur für die in besonderem Maße Interessierten bestimmt und wird nur in Abschnitt 4.1 benötigt und ist dort auch nicht zentral. Trotzdem ist es wichtig, auch einmal genauer über den Vorgang der *Modell-Erstellung* nachzudenken. Das Folgende ist ein Beispiel für diesen Vorgang.

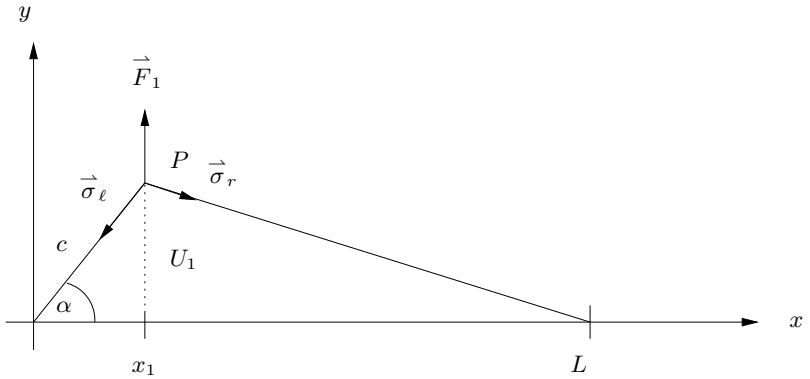
B.1 Auslenkung einer Saite

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.

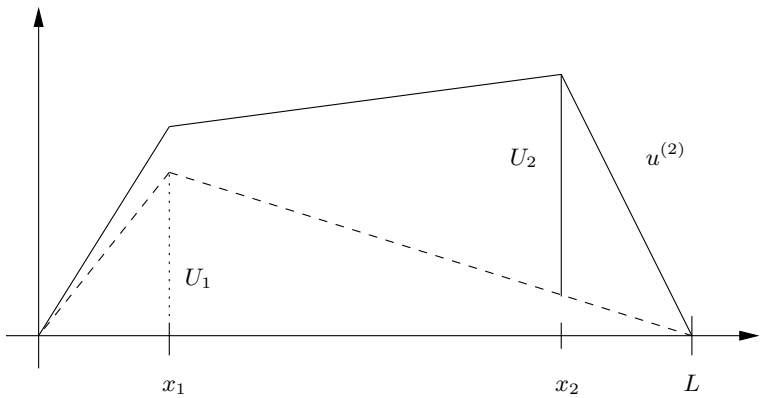
$$U_1 = \frac{(L - x_1)x_1}{|\vec{\sigma}| L} F_1 \ .$$

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

$$\int_0^L K(x, \xi) f(\xi) d\xi = u(x) \tag{2.3}$$

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 (2.3) nicht definiert. In diesem Fall ist unser Modell *mathematisch sinnlos.*
- Unser Modell (2.3) 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 (2.3) 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.

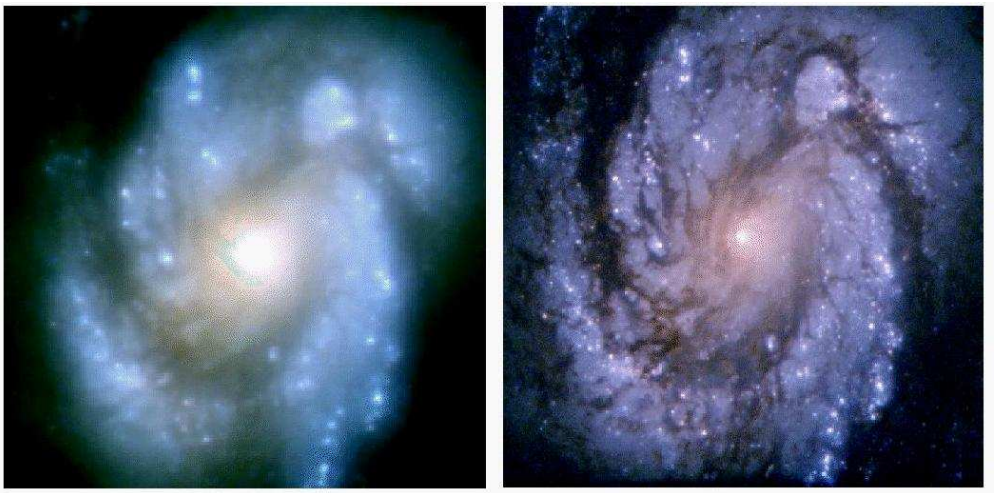


Abbildung 8: Bilder des Weltraumteleskops Hubble vor und nach dem Einbau der Korrekturoptik. Hier die Spiralgalaxie M100.

B.2 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 8). 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:

$$b(x) = \int_{\xi_l}^{\xi_r} K(x, \xi) w(\xi)\, d\xi \qquad \forall x \in [x_l, x_r] \tag{2.4}$$

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 gegebenem b ein w zu finden, so daß die Fredholmsche Integralgleichung erster Art (2.4) erfüllt ist.

Die Rekonstruktion des gewünschten Bildes kann sich durchaus sehen lassen (Abbildung 9).

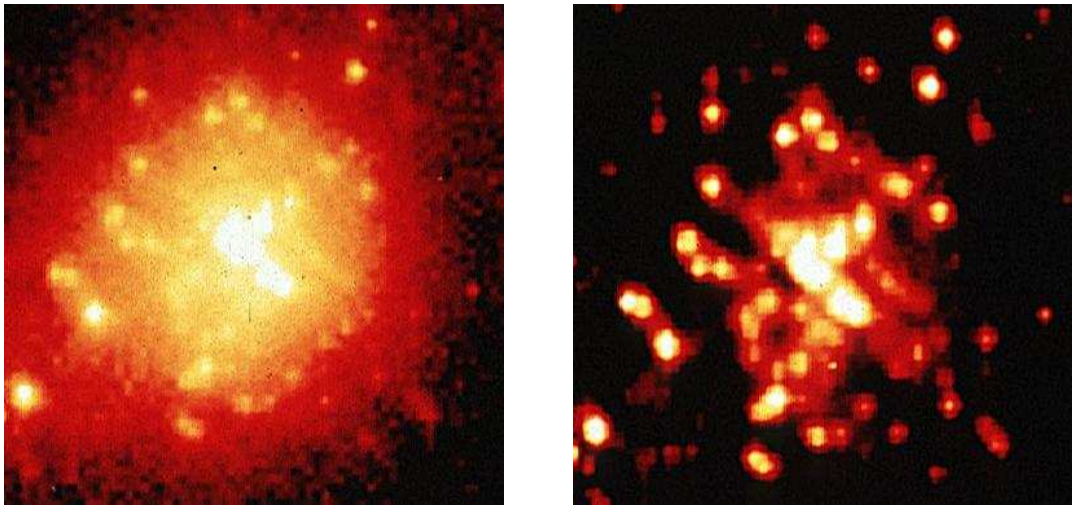


Abbildung 9: Bilder des Weltraumteleskops Hubble vor und nach der mathematischen Korrektur.

C Nochmal: Dreitermrekursion

Wir wollen kurz die Stabilität der beiden Algorithmen diskutieren, ohne allerdings eine detaillierte Stabilitätsanalyse wie in Abschnitt 2.4 vorzunehmen.

Algorithmus 2.9 beruht auf der sukzessiven Berechnung der x_k. Ausgehend von gestörten Eingangsdaten $\tilde{x}_0, \tilde{x}_1$ kommt dabei in jedem Schritt von k auf $k+1$ ein neuer Rundungsfehler der Größenordnung eps hinzu. Man kann nun x_{k-1}, x_k als Startwert für die restlichen $50-k$ Schritte auffassen, die eine gemeinsame Kondition von etwa

$$\left(\frac{\lambda_2}{\lambda_1}\right)^{50-k}$$

aufweisen. Wir setzen $q = \lambda_2/\lambda_1 \approx 5.83$. Dann wird klar, daß der Gesamtfehler ungefähr

$$\epsilon = \sum_{k=0}^{50} q^{50-k}\text{eps} \leq q^{50}\frac{\text{eps}}{1-q^{-1}}$$

beträgt. Die *zusätzliche* Fehlerverstärkung durch den Algorithmus ist also der Faktor $1/(1-q^{-1}) \approx 1.21$. Trotz der grauenhaften Ergebnisse ist Algorithmus 2.9 also stabil!

Natürlich möchte man sich nicht mit der Erkenntnis „Kondition zu groß — Problem unlösbar“ abfinden. Die rettende Idee ist nun, ein ganz anderes Problem mit anderen Eingabedaten zu formulieren, das auf das gleiche Ergebnis führt. Dies ist z.B. die Grundlage von Algorithmus 2.11. Die Eingabedaten bestehen hier nur aus dem *ersten* Folgenglied x_0! Schon x_1 wird durch die analytisch gewonnene Erkenntnis bestimmt, daß sich die Lösung als $x_0\lambda_1^k$ darstellen läßt.

Ein wesentlicher Nachteil von Algorithmus 2.11 ist, daß seine Anwendungsmöglichkeit auf *konstante Koeffizienten* a, b beschränkt ist. Im Falle

$$x_{k+1} + a_k x_k + b_{k-1} x_{k-1} = 0$$

gilt nämlich Satz 2.10 nicht mehr.

Ein Ausweg besteht in der *näherungsweisen* Lösung von Problem (2.9): Anstatt zu versuchen, die Folgenglieder $x_2, ..., x_{50}$ exakt zu berechnen, konstruieren wir für jedes $k = 1, ..., 50$ eine Näherung $x_k^{(n)}, n = 0, ..$ mit der Eigenschaft

$$x_k^{(n)} \longrightarrow x_k \ , \qquad n \to \infty.$$

Wir haben schon gesehen, daß die numerische Anwendung mathematisch exakter Verfahren auch nur (rundungs-) fehlerbehaftete Lösungen liefert. Daher ist es durchaus kein Nachteil, wenn ein Näherungsverfahren (beliebig genaue) Approximation liefert.

Zur Konstruktion eines brauchbaren Näherungsverfahrens für (2.9) betrachten wir die *zugehörige Rückwärtsrekursion*

$$y_{k-1} = 2y_k + y_{k+1} \ , \qquad k = n+1, ..., 1$$

für die Anfangswerte $y_{n+1} = 1$, $y_{n+2} = 0$. Nach Satz 2.10 ist die Lösung gegeben durch

$$y_k = \alpha \lambda_1^{-(n-k+2)} + \beta \lambda_2^{-(n-k+2)} \ , \qquad k = n, ..., 0 \ .$$

Dabei sind $\lambda_1 = \sqrt{2} - 1$, $\lambda_2 = \sqrt{2} + 1$ nach wie vor die Nullstellen des charakteristischen Polynoms

$$\lambda^2 + 2\lambda - 1 = 0,$$

denn $\lambda_1^{-1}, \lambda_2^{-1}$ sind die Nullstellen des charakteristischen Polynoms

$$\mu^2 - 2\mu - 1 = 0$$

der Rückwärtsiteration. Die Koeffizienten α, β erhält man nun aus

$$\begin{aligned} \alpha + \beta &= 0 \\ \alpha \lambda_1^{-1} + \beta \lambda_2^{-1} &= 1. \end{aligned}$$

Im Vergleich mit der Vorwärtsiteration dreht sich das Wachstum der zu λ_1 bzw. λ_2 gehörenden Beiträge zur Lösung um. So gilt insbesondere

$$y_k = \alpha \lambda_1^{-(n-k+2)} + \beta \lambda_2^{-(n-k+2)} \approx \alpha \lambda_1^{-(n-k+2)} \ , \qquad k = 0, ..., 50 \ ,$$

falls $n \gg 50$, denn es ist ja $\lambda_1^{-1} > 1$, $\lambda_2^{-1} < 1$. Daher ist für genügend große n

$$x_k^{(n)} := \frac{y_k}{y_0} \approx \frac{\alpha \lambda_1^{-(n-k+2)}}{\alpha \lambda_1^{-(n+2)}} = \lambda_1^k = x_k \ , \qquad k = 0, ..., 50 \ . \tag{3.5}$$

Beachte, daß $\alpha \neq 0$ vorliegt!

Beobachtung (3.5) ist die Grundlage unseres Näherungsverfahrens.

Algorithmus C.1 (Miller 1952)

Initialisierung:	Wähle $n \geq 50$ genügend groß.
Rückwärtsiteration:	$y_k = 2y_{k+1} + y_{k+2}, \quad k = n, ..., 0, \quad y_{n+1} = 1, \quad y_{n+2} = 0.$
Normierung:	$x_k^{(n)} = \frac{y_k}{y_0} \qquad k = 0, ..., 50$.

Bei Wahl von $n = 100$ liefert ein entsprechendes MATLAB-Programm das Ergebnis

```
x_50 = 7.264667356934824e-20
```

Das ist bis auf 13 Stellen genau das Ergebnis von Algorithmus 2.11. Wir wollen dieses ermutigende Resultat nun theoretisch absichern.

Satz C.2 *Für die in* (3.5) *berechneten Näherungen gilt*

$$\lim_{n\to\infty} x_k^{(n)} = x_k \ , \qquad k = 0, ..., 50 \ .$$

Beweis: Wegen $\alpha \neq 0$ haben wir

$$\frac{y_k}{y_0} = \frac{\alpha\lambda_1^{-(n-k+2)} + \beta\lambda_2^{-(n-k+2)}}{\alpha\lambda_1^{-(n+2)} + \beta\lambda_2^{-(n+2)}} = \frac{\lambda_1^k + \frac{\beta}{\alpha}\lambda_2^k \left(\frac{\lambda_1}{\lambda_2}\right)^{n+2}}{1 + \frac{\beta}{\alpha}\left(\frac{\lambda_1}{\lambda_2}\right)^{n+2}} \ .$$

Offenbar ist für jedes feste $k = 0, ..., 50$

$$\lim_{n\to\infty} \lambda_2^k \left(\frac{\lambda_1}{\lambda_2}\right)^{n+2} = 0$$

und es folgt die Behauptung. ■

Der Miller–Algorithmus läßt sich auf Differenzengleichungen mit variablen Koeffizienten verallgemeinern, wird dann allerdings etwas aufwendiger. Für Einzelheiten verweisen wir auf das Kapitel 6 im Lehrbuch von Deuflhard und Hohmann [1].

Wir fassen zusammen:

- Der naive Algorithmus 2.9 ist stabil, liefert aber aufgrund der schlechten Kondition des Problems keine brauchbaren Ergebnisse.
- Algorithmus 2.11, also die Auswertung der geschlossenen Lösung aus Satz 2.10 ist stabil und berechnet ein gut konditioniertes Problem, ist aber nur auf konstante Koeffizienten a_k, b_k anwendbar.
- Algorithmus C.1 ist stabil und löst ein gut konditioniertes Problem. Darüberhinaus ist er auf nichtkonstante Koeffizienten verallgemeinerbar, aber aufwendiger als Algorithmus 2.11.

Die Frage nach dem *besten* Algorithmus ähnelt der Frage nach dem besten Golfschläger (traditioneller Studentensport): Die Antwort ist sehr problemabhängig.

In unserem Fall gewinnt Algorithmus 2.11. Bei variablen Koeffizienten scheidet er gleich aus. Bei gut konditionierten Problemen (auch die gibt es!) wird man Algorithmus 2.9 nehmen. Für schlechte Kondition und variable Koeffizienten bleibt Algorithmus C.1.

Matrixdarstellung der 3-Term-Rekursion: Zum Abschluß dieses Kapitels wollen wir unseren zentralen Satz 2.10 noch in einen allgemeineren Zusammenhang stellen, der in den Vorlesungen *Lineare Algebra II* und *Computerorientierte Mathematik II* wieder auftauchen wird.

Wir können offenbar die 3–Term–Rekursion

$$x_{k+1} + ax_k + bx_{k-1} = 0 \tag{3.6}$$

auch in folgender Form schreiben

$$\begin{aligned} x_{k+1} &= -ax_k - bx_{k-1} \\ x_k &= 1x_k \ . \end{aligned}$$

Zur Abkürzung führen wir die Bezeichnungen

$$u_{k+1} = \begin{pmatrix} x_{k+1} \\ x_k \end{pmatrix} \qquad \text{Vektor}$$

$$A = \begin{pmatrix} -a & -b \\ 1 & 0 \end{pmatrix} \qquad \text{Matrix}$$

ein.

Wir definieren das Matrix–Vektor–Produkt

$$Au_k = \begin{pmatrix} -a & -b \\ 1 & 0 \end{pmatrix} \begin{pmatrix} x_k \\ x_{k-1} \end{pmatrix} \overset{\text{Def}}{=} \begin{pmatrix} -ax_k - bx_{k-1} \\ 1 \cdot x_k + 0 \cdot x_{k-1} \end{pmatrix}$$

gerade so, daß $u_{k+1} = Au_k$ gilt.

Weiter definieren wir

$$\lambda \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = \begin{pmatrix} \lambda v_1 \\ \lambda v_2 \end{pmatrix} \qquad \forall \lambda \in \mathbb{R} \ .$$

Es folgt

$$A(\lambda v) = \lambda A v \qquad \forall \lambda \in \mathbb{R} \ .$$

Eine Zahl λ heißt *Eigenwert* von A zum Eigenvektor $v = \begin{pmatrix} v_1 \\ v_2 \end{pmatrix}$ von A falls

$$Av = \lambda v \ .$$

Beobachtung: In diesem Fall ist

$$u_k = \alpha \lambda^k v \qquad \forall \alpha \in \mathbb{R}$$

Lösung der 3–Term–Rekursion (3.6), denn

$$Au_k = A(\alpha \lambda^k v) = \alpha \lambda^k A v = \alpha \lambda^{k+1} v = u_{k+1} \ .$$

Die Nullstellen λ_1, λ_2 des charakteristischen Polynoms

$$\lambda^2 + a\lambda + b = 0$$

aus Satz 2.10 sind gerade die Eigenwerte von A.

Beispiel:

$$\begin{aligned} a = 2, \quad b = -1 \quad A &= \begin{pmatrix} -2 & 1 \\ 1 & 0 \end{pmatrix} \\ \lambda_1 &= \sqrt{2} - 1 \\ v_1 &= \begin{pmatrix} 1 \\ 1 + \sqrt{2} \end{pmatrix} \\ Av_1 &= \begin{pmatrix} -2 + 1 + \sqrt{2} \\ 1 \end{pmatrix} = \begin{pmatrix} \sqrt{2} - 1 \\ (\sqrt{2} - 1)(\sqrt{2} + 1) \end{pmatrix} = \lambda_1 v_1 \end{aligned}$$

Literatur

[1] P. Deuflhard and A. Hohmann. *Numerische Mathematik I.* de Gruyter, 1993. Dreitermrekursion und die stabile Summation von Minimallösungen wird ausführlich in Kapitel 6 behandelt.