Noname manuscript No.
(will be inserted by the editor)

Gleitkommaarithmetik auf dem Prüfstand

Wie werden verifiziert(e) numerische Lösungen berechnet?

Siegfried Michael Rump

Received: date / Accepted: date

Zusammenfassung Um es vorweg zu nehmen, ungenaue numerische Resultate sind selten; zu selten, sich immer darum kümmern zu müssen, aber nicht selten genug, sie zu ignorieren.

Im folgenden sollen die Grenzen und Möglichkeiten von Gleitkommaarithmetik und von numerischen Verfahren untersucht und Eigenschaften und Fakten verdeutlicht werden, insbesondere anhand einiger Beispiele. Namentlich werden Algorithmen besprochen, die zwar nur Gleitkommaarithmetik nutzen, aber dennoch grundsätzlich nur korrekte Ergebnisse liefern.

Um auch das vorweg zu nehmen, korrekte Ergebnisse nicht-trivialer Probleme können mit Intervallarithmetik berechnet werden, auch wenn jene zuweilen immer noch in einem zweifelhaften Ruf steht. Hierauf wird öfter eingegangen, auch in einem eigenen Kapitel 15.

Der Artikel wendet sich insbesondere an Mathematiker, die bisher eher peripher mit Numerik zu tun hatten. Ich hoffe auf Nachsicht, daß wenig mehr als mathematisches Abiturwissen durchaus genügt und verweise auf [61] für diejenigen, die mehr bzw. andere Mathematik hinzufügen möchten. Allein, ohne die hier explizierten Grundlagen wird man den tieferen Sinn der dort vorgestellten Verfahren kaum gründlich verstehen können.

Schlüsselwörter Gleitkommaarithmetik - Rundungsfehler - Verifikationsmethoden

Mathematics Subject Classification 65G20 - 65Y04 - 68M07

1 Einleitung

Numerische Methoden gibt es zwar seit jeher, naturgemäß wurden sie aber erst mit dem Aufkommen elektronischer Rechenanlagen als eigenständige mathematische Disziplin interessant. Wie bekannt, waren die Anfänge nicht ganz problemlos, und die neue Disziplin „Numerische Mathematik“ mußte allerlei Verballhornungen erdulden.

In der Tat war es ungewohnt, statt mit reellen oder komplexen Zahlen mit Approximationen zu hantieren. Wir beginnen daher mit der allgemeinen Fragestellung, reelle Zahlen

Unterstützt durch das JST CREST Projekt, das MEXT Super Global University Projekt und das MEXT Leading University Projekt, Waseda Universität, Tokio, Japan.

Siegfried M. Rump, Institut für Zuverlässiges Rechnen, Technische Universität Hamburg und
Waseda University, Faculty of Science and Engineering, Tokyo, Japan

und Operationen auf einer Teilmenge zu approximieren und zeigen, daß unter schwachen Voraussetzungen weder Assoziativgesetze für Addition und Multiplikation noch Distributivgesetz gelten können. Auf Gleitkommaoperationen spezialisiert, die in der Regel mit einem (kleinen) Fehler behaftet sind, wird kurz umrissen, wie man aus Einzelfehlern auf den Fehler zusammengesetzter Operationen schließen kann.

Die Abschätzungen sind zwar beinahe scharf, im allgemeinen jedoch zu pessimistisch. Das Ziel des Aufsatzes wird daher sein, einerseits Fallstricke einiger numerischer Methoden aufzuzeigen, und andererseits Möglichkeiten zu besprechen, wie brauchbare, mathematisch korrekte Fehlerabschätzungen mit nicht allzu viel Mehraufwand berechnet werden können.

Insbesondere geht es um die Beurteilung numerischer Ergebnisse, fehlerfreie Gleitkomma-Transformationen, toleranzbehaftete Daten, automatische Differentiation und anderes mehr.

Es werden eine Reihe von Beispielen mit der neuesten Version 2016a des weitverbreiteten Softwarepakets Matlab berechnet [47]. Das hat den besonderen Vorteil, daß ausführbare Programmstücke gezeigt werden können. Die in Matlab verwandten numerischen Programme entsprechen zumeist dem state-of-the-art, obwohl gelegentlich Kompromisse zwischen Rechenzeit und Genauigkeit gemacht werden, siehe Kapitel 9.

Die Beispiele und Programmstücke für Verifikationsmethoden sind mit INTLAB [59], der toolbox für Zuverlässiges Rechnen, in Matlab und in Octave [53] ausführbar. Dass Programmpaket INTLAB ist von mir geschrieben und wird von mir seit 1998 stetig weiterentwickelt; es wird von einigen Tausend Nutzern in über 50 Ländern verwendet.

2 Prinzipielle Eigenschaften einer Arithmetik auf dem Rechner

Es sei $\mathbb{F} \subseteq \mathbb{R}$ eine Teilmenge der reellen Zahlen, so daß $0 \in \mathbb{F}, \mathbb{F} = -\mathbb{F}$ und $\mathbb{F} \backslash \{0\}$ diskret ist. Auf $\mathbb{F}$ sollen Operationen definiert werden, die die reellen Grundrechnungsarten approximieren. Die Menge $\mathbb{F}$ kann z.B. eine Menge von später definierten „Gleitkommazahlen“ sein; für den Moment ist es eine beliebige Menge mit den genannten Eigenschaften.

Da 0 der einzige potentielle Häufungspunkt in $\mathbb{F}$ ist und $0 \in \mathbb{F}$, gibt es zu jeder reellen Zahl $r \in \mathbb{R}$ ein eindeutig bestimmtes, nächstgelegenes Element in $\mathbb{F}$, es sei denn, r liegt in der Mitte zweier benachbarter Elemente in $\mathbb{F}$. Durch eine geeignete Definition für diesen Fall ergibt sich eine Rundung $\mathrm{fl} : \mathbb{R} \to \mathbb{F}$ mit minimalem Fehler. Dieses „geeignet“ ist weitgehend gleichgültig, man fordert nur eine naheliegende Symmetrieeigenschaft (R2), so daß

$$\forall r \in \mathbb{R}: \qquad |\mathrm{fl}(r) - r| = \min\{|f - r| : f \in \mathbb{F}\} \tag{R1}$$

$$\forall r \in \mathbb{R}: \qquad \mathrm{fl}(-r) = -\mathrm{fl}(r) \tag{R2}$$

$$\forall f \in \mathbb{F}: \qquad \mathrm{fl}(f) = f \tag{R3}$$

$$\forall r,s \in \mathbb{R}: \qquad r \le s \;\Rightarrow\; \mathrm{fl}(r) \le \mathrm{fl}(s) \tag{R4}$$

gilt. Man beachte, daß (R3) und (R4) aus der Bestapproximationseigenschaft (R1) folgen. Die Näherung $\tilde{\circ} : \mathbb{F} \times \mathbb{F} \to \mathbb{F}$ einer Grundoperation $\circ \in \{+, -, \cdot, /\}$ mit

$$\forall f,g \in \mathbb{F}: \qquad f \,\tilde{\circ}\, g := \mathrm{fl}(f \circ g) \tag{2.1}$$

realisiert offenbar eine bestmögliche Approximation der reellen Operation $\circ$:

Fakt 2.1 *Der Fehler jeder (einzelnen) Operation $\tilde{\circ}$ ist kleinstmöglich.*

Das ist die wohl naheliegendste Definition einer Arithmetik auf einer recht allgemeinen Teilmenge $\mathbb{F} \subset \mathbb{R}$, die jetzt auf eine spezifische Menge angewandt wird. Für gegebene Basis $2 \leq \beta \in \mathbb{N}$ sind *Gleitkommazahlen* f mit $2 \leq k \in \mathbb{N}$ Mantissenziffern $m_\nu \in \mathbb{N}, 0 \leq m_\nu < \beta$ und Exponent $e \in \mathbb{Z}$ von der Form

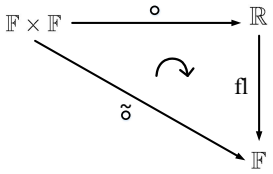
$$f := \pm 0, m_1 m_2 \ldots m_k \cdot \beta^e = \pm \sum_{\nu=1}^{k} m_\nu \beta^{e-\nu}. \tag{2.2}$$

Im folgenden seien β und k fest, wobei der Einfachheit halber in (2.2) der Exponentenbereich unbeschränkt ist.[1] Eine eindeutige Darstellung wird offenbar erreicht, wenn (außer für $f = 0$) $m_1 \neq 0$, d.h. Gleitkommazahlen als „normalisiert“ vorausgesetzt werden. Die Menge solcher Gleitkommazahlen sei mit $\mathbb{F} = \mathbb{F}_{\beta,k}$ bezeichnet. Es folgt $-\mathbb{F} = \mathbb{F}$, daß 0 der (einzige) Häufungspunkt von $\mathbb{F}$ ist, und wegen (2.1) die meist benutzte Fehlerabschätzung

$$\forall r \in \mathbb{R}: \qquad |\mathrm{fl}(r) - r| \leq \mathbf{u}|r| \quad \text{mit} \quad \mathbf{u} := 0.5 \cdot \beta^{1-k}. \tag{R5}$$

Dabei ist egal, zu welchem Nachbarn der Mittelpunkt zweier benachbarter Gleitkommazahlen gerundet wird, einzig (R1) und (R2) müssen gelten. Man nennt $\mathbf{u}$ die *relative Rundungsfehlereinheit*.

Die praktische Realisierbarkeit der Gleitkommaoperationen folgt leicht, wenn man einen Akkumulator mit $2k$ Stellen zur Basis β voraussetzt: Die Multiplikation ist dann gar exakt, ebenso Addition und Subtraktion, wenn die Exponenten der Operanden sich nicht um mehr als $k-1$ Stellen unterscheiden; falls doch, liefert eine Fallunterscheidung das korrekte Ergebnis, und Quotientenziffern können einzeln berechnet werden. Unnötig zu erwähnen, daß es sehr viel effizientere Methoden gibt [50].



Das sieht alles sehr einfach aus, und man mag sich fragen, warum nicht schon immer definiert wurde, daß nebenstehendes Diagramm gemäß (2.1) kommutiert. Nun, bis in die 1980er Jahre war es gar nicht so leicht, überhaupt Informationen über die tatsächliche Implementierung der Arithmetik in einer Rechenanlage zu bekommen. Das klingt aus heutiger Sicht verwunderlich, entspricht aber den Tatsachen (Wilkinson [81] bemängelte: „Need the arithmetic be so bad!“).

```
  10 000 000
-  9 999 999,7
  ------------
           1
```

Abbildung 2.1: Subtraktion ohne „guard digit“

Mehr noch, diverse Groß- und Superrechner implementierten Operationen mit einem relativen Fehler von bis zu 100% (sic!), während mit Definition (2.1) der maximale relative Fehler gemäß (R5) höchstens $\mathbf{u}$ beträgt - wie man es erwarten würde. Der Grund lag in der Verwendung eines Rechenakkumulators ohne „guard digit“, m.a.W., für k Mantissenstellen hatte der Rechenakkumulator ebenfalls k Stellen zur Basis β.

Das Ausmaß des Schreckens macht man sich leicht an einem 8-stelligen dezimalen Taschenrechner klar, siehe Abbildung 2.1: Beide Zahlen sind in 8 Dezimalstellen darstellbar, und zur Subtraktion müssen „wie in der Schule“ die Kommata untereinander geschoben werden. Jetzt wäre allerdings eine neunte Stelle für die letzte Ziffer 7 notwendig. Da

Abbildung 2.2: 100 Yen Laden[2]

[1] Dadurch entfallen Über- und Unterlauf ohne essentielle Probleme auszuklammern.

[2] Hier werden nützliche Dinge des Alltags zum Einheitspreis von 100 Yen ($\approx$ 80 €-cent) angeboten.

der Rechenakkumulator jedoch nur 8 Stellen hat, verschwindet diese einfach. Die letzte Ziffer 7 im Subtrahenden kann durch jede Ziffer von 0 bis 9 ersetzt werden, das Ergebnis ist immer gleich 1.

Dieses Prinzip kann man auf jedem Taschenrechner mit 8, 10 oder 12 Dezimalstellen ohne Exponent, den man u.a. als Ramschware für einen Euro (oder 100 Yen, siehe Abb. 2.2) bekommt, ausprobieren. Wie erwähnt lag der Arithmetik diverser Großrechenanlagen (Univac, Cray etc.) bis in die 1980/90er Jahre das gleiche Prinzip zugrunde, nur eben binär.

Dies führte 1985 zur Definition des IEEE 754 Arithmetik Standards [33], der 2008 [34] erweitert wurde. Diese Definition der Arithmetik erfüllt insbesondere (R1...5) und (2.1), und ist heute vom Laptop bis zum Großrechner in praktisch allen Rechenanlagen realisiert.[3]

3 Fehlende mathematische Strukturen

Nach Fakt 2.1 ist der Fehler jeder Operation minimal, und es stellt sich die Frage nach mathematischen Strukturen einer Gleitkommaarithmetik.

Es wird sich zeigen und wirkt ernüchternd, daß außer dem Kommutativgesetz an mathematischen Eigenschaften nichts erwartet werden kann. Selbst das Assoziativgesetz für Gleitkommaoperationen, grundlegend in fast allen mathematischen Strukturen, *kann* nicht gelten.

Lemma 3.1 *Für beliebiges $\mathbb{M} \subseteq \mathbb{R}$ mit $\mathbb{M} = -\mathbb{M}$ und $0 \in \mathbb{M}$ sei eine Rundung $fl : \mathbb{R} \to \mathbb{M}$ mit (R1) gegeben, und eine Addition auf $\mathbb{M}$ erfülle (2.1). Falls es $a,b \in \mathbb{M}$ mit $a < b$ gibt, die in $\mathbb{M}$ aufeinanderfolgend sind, also $a < x < b \Rightarrow x \notin \mathbb{M}$, und $\delta := (b-a)/2 \in \mathbb{M}$, so ist die Addition nicht assoziativ.*

Beweis. Aus $\mathbb{M} = -\mathbb{M}$ folgt $\gamma := -\delta \in \mathbb{M}$, und wegen (R1) gilt $\mathrm{fl}(m) \in \{a,b\}$ für $m := a+\delta = \frac{a+b}{2} = b+\gamma$. Angenommen, $\mathrm{fl}(m) = a$. Dann gilt $\big(b \tilde{+} \gamma\big) \tilde{+} \delta \neq b \tilde{+} \big(\gamma \tilde{+} \delta\big)$ wegen

$$\mathrm{fl}\big(\mathrm{fl}(b+\gamma)+\delta\big) = \mathrm{fl}(a+\delta) = a \neq b = \mathrm{fl}(b+0) = \mathrm{fl}\big(b+\mathrm{fl}(\gamma+\delta)\big)$$

und $\mathrm{fl}(0) = 0 \in \mathbb{M}$. Falls $\mathrm{fl}(m) = b$ betrachtet man $a+\delta+\gamma$. □

Für Gleitkommazahlen $\mathbb{F}$ nach (2.2) ist die Bedingung $(b-a)/2 \in \mathbb{F}$ *für alle* aufeinanderfolgenden $a,b \in \mathbb{F}$ erfüllt; Lemma 3.1 schließt aber die Festkomma-Addition aus, die ja bis auf Überlauf immer fehlerfrei ist.

In der Realität ist $\mathbb{F}$ endlich, und ein nicht-trivialer Homomorphismus schließt sich damit aus. Nach einem ähnlichen Prinzip wie in Lemma 3.1 gilt das sehr viel allgemeiner.

Lemma 3.2 *Unter den Voraussetzungen von Lemma 3.1 ist für eine Rundung $fl : \mathbb{R} \to \mathbb{F}$ gemäß* (R1) *die durch (2.1) definierte Addition kein Homomorphismus.*

Beweis. Wie im Beweis von Lemma 3.1 ist $\mathrm{fl}(m) \in \{a,b\}$ für $m := a+\delta$. Angenommen, $\mathrm{fl}(m) = a$. Dann gilt

$$\mathrm{fl}(m+\delta) = \mathrm{fl}(b) = b \neq a = \mathrm{fl}(a+\delta) = a \tilde{+} \delta = \mathrm{fl}(m) \tilde{+} \mathrm{fl}(\delta).$$

Falls $\mathrm{fl}(m) = b$ betrachtet man $m+\gamma$ für $\gamma := -\delta \in \mathbb{F}$. □

[3] „Intel inside“; bis auf die erwähnten Taschenrechner.

Für die weitere Überlegung setzen wir binäre Gleitkommazahlen, d.h. $\beta = 2$, nach (2.2) und $k \geq 2$ voraus mit $(R1 \ldots 5)$ und Operationen nach (2.1). Dann ist $\mathbf{u} = 2^{-k}$ die relative Rundungsfehlereinheit, und offenbar sind

$$1, 1+2\mathbf{u}, 1+4\mathbf{u}, \ldots, 2-2\mathbf{u}, 2$$

alle Gleitkommazahlen im Intervall $[1,2]$. Für $m \in \mathbb{N}, 1+2m\mathbf{u} \leq 2$ und $|\delta| < \mathbf{u}$ ist daher $\mathrm{fl}(1+2m\mathbf{u}+\delta) = 1+2m\mathbf{u}$. Der Nachfolger von 1 ist $1+2\mathbf{u}$, wegen des wechselnden Exponenten ist der Vorgänger von 1 gleich $1-\mathbf{u}$.

Lemma 3.3 *Für binäre Gleitkommazahlen nach (2.2) ist weder das Assoziativgesetz der Addition oder Multiplikation noch das Distributivgesetz erfüllt.*

Beweis. Nach Lemma 3.1 ist die Addition nicht assoziativ.

Für $a \in \mathbb{F}$ mit $1.5 < a < 2-2\mathbf{u}$, $b := 1+2\mathbf{u}$ und $c := 1-\mathbf{u}$ gilt $a+3\mathbf{u} < ab < a+4\mathbf{u}$, so daß $1.5 < a < a+4\mathbf{u} = \mathrm{fl}(ab) =: d \leq 2$. Der Vorgänger von d ist also $d-2\mathbf{u}$, so daß $\big(a \tilde{\cdot} b\big) \tilde{\cdot} c = \mathrm{fl}(dc) = d-2\mathbf{u} = a+2\mathbf{u}$ wegen $d-2\mathbf{u} \leq dc < d-\mathbf{u}$. Anderseits ist aber $1 < bc < 1+\mathbf{u}$, so daß $\mathrm{fl}(bc) = 1$ und damit $a \tilde{\cdot} \big(b \tilde{\cdot} c\big) = a \neq a+2\mathbf{u}$. Die Multiplikation ist also nicht assoziativ.

Für $1 < a < 2$ und $b := -(1-\mathbf{u})$ ist $a \tilde{\cdot} \big(1 \tilde{+} b\big) = \mathrm{fl}\big(a \cdot \mathrm{fl}(1+b)\big) = \mathrm{fl}(a\mathbf{u}) = a\mathbf{u}$, weil mit $a \in \mathbb{F}$ immer $a\mathbf{u} \in \mathbb{F}$. Andererseits gilt $\mathrm{fl}(a \cdot 1) = a$ und $\mathrm{fl}(ab) = -\mathrm{fl}\big(a(1-\mathbf{u})\big)$ wegen (R2). Aus $1 < a < 2$ folgt $1 \leq a-2\mathbf{u} < a(1-\mathbf{u}) < a-\mathbf{u}$ und daher $\mathrm{fl}(ab) = -(a-2\mathbf{u}) =: d$, so daß schließlich $a \tilde{\cdot} 1 \tilde{+} a \tilde{\cdot} b = \mathrm{fl}(a+d) = \mathrm{fl}(2\mathbf{u}) = 2\mathbf{u} > a\mathbf{u} = a \tilde{\cdot} \big(1 \tilde{+} b\big)$. □

Fakt 3.4 *Eine Gleitkommaarithmetik kann elementare Gesetze wie Assoziativgesetz oder Distributivgesetz prinzipiell nicht erfüllen.*

Es liegt also nicht einmal eine Halbgruppe vor, und die in der Mathematik üblichen Strukturen mit ihren mächtigen Folgerungen greifen nicht. Man ist auf Abschätzungen angewiesen, die jedoch oft alles andere als schön sind.

Will man mit Sicherheit korrekte Ergebnisse erzielen, ist es unerläßlich, das Werkzeug präzis zu verstehen.[4] Daher wird in dieser Notiz in mehreren Kapiteln die Gleitkommaarithmetik genau untersucht; andernfalls kann man in unvermutete Fallen tappen, selbst bei reiner Integer-Arithmetik.

So könnte man auf die Idee kommen [38], ein Gegenbeispiel für den großen Fermatschen Satz konstruieren zu wollen. In der Tat wird man schnell fündig: In der Programmiersprache C [40] berechnet man $864^3 + 9792^3 = 9824^3$ für den Datentyp „short int“. Die Überprüfung in verdoppelter Genauigkeit, zur Sicherheit, liefert das gleiche Ergebnis.[5]

Allerdings verwenden diese Datentypen nur jeweils $k = 16$ bzw. $k = 32$ Bits, und alle Ergebnisse werden in einer Art wrap-around, i.d.R. ohne Fehlermeldung, auf den Bereich von -2^ℓ bis $2^\ell - 1$ mit $\ell := k-1$ abgebildet. Das führt zu kuriosen Erscheinungen wie $x = 2^\ell - 1, y = x+1 \Rightarrow y = -2^\ell$ oder gar $x = -2^\ell, y = -x \Rightarrow y = x$. So erklärt sich auch das obige „Gegenbeispiel“, was der inadäquaten Benutzung, nicht der Arithmetik geschuldet ist.

4 Unvermeidbare Fehler

Ab jetzt verwenden wir das heute am meisten verbreitete, doppeltgenaue binäre Format $\mathbb{F} = \mathbb{F}_{2,53}$, also 53 binäre Mantissenbits (in IEEE 754 „binary64“ genannt). In allen Beispielen tritt kein Über- oder Unterlauf auf, so daß es keinen Unterschied zu der vereinfachten Betrachtung in (2.2) mit unbeschränktem Exponentenbereich gibt.

[4] Der Unfall, der in [78] detailliert beschrieben wird, entstand durch ebensolches Unverständnis.

[5] Bei den Simpsons [68] wird ein ähnliches Beispiel für Taschenrechner konstruiert.

Fragen wir also, was man a priori über den Fehler einer numerischen Rechnung sagen kann. Mit der Definition von Gleitkommaoperationen nach (2.1) wird der Fehler jeder einzelnen Operation zwar minimal, bei mehreren Operationen kann trotzdem ein beliebig großer Fehler entstehen.

Rundet man eine reelle Zahl, so ist nach (R5) der relative Fehler maximal gleich der relativen Rundungsfehlereinheit $\mathbf{u} = 2^{-53} \approx 1.1 \cdot 10^{-16}$. Gleiches gilt nach (2.1) auch für jede Gleitkommaoperation, also für $\circ \in \{+,-,\cdot,/\}$

$$\forall f,g \in \mathbb{F}: \qquad f \tilde{\circ} g = (f \circ g) \cdot (1+\varepsilon) \quad \text{für ein } |\varepsilon| \le \mathbf{u}. \tag{4.1}$$

Bildet man von einer Gleitkommazahl f zweimal hintereinander den Kehrwert, muß das Ergebnis zwar nicht wieder f sein, aber sehr „nahe dran“, denn nach (4.1) gilt für alle $f \in \mathbb{F}$:

$$g = 1 \tilde{/} \left(1 \tilde{/} f\right) = \mathrm{fl}(1/\mathrm{fl}(1/f)) = 1/\big((1/f) \cdot (1+\varepsilon_1)\big) \cdot (1+\varepsilon_2) = f \cdot \frac{1+\varepsilon_2}{1+\varepsilon_1}$$

für geeignete $\varepsilon_1, \varepsilon_2 \in [-\mathbf{u}, \mathbf{u}]$; es gilt aber nicht notwendigerweise $g = f$. In der Tat ist z.B.

$$f = 8 \cdot 10^{15} \quad \Rightarrow \quad g = 1 \tilde{/} \left(1 \tilde{/} f\right) = f - 1 = 7999999999999999.$$

Mathematisch betrachtet mutet das zumindest seltsam an. Ebenso ist es u.U. ein Unterschied, 1 Prozent Zins auf ein Kapital K als $0.01 \cdot K$ oder als $K/100$ zu berechnen. Die Differenz ist zwar gering, könnte aber in einer Buchhaltung Probleme bereiten.

Immerhin kann positiv vermerkt werden, daß die Resultate nach dem IEEE 754 Standard auf praktisch allen Rechnern vorhersagbar gleich sind.

Es scheint allerdings auch klar, daß obige Abschätzungen für mehrere Operationen rasch unübersichtlich und ausgesprochen unschön werden können. Das ist die m.E. zu Recht negativ berüchtigte „Epsilontik“.

Bleiben wir zunächst bei nur zwei Operationen und betrachten $f - g \cdot h$. Wiederum folgt nach (4.1)

$$f \tilde{-} g \tilde{\cdot} h = \mathrm{fl}(f - \mathrm{fl}(g \cdot h)) = \big(f - (g \cdot h) \cdot (1+\varepsilon_1)\big) \cdot (1+\varepsilon_2)$$

mit $|\varepsilon_\nu| \le \mathbf{u}$. Haben also f und gh gleiches Vorzeichen und sind von ähnlicher Größenordnung, kann offenbar ein sehr großer relativer Fehler entstehen. Tatsächlich gilt für $f = 10^{16}, g = 221349167, h = 45177491$

$$f,g,h \in \mathbb{F} \quad \text{und} \quad f \tilde{-} g \tilde{\cdot} h = 4, \quad \text{aber} \quad f - gh = 3. \tag{4.2}$$

Der Grund ist die beschränkte Mantissenlänge von 53 Bit: $gh \in \mathbb{R}$ ist die Mitte der aufeinanderfolgenden Gleitkommazahlen $10^{16}-4, 10^{16}-2 \in \mathbb{F}$; in welche auch gerundet wird, der Gesamtfehler ist gleich 1, ein riesiger relativer Fehler:

Fakt 4.1 *Der Fehler mehrerer Gleitkommaoperationen kann beliebig groß sein.*

Der Effekt in (4.2) wird „Auslöschung“ genannt: Werden zwei Zahlen mit gleichem Vorzeichen von gleicher Größenordnung voneinander abgezogen, entsteht ein großer relativer Fehler. Der Fehler passiert allerdings nicht bei der Subtraktion (die ist sogar fehlerlos!), sondern in der Berechnung der Operanden.

Die Frage ist, ob diese Effekte auch in größeren Berechnungen vorkommen, und ob sich etwaige Fehler nicht mehr oder weniger kompensieren. Letzteres ist oft der Fall, aber nicht immer: Als Beispiel sollen die Eigenwerte der Matrix in Abb. 4.3 berechnet werden.

$$A = \begin{pmatrix} 275 & -451 & 708 & -1880 & -287 \\ 137 & -218 & 334 & -924 & -180 \\ 0 & -2 & 6 & -4 & 11 \\ 2 & -6 & 13 & -19 & 13 \\ 29 & -46 & 70 & -195 & -39 \end{pmatrix}$$

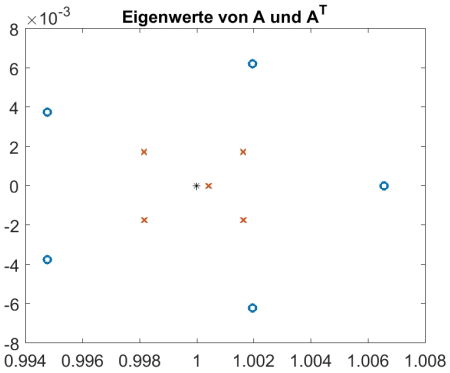


Abbildung 4.3: Eigenwertapproximationen

Dann liefert Matlab [47] mit dem Kommando `eig(A)` die in Abb. 4.3 mit blauen Kreisen markierten Approximationen der Eigenwerte von A in der komplexen Ebene. Berechnet man die Eigenwerte von A^T, so ergeben sich die mit roten „×“ markierten Approximationen. Es gibt allerdings keinerlei Warnung oder gar Fehlermeldung.

Die Matrix ist so konstruiert, daß sie einen fünffachen Eigenwert 1 (der Stern in der Abbildung) mit geometrischer Multiplizität 1 hat:

Fakt 4.2 *Einfachste numerische Algorithmen können in Gleitkommaartihmetik grob fehlerhafte Approximationen liefern, und zwar ohne Warnung.*

Gleitkommaarithmetik ist aber nun mal das Hilfsmittel, das zur Verfügung steht und Operationen mit faszinierender Geschwindigkeit ausführt. Also wird versucht, das Beste daraus zu machen.

Als nächstes wird untersucht, was denn das Beste ist. Wie im nächsten Kapitel erläutert, sind aus numerischer Sicht sowohl für die berechneten Eigenwerte von A als auch für die von A^T kaum bessere Näherungen zu erwarten und die Ergebnisse daher auf eine Art akzeptabel; das Fehlen einer geeigneten Warnung ist allerdings zweifelhaft.

5 Beurteilung numerischer Ergebnisse

Die Ergebnis-Genauigkeit einer Approximation wird man wohl am (relativen oder absoluten) Fehler in bezug auf das korrekte Ergebnis messen.[6]

Die zu erwartende (oder erhoffte) Ergebnis-Genauigkeit hängt allerdings nicht nur vom Problem, sondern auch von den Gegebenheiten ab. Eine auf die Sekunde genaue Zeitmessung wird man mit jeder Armbanduhr erzielen, aber nicht erreichen (und erwarten), wenn als Information nur der Stundenzeiger zur Verfügung steht. Insofern ist in jenem Fall eine auf die Minute genaue Zeitmessung akzeptabel, im ersten Fall nicht.

Ähnlich verhält es sich bei numerischen Problemen. Man definiert als *Konditionszahl* κ die Empfindlichkeit einer Lösung in bezug auf Änderung der Eingabedaten. Für eine Nullstelle $\hat{x} \in \mathbb{R}^n$ einer von k Parametern abhängigen Funktion $f : \mathbb{R}^{n\times k} \to \mathbb{R}^n$ also

$$\kappa(\hat{x}) := \lim_{\varepsilon\to 0} \sup \Big\{ \frac{\|x-\hat{x}\|}{\varepsilon\|\hat{x}\|} : f(x,p) = f(\hat{x},\hat{p}) = 0, \|p-\hat{p}\| \le \varepsilon\|\hat{p}\| \Big\}. \qquad (5.1)$$

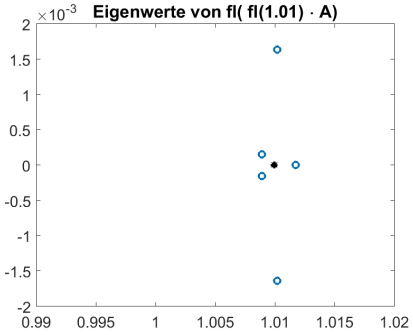
[6] Im angelsächsischen Raum diskriminiert man günstigerweise zwischen „precision“ und „accuracy“, der Rechen- und der Ergebnisgenauigkeit.

Manchmal ist die Konditionszahl a priori bekannt, z.B. sind die Eigenwerte symmetrischer Matrizen immer gut konditioniert. Es gilt [31] für $A, E \in \mathbb{R}^{n\times n}$ mit $A^T = A$ und $E^T = E$

$$\forall 1 \le i \le n: \qquad |\lambda_i(A+E) - \lambda_i(A)| \le \|E\|_2, \tag{5.2}$$

wobei $\lambda_n(A) \le \cdots \le \lambda_1(A)$ die (reellen) Eigenwerte und $\|\cdot\|_2$ die Spektralnorm bezeichnet. Das Resultat gilt unabhängig von der Größe der Störung E und gilt auch für mehrfache Eigenwerte, die Konditionszahl ist 1.

Andererseits hängen die Eigenvektoren mehrfacher Eigenwerte einer symmetrischen Matrix nicht stetig von den Eingabedaten ab: Zu einem k-fachen Eigenwert ist jeder nichttriviale Vektor eines k-dimensionalen Eigenraums X auch Eigenvektor, während jeder individuelle Vektor in X durch eine beliebig kleine Störung der Matrix zum (bis auf Normalisierung) eindeutigen Eigenvektor wird.[7] Nach Hadamard werden solche Probleme, bei denen die Lösung nicht stetig von den Eingabedaten abhängt, als *schlecht gestellt* (ill-posed) bezeichnet. Die Konditionszahl ist ∞.



Die Matrix A in Abb. 4.3 hat einen fünffachen Eigenwert mit geometrischer Vielfachheit 1. Dadurch ist der Eigenwert sehr empfindlich: Der relative Fehler aller Komponenten von A und $\tilde{A} := \mathrm{fl}\big(\mathrm{fl}(1.01)\cdot A\big)$ ist kleiner als $8.3\cdot 10^{-17}$, trotzdem sind die *tatsächlichen* Eigenwerte von $\tilde{A}$ die Kreise in nebenstehendem Bild, während $1.01\cdot A \in \mathbb{R}^{n\times n}$ den fünffachen Eigenwert 1.01 hat.

Die Matrix A in Abb. 4.3 hat absichtlich ganzzahlige Einträge, um allfällige Probleme mit Rundungen in das Gleitkommaformat $\mathbb{F}$ zu vermeiden. In der Regel ist das bei numerischen Problemen oft nicht der Fall: Die Eingabedaten sind keine Gleitkommazahlen. Das heißt, durch die Rundung in $\mathbb{F}$ entsteht üblicherweise ein kleiner Fehler, und es wird gar nicht das *tatsächliche*, sondern das in $\mathbb{F}$ *gerundete* Problem gelöst. Diese Betrachtungsweise findet sich bereits bei Turing [77, S. 306].

Ändert man aber A in der Größenordnung der relativen Rundungsfehlereinheit $\mathbf{u} = 10^{-16}$, wie es für allgemeines $A \in \mathbb{R}^{n\times n}$ durch Rundung in $\mathbb{F}^{n\times n}$ geschähe, so können die tatsächlichen Eigenwerte von der originalen 1 um bis zu 0.007 abweichen. Die numerischen Approximationen der Eigenwerte von A als auch die von A^T in Abb. 4.3 sind aber innerhalb dieser Toleranz und insofern in gewisser Weise akzeptabel.

Diese Betrachtungsweise ist die Grundlage der bereits in [52, S. 1092] erwähnten, aber erst von Wilkinson [79] vertieft betrachteten *Rückwärtsanalyse*: Wie müssen die Eingabedaten geändert werden, damit die durch einen Algorithmus berechneten Approximationen tatsächliche Lösung des perturbierten Problems sind. Weicht die notwendige (relative) Änderung nicht zu sehr von $\mathbf{u}$ ab, nennt man den Algorithmus *stabil*, die berechneten Approximationen sind im Rahmen des Bestmöglichen.

Fakt 5.1 *Moderne numerische Algorithmen sind in aller Regel numerisch stabil.*

Die Beurteilung eines Verfahrens und auch des Ergebnisses orientiert sich an den zur Verfügung stehenden Mitteln und an der Empfindlichkeit des Problems. Die Arbeitsmittel sollten eigentlich dem Problem angepaßt sein: Man wird auch den Rasen nicht mit einer Kneifzange bearbeiten wollen.

[7] Man ersetzt in der Eigenzerlegung $A = XDX^T$ einfach D durch $D+E$ für diagonales E.

Allein, in der Numerik kann man sich die Hilfsmittel aber nicht aussuchen. Es steht in aller Regel ein Computer mit viel Speicher, hoher Rechenleistung und mit doppeltgenauem Gleitkommaformat zur Verfügung; numerische Methoden versuchen eben, das Beste daraus zu machen.

Zum Nachweis der Stabilität eines numerischen Algorithmus schätzt man die Konditionszahl (5.1), also die maximale Änderung der Lösung eines perturbierten Problems ab. Das ist bei linearen Gleichungssystemen besonders einfach. Einfachheitshalber werden im folgenden nur die Matrixnormen $\|\cdot\|_p$ für $p \in \{1,2,\infty\}$ - also die Spaltensummen-, Euklid- bzw. Zeilensummennorm - oder die Frobeniusnorm betrachtet; Vektornormen werden immer als verträglich mit der benutzten Matrixnorm vorausgesetzt.

Lemma 5.2 *Seien $A,E \in \mathbb{R}^{n\times n}, x,b \in \mathbb{R}^n$, sei A nicht singulär, $Ax = b$ und $\kappa := \|A^{-1}\|\cdot\|A\|$ für eine Matrixnorm. Gilt $\kappa\varepsilon < 1$ für $0 < \varepsilon \in \mathbb{R}$, so folgt für alle $\|E\| \le \varepsilon\|A\|$ daß $A+E$ nicht singulär ist, und mit $(A+E)y = b$ gilt*[8]

$$\frac{\|y-x\|}{\|x\|} \le \frac{\kappa\varepsilon}{1-\kappa\varepsilon}. \tag{5.3}$$

Beweis. Wäre $A+E = A(I+A^{-1}E)$ singulär, so hätte $A^{-1}E$ einen Eigenwert -1, im Widerspruch zu $|-1| \le \rho(A^{-1}E) \le \|A^{-1}E\| \le \kappa\varepsilon < 1$ für den Spektralradius ρ. Wegen $A(y-x) = -Ey$ ist $\|y-x\| = \|A^{-1}Ey\| \le \kappa\varepsilon\big(\|x\| + \|y-x\|\big)$, und (5.3) folgt mit $\kappa\varepsilon < 1$. □

Bei einer relativen Störung ε der Matrix ändert sich die Lösung also maximal und i.d.R. auch tatsächlich um etwa $\kappa\varepsilon$. Aufgrund der Rundung von Eingabedaten in ein Gleitkommaformat darf man also von einem stabilen Algorithmus eine Approximation mit relativem Fehler nicht viel schlechter als $\kappa\mathbf{u}$ erwarten - aber in der Regel auch nicht viel besser.

Ein typisches Beispiel eines mathematisch korrekten, numerisch aber *nicht stabilen* Algorithmus ist die Benutzung von Normalgleichungen. Das Ausgleichsproblem bestimmt für gegebenes $A \in \mathbb{R}^{m\times n}, b \in \mathbb{R}^m$ ein $x \in \mathbb{R}^n$, das $\|Ax-b\|_2$ minimiert. Ist $m > n$ und A von vollem Rang, ist die Lösung eindeutig bestimmt und gleich der von $A^TAx = A^Tb$.

Betrachtet man die Konditionszahl in der Euklidnorm $\sqrt{\rho(A^TA)}$, so ergibt sich

$$\kappa(A^TA) = \frac{\sigma_1(A^TA)}{\sigma_n(A^TA)} = \frac{\sigma_1(A)^2}{\sigma_n(A)^2} = \kappa(A)^2,$$

wobei σ_i die Singulärwerte in nicht steigender Reihenfolge bezeichnet. Die Konditionszahl[9], der Verstärkungsfaktor, quadriert sich also!

Das läßt sich leicht an einem Beispiel demonstrieren. Die $n \times n$ Hilbert-Matrix H_n hat die Einträge $H_{ij} := (i+j-1)^{-1}$, die Konditionszahl wächst exponentiell mit der Dimension. Um Rundungsfehler zu vermeiden, wird die Matrix mit dem kleinsten gemeinsamen Vielfachen der Zahlen 1 bis $2n-1$ skaliert, und zur Herstellung eines Ausgleichsproblems noch eine $(n+1)$-te Zeile $(1,\ldots,n)$ angehängt. Für die so entstehende Matrix H_n^* wird eine rechte Seite $b := H_n^*e$ erzeugt mit $e := (1,\ldots,1)^T$. Die rechte Seite liegt also im Bild von H_n^*, und die Lösung des Ausgleichsproblems ist natürlich $(1,\ldots,1)^T$. Für $n = 8$ ist

[8] Läßt man Perturbationen $\|f\| \le \varepsilon\|b\|$ der rechten Seite zu, so kommt wg. $\|A^{-1}f\| \le \varepsilon\|A^{-1}\|\cdot\|b\| \le \kappa\varepsilon\|x\|$ auf der rechten Seite von (5.3) ein Faktor 2 hinzu.

[9] Für das Ausgleichsproblem wurde vereinfacht die Konditionszahl für die rechteckige Matrix A benutzt; genau genommen ist die Sache etwas komplizierter [10, 70].

$$A := H_8^* = \begin{pmatrix} 360360 & 180180 & ... & 45045 \\ 180180 & 120120 & ... & 40040 \\ & ... & & \\ 45045 & 40040 & ... & 24024 \\ 1 & 2 & ... & 8 \end{pmatrix} \quad \text{und} \quad b = \begin{pmatrix} 979407 \\ 659087 \\ ... \\ 261395 \\ 36 \end{pmatrix}, \tag{5.4}$$

wobei alle Komponenten von A, A^TA, b und A^Tb ohne Rundungsfehler berechnet werden. In Matlab liefert der Standard-Algorithmus (der \-Operator) als Lösung von $Au = b$ und $A^TAv = A^Tb$ jedoch

$$\tilde{u} = \begin{pmatrix} 1.000000000 \\ 1.000000000 \\ 1.000000000 \\ 1.000000002 \\ 0.999999995 \\ 1.000000006 \\ 0.999999996 \\ 1.000000001 \end{pmatrix} \quad \text{bzw.} \quad \tilde{v} = \begin{pmatrix} 1.003760327 \\ 0.831994745 \\ 2.876181603 \\ -7.844524188 \\ 22.032218345 \\ -25.573767720 \\ 18.035328777 \\ -3.360970393 \end{pmatrix}.$$

Mathematisch ist $u = v$, aber numerisch ist die Lösung über Normalgleichungen ein *instabiles* Verfahren.[10] Die Konditionszahlen $\kappa(A) \approx 10^9$ bzw. $\kappa(A^TA) \approx 10^{18}$ erklären den Unterschied zwischen $\tilde{u}$ und $\tilde{v}$: Für $\tilde{u}$ ist ein relativer Fehler von ca. $10^9\mathbf{u} \approx 10^{-7}$ zu erwarten, für $\tilde{v}$ hingegen $10^{18}\mathbf{u} \approx 10^2$, also keine richtige Dezimalstelle.

Es sei hinzugefügt, daß keine Warnung von Matlab ausgegeben wird bei der Berechnung von $\tilde{v}$. Wüßte man nichts über die Herkunft der Daten und würde die Lösung des Gleichungssystems $Bx = c$ mit $B := A^TA$ und $c = A^Tb$ numerisch berechnen, könnte aufgrund fehlender Warnung die Näherung $\tilde{v}$ für bare Münze genommen werden - ein möglicherweise fataler Fehler. Für eine sehr schöne Einführung in die numerische lineare Algebra siehe [5].

Selbst bei trivialsten mathematischen Entitäten kann ein numerischer Blick interessantes entdecken, z.B. in der „pq-Formel“. Für $p = -10^6$ und $q = 2^{-14}$, beides Elemente in $\mathbb{F}$, ergibt $x_{1,2} = -\frac{p}{2} \pm \sqrt{\frac{p^2}{4} - q}$ für die betragsmäßig kleinere Nullstelle

$$\tilde{x}_2 = 5.8208 \cdot 10^{-11} \quad \text{als Näherung für} \quad x_2 = 6.1035 \cdot 10^{-11}.$$

Der Grund ist wieder numerische Auslöschung, die für negatives p mit

$$x_1 = -\frac{p}{2} + \sqrt{\frac{p^2}{4} - q} \qquad \text{und} \qquad x_2 = \frac{q}{-\frac{p}{2} + \sqrt{\frac{p^2}{4} - q}} \tag{5.5}$$

vermieden wird (entsprechend für positives p).[11]

[10] Besser zerlegt man $A = QR$ in eine orthogonale Matrix Q und eine obere Dreiecksmatrix R. Offenbar folgt $\|Ax - b\|_2 = \|Rx - Q^Tb\|_2$, wobei die Gleichungen in $Rx = Q^Tb$ bequem durch Einsetzen der Reihe nach aufgelöst werden können (genau das macht der \-Operator).

[11] Mit (5.5) werden beide Nullstellen mit einem relativen Fehler kleiner $\mathbf{u}$ angenähert.

6 Können numerische Ungenauigkeiten vorhergesagt werden?

Leider nicht, jedenfalls nicht im Allgemeinen. Es gibt auch keine allgemeine Regel, wie arithmetische Ausdrücke wie die pq-Formel numerisch vollkommen stabil formuliert werden können. So vermeidet (5.5) zwar die Auslöschung der äußeren Summe, nicht aber in der Summe unterhalb der Wurzel. Jene ist zwar von geringer Bedeutung, aber kaum vermeidbar.

Offensichtlich kann man versuchen, mit höherer Genauigkeit (falls vorhanden) genauere Ergebnisse zu erzielen, was allerdings auch keine Gewähr für bessere Approximationen ist. Betrachten wir den Ausdruck

$$\begin{aligned} r = p^3\big(&p^{16} + 6561q^{16} - 17496p^2q^{14} + 20412p^4q^{12} - 13608p^6q^{10} \\ &+5670p^8q^8 - 1512p^{10}q^6 + 252p^{12}q^4 - 24p^{14}q^2\big) - q \qquad (6.1) \\ \text{für } p = 206987/2048 &\text{ und } q = 119504/2048. \end{aligned}$$

Tabelle 1 zeigt die berechneten Approximationen in verschiedenen Genauigkeiten, und in der letzten Zeile den auf neun Stellen korrekt gerundeten Wert. Man beachte, daß p,q und alle anderen Konstanten in (6.1) in $\mathbb{F}$ liegen, es also keine Konvertierungsfehler gibt.

Tabelle 1: Approximation von r in (6.1) in verschiedenen Genauigkeiten.

einfache Genauigkeit ($\mathbf{u} \approx 6 \cdot 10^{-8}$)	-58.3515625
doppelte Genauigkeit ($\mathbf{u} \approx 10^{-16}$)	$+1.19 \cdot 10^{24}$
20 Dezimalstellen Genauigkeit ($\mathbf{u} \approx 10^{-20}$)	$-3.63 \cdot 10^{19}$
30 Dezimalstellen Genauigkeit ($\mathbf{u} \approx 10^{-30}$)	$+1.06 \cdot 10^{9}$
40 Dezimalstellen Genauigkeit ($\mathbf{u} \approx 10^{-40}$)	-57.98
r	-58.3515625

Die in einfacher Genauigkeit berechnete Approximation ist korrekt, während in doppelter und auch mit 20 oder 30 Stellen Genaugkeit die Approximationen grob falsch sind.

Mathematisch ist $r = p^3(p^2 - 3q^2)^8 - q$. Wegen $p \approx 101$ und $q = 58.3515625$ entstehen in (6.1) Zwischenergebnisse von mindestens der Größenordnung $p^{14}q^2 \approx 4 \cdot 10^{31}$. Bei einem Endergebnis von der Größenordnung -58 kann bei weniger als 30 Dezimalstellen Rechengenauigkeit keine einzige richtige Stelle im Ergebnis erwartet werden. Das Beispiel ist so konstruiert, daß Ersetzen aller Potenzen p^3 durch einzelne Multiplikationen $p \cdot p \cdot p$ usw. qualitativ das gleiche Ergebnis liefert.

Darüber hinaus ist $t := (p^2 - 3q^2)^8 \approx 4.8 \cdot 10^{-37}$, und das Beispiel ist so konstruiert, daß sich in einfacher Genauigkeit der Kofaktor von p^3, der gleich t sein müßte, zufälligerweise zu Null auslöscht. Das Endergebnis ist damit $-q$ mit einem relativen Fehler kleiner als $|p^3t/(p^3t - q)| < 8.5 \cdot 10^{-33}$.

In den anderen Genauigkeiten löscht sich der Kofaktor nicht zu Null aus, daher die falschen Approximationen. Es ist daher eine reine Zufälligkeit, daß in einfacher Genauigkeit das korrekt gerundete Ergebnis berechnet wird.

Ebenso kann es passieren, daß in verschiedenen Genauigkeiten berechnete Approximationen zwar alle gleich, aber alle falsch sind:

$$f = 21b^2 - 2a^2 + 55b^4 - 10a^2b^2 + \frac{a}{2b} \quad \text{für} \quad a = 77617 \text{ und } b = 33096. \qquad (6.2)$$

Das Beispiel wurde öfter [13,42] analysiert. Für die Genauigkeiten in IEEE 754 ergibt sich

einfache Genauigkeit($\mathbf{u} \approx 6 \cdot 10^{-8}$)	$+1.1726040$
dopelte Genauigkeit($\mathbf{u} \approx 10^{-16}$)	$+1.172603940053179$
extended Genauigkeit($\mathbf{u} \approx 8 \cdot 10^{-25}$)	$+1.17260394005317863182387 4167317$
exakt	$-0.82739...$

Alle Ergebnisse in den in IEEE 754 zur Verfügung stehenden Genauigkeiten stimmen also überein, aber noch nicht einmal das Vorzeichen ist korrekt. Wieder ist das Beispiel so konstruiert, daß der links neben dem Bruch stehende Teil sich für alle Genauigkeiten zu Null auslöscht, der tatsächliche Wert aber -2 ist.

Fakt 6.1 *Die Durchführung einer Rechnung in verschiedenen Genauigkeiten läßt in der Regel keinen Rückschluß auf das tatsächliche Ergebnis zu.*

Zusammenfassend kann man sagen, daß, wenn man es darauf anlegt, mit derartigen Beispielen numerisch so ziemlich jeder Unsinn produziert werden kann.

7 A priori Abschätzung der Genauigkeit

Auch wenn nicht einmal das Assoziativgesetz gilt, kann der Fehler zusammengesetzter Operationen abgeschätzt werden. Betrachten wir als einfachstes Beispiel die Addition von n Gleitkommazahlen $x_i \in \mathbb{F}$. Nach (4.1) gilt für die gleitkommamäßige Summe $\tilde{s}$

$$\tilde{s} = \big(\ldots((x_1+x_2)(1+\varepsilon_1)+x_3)(1+\varepsilon_2)+\ldots+x_n\big)(1+\varepsilon_{n-1}) \quad \text{mit } |\varepsilon_i| \le \mathbf{u}.$$

Daraus folgt offenbar die seit den 1960er Jahren namentlich von Wilkinson [79] eingeführte Abschätzung

$$|\tilde{s} - \sum_{i=1}^{n} x_i| \le \big((1+\mathbf{u})^{n-1}-1\big)\mathbf{u} \sum_{i=1}^{n} |x_i|. \tag{7.1}$$

Der häßliche Faktor auf der rechten Seite wird für $(n-1)\mathbf{u} < 1$ üblicherweise [31] durch γ_{n-1} ersetzt, wobei $\gamma_k := \frac{k\mathbf{u}}{1-k\mathbf{u}}$ die Terme höherer Ordnung einbezieht.[12] Wünschenswert wäre in (7.1) ein Faktor $(n-1)\mathbf{u}$, sozusagen ein $\mathbf{u}$ für jede Operation - mehr kann man nicht erwarten.

Die Fehlerabschätzung (7.1) für die Summe ist unmittelbar klar. Bemerkenswert ist aber, daß durch die präzise Definition der Gleitkommaarithmetik weiterreichende Schlußfolgerungen möglich sind wie z.B. das bekannte Lemma von Sterbenz [50, Lemma 2, Seite 142]

$$\forall a,b \in \mathbb{F}: \qquad \frac{b}{2} \le a \le 2b \quad \Rightarrow \quad a-b \in \mathbb{F}. \tag{7.2}$$

Mit anderen Worten, falls a und b nicht allzuweit auseinander liegen, ist die gleitkommamäßige Subtraktion fehlerfrei. In [50] muß allerdings eine dreiviertel Seite für den Beweis aufgewandt werden mit vielerlei Bezug auf die Wertigkeit einzelner Bits.

Diese Art der Beweisführung, auch als Epsilontik verballhornt, ist nicht nur ziemlich unschön, sie ist auch durchaus fehleranfällig.[13] Daher werden Beweise insbesondere von der französischen Schule mittels (halb-)automatischen Beweissystemen wie COQ [2] verifiziert, z.B. in [4].

Es geht aber auch anders. Jüngst habe ich einen Kalkül entwickelt, der Beweise i.W. auf Ungleichungen zurückführt. Der Kalkül beruht auf der „unit in the first place“, der Wertigkeit des ersten Bits der binären Darstellung einer reellen Zahl:

$$\mathrm{ufp}(0) = 0 \qquad \text{und} \qquad \forall 0 \ne r \in \mathbb{R}: \ \mathrm{ufp}(r) := 2^{\lfloor \log_2(|r|) \rfloor},$$

[12] Früher wurde $1.01k\mathbf{u}$ benutzt, was zumindest für hinreichend kleines $\mathbf{u}$ und k korrekt ist.

[13] Der in [50] gelieferte Beweis ist natürlich korrekt.

so daß $\mathrm{ufp}(r) \le |r| < 2\mathrm{ufp}(r)$ für $r \neq 0$. Damit werden Gleitkommazahlen f als skalierte ganze Zahlen (U1) aufgefaßt. Aus der Definition und (2.2) folgt offenbar unter anderem:

$$f \in \mathbb{F}: \qquad f \in 2\mathbf{u} \cdot \mathrm{ufp}(f)\mathbb{Z} \tag{U1}$$

$$\ell \in \mathbb{Z},\ r \in \mathbb{R}: \qquad r \in 2^{\ell}\mathbf{u}\mathbb{Z},\ |r| \le 2^{\ell} \ \Rightarrow\ r \in \mathbb{F} \tag{U2}$$

$$r \in \mathbb{R},\ f \in \mathbb{F}: \qquad |r-f| < \mathbf{u} \cdot \mathrm{ufp}(r) \ \Rightarrow\ \mathrm{fl}(r) = f \tag{U3}$$

$$a,b \in \mathbb{F},\ f = \mathrm{fl}(a+b): \qquad f = a+b+\delta,\ |\delta| \le \mathbf{u} \cdot \mathrm{ufp}(a+b) \le \mathbf{u} \cdot \mathrm{ufp}(f) \tag{U4}$$

Der Beweis des Lemmas von Sterbenz reduziert sich damit auf zwei Zeilen: Aus der Prämisse (7.2) folgt $a,b \ge 0$ und $|a-b| \le \min(a,b) \le 2\min\big(\mathrm{ufp}(a),\mathrm{ufp}(b)\big) =: 2\sigma$. Nach (U1) ist $a,b \in 2\mathbf{u}\sigma\mathbb{Z}$ und daher $a-b \in 2\mathbf{u}\sigma\mathbb{Z}$, und (7.2) folgt aus (U2).

Mit dem Kalkül gelingt es, aus über 50 Jahre alten Standardabschätzungen die dort notwendigen höheren Terme zu eliminieren. Das sei beispielhaft an der Summation erläutert, wofür der erwähnte, wünschenswerte Faktor $(n-1)\mathbf{u}$ tatsächlich bewiesen werden kann [62]. Ein Hilfslemma wird benötigt.

Lemma 7.1 *Für $a,b \in \mathbb{F}$ gilt*

$$|fl(a+b) - (a+b)| \le |b|.$$

Beweis. Für $|(a+b)-a| < \mathbf{u} \cdot \mathrm{ufp}(a+b)$ folgt $\mathrm{fl}(a+b) = a$ aus (U3); andernfalls liefert (U4)

$$|b| = |(a+b)-a| \ge \mathbf{u} \cdot \mathrm{ufp}(a+b) \ge |\mathrm{fl}(a+b) - (a+b)|. \qquad \square$$

Lemma 7.2 *Für $x_1,\ldots,x_n \in \mathbb{F}$ seien $\tilde{s}_1 := x_1$, $s_k := \tilde{s}_{k-1} + x_k$ und $\tilde{s}_k := fl(s_k)$ für $2 \le k \le n$. Dann gilt*

$$|\tilde{s}_n - \sum_{i=1}^{n} x_i| \le (n-1)\mathbf{u} \sum_{i=1}^{n} |x_i|. \tag{7.3}$$

Beweis[14] durch vollständige Induktion. Der Induktionsanfang ist trivial, und

$$\Delta := |\ \tilde{s}_n - \sum_{i=1}^{n} x_i\ | = |\tilde{s}_n - s_n + \tilde{s}_{n-1} - \sum_{i=1}^{n-1} x_i| \le |\tilde{s}_n - s_n| + (n-2)\mathbf{u} \sum_{i=1}^{n-1} |x_i|\ . \tag{7.4}$$

Angenommen, $|x_n| \le \mathbf{u}\sum_{i=1}^{n-1} |x_i|$. Dann folgt aus Lemma 7.1

$$|\tilde{s}_n - s_n| = |\mathrm{fl}(\tilde{s}_{n-1}+x_n) - (\tilde{s}_{n-1}+x_n)| \le |x_n| \le \mathbf{u} \sum_{i=1}^{n-1} |x_i|\ ,$$

und (7.4) liefert die Behauptung. Andernfalls ergibt (U4)

$$|\tilde{s}_n - s_n| \le \mathbf{u} \cdot \mathrm{ufp}(s_n) \le \mathbf{u}|s_n| = \mathbf{u}|\tilde{s}_{n-1} - \sum_{i=1}^{n-1} x_i + \sum_{i=1}^{n} x_i|\ ,$$

so daß mit (7.4)

$$\begin{aligned} \Delta &\le \mathbf{u}\big[\ (n-2)\mathbf{u}\textstyle\sum_{i=1}^{n-1} |x_i| + \sum_{i=1}^{n} |x_i|\ \big] + (n-2)\mathbf{u}\textstyle\sum_{i=1}^{n-1} |x_i| \\ &< \mathbf{u}\big[\ (n-2)|x_n| + |x_n| + \textstyle\sum_{i=1}^{n-1} |x_i|\ \big] + (n-2)\mathbf{u}\textstyle\sum_{i=1}^{n-1} |x_i| \\ &= (n-1)\mathbf{u}\textstyle\sum_{i=1}^{n} |x_i| \end{aligned}$$

[14] Lemma 7.2 ist für sukzessive Summation formuliert, gilt aber [36] für beliebige Reihenfolgen der Summation.

ebenso die Behauptung (7.3) folgt. □

Der ufp-Kalkül [62] umfaßt nur wenige, grundlegende Eigenschaften des Gleitkommaformats (2.2), die Beweisführungen reduzieren (wie oben) sich auf die Anwendung von Ungleichungen.

Standardabschätzungen für den Fehler von Summen, Skalarprodukten, *LU*-Zerlegung, Cholesky-Zerlegung und dergleichen mehr benutzten jeher [79,71,72,31] einen Faktor $\gamma_k = \frac{k\mathbf{u}}{1-k\mathbf{u}}$, um höhere Terme abzufangen.

In all diesen Fällen [62,36,64] können die Faktoren mithilfe des ufp-Kalküls, unabhängig von der Reihenfolge von Summationen und ohne Beschränkung von n, durch den einfachen Faktor $k\mathbf{u}$ ersetzt werden, also zum Beispiel

$$|\tilde{L}\tilde{U} - A| \leq n\mathbf{u}|\tilde{L}||\tilde{U}| \qquad \text{oder} \qquad |\tilde{G}\tilde{G}^T - A| \leq (n+1)\mathbf{u}|\tilde{G}^T||\tilde{G}| \tag{7.5}$$

für die *berechneten LU*-Faktoren $\tilde{L},\tilde{U}$ oder den Cholesky-Faktor $\tilde{G}$ einer $n \times n$-Matrix A, und so weiter. Das ist auch von praktischer Bedeutung, vornehmlich eher eine Frage der Schönheit.

Fakt 7.3 *Obwohl elementare Gesetze wie das Assoziativgesetz nicht erfüllt sind, können sehr wohl mathematisch korrekte Fehlerabschätzungen hergeleitet werden. Diese sind zwar beinahe scharf, doch in der Regel pessimistisch.*

Für die Beurteilung der Approximation der Lösung nutzt ein kleines Residuum $|\tilde{L}\tilde{U} - A|$ allerdings wenig, außerdem ist der wahre Faktor in der Regel deutlich kleiner als $n\mathbf{u}$.

Für $B = A^T A$ mit der Matrix A aus (5.4) etwa gilt $|LU - B| \leq 1.12\mathbf{u}|LU|$, d.h. der relative Fehler zwischen LU und B entspricht dem relativen Rundungsfehler. Dieser Fehler multipliziert sich nach Lemma 5.2 jedoch mit der Konditionszahl $\kappa(B) \approx 10^{18}$; die Approximation ist also, wie nach (5.4) gezeigt, wertlos.

Das gilt nicht immer. Auch wenn die Cholesky-Zerlegung ausschließlich in Gleitkommaarithmetik durchgeführt wird, können trotz Fakt 7.3 korrekte und gute Fehlerschranken für die Lösung eines Gleichungssystems berechnet werden, siehe Kapitel 17.

8 Fehlerfreie Gleitkommaoperationen

Interessanterweise kann man sich die präzise Definition der Gleitkommaarithmetik zunutze machen und fehlerfrei rechnen. Es seien $a,b \in \mathbb{F}$, dann ist mit $x := a \tilde{+} b$ der Fehler $\delta := a+b-x$ in $\mathbb{F}$ betragsmäßig minimal. Man kann zeigen [41,50], daß nicht nur immer $\delta \in \mathbb{F}$ gilt, sondern darüber hinaus der Fehler δ mittels Gleitkommaoperationen berechenbar ist:[15]

Lemma 8.1 *Für $a,b \in \mathbb{F}$ mit $|a| \geq |b|$ und*

$$x := a \tilde{+} b, \quad y := (a \tilde{-} x) \tilde{+} b \tag{8.1}$$

gilt $a+b = x+y$.

Das gleiche Prinzip gilt ebenso für Multiplikation, Division und Quadratwurzel, wobei für $x := a \tilde{\cdot} b$, $x := a \tilde{/} b$ und $x := \mathrm{fl}(\sqrt{a})$ jeweils y so berechnet wird, daß $ab = x+y$, $a = bx+y$ bzw. $a = x^2+y$ gilt [14]. In jenen Fällen ist allerdings möglicher Über- und Unterlauf zu berücksichtigen.

[15] Das gilt auch bei beschränktem Exponentenbereich, da es keinen Überlauf geben kann; im Unterlaufbereich ist die Addition und Subtraktion fehlerfrei.

Im erweiterten IEEE-Standard [34] von 2008 ist die Berechnung von y besonders einfach, da eine FMA-Operation („*fused multiply-and-add*“) eingeführt wurde: Es wird $x := FMA(a,b,c) = \mathrm{fl}(ab+c)$ für $a,b,c \in \mathbb{F}$ mit einer Rundung berechnet. In Beispiel (4.2) würde $FMA(-g,h,f)$ also das korrekte Ergebnis $\mathrm{fl}(f-gh) = 3$ liefern. Für die fehlerfreien Transformationen von Multiplikation, Division und Quadratwurzel liefert $FMA(a,b,-x)$, $FMA(-b,x,a)$ bzw. $FMA(-x,x,a)$ das korrekte Ergebnis für y.

Fakt 8.2 *Die vier Grundrechnungsarten und die Quadratwurzel können unter ausschließlicher Verwendung von Gleitkommaarithmetik fehlerfrei ausgeführt werden.*

Die FMA-Operation ist bereits auf einigen, aber nicht allen Rechnern verfügbar. Nachdem der erste IEEE-Standard [33] in kurzer Zeit auf praktisch allen Rechnern implementiert wurde, bleibt zu hoffen, daß dies auch bald für die FMA-Operation der Fall sein wird. Im übrigen kann man diese auch wieder mit den üblichen Gleitkommaoperationen simulieren.

Mit diesem Prinzip können zwar die vier Grundrechnungsarten und die Quadratwurzel fehlerfrei ausgeführt werden, für zusammengesetzte Operationen müßte man allerdings mit Paaren von Gleitkommazahlen rechnen können. Das wird in [16,83] auch getan. Die Arithmetik auf Paaren ist zwar nicht mehr fehlerfrei, immerhin hat man damit aber die (Rechen-)Genauigkeit quasi verdoppelt. Damit erhöht sich in der Regel auch die Ergebnisgenauigkeit, muß aber nicht, wie die Beispiele (6.1) oder (6.2) zeigen.

Man kann den Prozeß iterieren und insbesondere für Skalarprodukte praktisch beliebig genaue Ergebnisse berechnen [51,54]. Das hat u.a. Anwendungen in der Bildverarbeitung [67], wo das Vorzeichen eines Skalarproduktes darüber entscheidet, ob ein Punkt sichtbar ist oder nicht. Wiederum sind Skalarprodukte ein wichtiger Schritt in Richtung korrekte Ergebnisse, allerdings kein Allheilmittel.

9 Praktische Kompromisse

Einer der am häufigsten benutzten numerischen Algorithmen überhaupt ist die Gauß-Elimination mit partieller Pivotisierung zur Lösung allgemeiner linearer Gleichungssysteme. Interessanterweise ist dies *kein* stabiler Algorithmus im Sinne von Kapitel 5. Das macht man sich wie folgt klar.

Wie in den vorigen Beispielen (5.5) oder (6.1) numerische Auslöschung die Hauptursache für numerischen Schwierigkeiten war, so kann man eine Art Faustregel aufstellen, daß zumindest die Gefahr einer falschen Näherung besteht, wenn während der Ausführung eines Algorithmus betragsmäßig große Zahlen vorkommen bei vergleichsweise kleinem Endergebnis.

Bei der Gauß-Elimination mißt man das mit dem „growth factor“ [31]

$$g(A) := \frac{\max_{i,j,k} |A_{ij}^{(k)}|}{\max_{i,j} |A_{ij}|}, \tag{9.1}$$

wobei $A_{ij}^{(k)}$ die Matrixelemente im k-ten Eliminationsschritt bezeichnet. Aufgabe der Pivotisierung ist es, $g(A)$ klein, möglichst in der Nähe von 1 zu halten. Das ist bei partieller Pivotisierung in aller Regel auch der Fall, während die bekannten Ausnahmen ausschließlich konstruierte, akademische Beispiele waren.

Wright [82] gab 1992 allerdings eine Reihe praktischer Beispiele an (siehe auch [30]), für die die Gauß-Elimination verheerende Approximationen berechnet, obwohl die Matrizen bestens konditioniert sind.

Diskretisiert man nach Foster [19] z.B. das Randwertproblem $\dot{x} = x - 1$ mit $x(0) = x(T)$ auf $[0,T]$ mit der Trapezregel, so entsteht für 70 Gitterpunkte ein lineares Gleichungssystem mit einer Konditionszahl $\kappa(A) \approx 30$. Bei Rechnung in doppelter Genauigkeit können also etwa 14 korrekte Dezimalstellen erwartet werden.

Matlab liefert mittels der Gauß-Elimination einen „Lösungsvektor“ $\tilde{x}$, dessen letzte 10 Komponenten $(\ldots, 0.7041, 1.4082, 0, 0, 0, 0, -22.5306, 0, 0, 1)$ sind, die exakte Lösung ist aber offenbar die Konstante 1. Die Ursache ist der growth factor von $g(A) \approx 5.4 \cdot 10^{17}$, trotz der Konditionszahl $\kappa(A) \approx 30$.

Es gibt eine Reihe von alternativen Pivotisierungstechniken, die das Problem lösen wie totale Pivotisierung, bei der das betragsmäßige Maximum nicht nur in der Pivotspalte, sondern im gesamten verbleibenden Block gesucht wird, oder auch „rook“-pivoting, siehe [31].

Allerdings erfordern diese Methoden zum Teil einen erheblichen Mehraufwand an Rechenzeit. Da die Problemfälle praktisch äußerst selten beobachtet werden, benutzt man weiterhin die potentiell instabile Gauß-Elimination mit partieller Pivotisierung.

Im betrachteten Beispiel könnte mit kaum Mehraufwand Abhilfe geschaffen werden: Für ein gegebenes Gleichungssystem $Ax = b$ sei $\tilde{x}$ die mittels Gauß-Elimination berechnete Näherung, und d die ebenso ermittelte Näherungslösung von $Ad = b - A\tilde{x}$. Skeel [69] zeigte 1980, daß eine einzige Residueniteration $\tilde{x} \to \tilde{x} + d$ eine rückwärts stabile Näherung liefert, auch wenn das Residuum in gleicher Rechengenauigkeit berechnet wird. In der Tat werden im obigen Beispiel so etwa 14 korrekte Dezimalstellen in jeder Lösungskomponente erzielt.

10 Vorwärtsfehler

Rückwärts stabil bedeutet „nur“, daß eine berechnete Näherung $\tilde{x}$ die tatsächliche Lösung eines leicht gestörten Gleichungssystems ist, also $(A + \Delta A)\tilde{x} = b$ für $\|\Delta A\| \approx \mathbf{u}\|A\|$. Wie besprochen ist das das Beste, was man mit den zur Verfügung stehenden Mitteln, d.h. mittels Gleitkommaarithmetik, erreichen kann.

Trotzdem ist nicht nur praktisch sondern auch mathematisch der „Vorwärtsfehler“ interessant, also etwa $r := \|A^{-1}b - \tilde{x}\|_\infty$, der für jede Lösungskomponente $|\left(A^{-1}b\right)_i - \tilde{x}_i| \leq r$ impliziert. Die Berechnung von r setzt voraus, daß A nicht singulär ist.

Nota bene ist das für den Rückwärtsfehler nicht notwendig, und in der Tat kann eine singuläre Matrix numerisch problemlos „invertiert“ werden - allerdings, so weit bekannt, immer mit einer Warnung verbunden.

In den Anfangszeiten des numerischen Rechnens wurde viel über den Vorwärtsfehler nachgedacht. In einer viel beachteten, 80-seitigen Arbeit [52] untersuchten v. Neumann und Goldstine 1947 die prinzipielle numerische Lösbarkeit linearer Gleichungssysteme. Sie kamen zu dem ernüchternden Schluß, daß zum Beispiel in einer 24-Bit Festpunktarithmetik (entsprechend einfacher Genauigkeit nach IEEE 754) wohl nur Gleichungssysteme bis zu einer Dimension $n \leq 9$ (sic!) mit akzeptabler Ergebnisgenauigkeit gelöst werden können.

Damals untersuchte man nur den Vorwärtsfehler, und tatsächlich ergibt die Hintereinanderreihung der Abschätzungen (4.1) ein sehr düsteres Bild. Das setzt allerdings implizit voraus, Rundungsfehler seien mehr oder wenig zufällig. In Wirklichkeit sind sie aber außerordentlich korreliert.

Fakt 10.1 *Rundungsfehler sind in der Regel nicht unabhängig.*

Dieser Sachverhalt wurde in [76] für die Gauß-Elimination analysiert. Insbesondere spielt eine Rolle, daß die Korrekturen in jedem Eliminationsschritt vom Rang 1 sind und daher die Vorzeichen der Korrektur nicht unabhängig sind.

Außerdem kann an einem einfachen praktischen Beispiel gezeigt werden, daß eine Näherungsinverse einer sehr schlecht konditionierten Matrix nutzbare Information enthält. Die bereits betrachtete skalierte Hilbert-Matrix ist für $n = 15$ mit $A_{ij} = c/(i+j-1)$ und $c := \mathrm{kgV}(1,\ldots,2n-1) = 2329089562800$ exakt darstellbar in $\mathbb{F}$ und hat eine Konditionszahl von $\kappa(A) \approx 6 \cdot 10^{20}$. Bei Rechnung in doppelter Genauigkeit wird man von einer Näherungsinversen R keine richtige Stelle erwarten können. In der Tat liefert Matlab

$$R = \begin{pmatrix} 7.0\cdot 10^{-11} & -5.7\cdot 10^{-9} & \ldots & +1.1\cdot 10^{-4} & -1.8\cdot 10^{-5} \\ -5.7\cdot 10^{-9} & 6.3\cdot 10^{-7} & \ldots & -2.2\cdot 10^{-2} & +3.6\cdot 10^{-3} \\ \ldots & & & \ldots & \\ +1.1\cdot 10^{-4} & -0.0221 & \ldots & 728.3 & -356.2 \\ -1.8\cdot 10^{-5} & +3.6\cdot 10^{-3} & \ldots & -356.2 & 96.84 \end{pmatrix} \tag{10.1}$$

während (alle Einträge auf zwei bis drei Stellen gerundet)

$$A^{-1} = \begin{pmatrix} 9.7\cdot 10^{-11} & -1.1\cdot 10^{-8} & \ldots & -3.6\cdot 10^{-3} & +5.0\cdot 10^{-4} \\ -1.1\cdot 10^{-8} & 1.6\cdot 10^{-6} & \ldots & +0.756 & -0.105 \\ \ldots & & & \ldots & \\ -3.6\cdot 10^{-3} & +0.756 & \ldots & 9.8\cdot 10^{5} & -1.4\cdot 10^{5} \\ +5.0\cdot 10^{-4} & -0.105 & \ldots & -1.4\cdot 10^{5} & 2.0\cdot 10^{4} \end{pmatrix} \tag{10.2}$$

Für mehr als die Hälfte der Komponenten von R ist nicht einmal das Vorzeichen korrekt. Für $P := \mathrm{fl}(RA)$ sei Q die in doppelter Genauigkeit berechnete Näherungsinverse von $P \in \mathbb{F}^{n\times n}$.

Ohne Rundungsfehler wäre $QR = A^{-1}$, mit Rundungsfehler stimmen $\mathrm{fl}(QR)$ und A^{-1} immerhin auf mindestens 7 Dezimalstellen in allen Komponenten überein, d.h., in dem an sich unbrauchbaren R steckt einiges an Information. Ein auf dieser Beobachtung basierendes Verfahren wurde in [60] vorgestellt und analysiert.

11 Gaußsche Fehlerrechnung

In der Gaußschen Fehlerrechnung [21,73] wird von fehlerbehafteten Größen $A := a \pm \Delta a$ ausgegangen, wobei der Fehler Δa als so klein gegenüber a angenommen wird, daß quadratische Fehler vernachlässigt werden können. Es folgt

$$\begin{aligned} A+B &= a+b \pm (\Delta a + \Delta b), & AB &= ab \pm (\Delta a|b| + |a|\Delta b), \\ A-B &= a-b \pm (\Delta a + \Delta b), & A/B &= a/b \pm (\Delta a|b| + |a|\Delta b)/b^2, \end{aligned} \tag{11.1}$$

ähnlich den Regeln für Differentiation. Offenbar addieren sich die absoluten Fehler für die Addition und Subtraktion, während sich nach

$$\frac{\Delta(AB)}{|ab|} = \frac{\Delta a}{|a|} + \frac{\Delta b}{|b|} = \frac{\Delta(A/B)}{|a/b|}$$

für Multiplikation und Division die relativen Fehler addieren. Ist das Ergebnis einer Addition oder Subtraktion von gleicher Größenordnung wie die Operanden, bedeutet das ebenfalls einen kleinen relativen Fehler; bei Auslöschung jedoch, betragsmäßig kleinem Ergebnis gegenüber den Operanden, kann der relative Fehler wie bereits festgestellt groß werden.

Im 19. Jahrhundert wurde diese Art der Fehlerrechnung und -abschätzung ausgiebig benutzt [66], um Ergebnisse abzusichern. Um mathematisch korrekte Resultate zu erzielen, werden wir die quadratischen Fehler auch noch berücksichtigen.

12 Gerichtete Rundung

In Kapitel 2 wurde naheliegenderweise die Rundung als bestmögliche Approximation definiert, die, bis auf die Mittelpunkte aufeinanderfolgender Elemente in $\mathbb{F}$, eindeutig ist. Darüber hinaus werden gerichtete Rundungen $\mathrm{fl}_\nabla, \mathrm{fl}_\Delta : \mathbb{R} \to \mathbb{F}$ definiert mit

$$\forall r \in \mathbb{R}: \quad \mathrm{fl}_\nabla(r) := \max\{f \in \mathbb{F} : f \le r\} \quad \text{und} \quad \mathrm{fl}_\Delta(r) := \min\{f \in \mathbb{F} : r \le f\}. \qquad (12.1)$$

Beide Rundungen sind im IEEE 754 Arithmetik Standard [33,34] enthalten und heute auf praktisch jedem Rechner verfügbar. Aus der Definition folgt

Lemma 12.1 *Für $r \in \mathbb{R}$ gilt $r \in \mathbb{F} \Leftrightarrow fl_\nabla(r) = fl_\Delta(r)$.*

Wie in (2.1) für die Rundung zum nächstgelegenen Element in $\mathbb{F}$, definiert man für $\circ \in \{+,-,\cdot,/\}$ Gleitkommaoperationen mithilfe der gerichteten Rundungen, also für $f,g \in \mathbb{F}$:

$$f \,\tilde{\circ}_\nabla\, g := \mathrm{fl}_\nabla(f \circ g) \quad \text{und} \quad f \,\tilde{\circ}_\Delta\, g := \mathrm{fl}_\Delta(f \circ g).$$

Es folgt für $\circ \in \{+,-,\cdot,/\}$ die bemerkenswerte Eigenschaft

$$\forall f,g \in \mathbb{F}: \qquad f \,\tilde{\circ}_\nabla\, g \;\le\; f \circ g \;\le\; f \,\tilde{\circ}_\Delta\, g, \qquad (12.2)$$

und auch $f \circ g \in \mathbb{F} \Leftrightarrow f \,\tilde{\circ}_\nabla\, g = f \,\tilde{\circ}_\Delta\, g$. Das heißt, für alle elementaren Operationen sind bestmögliche Schranken auf praktisch jedem Computer direkt berechenbar.

In der Regel werden die gerichteten Rundungen so realisiert, daß in einem Kontrollwort eine Information hinterlassen wird die bewirkt, daß fortan *jede* Operation mit der spezifizierten Rundung ausgeführt wird.

Das macht eine mathematische Beschreibung schwerfällig, und es scheint am einfachsten, kurze Programmstücke zu verwenden. Dadurch wird auch klar, daß Gleitkomma- und keine reellen Operationen verwendet werden. In folgendem, in Matlab und Octave ausführbaren INTLAB-Code [59]

```
setround(-1), qinf = 15/11,  xinf = 11*qinf
setround(0),  q = 15/11,     x = 11*q
setround(+1), qsup = 15/11,  xsup = 11*qsup
```

schaltet `setround(rnd)` für `rnd` $= -1,0,1$ die Rundung nach unten, zum nächstgelegenen Element in $\mathbb{F}$ bzw. nach oben um.

Mathematisch ist `qinf` $= 15 \,\tilde{/}_\nabla\, 11$, `xinf` $= 11 \,\tilde{\cdot}_\nabla$ `qinf` und so weiter. Da $15/11 \notin \mathbb{F}$ sind die berechneten Werte `qinf` und `qsup` die beiden Nachbarn von $15/11$ in $\mathbb{F}$, und `q` ist gleich dem nähergelegenen Nachbarn.

Aus `qinf` $< 15/11 <$ `qsup` folgt außerdem `xinf` $< 15 <$ `xsup`, aber nicht `x` $= 15$; im vorliegenden Beispiel ist `x` $=$ `xinf` gleich dem Vorgänger von 15.

Fakt 12.2 *Für die vier Grundrechnungsarten $\circ \in \{+,-,\cdot,/\}$ und $a,b \in \mathbb{F}$ berechnen gerichtete Rundungen das engstmögliche Intervall $[f_1,f_2]$ mit $f_\nu \in \mathbb{F}$ und $a \circ b \in [f_1,f_2]$. Das Ergebnis $a \circ b$ der reellen Operation ist genau dann eine Gleitkommazahl, wenn $f_1 = f_2$ ist.*

13 Korrekte Fehlerschranken mittels Gleitkommaarithmetik

Nach dem Befehl `setround(−1)` bzw. `setround(1)` werden wie gesagt *alle* Operationen mit Rundung nach unten bzw. nach oben ausgeführt. Wendet man das nacheinander auf die Operationen eines Skalarproduktes an, so folgt für Vektoren $x, y \in \mathbb{F}^n$ die bemerkenswerte Eigenschaft, daß

```
setround(-1), sinf = x'*y;
setround(1),  ssup = x'*y;
```

Gleitkommazahlen $\texttt{sinf}, \texttt{ssup} \in \mathbb{F}$ mit $\texttt{sinf} \le x^T y \le \texttt{ssup}$ berechnet. Aus `sinf=ssup` folgt $x^T y \in \mathbb{F}$, aber nicht umgekehrt. Das Prinzip ist ebenso auf Matrizen anwendbar. Für $A, B, C \in \mathbb{F}^{n\times n}$ sei

```
setround(-1), Sinf = A*B - C;
setround(1),  Ssup = A*B - C;
```

Für $n = 1$ folgt aus den gerichteten Rundungen

$$\begin{aligned}\texttt{Sinf} &= \mathrm{fl}_\nabla\big(\mathrm{fl}_\nabla(AB) - C\big) \le \mathrm{fl}_\nabla(AB) - C \le AB - C \\ &\le \mathrm{fl}_\Delta(AB) - C \le \mathrm{fl}_\Delta\big(\mathrm{fl}_\Delta(AB) - C\big) = \texttt{Ssup}.\end{aligned}$$

Das Argument ist für elementweisen Vergleich ebenso für $n > 1$ anzuwenden, also $\texttt{Sinf} \le AB - C \le \texttt{Ssup}$ für Matrizen A, B, C. Man beachte, daß der Schluß auf $C - AB$ in dieser Weise nicht anwendbar ist.

Damit kann man bereits die Regularität einer Matrix $A \in \mathbb{F}^{n\times n}$ mit folgenden Matlab/Octave Anweisungen allein mittels Gleitkommaoperationen nachweisen. Hierbei bezeichnet `eye(n)` die $n \times n$-Einheitsmatrix, `inv(A)` berechnet eine Näherungsinverse von A, `abs(A)` ist die Matrix der Absolutbeträge, das Maximum zweier Matrizen ist die Matrix der Maxima und `sum(A)` ist der Zeilenvektor der Spaltensummen. Ohne Rundungsfehler ist `max(sum(A))` also gleich der Spaltensummennorm $\|A\|_1$ von A.

```
R = inv(A);  I = eye(n);
setround(-1), Cinf = R*A - I;
setround(1),  Csup = R*A - I;
C = max( abs(Cinf) , abs(Csup) );
normC = max(sum(C));
```

Lemma 13.1 *Aus* `normC` < 1 *für* $\texttt{A} \in \mathbb{F}^{n\times n}$ *folgt* $det(\texttt{A}) \neq 0$.

Beweis. Aus der jeweiligen Rundung folgt $\texttt{Cinf} \le R\texttt{A} - I \le \texttt{Csup}$. Da Absolutbetrag und Maximum fehlerfrei sind, folgt $|R\texttt{A} - I| \le C$ und, wegen der Rundung nach oben, $\|C\|_1 \le \texttt{normC}$. Aus $\texttt{normC} < 1$ folgt also $\|R\texttt{A} - I\|_1 < 1$ und damit die Regularität von `A` (und `R`). □

`Bemerkung 1.` Es gibt keine Voraussetzungen an die Güte der Näherungsinversen `R`. Die Aussage ist für jedes `R` korrekt; für zu 'schlechtes' `R` wird die Voraussetzung $\texttt{normC} < 1$ allerdings nicht erfüllt sein.

`Bemerkung 2.` Für $\texttt{normC} \ge 1$ ist keine Aussage über die Regularität möglich.

Wendet man die Methode auf die skalierte Hilbert-Matrix an, so ergibt sich $\texttt{normC} < 0.69$ für $n = 12$, was die Regularität der (tatsächlichen) Hilbert-Matrix nachweist. Dabei reagiert Matlab übrigens übervorsichtig, gibt eine Warnung bei der Berechnung von `inv(A)` aus und schätzt die Konditionszahl auf $\kappa(A) \approx 4.1 \cdot 10^{16}$; tatsächlich gilt $\kappa(A) \approx 1.8 \cdot 10^{16}$.

Für $n = 13$ ist $\texttt{normC} > 10$, daher erlaubt das Verfahren keine Aussage über die Regularität der 13×13 Hilbert-Matrix.

Der oben skizzierte Algorithmus ist eine triviale Version eines *Verifikationsverfahrens*, d.h. eines Verfahrens, das mathematisch korrekte Aussagen mittels Gleitkommaarithmetik berechnet. Es gilt dabei, die Kluft zwischen hinreichend und notwendig, also die Fälle, in denen zwar $det(A) \neq 0$ aber $\texttt{normC} \geq 1$ ist, so gering wie möglich zu halten.

Fakt 13.2 *Die mathematische Korrektheit einer Verifikationsmethode gilt unter der Voraussetzung, daß die Arithmetik, der Compiler, das Betriebssystem und so weiter gemäß ihren Spezifikationen korrekt implementiert sind.*

Der klassische mathematische Beweis wird auf dem Papier geführt, und in der reinen Form ist er ohne Hilfsmittel nachvollziehbar. Allerdings würde man heute wohl kaum die Faktorisierung von $2^{67} - 1 = 147573952589676412927 = 761838257287 \cdot 193707721$ per Hand an der Tafel vorrechnen wie Frank Cole 1903 auf der Jahrestagung der American Mathematical Society [11].

Die Akzeptanz von Hilfsmitteln ist von deren Komplexität abhängig: Je einfacher, desto besser. Die Benutzung von Gleitkommaoperationen ist zwar nicht eben kompliziert, steht aber gemeinhin am unteren Ende der Akzeptanzskala; ein Taschenrechner benutzt zwar auch genäherte Operationen, wäre aber wohl eher hoffähig.

In Kapitel 2 wurde jedoch gezeigt, daß gerade Taschenrechner bis zum heutigen Tag fatal fehlerhaft sind, vielleicht ja, weil sich niemand recht dafür interessiert. Auch reine Integer-Arithmetik (ebd.) sollte der Spezifikation gemäß benutzt werden.

Die Gleitkommaarithmetik ist nach dem IEEE 754 Standard präzis definiert und für Verifikationsmethoden brauchbar. In der Frühzeit machte diese zwar auch mit Fehlern Furore [45], heute kann man zumindest durch millionenfaches Testen von einer gewissen Sicherheit ausgehen. Allein, die Voraussetzung in Fakt 13.2 bleibt.

14 Toleranzbehaftete Eingabedaten

Bis jetzt wurden als Eingabedaten nur Gleitkommazahlen benutzt und zum Beispiel die Regularität der skalierten Hilbert-Matrix nachgewiesen. Für den Nachweis der Nicht-Singularität einer originalen Hilbert-Matrix H mit $H_{ij} := 1/(i+j-1)$ ist diese Methode für $n \geq 2$ nicht anwendbar, da etwa $1/3 \notin \mathbb{F}$. Man könnte allerdings mit

```
setround(-1), for i=1:n, for j=1:n, Hinf(i,j) = 1/(i+j-1); end, end
setround(1),  for i=1:n, for j=1:n, Hsup(i,j) = 1/(i+j-1); end, end
```

Matrizen $\texttt{Hinf}, \texttt{Hsup} \in \mathbb{F}^{n \times n}$ berechnen, die $\texttt{Hinf} \leq H \leq \texttt{Hsup}$ (mit komponentenweisem $\leq$) erfüllen.[16] Mit geeigneten Fehlerabschätzungen kann so die Regularität *jeder* Matrix A mit $\texttt{Hinf} \leq A \leq \texttt{Hsup}$ verifiziert werden, insbesondere die von H.

Jene „geeigneten Fehlerabschätzungen“ sind im Grunde trivial, allerdings wäre es ziemlich mühselig, diese immer wieder neu durchzuführen.

Eine einfache Abhilfe schafft das Rechnen mit Intervallen, was jedoch einer Vorrede bedarf. Die sogenannte Intervallrechnung oder gar „Intervallmathematik“ ist in bösen Verruf geraten. Im nächsten Kapitel wird darauf eingegangen, zunächst aber erklärt, worum es eigentlich geht.[17]

[16] Hierbei wird benutzt, daß $i + j - 1 \in \mathbb{F}$ gilt, jedenfalls für praktische Werte von n.

[17] Mir sind schon mehrfach Aussagen des Kalibers „Ich weiß nicht, worum es bei Intervallrechnung genau geht, aber ich halte nichts davon.“ begegnet.

Für reelle Intervalle $A := [\underline{a}, \overline{a}]$ und $B := [\underline{b}, \overline{b}]$ und $\circ \in \{+, -, \cdot, /\}$ sei $v := [\underline{a} \circ \underline{b}, \underline{a} \circ \overline{b}, \overline{a} \circ \underline{b}, \overline{a} \circ \overline{b}] \in \mathbb{R}^4$. Dann folgt für $P := [\min(v), \max(v)]$ offenbar

$$P = \{a \circ b : a \in A, b \in B\} \in \mathbb{IR} := \{[\underline{a}, \overline{a}] : \underline{a}, \overline{a} \in \mathbb{R}, \underline{a} \le \overline{a}\},$$

vorausgesetzt $0 \notin B$ für die Division. Das Einschließungsintervall $P \in \mathbb{IR}$ ist bestmöglich.

Um Schranken mittels Gleitkommaarithmetik berechnen zu können, sei $\mathbb{IF} := \{[\underline{a}, \overline{a}] : \underline{a}, \overline{a} \in \mathbb{F}, \underline{a} \le \overline{a}\}$. Ist $A, B \in \mathbb{IF}$ und bezeichnet $\underline{v}, \overline{v}$ den Vektor v, wobei die Operationen mit Rundung nach unten bzw. oben ausgeführt werden, so gilt für $Q := [\min(\underline{v}), \max(\overline{v})]$

$$Q = \bigcap \{X \in \mathbb{IF} : a \circ b \in X \ \forall a \in A, b \in B\} \in \mathbb{IF}. \tag{14.1}$$

Die Einschließung $Q \in \mathbb{IF}$ ist also wieder bestmöglich. Für die praktische Implementierung geht man effizienter vor und definiert etwa $A + B = [\mathrm{fl}_\nabla(\underline{a} + \underline{b}), \mathrm{fl}_\Delta(\overline{a} + \overline{b})]$, $A - B = [\mathrm{fl}_\nabla(\underline{a} - \overline{b}), \mathrm{fl}_\Delta(\overline{a} - \underline{b})]$, und Multiplikation und Division mit einigen Fallunterscheidungen. Man beachte, daß $[\underline{a}, \overline{a}] \in \mathbb{IF}$ alle *reellen* Zahlen x mit $\underline{a} \le x \le \overline{a}$ enthält.

Die Menge der Intervallvektoren ist das Kartesische Produkt $(\mathbb{IF})^n$. Für Addition, Subtraktion und Skalarprodukt von Intervallvektoren werden die reellen Operationen durch die entsprechenden Intervalloperationen ersetzt. Nimmt man alle Komponenten $[\underline{a_i}, \overline{a_i}], [\underline{b_i}, \overline{b_i}]$ als unabhängig voneinander an, so ist das Ergebnis gemäß (14.1) wiederum bestmöglich.

Schließlich ist die Menge der $m \times n$ Intervallmatrizen das Kartesische Produkt $(\mathbb{IF})^{m \times n}$, und Addition, Subtraktion und Multiplikation werden wieder definiert, indem die reellen Operationen durch die entsprechenden Intervalloperationen ersetzt werden. Man beachte, daß wieder, Unabhängigkeit aller Intervallkomponenten vorausgesetzt, das Resultat gemäß (14.1) bestmöglich ist.

Praktisch gesehen definiert man einen Intervall-Datentyp so, daß, gleichgültig ob Skalar, Vektor oder Matrix, die oben beschriebenen Intervalloperationen immer verwandt werden, wenn mindestens ein Operand ein Intervall ist. Zum Beispiel berechnet INTLAB durch $\mathtt{c} = \mathtt{intval}(3)/7 \in \mathbb{IF}$ das engstmögliche Intervall mit Gleitkommagrenzen, das $3/7$ enthält.

Die engstmögliche Einschließung der tatsächlichen Hilbert-Matrix kann also einfacher durch

$$\mathtt{for\ i = 1:n,\ for\ j = 1:n,\ H(i,j) = intval(1)/(i+j-1);\ end,\ end} \tag{14.2}$$

berechnet werden. Dies ist wieder, wie in allen weiteren Beispielen, ausführbarer Code in INTLAB. Mit $\mathtt{H*H}$ wird also z.B. eine Matrix berechnet, die insbesondere H^2 enthält.

Man greift auf die Grenzen einer Intervallmatrix $\mathtt{A}$ mit $\mathtt{inf(A)}$ und $\mathtt{sup(A)}$ zu und definiert damit den „Mittelpunkt“ als $\mathtt{mid(A)} := \mathtt{0.5*(inf(A)+sup(A))}$. Man beachte, daß der tatsächliche Mittelpunkt von $A \in \mathbb{IF}^{m \times n}$ oft nicht in $\mathbb{F}^{m \times n}$, $\mathtt{mid(A)}$ also nur „in der Nähe“ liegt.

Für die Einschließung $\mathtt{H}$ der Hilbert-Matrix ist das so, da die Grenzen aufeinanderfolgende Gleitkommazahlen sind. Für den Nachweis der Regularität ist das aber unerheblich:

Lemma 14.1 *Es sei* $\mathtt{A} \in \mathbb{IF}^{n \times n}$ *und* `normC` *nach*

$$\mathtt{R = inv(mid(A));\quad normC = max(sum(eye(n) - R*A));}$$

berechnet. Aus $\mathtt{normC} < 1$ *folgt dann* $\det(A) \ne 0$ *für alle* $A \in \mathtt{A}$.

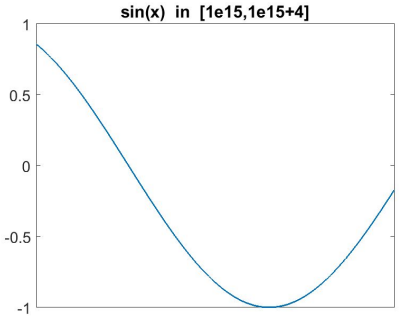
Beweis. Es sei $A \in \mathtt{A}$ fest aber beliebig. Für $\mathtt{P} := \mathtt{R*A} \in \mathbb{IF}^{n \times n}$ gilt dann $RA \in \mathtt{P}$ und, da ein Operand der Subtraktion vom Typ Intervall ist, auch $I - RA \in \mathtt{eye(n)} - \mathtt{R*A}$. Wie im Beweis von Lemma 13.1 folgt $\|I - RA\|_1 \le \mathtt{normC} < 1$ und damit die Regularität von A (und von R). □

Man beachte, daß dies den Nachweis der Regularität aller Matrizen innerhalb einer Intervallmatrix beinhaltet, ein NP-hartes Problem [56].

Für die praktische Anwendbarkeit von Intervallrechnung ist die Auswertung nicht-linearer Funktionen f über einem Intervall `X` notwendig, genauer, die Berechnung einer Einschließung von $\{f(x) : x \in X\}$. Für monotone Funktionen f ist das nicht so schwierig: Zum Beispiel genügen für die Fehlerfunktion $\mathrm{erf} : \mathbb{R} \to \mathbb{R}$ Funktionen $\underline{\mathrm{erf}}, \overline{\mathrm{erf}} : \mathbb{F} \to \mathbb{F}$ mit

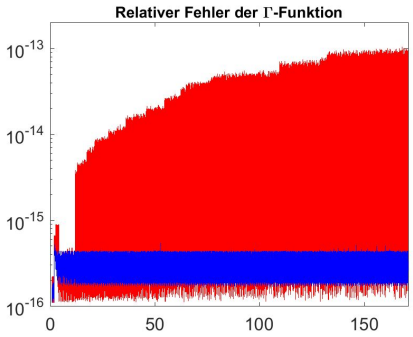
$$\forall x \in \mathbb{F} : \quad \underline{\mathrm{erf}}(x) \le \mathrm{erf}(x) \le \overline{\mathrm{erf}}(x).$$

Jene könnten durch intervallmäßige Auswertung der Taylorreihe gewonnen werden; es gibt allerdings wesentlich ausgefeiltere Methoden.



Periodische Funktionen wie der Sinus, insbesondere für große Argumente, sind schwieriger. Man benötigt eine (verifizierte) Argumentreduktion und diverse Fallunterscheidungen. In INTLAB ist das für die elementar-transzendenten Funktionen und einige weitere wie die Fehler- oder Gamma-Funktion gemacht, wobei die Resultate in aller Regel nahezu scharf sind. Zum Beispiel ergibt `sin(infsup(1e15,1e15+4))` die Einschließung $[-1, 0.85827279317024]$, wobei die untere Grenze -1 scharf ist, und die letzte Ziffer 4 der oberen Grenze nicht durch 3 ersetzt werden darf.

Numerische Bibliotheken implementieren Standardfunktionen in aller Regel sehr genau einschließlich periodischer Funktionen mit großem Argument [55], aber nicht immer. Nebenstehendes Bild zeigt den relativen Fehler der in Matlab eingebauten Γ-Funktion für positive Abszisse in rot. Der Fehler steigt bis 10^{-13}, während der Fehler der mit INTLAB berechneten Einschließung für alle Argumente kleiner als $6 \cdot 10^{-16}$ bleibt.



In einzelnen Fällen ist die numerische Approximation gar unbrauchbar: Zum Beispiel liefert

```
x = -0.9999999999999999
y = gamma(x), Y = gamma(intval(x))
```

die Matlab-Näherung $y = -5.5451 \cdot 10^{15}$, was fast um einen Faktor 2 falsch ist, wie die von INTLAB berechnete Einschließung $Y = -9.007199254740992 \cdot 10^{15} \pm 4$ zeigt. Selbstredend wird die Matlab-Näherung ohne Warnung berechnet.

Die Schattenseite der Intervallrechnung sind potentielle Überschätzungen, da Intervalle grundsätzlich als unabhängig voneinander betrachtet werden. Für $X = [\underline{x}, \overline{x}], Y = [\underline{y}, \overline{y}]$ ist ja $X + Y = [\underline{x} + \underline{y}, \overline{x} + \overline{y}]$ und $X - Y = [\underline{x} - \overline{y}, \overline{x} - \underline{y}]$. Das bedeutet für den Durchmesser

$$d(X+Y) = d(X) + d(Y) \quad \text{und} \quad d(X-Y) = (\overline{x} - \underline{y}) - (\underline{x} - \overline{y}) = d(X) + d(Y).$$

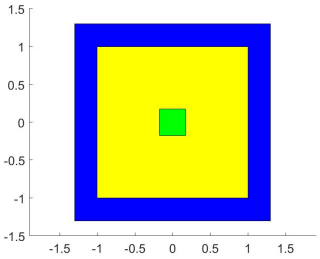
Die Durchmesser addieren sich also sowohl für Addition als auch für Subtraktion; etwas zugespitzt:

Fakt 14.2 *Durch fortgesetzte Additionen und Subtraktionen werden Intervalldurchmesser immer größer, in der Regel aber nur durch die Multiplikation mit einer betragsmäßig kleinen Zahl (oder Division durch eine große) wieder kleiner.*

Bei mehreren Operationen kann das entstehen, was treffenderweise als „wrapping effect“ bezeichnet wird. Durch Datenabhängigkeit folgt

$$A = \frac{1}{10}\begin{pmatrix} 6 & 7 \\ -7 & 6 \end{pmatrix},\ X = \begin{pmatrix} [-1,1] \\ [-1,1] \end{pmatrix} \quad \Rightarrow \quad AX = \begin{pmatrix} [-1.3,1.3] \\ [-1.3,1.3] \end{pmatrix}, \tag{14.3}$$

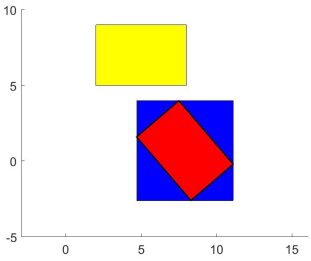
obwohl der Spektralradius $\rho(A) \approx 0.922$ kleiner als 1 ist. Bei fortgesetzter Multiplikation ist $A^m X$ als $A(A(\ldots(AX)\ldots))$ ausgeführt offenbar der Würfel mit Seitenlängen $2 \cdot 1.3^m$ (entsprechend der Zeilensummennorm $\|A\|_\infty = 1.3$), wird also immer größer obwohl $\{A^m x : x \in X\}$ gegen Null konvergiert. In untenstehender Graphik ist gelb das originale X und blau AX in gewöhnlicher Intervallarithmetik.



Eine Möglichkeit der Verbesserung ist, die Matrix (intervallmäßig) zu potenzieren und erst dann mit X zu multiplizieren, also $(A^m)X$. Damit entsteht für $m = 25$ das in nebenstehender Graphik grün gezeichnete Quadrat, in etwa von der Güte des engsten Intervalls, das $\{A^{25}x : x \in X\}$ enthält. Der Aufwand der Matrix-Multiplikationen im Vergleich zu den Matrix-Vektor-Produkten steigt für $n \times n$ Matrizen jedoch von ca. mn^2 auf mn^3.

Eine andere Möglichkeit ist die sogenannte Affine Arithmetik [1,18,8], die allerdings deutlich aufwendiger ist. Hier wird das Intervall $[2,8]$ z.B. als $Y := \{y \in \mathbb{R} : y = 5 + 3\varepsilon_1, |\varepsilon_1| \le 1\}$ definiert. Während in gewöhnlicher Intervallarithmetik aber $Y - Y$ zu $[-6,6]$ berechnet wird, folgt für Affine Arithmetik $Y - Y = \{x \in \mathbb{R} : x = (5-5) + (3-3)\varepsilon_1, |\varepsilon_1| \le 1\} = [0,0]$!

Die Kehrseite ist, daß jede neue Definition einer Intervallgröße einen neuen Paramater kreieren muß, also etwa $Z := \{z \in \mathbb{R} : z = 7 + 2\varepsilon_2, |\varepsilon_2| \le 1\}$ für das Intervall $[5,9]$. Die Anzahl der Parameter kann in gewissen Grenzen kontrolliert werden [39], trotzdem steigt der Rechenaufwand erheblich. Die Differenz der (unabhängigen) Intervalle Y und Z wird zwar wieder



$$Y - Z = \{x \in \mathbb{R} : x = -2 + 3\varepsilon_1 - 2\varepsilon_2, |\varepsilon_1|, |\varepsilon_2| \le 1\} = [-7,3]$$

wie für gewöhnliche Intervallarithmetik, als Komponenten in einem Vektor wird die Abhängigkeit von Y und Z aber ausgenutzt:

$$A \cdot \begin{pmatrix} Y \\ Z \end{pmatrix} = \{A \begin{pmatrix} 5 + 3\varepsilon_1, \\ 7 + 1.4\varepsilon_2 \end{pmatrix} : |\varepsilon_1|, |\varepsilon_2| \le 1\} = \{\begin{pmatrix} 7.9 + 1.8\varepsilon_1 + 1.4\varepsilon_2, \\ 0.7 - 2.1\varepsilon_1 + 1.2\varepsilon_2 \end{pmatrix} : |\varepsilon_1|, |\varepsilon_2| \le 1\}.$$

Der Ausgangsvektor $(Y,Z)^T$ ist oben wieder gelb, das gewöhnliche Intervallprodukt blau und das Ergebnis der Affinen Arithmetik rot gezeichnet.

Bekanntermaßen können mit Fraktalen hübsche Formen erzeugt werden, etwa die Julia-Menge der Iteration $z_{k+1} := z_k^2 + c$. Um schöne Bilder zu erzeugen, werden z.B. bis zu 500 Iterationen für einen Startwert z_0 gleitkommamäßig durchgeführt. Ist $|z_k| > \max(|z_0|, 2)$, ist die Folge divergent und der Punkt z_0 wird in der komplexen Ebene in Abhängigkeit von k eingefärbt. Ist $|z_{500}| \le \max(|z_0|, 2)$, geht man von Konvergenz aus und färbt z_0 schwarz.

Rundungsfehler werden dabei ignoriert, d.h. mathematisch gesehen ist unklar, ob die erzeugten Bilder mit der tatsächlichen Julia-Menge etwas zu tun haben.

Dabei kann leicht ein verifiziertes Bild erzeugt werden. Gilt $Z_{k+1} \subseteq \bigcup_{i=1}^{k} Z_i$ für ein Anfangsintervall $Z_0 \in \mathbb{IC}$, so ist die Folge für alle $z_0 \in Z_0$ konvergent, und gilt $|z_k| > \max(|z_0|, 2)$ für alle $z_k \in Z_k$, so ist die Folge für alle $z_0 \in Z_0$ divergent. Kann keine der Bedingungen verifiziert werden, wird Z_0 bisektiert. Mit Affiner Arithmetik erhält man damit z.B. Abb. 14.4, wobei alle schwarz eingefärbten Punkte (mit Sicherheit) eine konvergente, und die rot eingefärbten sicher eine divergente Folge erzeugen. Ein schmaler, nicht entschiedener gelber Bereich bleibt, per definitionem eine Einschließung der Julia-Menge.

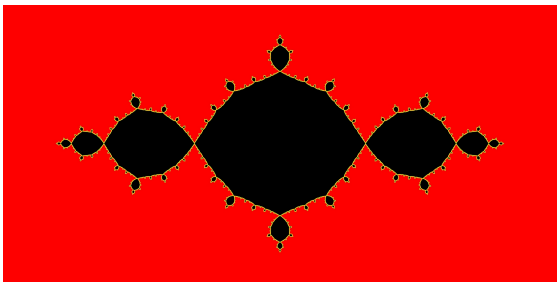


Abbildung 14.4: Julia-Menge der Iteration $z^2 + c$

15 Naive Intervallrechnung

Bevor wir zu Anwendungen kommen, zuvor wie angekündigt einige Anmerkungen zum umstrittenen Ruf der Intervallrechnung.

Die erste umfassende Darstellung der Intervallarithmetik[18] mit Anwendungen gab 1956 Sunaga [74]. Seine handschriftlich auf Japanisch verfaßte Diplomarbeit (sic!) war jedoch kaum zugänglich und ihrer Zeit voraus. In der Folge wurden seine Resultate wiederentdeckt und namentlich durch Moore [48,49] populär gemacht.

Für Intervalle A, B und eine Operation $\circ \in \{+, -, \cdot, /\}$ gilt $a \circ b \in A \circ B$ für alle $a \in A, b \in B$, vorausgesetzt $0 \notin B$ für die Division. Offensichtlich gilt das ebenso für eine Folge von Operationen (einschließlich Standardfunktionen):

Fakt 15.1 *Ersetzt man in einem arithmetischen Ausdruck oder Programm jede Gleitkommazahl f durch ein Intervall $[f, f]$ und jede Operation durch die entsprechende Intervalloperation, so wird, falls kein Nenner-Intervall die Null enthält, ein korrekte Einschließung des tatsächlichen Werts des Ausdrucks bzw. der Ergebnisse des Programms berechnet.*

Diese bemerkenswerte Eigenschaft gilt ebenso für Algorithmen, zum Beispiel für die Gauß-Elimination.[19] Von dieser, an sich richtigen Aussage, waren einige Protagonisten ab Mitte der 1960er bis in die 1970er Jahre derart elektrisiert, daß sie darin eine Art Allheilmittel gegen potentielle numerische Fehler sahen. Das wurde verbal auch drastisch ausgedrückt, wobei leider nicht nur der eigene Fortschritt gelobt, sondern die Ignoranz der „Unwissenden“ zum Teil derb diffamiert wurde.

Es sei angemerkt, daß Rechner in dieser Zeit nicht so einfach zugänglich waren wie heute, man sich also nicht einfach hinsetzen und die Sache über Spielzeugbeispiele hinaus mal ausprobieren konnte. Die angegriffenen Numeriker haben das aber getan und rasch festgestellt, daß die Aussage an sich zwar richtig, die Resultatintervalle aber schnell sehr weit und unbrauchbar wurden.

[18] Nach Wilkinson [80] dachte Turing bereits 1946 über eine Arithmetik mit Fehlerschranken und eine hierfür geeignete Hardware nach.

[19] auch „IGA“, Intervall Gauß-Algorithmus genannt.

Heute kann man analysieren [61, Kapitel 10.1: The failure of the naive approach: interval Gaussian elimination (IGA)], daß das Verfahren außer in Spezialfällen nicht funktionieren *kann* (das exponentielle Wachstum der Rundungsfehler findet sich bereits in [32]).

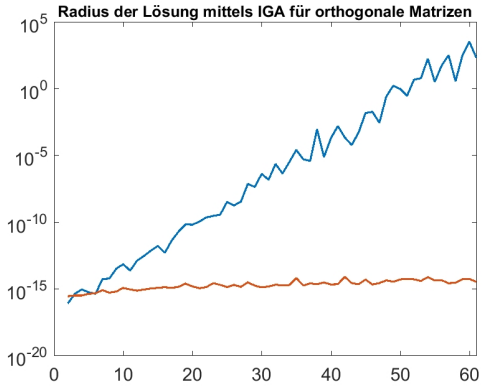


Abbildung 15.5: IGA vs. Verifikationsmethoden

Als Beispiel berechnen wir mit IGA die erste Spalte der Inversen einer zufälligen orthogonalen Matrix (welche gleich der ersten Zeile der Matrix ist). Man beachte, daß die Konditionszahl der Matrix bestmöglich gleich 1 ist. Für Dimensionen bis $n = 61$ zeigt die blaue Kurve in Abb. 15.5 in logarithmischem Maßstab den Median der Radien der von IGA berechneten Einschließungskomponenten.

Ab $n > 61$ bricht IGA wg. Null in allen Elementen der Pivotspalte ab; die (riesige) Intervalleinschließung der ersten Lösungskomponente für $n = 61$ ist $[-3.03 \cdot 10^7, 3.03 \cdot 10^7]$.

Darunter ist in rot der Median der Radien der Einschließung einer einfachen, im nächsten Kapitel erläuterten Verifikationsmethode aufgetragen. Bei der Konditionszahl 1 erwartet man diese Güte von etwa 15 Stellen Genauigkeit. Darüber hinaus ist jene Methode z.B. für $n = 60$ um den Faktor 40 schneller als IGA.

Für den mittels IGA berechneten Faktor $\mathbf{U} \in \mathbb{IF}^{n\times n}$ und $A = LU$ gilt [61, (10.5)]

$$\mathrm{rad}(\mathbf{U}) \geq \text{upper triangle}\big(\langle L\rangle^{-1}\mathrm{rad}(A)\big), \tag{15.1}$$

wobei $\langle M\rangle$ die durch

$$\langle M\rangle_{ij} := \begin{cases} |M_{ij}| & \text{for } i = j \\ -|M_{ij}| & \text{otherwise} \end{cases} .$$

definierte Ostrowskische Vergleichsmatrix bezeichnet. Man kann davon ausgehen, daß nach dem ersten Eliminationsschritt alle Matrixelemente kleine Intervalle sind, was $\mathrm{rad}(A) \approx \mathbf{u}|A|$ entspricht, so daß man von der unteren Schranke $\mathrm{rad}(\mathbf{U}) \geq \text{upper triangle}\big(\mathbf{u}\langle L\rangle^{-1}|A|\big)$ ausgehen kann, unabhängig von der Konditionszahl.

Damit wird in [61] gezeigt, daß unter vernünftigen Annahmen erwartet werden kann, daß die Radien der durch IGA bestimmten Lösung mit der Dimension exponentiell wachsen und IGA spätestens ab Dimension 150 abbricht. Das gilt im Allgemeinen, nicht für M-Matrizen oder diagonal-dominante Matrizen.

Das Verhalten ist ähnlich der erwähnten Analyse [52] von v. Neumann und Goldstine, da jede Intervalloperation den schlimmsten Fall abschätzt, ohne Abhängigkeiten der Eingabedaten zu berücksichtigen; Rundungsfehler sind aber nicht unabhängig (siehe Fakt 10.1).

Diese Fakten und vor allem die historischen Umstände haben den Ruf der Intervallrechnung nachhaltig beschädigt. Klar ist jedoch nur:

Fakt 15.2 *Ersetzt man in einem numerischen Algorithmus die Operationen durch Intervalloperationen, führt das in aller Regel zu keinem brauchbaren Verfahren.*

Das wirft per se kein schlechtes Licht auf die Intervallarithmetik, sondern auf deren falsche Anwendung: Faßt man ein Skalpell an der Schneide an, sollte man sich nicht wundern, wenn das Resultat einer Operation nicht den Erwartungen entspricht. Im Gegensatz zu naiver Anwendung von Intervallarithmetik gilt:

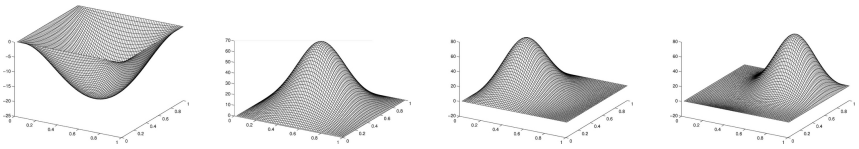
Fakt 15.3 *Verifikationsmethoden basieren auf mathematischen Theoremen, deren Voraussetzungen auf dem Rechner verifiziert werden, zum Beispiel mittels Intervallarithmetik.*

Die Kunst besteht darin, die Voraussetzungen so zu formulieren, daß der Nachweis bei nicht allzu schlecht konditionierten Problemen auch gelingt.

Immerhin werden heute in diesem Sinne nicht-triviale Probleme gelöst. Zum Beispiel vermutete man seit den 1980er Jahren, daß

$$\Delta u + u^2 = s \cdot sin(\pi x_1) sin(\pi x_2),\ x \in \Omega := (0,1)^2,\ \ u = 0 \ \text{ auf } \ \delta\Omega$$

für großes s mindestens vier Lösungen hat, und 2003 [9] wurden mit Hilfe von Verifikationsmethoden für $s = 800$ tatsächlich vier verschiedene Lösungen identifiziert:



Oder aber die 400 Jahre alte Kepler-Vermutung zur dichtesten Packung von Kugeln, für deren Beweis Hales Teilprobleme ebenfalls mithilfe von Verifikationsmethoden löste [26, 27]. Der Original-Beweis von 1998 umfaßte umfangreiche Computerprogramme; erst kürzlich wurde er mittels des automatischen Beweissystems Hol Light [29] von Hales und einer Schar von Mitstreitern verifiziert [28].

Bevor aber also allzuviel Euphorie aufkommt, sei auf die Kritik an Verifikationsmethoden in Kapitel 23 verwiesen.

16 Eine einfache Verifikationsmethode

Für ein lineares Gleichungssystem $Ax = b$ wird im folgenden eine einfache Verifikationsmethode beschrieben, an der man eine Reihe von Prinzipien bereits ablesen kann. Die Methode ist durchaus funktionsfähig, auch für schlecht konditionierte Matrizen, es existieren jedoch deutlich bessere basierend auf H-Matrizen und Perron-Frobenius Theorie [63].

Es seien $A, R \in \mathbb{R}^{n\times n}$ und $b, \tilde{x} \in \mathbb{R}^n$ gegeben mit $\|I - RA\| < 1$. Dann sind R und A nicht singulär und mit $E := I - RA$ liefert eine Standardformel $A^{-1}b - \tilde{x} = (RA)^{-1}R(b - A\tilde{x}) = (I-E)^{-1}R(b - A\tilde{x})$, und daher in jeder Matrixnorm

$$\|A^{-1}b - \tilde{x}\| \le \frac{\|R(b - A\tilde{x})\|}{1 - \|E\|}\,. \tag{16.1}$$

Mathematisch gesehen sind R und $\tilde{x}$ beliebig; eine Einschließung wird um so besser oder überhaupt erst möglich, je kleiner die Residuen $\|b - A\tilde{x}\|$ und insbesondere $\|I - RA\|$ sind.

```
function X = VerLinSys(A,b)
   R = inv(A);
   normE = norm(eye(size(A))-R*intval(A),inf);
   if normE<1
      xs = R*b;
      delta = norm(R*(b-A*intval(xs)),inf);
      X = midrad( xs , mag(delta/(1-normE)) );
   else
      X = NaN(size(b));
   end
```

Benutzt man zum Beispiel die ∞-Norm in (16.1), so liefert obiges, ausführbare Programm eine Einschließung der Lösung des linearen Gleichungssystems $Ax = b$ einschließlich des Nachweises der Regularität von A.

Zur Erläuterung: Die Matrix `eye(size(A))` in Zeile 3 ist die zu A passende Einheitsmatrix, `normE` $\in \mathbb{IF}$ eine Einschließung von $\|I - RA\|_\infty$; in Zeile 4 ist `normE` < 1 erfüllt, falls alle $r \in \mathbb{R}$ mit $r \in$ `normE` kleiner 1 sind und damit insbesondere $\|I - RA\|_\infty$; `mag` ist das Maximum aller Absolutbeträge eines Intervalls (oder -Vektors/-Matrix); `midrad(m,r)` erzeugt ein Intervall mit Mittelpunkt m und Radius r (ebenso für Vektoren und Matrizen), und schließlich zeigt `NaN`, Not a Number, an, daß keine Einschließung berechnet werden konnte.

Man beachte, daß durch die Verwendung von `intval(A)` und `intval(xs)` sichergestellt wird, daß in den Zeilen 3 und 6 jeweils nur Intervalloperationen verwendet werden, während `R` und `xs` in den Zeilen 2 bzw. 5 gleitpunktmäßig berechnet werden.

Verifikationsmethoden basieren auf möglichst guten Näherungen für die Lösung oder für andere Größen und bergen den Vorteil:

Fakt 16.1 *Es gilt folgende Dichotomie: Verifikationsmethoden liefern entweder eine korrekte Einschließung oder aber eine Fehlermeldung. Fehlerhafte Aussagen, insbesondere ohne Warnung, sind nicht möglich.*

Eine Fehlermeldung wird oben durch ein NaN-Resultat angezeigt.[20]

Mit Verifikationsmethoden können außerdem Eingabedaten mit Toleranzen einfach behandelt werden. Möchte man etwa ein lineares Gleichungssystem mit der originalen Hilbert-Matrix H lösen, so kann für `A` in obigem Algorithmus die in (14.2) definierte Intervallmatrix `H` verwandt werden, die (insbesondere) die Original Hilbert-Matrix H enthält: Einzig die zweite Zeile ist in `R` $=$ `inv(mid(A))`; zu ändern. Mit dieser Änderung wird eine Einschließung aller Gleichungssysteme $Ax = b$ mit $A \in$ `A` berechnet, insbesondere die von $Hx = b$. Die rechte Seite b kann natürlich ebenso ein Intervallvektor sein.

An obigem simplen Algorithmus, mit dem die untere Kurve in Abbildung 15.5 erzeugt wurde, kann man bereits Prinzipien von Verifikationsmethoden ablesen:

Prinzip 1. Die Operanden von Intervalloperationen sollten möglichst Eingabedaten sein, nicht bereits berechnete Einschließungen.

Im Gegensatz dazu basiert im naiven Intervall Gauß-Algorithmus jeder Eliminationsschritt auf dem vorhergehenden, die Intervalle blähen sich immer weiter auf. Die nach (14.3) angewandte Technik, $A^k X$ statt $A(A(\ldots(AX)\ldots))$ zu berechnen entspricht dem genannten Prinzip 1 und wird etwa in [43] für die Lösung von Anfangswertproblemen verwendet.

Prinzip 2. Es wird nicht die Lösung selbst, sondern der Fehler in bezug auf eine Näherungslösung eingeschlossen.

Diese in [58] eingeführte Strategie hat den Vorteil, daß sich ein größerer Radius der Einschließung des Fehlers nur indirekt auf die Qualität der Einschließung des Ergebnisses auswirkt.

Aus dem ersten Prinzip folgen prinzipielle Unterschiede zwischen numerischen und Verifikationsmethoden:

Direkte Methoden wie der Gauß-Algorithmus oder das cg-Verfahren liefern, so das Problem lösbar ist, das korrekte Ergebnis. Daraus wird ein numerischer Algorithmus, indem die reellen Operationen durch Gleitkommaoperationen ersetzt werden. Für die Übertragung ist nicht jedes mathematische Verfahren geeignet (wie die Lösung des Ausgleichsproblems

[20] Kritiker mögen einwenden, daß diese Dichotomie ja leicht zu erreichen ist, indem immer eine Fehlermeldung ausgegeben wird. Allein, das Ziel von Verifikationsmethoden ist, wie gesagt, die Kluft zwischen notwendig und hinreichend so gering wie möglich zu halten.

mittels Normalgleichungen), das ist aber eher die Ausnahme. Für Verifikationsmethoden ist diese Vorgehensweise hingegen ungeeignet, siehe Fakt 15.2: für jedes Problem ist ein spezielles Verfahren zu entwickeln.

Iterative Methoden wie das SSOR- oder Quasi-Newton-Verfahren liefern eine Folge, die unter gewissen Voraussetzungen gegen die Lösung konvergiert. Iterative Methoden können mit Intervallarithmetik nicht ohne weiteres in Verifikationsmethoden umgewandelt werden.

17 Korrekte Fehlerschranken ausschließlich in Gleitkommaarithmetik

Werden Zerlegungen wie die Gauß-Elimination in Gleitkommaarithmetik ausgeführt, kann a priori kaum etwas über den Fehler gesagt werden; a priori Abschätzungen wie die erwähnte bei v. Neumann und Goldstine [52] sind außerordentlich pessimistisch. Jene beziehen sich aber ausdrücklich nur auf die Cholesky-Zerlegung, und es ist wie eine Ironie der Geschichte das ausgerechnet diese hier eine Ausnahme macht: Allein aus der Tatsache, daß die rein gleitpunktmäßige Cholesky-Zerlegung[21] durchführbar ist, d.h. keine negativen Elemente auf der Diagonalen entstehen, folgen gute Fehlerabschätzungen für ein lineares Gleichungssystem. Folgendes, auf [15] basierende Lemma wird dazu benötigt.

Lemma 17.1 *Für symmetrisches $A \in \mathbb{F}^{n\times n}$ sei die gleitpunktmäßige Cholesky-Zerlegung durchführbar, und sei $\tilde{G}$ der berechnete Cholesky-Faktor. Dann gilt, falls $(n+1)\mathbf{u} < 1$, $\tilde{G}^T\tilde{G} = A + \Delta A$ mit $|\Delta A| \le \gamma_{n+1} dd^T$ für $\gamma_k := \frac{k\mathbf{u}}{1-k\mathbf{u}}$ und $d_i := A_{ii}^{1/2}$.*

Beweis. Aus der Durchführbarkeit des Verfahrens folgt $A_{ii} \ge 0$, also ist d reell. Nach (7.5) ist $\tilde{G}^T\tilde{G} = A + \Delta A$ mit $|\Delta A| \le (n+1)\mathbf{u}|\tilde{G}^T||\tilde{G}|$, für die i-te Spalte $\tilde{g}_i \in \mathbb{F}^n$ von $\tilde{G}$ gilt

$$\|\tilde{g}_i\|_2^2 = |\tilde{g}_i|^T|\tilde{g}_i| = \tilde{g}_i^T\tilde{g}_i = A_{ii} + \Delta A_{ii} \le A_{ii} + (n+1)\mathbf{u}|\tilde{g}_i|^T|\tilde{g}_i|,$$

und daher $\|\tilde{g}_i\|_2^2 \le (1-(n+1)\mathbf{u})^{-1}A_{ii}$. Mit Cauchy-Schwarz folgt

$$|\tilde{g}_i^T||\tilde{g}_j| \le \|\tilde{g}_i\|_2\|\tilde{g}_j\|_2 \le (1-(n+1)\mathbf{u})^{-1}\left(A_{ii}A_{jj}\right)^{1/2}$$

und $|\tilde{G}^T||\tilde{G}| \le (1-(n+1)\mathbf{u})^{-1}dd^T$, und damit das Resultat. □

Mit (5.2) folgt also insbesondere

$$\lambda_n(A) \ge \lambda_n(\tilde{G}^T\tilde{G}) - \|\Delta A\|_2 \ge -\|\,|\Delta A|\,\|_2 \ge -\gamma_{n+1}\|dd^T\|_2 = -\gamma_{n+1}\mathrm{trace}(A), \qquad (17.1)$$

da $\tilde{G}^T\tilde{G}$ positiv semidefinit ist. Allein die Durchführbarkeit der Cholesky-Zerlegung impliziert also eine Abschätzung des kleinsten Eigenwerts von A. Der Beweis der Einschließung benutzt die gegenüber (5.2) verfeinerte Abschätzung [23]

$$\forall 1 \le i \le n: \qquad \lambda_i(A) + \lambda_n(E) \le \lambda_i(A+E) \le \lambda_i(A) + \lambda_1(E) \qquad (17.2)$$

für $A, E \in \mathbb{R}^{n\times n}$ mit $A^T = A$ und $E^T = E$. Daraus folgt insbesondere, daß sich für positiv definites E sämtliche Eigenwerte von A vergrößern.

Lemma 17.2 *Für symmetrisches $A \in \mathbb{F}^{n\times n}$ sei $B = A - D \in \mathbb{F}^{n\times n}$ für diagonales D mit $D \ge 2\alpha I$ und $\alpha \ge \gamma_{n+1}\mathrm{trace}(A) > 0$. Ist die gleitpunktmäßige Cholesky-Zerlegung von B durchführbar, so ist A regulär, und für $\tilde{x} \in \mathbb{R}^n$ gilt*

$$\|A^{-1}b - \tilde{x}\|_2 \le \alpha^{-1}\|A\tilde{x} - b\|_2\,. \qquad (17.3)$$

[21] Für dieses Kapitel sei Über- oder Unterlauf ausgeschlossen.

Beweis. Es sei $\tilde{G}$ der berechnete Cholesky-Faktor von B und $d_i := B_{ii}^{1/2}$. Lemma 17.1, (17.1) und zweimalige Benutzung von (17.2) zeigen

$$\begin{aligned}\lambda_n(A) - 2\alpha &= \lambda_n(A - 2\alpha I) \geq \lambda_n(B) + \lambda_n(A - 2\alpha I - B) = \lambda_n(B) + \lambda_n(D - 2\alpha I) \\ &\geq \lambda_n(B) \geq -\gamma_{n+1}\mathrm{trace}(B) \geq -\gamma_{n+1}\mathrm{trace}(A) \geq -\alpha,\end{aligned}$$

also $\lambda_n(A) \geq \alpha > 0$. Da die Singulärwerte $\sigma_i(A)$ einer symmetrischen Matrix immer gleich den Beträgen der Eigenwerte sind, folgt $\sigma_n(A) = \lambda_n(A) \geq \alpha > 0$. Damit ist A regulär und

$$\|A^{-1}b - \tilde{x}\|_2 \leq \|A^{-1}\|_2 \|b - A\tilde{x}\|_2 = \sigma_n(A)^{-1}\|b - A\tilde{x}\|_2 \leq \alpha^{-1}\|b - A\tilde{x}\|_2. \qquad \square$$

Man beachte, daß die positive Definitheit von A nicht vorausgesetzt sondern a posteriori gezeigt wird.

Die Güte der Abschätzung (17.3) wird nahezu bestmöglich, wenn α etwas kleiner als $\frac{1}{2}\sigma_n(A)$ gewählt wird, so daß die Cholesky-Zerlegung von $A - 2\alpha I$ „gerade noch“ durchführbar ist. Darüber hinaus verbessert man $\tilde{x}$ mit einer Residueniteration.

Das Verfahren in Kapitel 16 zur Lösungseinschließung eines linearen Gleichungssystems benutzt eine Näherungsinverse A, die i.A. auch für spärliche Matrizen voll besetzt ist; dagegen wird in Lemma 17.2 nur eine Cholesky-Zerlegung näherungsweise berechnet.

In Zahlen: Ein lineares Gleichungssystem mit symmetrischer $n \times n$ Toeplitz-Matrix mit 2 und -1 auf der Haupt- bzw. den Nebendiagonalen wird mit Lemma 17.2 für $n = 10^6$ auf einem Laptop in 0.3 Sekunden mit einem Speicherbedarf von ca. 16 Mega-Byte verifiziert gelöst, eine Näherungsinverse scheitert hingegen an benötigten 8 Tera-Byte (abgesehen von geschätzten 100 Stunden Rechenzeit).

18 Anwendungsbereich und prinzipielle Grenzen von Verifikationsmethoden

In der Computer Algebra [44,46] wird mathematisch exakt gerechnet, Rundungsfehler sind ein Fremdwort. Dabei werden Langzahlen simuliert, das sind ganze oder rationale Zahlen mit beliebig vielen Ziffern - solange der Speicher reicht. Auch mit algebraischen Zahlen wird exakt gerechnet, etc. pp.

Damit sind erstaunliche und nicht-triviale mathematische Beweise möglich wie zum Beispiel im berühmten Algorithmus von Risch [57]: Für eine Funktion, die mit arithmetischen Operationen, rationalen Potenzen und elementaren Standardfunktionen definiert ist, wird bewiesen, ob in der gleichen Funktionenmenge ein uneigentliches Integral existiert oder nicht und ggf. auch berechnet. Die Rechenzeit ist endlich und wird im Vorhinein abgeschätzt.

Der Beweis der Singularität einer Matrix mittels LU-Zerlegung oder Berechnung der Determinante ist also trivial.[22] Das Rechnen mit Langzahlen hat allerdings seinen Preis, wenn auch sehr schnelle Implementierungen [20] existieren.

Verifikationsmethoden basieren auf Gleitkommaarithmetik und sind daher ausgesprochen schnell, was jedoch den Anwendungsbereich limitiert: Da jede Operation i.d.R. mit einem kleinen Fehler behaftet ist, gilt:

Fakt 18.1 *Verifikationsmethoden sind auf schlecht gestellten Probleme prinzipiell nicht anwendbar, da deren Lösung nicht stetig von den Eingabedaten abhängt.*

[22] In [17] wird allerdings von Fehlern in Mathematica und Maple berichtet: Die Determinante ganzzahliger 14×14 Matrizen (alle Einträge betragsmäßig kleiner als 1000) wird falsch berechnet. Fehler passieren; das eigentlich Schlimme ist, daß der Fehler, obwohl bekannt, über mehrere Versionen nicht korrigiert wurde.

So beweist der Algorithmus in Kapitel 13 die Regularität einer Matrix; der Beweis, daß z.B. eine Matrix den Rang $n-1$ hat, ist jedoch Computer Algebra Systemen vorbehalten, da in jeder ε-Umgebung einer singulären Matrix eine reguläre existiert.

Ist jedoch $\mathrm{Rang}(A) \le n-1$ bekannt, so gelingt der Beweis von $\mathrm{Rang}(A) = n-1$ auch mit Verifikationsmethoden. Hierzu berechnet man beispielsweise eine (näherungsweise) Singulärwertzerlegung $A \approx U\Sigma V^T$ und eine Einschließung von $E = U^T AV$. Ist D der in Diagonalanteil von E und sind $n-1$ Diagonalelemente von D größer als $\|E-D\|_2$, so ist $Rang(A) = n-1$ nach einem Perturbationssatz für Singulärwerte [23, Korollar 8.6.2].

Ebenso kann eine Verifikationsmethode zwar einen komplexen Kreis berechnen, in dem zwei Nullstellen eines Polynoms liegen, aber nicht nachweisen, ob es eine doppelte oder zwei einfache sind.

Fakt 18.1 widerspricht nicht der in Kapitel 14 gezeigten Einschließung einer Julia-Menge, die ja keine innere Punkte hat. In Abbildung 14.4 sind die schwarz (konvergent) und rot (divergent) gezeigten Bereiche verifiziert, und mathematisch folgt, daß im Komplement beider (gelb) die Julia-Menge liegen muß.

19 Nichtlineare Optimierung

In der globalen Optimierung sucht man für $f : \mathbb{R}^n \to \mathbb{R}$ das Minimum auf einer gegebenen Menge $X \subseteq \mathbb{R}^n$, ggf. eingeschränkt durch Nebenbedingungen $g(x) = 0$ und/oder $h(x) \le 0$. Ein triviales Problem dieser Art ist die Berechnung des Minimums der Funktion

$$f(x) := \cos(x^2) + \mathrm{atan}\big(x - \mathrm{erf}(x) - \mathrm{asinh}(x^3)\big) \qquad \text{auf} \quad [-5,5]. \tag{19.1}$$

Optisch gesehen (siehe Abb. 19.6) ist das Minimum etwa bei $x = 3.07$, numerisch berechnet die Matlab-Funktion `fminbnd` jedoch $x \approx 1.79$ als Abszisse des Minimums auf $[-5,5]$.

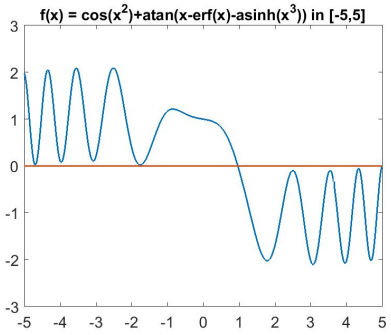


Abbildung 19.6: Minimierung einer Funktion in einer Veränderlichen

Numerische Algorithmen sind auf die Funktionswerte an endlich vielen, diskreten Stellen angewiesen, um das globale Minimum zu finden. Dadurch kann ein Algorithmus leicht in einem lokalen Minimum landen, wie für die Funktion in (19.1).

Die Auswertung an endlich vielen Punkten unterschätzt i.d.R. den Wertebereich. Mittels Intervallarithmetik kann der Funktionswert über einem Intervall eingeschlossen werden, wobei der tatsächliche Wertebereich zwar i.d.R. überschätzt, aber dennoch immer korrekt eingeschlossen wird.

Der numerischen „Vorschlag“ $\tilde{x} \approx 1.79$ kann mit einer offensichtlichen Bisektionsstrategie gekoppelt werden, um sicher das globale Minimum zu finden: Das globale Minimum ist in einem Intervall $X \in \mathbb{IF}$ sicher *nicht* enthalten, wenn das Minimum von $Y := F(X)$, d.i. die intervallmäßige Auswertung über X, die den Wertebereich $\{f(x) : x \in X\}$ einschließt, größer ist als $f([\tilde{x},\tilde{x}])$. Zum Beispiel ergibt

$$\texttt{f(infsup}(-5,-2.5)\texttt{)},\ \texttt{f(infsup}(-2.5,-1.25)\texttt{)},\ \texttt{f(infsup}(-1.25,0)\texttt{)}$$

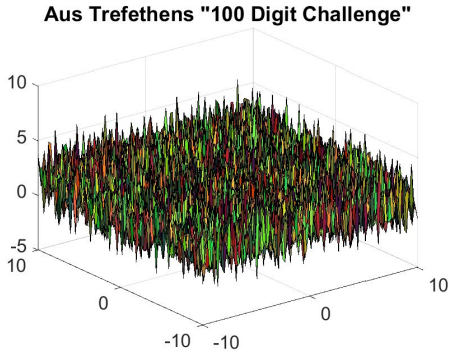
der Reihe nach $[-1.6,2.4],[-1.2,2.3],[-0.9,2.2]$, während $\texttt{f}(\texttt{intval}(\texttt{1.79})) \subseteq [-2.1,-2]$, jeweils auf zwei Stellen gerundet. Aus $f(x) > -2$ für $x \in [-5,0]$ folgt, daß die Abszisse des Minimums positiv sein muß.[23]

Testbeispiele für globale Optimierungsalgorithmen verwenden häufig periodische Funktionen über einem großen Argumentbereich, um viele lokale Minima zu erzeugen. Zum Beispiel gab Trefethen in seiner „SIAM 100-Digit Challenge“ [75] zehn Probleme mit jeweils einer skalaren Lösung und lobte für jeweils 10 korrekte Dezimalstellen der Lösung 100 US$ aus. Die Probleme hatten es in sich und regten verschiedenste Lösungen an [7,6].

Eines der Probleme war das globale Minimum von

$$f(x,y) := e^{sin(50x)} + sin(60e^y) + sin(70sin(x)) + sin(sin(80y)) - sin(10(x+y)) + (x^2+y^2)/4$$

auf $[-10,10]^2$ zu finden. Was für numerische Algorithmen schwer ist, da nur an diskreten Stellen ausgewertet wird, ist, jedenfalls in diesem Fall, für Intervallarithmetik besonders einfach: Der Wertebereich des Sinus ist auf $[-1,1]$ beschränkt, unabhängig vom Argument. Dadurch werden allfällige Überschätzungen von vornherein eingedämmt.



Die oben angedeutete, triviale Bisektionsstrategie ist allerdings zu kurz gegriffen. Deutlich bessere Ergebnisse erzielt man, wenn Ableitungen zur Verfügung stehen (und zwar über Intervallvektoren), die in Kapitel 21 besprochen werden. Enthält nämlich $\{\frac{\partial f}{\partial x}(x) : x \in X\}$ in wenigstens einer Komponente nicht die Null, kann X entweder entfernt, oder, falls X Randpunkte enthält, können jene Komponenten auf den Randwert reduziert werden.

In der Tat berechnet man auf einem Laptop in ca. 1.6 Sekunden eine Einschließung des globalen Minimums zu $\hat{x} \in (-0.0244030796943752, 0.2106124271553558)^T \pm 3 \cdot 10^{-16}$ mit $f(\hat{x}_1,\hat{x}_2) \in -3.306868647475 \pm 2 \cdot 10^{-13}$.

Es ist allerdings leicht, die Funktion für Intervallauswertung „schwierig“ zu machen, d.h., große Überschätzungen zu provozieren. Minimiert man statt f die Funktion

$$g(x,y) := f(x,y) + \cosh((x+y)/3)^2 - \sinh((x+y)/3)^2,$$

so wird der Funktionswert offenbar nur konstant um 1 erhöht; bei der intervallmäßigen Auswertung werden jedoch große Überschätzungen provoziert. Das korrekte Ergebnis wird zwar wieder berechnet, allerdings benötigt der Algorithmus jetzt eine Dreiviertelstunde!

Der Grund ist, daß für sehr enge Intervalle wie $X = 6 \pm 10^{-6}$ durch Intervallabhängigkeiten $\cosh(X)^2 - \sinh(X)^2$ zu $[-0.6276, 2.6276]$ eingeschlossen wird. Das macht es schwer, selbst enge Boxen „loszuwerden“.

In diesem Fall löst die Mittelpunkts-Entwicklung $\{f(x) : x \in X\} \subseteq f(\tilde{x}) + f'(X)(X-\tilde{x})$, die mittels der im Anschluß besprochenen automatischen Differentiation berechnet werden kann und $[0.9999997, 1.0000004]$ liefert, das Problem; in anderen Fällen ist die Überschätzung einfach groß.

[23] Tatsächlich schließt $\texttt{f}(\texttt{intval}(\texttt{1.79}))$ nicht $f(1.79)$, sondern $f(\mathrm{fl}(1.79))$ ein; das ist für die Schlußfolgerung jedoch ohne Belang.

Intervallarithmetik kann also sehr schlechte Einschließungen produzieren, geschickt eingesetzt aber trotzdem sehr hilfreich sein: Sahinidis und Tawaralani erhielten 2006 den renommierten Beale-Orchard-Hays-Preis für ihr Paket BARON [65], das laut Begründung „Techniken der automatischen Differentiation, Intervallarithmetik und anderer Gebiete zu einem automatischen, modularen und relativ effizienten Löser für sehr schwierige globale Optimierungsprobleme vereint."

20 Güte der Einschließungen bei toleranzbehafteten Daten

Für toleranzbehaftete Eingabedaten stehen numerisch Monte-Carlo Methoden zur Verfügung, die, falls die Eingabedaten unabhängig voneinander sind, Verifikationsmethoden unterlegen sein können: Es sei $A \in \mathbb{IF}^{100\times 100}$ die Intervallmatrix mit den ersten 10^4 Primzahlen der Reihe nach in den Spalten[24], wobei jede Matrixkomponente mit einer Toleranz von $\pm 2^{-13}$ behaftet ist.

Es sei $\alpha \in \mathbb{IR}$ die Menge der $(1,1)$-Elemente aller Inversen von Matrizen aus der Intervallmatrix A. Die in Kapitel 16 beschriebene Verifikationsmethode berechnet $\alpha \subseteq \mu \pm 0.001851$ mit $\mu = -0.019973$.

Es ist aber nicht klar, inwiefern die Weite des Radius der Empfindlichkeit des Problems oder aber Überschätzungen durch Intervallrechnung zuzuschreiben ist. Mit der in [61, Kapitel 10.6] beschriebenen Methode kann eine untere Schranke der Empfindlichkeit nachgewiesen werden, und zwar praktisch ohne Mehraufwand. Im vorliegenden Fall beträgt der tatsächliche Radius mindestens 0.001676, nicht viel kleiner als die obere Schranke 0.001851.

Der Radius von α kann mit Monte-Carlo-Methoden geschätzt werden, z.B. indem für K Randmatrizen, d.h. jedes Matrixelement wird zufällig mit maximaler Toleranz gewählt, jeweils die Inverse berechnet wird. Die Ergebnisse sind wie folgt:

	rad(α)	Rechenzeit [sec]
Verifikationsmethode	$\in [0.001676, 0.001851]$	0.01
$K = 100$	≥ 0.000053	0.2
$K = 10^3$	≥ 0.000088	2
$K = 10^4$	≥ 0.000124	20
$K = 10^5$	≥ 0.000121	201

Vom Prinzip der Sache her berechnet Monte-Carlo die Mindest-Empfindlichkeit, die im obigen Beispiel die tatsächliche trotz 10^5 Tests um mehr als einen Faktor 10 unterschätzt (durch eine Zufälligkeit ist diese für 10^5 Datensätze kleiner als für 10^4).

Das Gesagte gilt jedoch nur für voneinander unabhängige Eingabedaten. Hängen jene linear von Parametern ab, so gibt es zwar Verifikationsmethoden [61, Kapitel 10.7], nicht jedoch für allgemeine Abhängigkeiten, die wiederum für Monte-Carlo-Verfahren überhaupt kein Problem sind.

Fakt 20.1 *Für toleranzbehaftete Eingabedaten schließt das von einer Verifikationsmethode berechnete Ergebnis zwar die Lösung aller Probleme mit Daten innerhalb der Toleranzen ein, die Eingabedaten werden dabei in der Regel jedoch als unabhängig voneinander betrachtet. Das kann zu erheblichen Überschätzungen führen.*

[24] Die Primzahlen werden hier nur zur einfachen Angabe eines reproduzierbaren Beispiels verwandt.

21 Automatische Differentiation

Die Berechnung der Ableitung durch finite Differenzen ist bekanntlich nicht sonderlich stabil. Zum Beispiel ist nebenstehend der relative Fehler von $D := \big(f(\tilde{x}+h) - f(\tilde{x})\big)/h$ für f aus (19.1) an der Stelle $\tilde{x} = 5$ gegenüber $f'(5)$ in Abhängigkeit von h gezeigt.

Mit abnehmendem h wird der Fehler erwartungsgemäß kleiner, für zu kleines h wächst er jedoch erratisch wg. zunehmender Auslöschung wieder an. Der insgesamt kleinste Fehler wird bei $h \approx 5 \cdot 10^{-8}$ erreicht; es werden in jedem Fall höchstens 7 bis 8 korrekte Dezimalstellen der Ableitung erreicht.

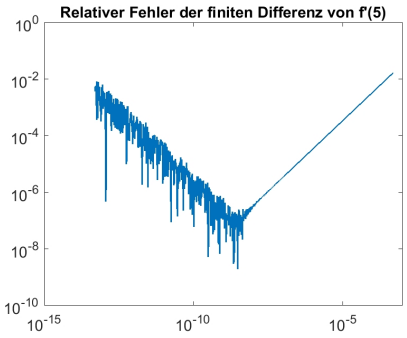


Abbildung 21.7: Finite Differenz für f aus (19.1)

Praktisch gesehen liegt natürlich weder eine Graphik für verschiedene Werte von h vor, noch kennt man den korrekten Wert der Ableitung. Mit einer einfachen Heuristik kann aber ein guter Wert für h geschätzt werden.

Der Fehler von D setzt sich aus dem Approximationsfehler der finiten Differenz und dem Rundungsfehler bei der Berechnung zusammen. Ersterer ist

$$\frac{D - f'(\tilde{x})}{f'(\tilde{x})} \approx \frac{h f''(\tilde{x})}{2 f'(\tilde{x})}.$$

Gehen wir davon aus, daß die berechneten Funktionswerte $\tilde{f}(\tilde{x}+h)$ und $\tilde{f}(\tilde{x})$ einen kleinen relativen Fehler aufweisen, also $\tilde{f}(\tilde{x}+h) = f(\tilde{x}+h)(1+\varepsilon_1)$ und $\tilde{f}(\tilde{x}) = f(\tilde{x})(1+\varepsilon_2)$ mit $|\varepsilon_\nu| \le \mathbf{u}$. Dann ist nach dem Lemma von Sterbenz (7.2) die Subtraktion für hinreichend kleines h und ebenso die Division, für h eine Potenz von 2, fehlerfrei, so daß der berechnete Wert $\tilde{D}$ sich zu

$$\tilde{D} = D + \frac{\varepsilon_2 f(\tilde{x}+h) - \varepsilon_1 f(\tilde{x})}{h} \quad \text{oder} \quad \frac{\tilde{D} - D}{D} \approx \frac{\varepsilon_2 f(\tilde{x}+h) - \varepsilon_1 f(\tilde{x})}{h f'(\tilde{x})}$$

ergibt. Nimmt man weiter an, daß $f(\tilde{x})$, $f'(\tilde{x})$ und $f''(\tilde{x})$ von ähnlicher Größenordnung sind und setzt die beiden Fehler gleich, ergibt sich ein ungefährer Wert $h \approx 2\sqrt{\mathbf{u}}$, ähnlich wie in der Graphik. Damit folgt auch, daß die Ableitung durch finite Differenzen i.allg. nicht wesentlich besser als mit einem relativen Fehler von $\sqrt{\mathbf{u}} \approx 10^{-8}$ approximiert werden kann.

Mit dem Prinzip der sogenannten „automatischen Differentiation“ werden Ableitungen nicht nur schnell, sondern auch numerisch stabil und genau berechnet [24]. Die Methode geht mindestens in die 1950er Jahre zurück und wurde immer wieder vergessen und neu veröffentlicht (ebd.). Ein Grund dafür mag sein, daß die computer-technischen Hilfsmittel noch nicht zur Verfügung standen.

Das *Prinzip* ist (vielleicht gerade zu) einfach. Für $f, g : \mathbb{R} \to \mathbb{R}$ und $\tilde{x} \in \mathbb{R}$ sei

$$f(\tilde{x}) = 0, \;\; f'(\tilde{x}) = 1, \;\; g(\tilde{x}) = 2 \quad \text{und} \quad g'(\tilde{x}) = 3$$

bekannt. Dann folgt offenbar, ohne weitere Kenntnis von f und g, daß

$$h := f + g \;\Rightarrow\; h'(\tilde{x}) = 4, \quad h := f \cdot g \;\Rightarrow\; h'(\tilde{x}) = 2 \quad \text{oder} \quad h := e^f \;\Rightarrow\; h'(\tilde{x}) = 1$$

ist. Ist eine Funktion $f : \mathbb{R} \to \mathbb{R}$ durch einen Ausdruck oder ein Programmstück gegeben, so definiert die automatische Differentiation *für festes* $\tilde{x} \in \mathbb{R}$ eine Paar-Arithmetik. Die

abhängige Variable wird durch $(\tilde{x},1)$ ersetzt, Konstanten ζ durch $(\zeta,0)$, und Operationen gemäß den üblichen Regeln der Differentiation wie

$$(a,\alpha)+(b,\beta)=(a+b,\alpha+\beta),\quad (a,\alpha)\cdot(b,\beta)=(ab,\alpha b+a\beta),\quad e^{(a,\alpha)}=(e^a,\alpha e^a)$$

definiert. Das Endergebnis (c,γ) erfüllt dann offenbar $c=f(\tilde{x})$ und $\gamma=f'(\tilde{x})$. Das Vorgehen hat eine gewisse Ähnlichkeit mit den im Risch-Algorithmus verwendeten Differentialkörpern [37]. Im Unterschied dazu und zu symbolischer Differentiation ist die automatische Differentiation jedoch eine reine Rechenvorschrift für ein festes $\tilde{x}$.

Das Ergebnis der finiten Differenz für $f'(5)$ mit f aus (19.1) war maximal auf ca. 8 Dezimalstellen genau, siehe Abb. 21.7; mit automatischer Differentiation und Intervallrechnung wird $f'(5)\in 1.444190873686714\pm 1.12\cdot 10^{-15}$ berechnet.

Für $f:\mathbb{R}^n\to\mathbb{R}$ kann das gleiche Schema, nur mit partiellen Ableitungen und $(n+1)$-Tupeln, angewandt werden. Das Ergebnis ist dann $f(\tilde{x})$ und der Gradient $\frac{\partial f}{\partial x}(\tilde{x})$. Der Rechenaufwand dieser sogenannten „Vorwärts-Differentiation“ ist jedoch erheblich, etwa das $4n$-fache einer Funktionsauswertung.

Der eigentliche Durchbruch kam mit der sogenannten „Rückwärts-Differentiation“ [24], die für beliebiges n den Funktionswert und den Gradienten in etwa der 4-fachen Rechenzeit einer Funktionsauswertung berechnet. Der Trick ist, die Ableitung eines Ausdrucks oder Programms mit Matrizen M_ν und einen Vektor p als ein Produkt $M_kM_{k-1}\ldots M_1p$ hinzuschreiben, das jeweils vorwärts (von links) als $((\ldots(M_kM_{k-1})\ldots)M_1)p$, oder aber als $M_k(M_{k-1}(\ldots(M_1p)\ldots))$ „rückwärts“ berechnet werden kann [61, Kapitel 11].

Programmiertechnisch ist die Vorwärtsvariante einfachst mit einem Operator-Konzept programmierbar, während die Rückwärtsdifferentiation im Prinzip die gleichzeitige Kenntnis aller Zwischenergebnisse voraussetzt. Den Prozeß effizient durchzuführen ist programmiertechnisch recht anspruchsvoll; es stehen hierfür aber Programmpakete wie ADIFOR [3] oder ADOL-C [25] zur Verfügung: Eingabe ist ein Programm, das einen Funktionswert berechnet, Ausgabe ist wiederum ein Programm, das in etwa der vierfachen Rechenzeit Funktionswert und Gradient liefert.

Es folgt eine bemerkenswerte Eigenschaft:

Fakt 21.1 *Ersetzt man bei der automatischen Differentiation das Argument $\tilde{x}\in\mathbb{F}^n$ durch einen Intervallvektor $X\in\mathbb{IF}^n$, so erhält man eine Einschließung des Wertebereichs $\{f(x):x\in X\}$ und des Gradienten $\{\frac{\partial f}{\partial x}(x):x\in X\}$ ohne weitere Analyse der Funktion.*

Das gilt i.W. für beliebige, durch Ausdrücke oder Programme gegebene Funktionen, wobei alle Operationen und Standardfunktionen durch die entsprechenden Intervall-Operationen und -Funktionen ersetzt werden.

Betrachten wir als Beispiel wieder die Funktion f aus (19.1) auf dem Intervall $[0.5,1]$. Dann liefert der ausführbare Code

```
f = @(x) cos(x^2)+atan(x-erf(x)-asinh(x^3));
Y = f(gradientinit(infsup(0.5,1)))
```

ohne weitere Analyse der Funktion das Ergebnis

```
intval gradient value Y.x =
[   -0.3456,    1.3099]
intval gradient derivative(s) Y.dx =
[   -4.5386,   -0.1928]
```

Das bedeutet $\{f(x) : x \in X\} \subseteq [-0.3456, 1.3099]$ und $\{f'(x) : x \in X\} \subseteq [-4.5386, -0.1928]$, so daß X kein Extremum von f enthalten[25] und deshalb in einem Algorithmus für globale Optimierung ausgeschlossen werden kann.

22 Nichtlineare Gleichungssysteme

Für eine stetig differenzierbare Funktion $f : \mathbb{R}^n \to \mathbb{R}^n$ folgt für $\tilde{x}, x \in \mathbb{R}^n$ nach dem Mittelwertsatz, daß $\xi_1, \ldots, \xi_n$ in $\{\lambda\tilde{x} + (1-\lambda)x : \lambda \in [0,1]\}$ existieren mit

$$f(x) = f(\tilde{x}) + \begin{pmatrix} \nabla f_1(\xi_1) \\ \cdots \\ \nabla f_n(\xi_n) \end{pmatrix} \big(x - \tilde{x}\big), \tag{22.1}$$

wobei $f_i : \mathbb{R}^n \to \mathbb{R}$ die i-te Komponentenfunktion von f bezeichnet. Die automatische Differentiation erlaubt für eine durch einen Ausdruck oder Programm gegebene Funktion f und einen Intervallvektor $X \in \mathbb{IF}^n$ eine Intervallmatrix $J_f(X) \in \mathbb{IF}^{n \times n}$ so zu berechnen, daß die Zeilen von $J_f(X)$ jeweils alle in Frage kommenden $\nabla f_k(\xi_k)$ enthalten. Daher gilt

$$\forall \tilde{x}, x \in X : \quad f(x) \in f(\tilde{x}) + J_f(X)(x - \tilde{x})$$

für $X \in \mathbb{IF}^n$. Der Grund ist, daß jede Zeile von $J_f(X)$ mittels automatischer Differentiation einzeln berechnet wird; hier ist die Überschätzung gegenüber der Menge der Jacobi-Matrizen $\{\frac{\partial f}{\partial x}(x) : x \in X\}$ also von Vorteil.

Für $R \in \mathbb{R}^{n \times n}$, $X \in \mathbb{IR}^n$ und $\tilde{x} \in X$ sei $g : \mathbb{R}^n \to \mathbb{R}^n$ durch $g(\delta) := \delta - Rf(\tilde{x} + \delta)$ definiert. Dann existiert für $0 \in X$ und $\tilde{x} + \delta \in X$ jeweils eine Matrix $M \in J_f(\tilde{x} + X)$ mit

$$g(\delta) = \delta - R\big(f(\tilde{x}) + M\delta\big) = -Rf(\tilde{x}) + (I - RM)\delta,$$

und daher

$$\forall x \in X : \quad g(x) \in -Rf(\tilde{x}) + \big(I - R \cdot J_f(\tilde{x} + X)\big)X. \tag{22.2}$$

Man beachte, daß mittels Intervallarithmetik und automatischer Differentiation eine Einschließung $Y \in \mathbb{IF}^n$ der rechten Seite von (22.2) berechnet werden kann. Ist $Y \subseteq X$, so folgt $\{g(x) : x \in X\} \subseteq X$, und mit dem Fixpunktsatz von Brouwer die Existenz eines $\hat{x} \in X$ mit $g(\hat{x}) = \hat{x}$ und damit $Rf(\tilde{x} + \hat{x}) = 0$. Man kann zeigen [61], daß die schärfere Forderung $Y \subseteq \mathrm{int}(X)$, also Y im topologisch Innern von X enthalten, die Regularität von R impliziert und damit $f(\tilde{x} + \hat{x}) = 0$.

Alle Operationen in (22.2) einschließlich der Berechnung von $f(\tilde{x})$ sind intervallmäßig auszuführen. In [61, Kapitel 13] ist ein ausführbares INTLAB Programm mit nur 14 Zeilen angegeben. Eingabe sind die Funktion f und eine Näherungslösung $\tilde{x}$, an deren Güte aber keine Forderungen gestellt wird.

Man beachte, daß die in Kapitel 16 angegebenen Prinzipien eingehalten werden: Die Einschließungsbedingung $Y \subseteq \mathrm{int}(X)$ wird direkt, nicht in mehreren, voneinander abhängigen Schritten, aus den Eingabedaten berechnet, und es wird der Fehler gegenüber der Näherung $\tilde{x}$ eingeschlossen.

Wie in Fakt 15.3 formuliert, basieren Verifikationsmethoden auf mathematischen Theoremen, deren Voraussetzung auf dem Rechner verifiziert werden. An (22.2) kann man ablesen, warum die hinreichende Bedingung $Y \subseteq \mathrm{int}(X)$ für nicht allzu schlecht konditionierte Probleme auch tatsächlich verifiziert wird: Praktisch nimmt man für X einen engen,

[25] Die Ausgabe ist so programmiert, daß die angezeigten Intervalle korrekte Einschließungen sind.

symmetrischen Intervallvektor mit Mittelpunkt Null, und für R eine Näherungsinverse der Jacobi-Matrix von f ausgewertet an der Stelle $\tilde{x}$. Der Intervallanteil in (22.2), von dem Überschätzungen zu befürchten wären, ist der zweite Summand. Jener ist aber das Produkt zweier, i.d.R. betragsmäßig kleiner Größen, potentielle Überschätzungen werden also eingedämmt (siehe Fakt 14.2).

Für eine zu schlechte Anfangsnäherung $\tilde{x}$ oder Näherungsinverse R wird für nichtlineare Gleichungssysteme kein falsches, sondern gar kein Ergebnis geliefert (Fakt 16.1), weil die Einschließungsbedingung $Y \subseteq \mathrm{int}(X)$ nicht erfüllt ist:

Fakt 22.1 *Verifikationsmethoden sind in der Regel auf gute Anfangsnäherungen angewiesen, für deren Güte a priori keine Voraussetzungen gelten. Zu schlechte Näherungen werden als solche erkannt und können zu keinen falschen Ergebnissen führen.*

Wie für lineare Gleichungssysteme ist mit obigen Ausführungen nur das Prinzip dargestellt; für eine praktische Implementierung sind einige Fragen zu klären, insbesondere die Wahl eines adäquaten X und was zu tun ist, wenn für das gewählte X die Einschließungsbedingung $Y \subseteq \mathrm{int}(X)$ nicht verifiziert werden kann. Das sprengt jedoch den Rahmen dieser Notiz; Details finden sich in [58, 61].

Betrachten wir als Beispiel die Einschließung einer Nullstelle von $\frac{\partial f}{\partial x} : \mathbb{R}^n \to \mathbb{R}^n$, also einen stationären Punkts der Funktion $f : \mathbb{R}^n \to \mathbb{R}$ aus dem Minimierungsproblem [12]

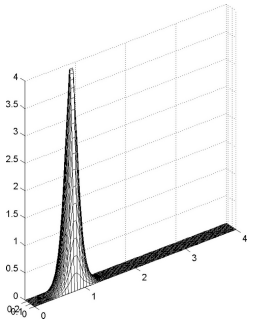
$$\text{Minimiere} \quad f(x) = \sum_{i=1}^{n-4} (3 - 4x_i)^2 + (x_i^2 + 2x_{i+1}^2 + 3x_{i+2}^2 + 4x_{i+3}^2 + 5x_n^2)^2 \,. \tag{22.3}$$

Die in [12] angegebene Startnäherung ist $(1, \ldots, 1)^T \in \mathbb{R}^n$. Damit wird für $n = 10000$ in 42 Sekunden auf einem Laptop eine in jeder Komponente auf mindestens 15 Dezimalstellen genaue Einschließung berechnet. Ein ähnliches Verfahren, das die spärliche Struktur der Jacobi-Matrix mehr ausnutzt, benötigt 10 Sekunden für eine auf mindestens 13 Dezimalstellen genaue Einschließung.

Das beschriebene Verfahren kann die Lösung eines diskreten, nichtlinearen Gleichungssystems einschließen; für die Einschließung der Lösung eines kontinuierlichen Problems wie das am Ende von Kapitel 15 aus [9] genannte sind andere Methoden notwendig.

Fakt 22.2 *Die Lösung einer Diskretisierung kann von der tatsächlichen Lösung eines kontinuierlichen Problems grundverschieden sein.*

Nebenstehende Graphik aus [61] zeigt die verifizierte Einschließung einer Lösung der diskretisierten Emden-Gleichung $-\Delta u = u^2$ mit Dirichlet-Randbedigungen auf dem Rechteck $[0, 0.25) \times [0, 4]$ mit 32 und 64 Gitterpunkten entlang der kurzen bzw. langen Seite. Nach dem Satz von Gidas, Ni, und Nirenberg [22] muß die Lösung des kontinuierlichen Problems aber zentral-symmetrisch zum Mittelpunkt des Rechtecks sein, hat mit der Lösung des diskretisierten Problem also nichts zu tun.



23 Kritik an Verifikationsmethoden: Schwächen und Stärken

Numerische Verfahren ersetzen in mathematischen Verfahren wie der Gauß-Elimination oder einem Runge-Kutta-Verfahren die reellen Operationen durch Gleitkommaoperationen.

In der Regel entsteht so ein brauchbares Verfahren. Für ein *gutes* numerisches Verfahren ist oft mehr notwendig, etwas, was man vielleicht als einen „numerischen Blick“ auf die Mathematik bezeichnen könnte.

Das Ersetzen der reellen Operationen durch Intervalloperationen ist fast zwangsläufig zum Scheitern verurteilt (Fakt 15.2). Das bedeutet, daß für jedes Problem eine jeweils geeignete Verifikationsmethode zu entwickeln ist, und zwar so, daß die Voraussetzungen der zugrunde liegenden mathematischen Theoreme auf dem Rechner nicht nur verifiziert werden können, sondern für nicht allzu schlecht konditionierte Probleme auch tatsächlich verifiziert werden (Fakt 15.3).

Offensichtlicher Vorteil der Verifikationsmethoden ist, daß ausschließlich korrekte Ergebnisse geliefert werden, oder aber eine Fehlermeldung (Fakt 16.1). Ungenaue oder falsche Näherungen und gar ohne Warnung sind nicht möglich.

Verifikation hat ihren Preis. Für viele Problemstellungen ist ein Verifikationsalgorithmus um ca. den Faktor 5 langsamer als ein rein numerischer Algorithmus, das ist der Preis für die Garantie. Ein entsprechender Aufwand in einem numerischen Algorithmus würde wohl auch eine bessere numerische Approximation berechnen, allein, die garantierte mathematische Korrektheit fehlt im allgemeinen (Fakt 4.2 und 6.1).

Verifikationsmethoden können fast ohne Mehraufwand Probleme mit toleranzbehafteten Daten lösen. Lineare Datenabhängigkeiten können zwar berücksichtigt werden, i. allg. werden die Daten jedoch als unabhängig voneinander behandelt (Fakt 20.1). Numerische Methoden können zwar nichtlineare Abhängigkeiten leicht berücksichtigen, die tatsächliche Empfindlichkeit der Lösung jedoch womöglich unterschätzen.

Es gibt Problemklassen wie semi-definite Optimierung, bei denen die über die Berechnung der Näherung hinaus notwendige Rechenzeit für die Verifikation vom Prinzip des Ansatzes her fast vernachlässigbar ist [35].

Andererseits entziehen sich Problemklassen wie schlecht gestellte Probleme oder iterative Verfahren den Verifikationsmethoden (Fakt 18.1).

Intervallrechnung berechnet rigorose Einschließungen des Wertebereichs einer Funktion und deren Ableitungen über einem Intervallvektor ohne weitere Analyse der Funktion. Das ist insbesondere vorteilhaft bei globalen Optimierungsproblemen. Dabei entsteht, je nach Problem, möglicherweise eine (erhebliche) Überschätzung, allein das Ergebnis ist immer korrekt (Fakt 21.1). Stärken und Schwächen liegen hier nahe beieinander.

Wohl fast alle Methoden, so kann man zusammenfassen, haben Licht- und Schattenseiten, und Verifikationsmethoden machen keine Ausnahme. Es wurden zwar nicht-triviale Probleme wie zum Beispiel die am Ende von Kapitel 15 erwähnten gelöst, doch für vermeintlich nicht eben schwere Probleme wie lineare Gleichungssysteme mit spärlich besetzter, symmetrischer Matrix gibt es zwar Ansätze, aber keine wirklich zufriedenstellende Verifikationsmethode:

Challenge 10.15 in [61]: Derive a verification algorithm which computes an inclusion of the solution of a linear system with a general symmetric sparse matrix of dimension $10,000$ with condition number 10^{10} in IEEE 754 double precision, and which is not more than 10 times slower than the best numerical algorithm for that problem.

Die Konditionszahl schließt das in Kapitel 17 besprochene Verfahren angewandt auf Normalgleichungen aus; das Problem harrt weiter eine Lösung.

Braucht man Verifikationsmethoden? Wie eingangs gesagt, sind ungenaue numerische Resultate selten; zu selten, sich immer darum kümmern zu müssen, aber nicht selten genug, sie zu ignorieren.

Es sind Fälle bekannt [78], in denen rein numerische Probleme zu tragischen Unfällen führten. Wenn im genannten Fall auch Unkenntnis einfacher numerischer Sachverhalte verantwortlich war, so sind für sicherheitsrelevante Fragestellungen verifiziert korrekte Ergebnisse wohl wünschenswert.

In [80] beschreibt Wilkinson, Mitbegründer der modernen Fehleranalyse, Überschätzungen bei naiver Anwendung von Intervalloperationen und den Mangel, Korrelationen zu verarbeiten, und fährt fort:

> „This does not imply that interval arithmetic is useless, but it does place severe restrictions on the way it can be applied. In general it is best in algebraic computations to leave the use of interval arithmetic as late as possible so that it effectively becomes an a posteriori weapon."

Verifikationsmethoden basieren auf mathematischen Theoremen, deren Voraussetzungen auf dem Rechner verifiziert werden (Fakt 15.3). Insofern erfordert die Entwicklung effizienter Verifikationsmethoden (im Sinne von Geschwindigkeit und Anwendbarkeit) mathematische, numerische und algorithmische Kenntnisse.

Danksagung. Meinen herzlichen Dank an die Herren Bünger und Hanke-Bourgeois für ihre große Mühe und viele wertvolle Kommentare.

Literatur

1. M.V.A. Andrade, J.L.D. Comba, and J. Stolfi. Affine Arithmetic. Presented at INTERVAL'94, 1994.
2. Y. Bertot and P. Castéran. *Interactive Theorem Proving and Program Development Coq'Art: The Calculus of Inductive Constructions*. Texts in Theoretical Computer Science, EATCS Series. Springer, 2004.
3. C.H. Bischof, A. Carle, G. Corliss, and A. Griewank. ADIFOR - Generating Derivative Codes from Fortran Programs. Technical report, Mathematics and Computer Science Devision, Argonne National Laboratory, 1991.
4. Sylvie Boldo. Pitfalls of a full floating-point proof: Example on the formal proof of the Veltkamp/Dekker algorithms. *Proceedings of Automated Reasoning IJCAR, Seattle*, pages 52–66, 2006.
5. F. Bornemann. *Numerische lineare Algebra - Eine konzise Einführung mit MATLAB und Julia*. Springer, 2016.
6. F. Bornemann. The SIAM 100-Digit Challenge: A Decade later. *Jahresberichte der DMV*, erscheint 2016.
7. F. Bornemann, D. Laurie, S. Wagon, and J. Waldvogel. *The SIAM 100-Digit Challenge—A Study in High-Accuracy Numerical Computing*. SIAM, Philadelphia, 2004.
8. O. Bouissou, E. Goubault, J. Goubault-Larrecq, and S. Putot. A generalization of p-boxes to affine arithmetic. *Computing*, 94(2-4):180–201, 2012.
9. B. Breuer, P. J. McKenna, and M. Plum. Multiple solutions for a semilinear boundary value problem: a computational multiplicity proof. *J. Differential Equations*, 195:243–269, 2003.
10. S. Chandrasekaran and I. Ipsen. Perturbation Theory for the Solution of Systems of Linear Equations. Technical Report YALEU/DCS/RR-866, Yale University, Dept. of Computer Science, October 1991.
11. F.N. Cole. On the factoring of large numbers. *Bull. Amer. Math. Soc.*, 10:134–137, 1903.
12. A.R. Conn, N.I.M. Gould, M. Lescrenier, and Ph.L. Toint. Performance of a multifrontal scheme for partially separable optimization. Technical Report 88/4, Dept of Mathematics, FUNDP (Namur, B), 1988.
13. A. Cuyt, B. Verdonk, S. Becuwe, and P. Kuterna. A Remarkable Example of Catastrophic Cancellation Unraveled. *Computing*, 66(3):309–320, 2001.
14. T.J. Dekker. A floating-point technique for extending the available precision. *Numerische Mathematik*, 18:224–242, 1971.
15. J.B. Demmel. On floating point errors in Cholesky. LAPACK Working Note 14 CS-89-87, Department of Computer Science, University of Tennessee, Knoxville, TN, USA, 1989.
16. J.B. Demmel, G. Henry, and W. Kahan. XBLAS: A Draft Proposal to the BLAS Technical Forum for Extra Precise BLAS, 1997.
17. Antonio J. Durán, Mario Pérez, and Juan Luis Varona. Misfortunes of a mathematicians' trio using computer algebra systems: Can we trust? *Notices of the ACM*, 61(10), 2014.

18. L.H. de Figueiredo and J. Stolfi. Affine arithmetic: Concepts and applications. *Numerical Algorithms*, 37(1-4):147–158, 2004.
19. L.V. Foster. Gaussian elimination with partial pivoting can fail in practice. *Siam J. Matrix Anal. Appl.*, 14:1354–1362, 1994.
20. L. Fousse, G. Hanrot, V. Lefèvre, P. Pélissier, and P. Zimmermann. MPFR: A Multiple-Precision Binary Floating-Point Library With Correct Rounding. Research Report RR-5753, INRIA, 2005. Code and documentation available at `http://hal.inria.fr/inria-00000818`.
21. C.F. Gauß. *Theoria motus corporum coelestium in sectionibus conicis solem ambientium (Theorie der Bewegung der Himmelskörper, die in Kegelschnitten die Sonne umlaufen)*. Hamburg. F. Perthes und I. H. Besser, 1809.
22. B. Gidas, W. Ni, and L. Nirenberg. Symmetry and related problems via the maximum principle. *Comm. Math. Phys.*, 68:209–243, 1979.
23. G.H. Golub and Ch. Van Loan. *Matrix Computations*. Johns Hopkins University Press, Baltimore, fourth edition, 2013.
24. A. Griewank. A Mathematical View of Automatic Differentiation. In *Acta Numerica*, volume 12, pages 321–398. Cambridge University Press, 2003.
25. A. Griewank, D. Juedes, H. Mitev, J. Utke, O. Vogel, and A. Walther. ADOL-C: A Package for the Automatic Differentiation of Algorithms Written in C/C++. *ACM Trans. Math. Software*, 22(2):131–167, 1995.
26. T.C. Hales. The Kepler conjecture. Manuscript, 1998. `http://www.math.lsa.umich.edu/~hales/countdown/`.
27. T.C. Hales. A proof of the Kepler conjecture. *Annals of Mathematics*, 162:1063–1183, 2000.
28. T.C. Hales, M. Adams, G. Bauer, D.T. Dang, J. Harrison, T.L. Hoang, C. Kaliszyk, V. Magron, S. McLaughlin, T.T. Nguyen, T.Q. Nguyen, T. Nipkow, S. Obua, J. Pleso, J. Rute, A. Solovyev, A.H. Ta, T.N. Tran, D.T. Trieu, J. Urban, K.K. Vu, and R. Zumkeller. A formal proof of the Kepler conjecture. Manuscript, 2015. `http://arxiv.org/abs/1501.02155`.
29. J. Harrison. Hol light: An overview. *Theorem Proving in Higher Order Logics*, pages 60–66, 2009.
30. D.J. Higham and N.J. Higham. Large Growth Factors in Gaussian Elimination with Pivoting. *SIAM J. Matrix Anal. Appl. (SIMAX)*, 10:155–164, 1989.
31. N. J. Higham. *Accuracy and stability of numerical algorithms*. SIAM Publications, Philadelphia, 2nd edition, 2002.
32. H. Hotelling. Some new methods in matrix calculation. *Ann. Math. Statistics*, 14:1–34, 1943.
33. *ANSI/IEEE 754-1985: IEEE Standard for Binary Floating-Point Arithmetic*. New York, 1985.
34. *ANSI/IEEE 754-2008: IEEE Standard for Floating-Point Arithmetic*. New York, 2008.
35. C. Jansson, D. Chaykin, and C. Keil. Rigorous Error Bounds for the Optimal Value in Semidefinite Programming. *SIAM Journal on Numerical Analysis*, 46(1):180–200, 2007.
36. C.-P. Jeannerod and S.M. Rump. Improved error bounds for inner products in floating-point arithmetic. *SIAM. J. Matrix Anal. & Appl. (SIMAX)*, 34(2):338–344, 2013.
37. I. Kaplanski. An Introduction to Differential Algebra. [Russian translation], Illinois, 1959.
38. M. Kashiwagi. private communication.
39. M. Kashiwagi. An algorithm to reduce the number of dummy variables in affine arithmetic. In *SCAN conference*, Novosibirsk, 2012.
40. B.W. Kernighan and D.M. Ritchie. *The C Programming Language*. Prentice-Hall software series. Prentice Hall, 1988.
41. D.E. Knuth. *The Art of Computer Programming: Seminumerical Algorithms*, volume 2. Addison Wesley, Reading, Massachusetts, third edition, 1998.
42. E. Loh and W. Walster. Rump's Example Revisited. *Reliable Computing*, 8(3):245–248, 2002.
43. R. Lohner. *Einschließung der Lösung gewöhnlicher Anfangs- und Randwertaufgaben und Anwendungen*. PhD thesis, University of Karlsruhe, 1988.
44. Maple. *Release 2015, Reference Manual*, 2015.
45. J. Markoff. Flaw undermines accuracy of pentium chip. *New York Times*, November 23, 1994.
46. Mathematica. *Release 10.4, Reference Manual*, 2016.
47. MATLAB. User's Guide, Version 2016a, the MathWorks Inc., 2016.
48. R.E. Moore. *Interval Arithmetic and Automatic Error Analysis in Digital Computing*. Dissertation, Stanford University, 1963.
49. R.E. Moore. *Interval Analysis*. Prentice-Hall, Englewood Cliffs, N.J., 1966.
50. J.-M. Muller, N. Brisebarre, F. de Dinechin, C.-P. Jeannerod, V. Lefèvre, G. Melquiond, N. Revol, D. Stehlé, and S. Torres. *Handbook of Floating-Point Arithmetic*. Birkhäuser, Boston, 2009.
51. A. Neumaier. Rundungsfehleranalyse einiger Verfahren zur Summation endlicher Summen. *Zeitschrift für Angew. Math. Mech. (ZAMM)*, 54:39–51, 1974.

52. J.v. Neumann and H.H. Goldstine. Numerical Inverting of Matrices of High Order. *Bull. Amer. Math. Soc. 53*, pages 1021–1099, 1947.
53. GNU Octave. Release 4.0.1 User's Guide, 2016. `http://www.gnu.org/software/octave/`.
54. T. Ogita, S.M. Rump, and S. Oishi. Accurate sum and dot product. *SIAM Journal on Scientific Computing (SISC)*, 26(6):1955–1988, 2005.
55. M. Payne and R. Hanek. Radian Reduction for Trigonometric Functions. *SIGNUM Newsletter*, 18:19–24, 1983.
56. S. Poljak and J. Rohn. Radius of nonsingularity. No. 88-117, Universitas Carolina Pragensis, 1988.
57. R.H. Risch. The problem of integration in finite terms. *Transactions of the American Mathematical Society*, 139:167–189, 1969.
58. S.M. Rump. *Kleine Fehlerschranken bei Matrixproblemen*. PhD thesis, Universität Karlsruhe, 1980.
59. S.M. Rump. INTLAB - INTerval LABoratory. In Tibor Csendes, editor, *Developments in Reliable Computing*, pages 77–104. Kluwer Academic Publishers, Dordrecht, 1999.
60. S.M. Rump. Inversion of extremely ill-conditioned matrices in floating-point. *Japan J. Indust. Appl. Math. (JJIAM)*, 26:249–277, 2009.
61. S.M. Rump. Verification methods: Rigorous results using floating-point arithmetic. *Acta Numerica*, 19:287–449, 2010.
62. S.M. Rump. Error estimation of floating-point summation and dot product. *BIT Numerical Mathematics*, 52(1):201–220, 2012.
63. S.M. Rump. Accurate solution of dense linear systems, Part I: Algorithms in rounding to nearest. *Journal of Computational and Applied Mathematics (JCAM)*, 242:157–184, 2013.
64. S.M. Rump and C.-P. Jeannerod. Improved backward error bounds for LU and Cholesky factorizations. *SIAM. J. Matrix Anal. & Appl. (SIMAX)*, 35(2):684–698, 2014.
65. N.V. Sahinidis and M. Tawaralani. A polyhedral branch-and-cut approach to global optimization. *Math. Programming*, B103:225–249, 2005.
66. O. Schloemilch. *Handbuch der Mathematik, Erster Band: Elementarmathematik*. Trewendt, Breslau, 1881.
67. J.R. Shewchuk. Adaptive precision floating-point arithmetic and fast robust geometric predicates. *Discrete Comput. Geom.*, 18(3):305–363, 1997.
68. S. Singh. Homers Formel. *ZEIT Online*, 46:14. November, 2013.
69. R. Skeel. Iterative Refinement Implies Numerical Stability for Gaussian Elimination. *Math. Comp.*, 35(151):817–832, 1980.
70. G.W. Stewart and J. Sun. *Matrix Perturbation Theory*. Academic Press, 1990.
71. J. Stoer. *Einführung in die Numerische Mathematik I*. Springer-Verlag, New York, 1999.
72. J. Stoer and R. Burlirsch. *Einführung in die Numerische Mathematik II*. Springer-Verlag, New York, 2005.
73. K. Strubecker. *Einführung in die höhere Mathematik*. Oldenbourg Verlag, München, 1967.
74. T. Sunaga. Geometry of Numerals. Master's thesis, University of Tokyo, February 1956.
75. L.N. Trefethen. The SIAM 100-Dollar, 100-Digit Challenge. *SIAM-NEWS*, 35(6):2, 2002.
76. L.N. Trefethen and R. Schreiber. Average-case stability of gaussian elimination. *SIAM J. Matrix Anal. Appl.*, 11(3):335–360, 1990.
77. A.M. Turing. Rounding-off errors in matrix processes. *Q. J. Mech. Appl. Math.*, 1:287–308, 1948.
78. Patriot missile defense: Software problem led to system failure at Dhahran, Saudi Arabia. Report GAO/IMTEC-92-26, Information Management and Technology Division, United States General Accounting Office, Washington, D.C. February 1992. 16 pp.
79. J.H. Wilkinson. Error Analysis of Floating-point Computation. *Numerische Mathematik*, 2:319–340, 1960.
80. J.H. Wilkinson. Modern Error Analysis. *SIAM Review (SIREV)*, 13:548–568, 1971.
81. J.H. Wilkinson. Some comments from a numerical analyst [1970 Turing Lecture anläßlich der Verleihung des ACM Turing Awards (Ausarbeitung)]. *Journal of the ACM*, 18(2):137–147, 1971.
82. S.J. Wright. A collection of problems for which Gaussian elimination with partial pivoting is unstable. *SIAM J. Sci. Comput. (SISC)*, 14(1):231–238, 1993.
83. XBLAS: A Reference Implementation for Extended and Mixed Precision BLAS. `http://crd.lbl.gov/~xiaoye/XBLAS/`.

Siegfried Michael Rump studierte Mathematik, Physik und Informatik an der Universität Kaiserslautern, wurde in Mathematik an der Universität Karlsruhe 1980 promoviert und 1983 habilitiert. Er wechselte ins IBM Forschungslabor Böblingen, wo seine Algorithmen zum Programmprodukt ACRITH wurden, und nahm 1987 den Ruf auf den Lehrstuhl für Zuverlässiges Rechnen an der Technischen Universität Hamburg an. Er war bzw. ist Editor internationaler Zeitschriften wie LAA und ACM TOMS, und er ist IFIP Silver Core member. Seit 1998 entwickelt er INTLAB, die Matlab/Octave toolbox für Reliable Computing mit einigen Tausend Nutzern in über 50 Ländern. Er verbrachte mehrere Forschungsaufenthalte an der Sorbonne Universität Marie Curie in Paris und hat seit 2002 eine zweite Professur an der Waseda Universität in Tokio inne. Seine Forschungsinteressen sind Matrixtheorie, Verifikationsmethoden und Wissenschaftliches Rechnen.

[Der Autor fischt in (der) Wirklichkeit nur mathematisch.]