1 Einleitung

Dieses Buch soll die Kunst des Rechnens auf Rechnern lehren.

Aber weil die Rechenergebnisse verlässlich sein sollen und die Mathematik die einzig verlässliche Wissenschaft ist, braucht man Mathematik, um verlässlich zu rechnen. Die erforderlichen Rechenverfahren (Algorithmen) laufen zwar auf Computern ab und könnten der Informatik zugeordnet werden, ihr Verständnis erfordert aber wesentlich mehr Mathematik als Informatik, und deshalb ist dieses Buch ein *Mathematik*buch, das von Rechenverfahren auf Rechnern handelt. Es sollte aber auch für Studierende der Informatik verständlich sein, wenn die mathematischen Grundkenntnisse nicht allzu lückenhaft sind.

Bevor das Rechnen überhaupt beginnt, muss erst einmal ein mathematisch sauber definiertes Problem vorliegen. Dieses stammt normalerweise gar nicht aus der Mathematik, sondern aus einer Anwendung in Wirtschaft, Technik oder Wissenschaft. Deshalb spricht man auch vom "Wissenschaftlichen Rechnen" als Grenzgebiet von Mathematik und Informatik, wobei Anwendungen in der Ökonomie und im Ingenieurwesen einbezogen werden. Der Weg von einem Anwendungsproblem zu einem rechnerisch lösbaren mathematischen Problem wird auch als *Mathematisierung* bezeichnet. Er erfordert spezifische Kenntnisse aus dem jeweiligen Anwendungsbereich und kann hier nur in einem Beispiel im folgenden Abschnitt 1.1 behandelt werden. Die aus der Mathematisierung entstehenden mathematischen Probleme erfordern Lösungsverfahren, die aus gewissen Standardbausteinen zusammengesetzt sind. Diese Standardbausteine und ihr mathematischer Hintergrund bilden den Kern dieses Buches. Die Lösungsverfahren liefern aber in der Regel keine exakten oder wahren Lösungen, sondern fehlerbehaftete Näherungslösungen. Deshalb gehen wir in Abschnitt 1.2 auf Fehler ein. Das schließt die Fehler ein, die durch die begrenzte Stellenzahl heutiger Rechner verursacht sind.

Der Rest dieser Einleitung enthält notwendige Dinge, die für alle nachfolgenden Kapitel wichtig sind: die Landau-Symbole in Abschnitt 1.3, die leider nicht überall zum Standardstoff der Mathematik-Grundausbildung gehören, und schließlich in Abschnitt 1.4 einiges zur Praxis des Rechnens auf Rechnern.

1.1 Vom Problem zum Programm

Der erste Schritt auf dem Weg zur Lösung eines Problems aus den Anwendungen erfordert eine *Mathematisierung*. Dadurch wird ein beispielsweise aus der Physik, der Wirtschaft oder der Technik stammendes *Anwendungsproblem* in ein (*primäres*) mathematisches Problem umgewandelt.

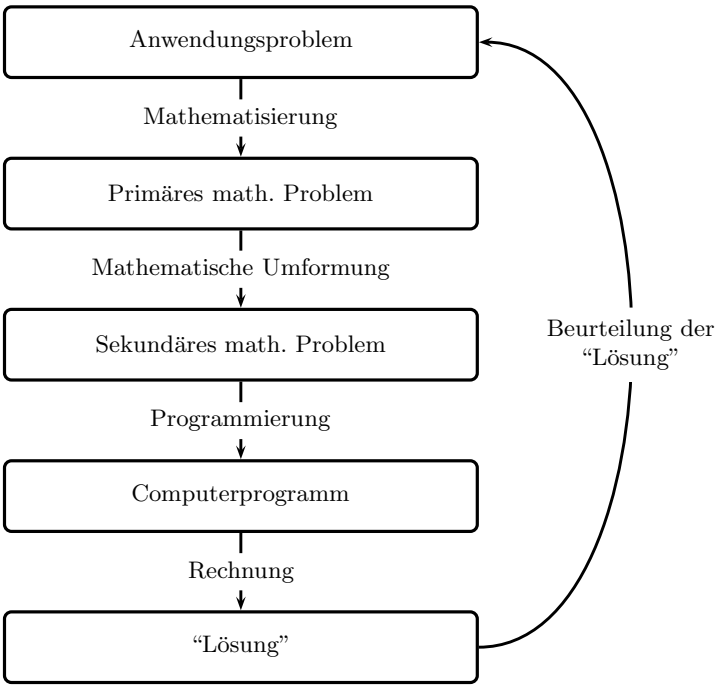

Abb. 1.1. Vom Anwendungsproblem zur Lösung

Beispiel 1.1. Wie ist das zeitliche Verhalten der Temperatur in einem Metallstab zu beschreiben, der zu einer bekannten Anfangszeit eine bestimmte Temperaturverteilung hat und an beiden Enden auf konstanter Temperatur gehalten wird? Dieses typische Anwendungsproblem soll im folgenden genauer untersucht werden.

Durch Einführung *mathematischer Objekte* (Variablen, Konstanten, Gleichungen, Abbildungen etc.), die als idealisierte Surrogate für die Bestandteile des Anwendungsproblems dienen, wird das ursprüngliche Problem in ein rein mathematisches transformiert. Gleichzeitig wird dabei eine *Interpretation* der neu eingeführten mathematischen Objekte festgelegt, die jedem Objekt eine

Bedeutung oder einen *Sinn* im Bereich der jeweiligen Anwendung zuweist. Oft werden dabei zusätzliche Annahmen eingebracht, die es fraglich sein lassen, ob die mathematischen Objekte noch die volle Wirklichkeit des Anwendungsproblems widerspiegeln.

Beispiel 1.2. Der Metallstab des Beispiels 1.1 wird als unendlich dünn und (abgesehen von den Enden) als vollständig isoliert angesehen. Seine Länge sei L, eine positive reelle Zahl. Jeder seiner Zwischenpunkte hat dann eine reelle "Koordinate" $x \in [0, L] \subseteq \mathbb{R}$. Die Temperatur zur Zeit t an der Stelle x sei $u(t, x)$. Ferner wird angenommen, das physikalische Phänomen der Wärmeleitung sei adäquat beschrieben durch die Differentialgleichung

$$\gamma \, \frac{\partial^2 u(t,x)}{\partial x^2} = \frac{\partial u(t,x)}{\partial t} \tag{1.1}$$

mit einer Konstanten γ, die ein Maß für die Wärmeleitfähigkeit des Materials ist. Zur Anfangszeit t_0, die ohne Einschränkung als Null angenommen wird, sei die Temperaturverteilung im Stab durch eine reellwertige stetige Funktion u_0 auf $[0, L]$ gegeben. Die Funktion u_0 sei, so teilt der Anwender mit, nicht überall bekannt; neben den Werten $u_0(0)$ und $u_0(L)$ an den Enden kenne man nur ihre Werte an gewissen Zwischenstellen $u_0(jL/9)$, $1 \leq j \leq 8$. Gesucht ist dann eine in x zweimal und in t einmal stetig differenzierbare Funktion $u(t, x)$, die für alle $x \in [0, L]$ und alle $t \geq 0$ der Differentialgleichung (1.1) genügt und die Zusatzbedingungen

$$\begin{aligned} u(0,x) &= u_0(x) \text{ für alle } x \in [0,L], \\ u(t,0) &= u_0(0) \text{ für alle } t \geq 0, \\ u(t,L) &= u_0(L) \text{ für alle } t \geq 0 \end{aligned}$$

erfüllt. Dies ist das durch Mathematisierung des Anwendungsproblems aus Beispiel 1.1 entstandene (primäre) mathematische Problem.

Es muss festgehalten werden, dass das mathematische Problem in der Regel nur ein stark idealisiertes Abbild des Anwendungsproblems ist. Selbst dann, wenn die Lösung des mathematischen Problems existiert und berechnet werden kann, ist durch zusätzliche Überlegungen, die spezielle Methoden des jeweiligen Anwendungsbereichs erfordern und der Verantwortung des Anwenders überlassen bleiben müssen, im Einzelfall nachzuprüfen, ob die Lösung des mathematischen Problems bei entsprechender Rückinterpretation (*Ent*mathematisierung) auch eine praktisch brauchbare Lösung des Anwendungsproblems liefert. Weil die Mathematisierung fundierte Kenntnisse des jeweiligen Anwendungsproblems erfordert, kann sie in diesem Buch nicht adäquat behandelt werden.

Das aus einem Anwendungsproblem entstandene (primäre) mathematische Problem ist oft einer praktischen Behandlung unzugänglich. In vielen

Fällen kann man Existenz und Eindeutigkeit einer Lösung theoretisch untersuchen und exakt beweisen; eine konstruktive Methode zur Berechnung einer Lösung ist damit noch nicht gegeben.

Deshalb nimmt man an dem gegebenen mathematischen Problem noch weitere Umformungen vor (oft unter Informationsverlust infolge von Vereinfachungen), bis ein konstruktiv lösbares (*sekundäres*) mathematisches Problem entsteht. Dessen Form ist von den zur Verfügung stehenden Rechenhilfsmitteln abhängig, weil letztere zur praktischen Lösung des Problems herangezogen werden müssen. Für Mathematiker, die an Problemen aus den Anwendungen arbeiten, ist deshalb die Kenntnis möglichst vieler konstruktiver *Lösungsverfahren* erforderlich. Diese Lösungsverfahren bestehen oft aus gewissen Standardbausteinen, und diese werden den Kern dieses Buches bilden.

Beispiel 1.3. Das Problem aus Beispiel 1.2 ist nach klassischen Sätzen der reellen Analysis eindeutig lösbar, wenn die Funktion u_0 vorgegeben ist. Da u_0 zunächst nur punktweise bekannt ist, hat man eine mehr oder weniger willkürliche Festlegung in den Zwischenpunkten zu treffen. Das kann durch eines der in diesem Buch behandelten *Interpolationsverfahren* geschehen. Der Fehler zwischen der "wahren" Funktion u_0 und der durch Interpolation konstruierten Funktion $\tilde{u}_0$ beeinflusst das Ergebnis. Ist $\tilde{u}(t,x)$ die Lösung zur Anfangsfunktion $\tilde{u}_0$, so gilt (nach dem Maximumprinzip aus der Theorie parabolischer Anfangsrandwertprobleme) die Fehlerabschätzung

$$|u(t,x) - \tilde{u}(t,x)| \le \max\{|\, u_0(y) - \tilde{u}_0(y)| \;\big|\; y \in [0,L]\}$$

für alle $t \ge 0$ und alle $x \in [0,L]$. Dies erlaubt, den beim Übergang zum Ersatzproblem entstehenden Fehler zu kontrollieren, wenn man den Interpolationsfehler kennt. Man sieht hier, dass Interpolation als Standardbaustein auftritt und dass Fehler und ihre Abschätzungen eine zentrale Rolle spielen.

Aber es sind noch weitere Möglichkeiten vorhanden, das Problem umzuformen. Geht man von $u(t,x)$ zu

$$v(t,x) := u\left(t\frac{L^2}{\pi^2\gamma}, x\frac{L}{\pi}\right) - \frac{x}{\pi}u_0(L) - \frac{\pi - x}{\pi}u_0(0)$$

über, so löst v das analoge Problem mit $L = \pi,\ \gamma = 1$ und der Startfunktion

$$v_0(x) := u_0\left(x\frac{L}{\pi}\right) - \frac{x}{\pi}u_0(L) - \frac{\pi - x}{\pi}u_0(0)$$

mit $v_0(0) = v_0(\pi) = 0$. Man kommt auch leicht von v zu u zurück, sodass man durch die Spezialisierung nichts verloren hat. Die vorübergehend eingeführte neue Funktion v kann man dann wieder ignorieren und das Ausgangsproblem auf $L = \pi,\ \gamma = 1$ und $u_0(0) = u_0(\pi) = 0$ eingeschränkt ansehen.

Je nachdem, welche Rechenhilfsmittel bzw. welche Standardrechenverfahren man anstrebt, lassen sich nun verschiedene Ersatzprobleme formulieren:

Beispiel 1.4. Das im obigen Sinn eingeschränkte Problem hat für die spezielle Randfunktion $u_k(x) := \sin(kx)$ für $k \in \mathbb{N}$ die Lösung $\exp(-k^2 t)\sin(kx)$. Setzt man die Randfunktion $u_0(x)$ an als Linearkombination

$$u_0(x) = \sum_{k=1}^{8} a_k \sin(kx),$$

so ist eine Lösung der Differentialgleichung gegeben durch

$$u(t,x) = \sum_{k=1}^{8} a_k \exp(-k^2 t)\sin(kx),$$

und die noch unbekannten Koeffizienten a_k kann man aus den gegebenen Daten $u_0(j\pi/9)$, $1 \le j \le 8$, zu berechnen versuchen, indem man das lineare Gleichungssystem

$$\sum_{k=1}^{8} a_k \sin(kj\pi/9) = u_0(j\pi/9), \qquad 1 \le j \le 8, \tag{1.2}$$

aufstellt.

Diese Strategie ist für die Transformation eines mathematischen Problems in ein praktisch behandelbares Ersatzproblem typisch: durch *Diskretisierung* macht man das Problem finit. In diesem speziellen Fall wird statt der allgemeinen gesuchten Funktion aus einem unendlichdimensionalen Raum eine speziell angesetzte Linearkombination aus einem endlichdimensionalen Teilraum als Näherungslösung gesucht; das Problem kann dann auf ein lineares Gleichungssystem reduziert werden.

Beispiel 1.5. Eine andere Diskretisierung ergibt sich, wenn man ausnutzt, dass die zweite Ableitung einer Funktion $f \in C^2(\mathbb{R})$ an einer Stelle x näherungsweise durch den *Differenzenquotienten* (vgl. Definition 8.13)

$$f''(x) \approx (f(x+h) - 2f(x) + f(x-h))/h^2$$

bei kleinem $h \neq 0$ gegeben ist (Beweis durch *Taylor*-Entwicklung von f um x). Dann kann man statt $u(t,x)$ die Funktionen

$$v_j(t) \approx u(t, j\pi/9), \qquad 0 \le j \le 9,$$

konstruieren mit $v_0(t) = v_9(t) = 0$ und

$$\begin{aligned} v_j(0) &= u_0(j\pi/9) \\ v_j'(t) &= \frac{81}{\pi^2}\,(v_{j+1}(t) - 2v_j(t) + v_{j+1}(t)) \end{aligned} \tag{1.3}$$

für $1 \le j \le 8$, denn man hat

$$\begin{aligned} v_j'(t) &\approx \frac{\partial u}{\partial t}(t, j\pi/9) = \frac{\partial^2 u}{\partial x^2}\,(t, j\pi/9) \\ &\approx \left(\frac{\pi}{9}\right)^{-2} (u(t,(j+1)\pi/9) - 2u(t, j\pi/9) + u(t,(j-1)\pi/9)) \\ &\approx \frac{81}{\pi^2}\,(v_{j+1}(t) - 2v_j(t) + v_{j-1}(t)). \end{aligned}$$

Dieses Ersatzproblem *(Linienmethode)* ist leicht lösbar, wenn man geeignete Hilfsmittel zur Lösung der gewöhnlichen Differentialgleichungen (1.3) hat. Die Diskretisierung erfolgt hier durch Übergang zu endlich vielen x-Koordinaten; dann hat man aber immer noch das System (1.3) zu lösen, was eventuell zu einer weiteren Diskretisierung in der t-Richtung führt. Ferner liefert die Lösung des Ersatzproblems (1.3) keine Lösung des ursprünglichen Problems; man muss $u(x,t)$ aus den $v_j(t)$ näherungsweise (z.B. durch Interpolation in x-Richtung bei festem t) berechnen und eine gründliche Fehleranalyse durchführen.

Beispiel 1.6. Ersetzt man auch noch die t-Ableitung von u durch eine geeignete Differenz

$$\frac{\partial u}{\partial t}(t,x) \approx (u(t+\Delta t, x) - u(t,x))/\Delta t$$

für ein kleines $\Delta t > 0$, so kann man die Unbekannten

$$v_{j,k} \approx u(k\Delta t,\ j\pi/9)$$

für $k \geq 1,\ j = 1, 2, \ldots, 8$ einführen und die Differentialgleichung (1.3) durch

$$\frac{81}{\pi^2}\,(v_{j+1,k} - 2v_{j,k} + v_{j-1,k}) = (v_{j,k+1} - v_{j,k})/\Delta t$$

ersetzen, wobei man die Randwerte als

$$\begin{aligned} v_{0,k} &= v_{9,k} = 0, \quad k \geq 0, \\ v_{j,0} &= u_0(j\pi/9),\ j = 1, 2 \ldots, 8, \end{aligned}$$

vorgibt. Umgeformt als

$$v_{j,k+1} = v_{j,k} + 81\Delta t(v_{j+1,k} - 2v_{j,k} + v_{j-1,k})/\pi^2 \tag{1.4}$$

erhält man eine simple Rekursionsformel, mit der man schrittweise für $k = 0, 1, 2, \ldots$ arbeiten kann. Dieser Ansatz sieht bestechend einfach aus, hat aber schwerwiegende Nachteile:

- er liefert nur Näherungswerte auf den Gitterpunkten $x = j\pi/9,\ t = k\Delta t$, und man hat eine komplizierte Fehlerabschätzung zu machen;
- führt man die Rechnung bei festem $t = T$ aus mit verschiedenen Werten von k und Δt mit $k\Delta t = T$, so erhält man nur für sehr kleine Δt (und entsprechend große k) einigermaßen brauchbare Ergebnisse.

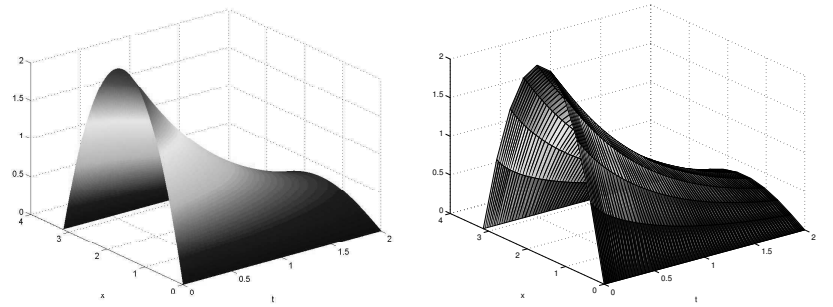

Abb. 1.2. Sinusansatz und Linienmethode

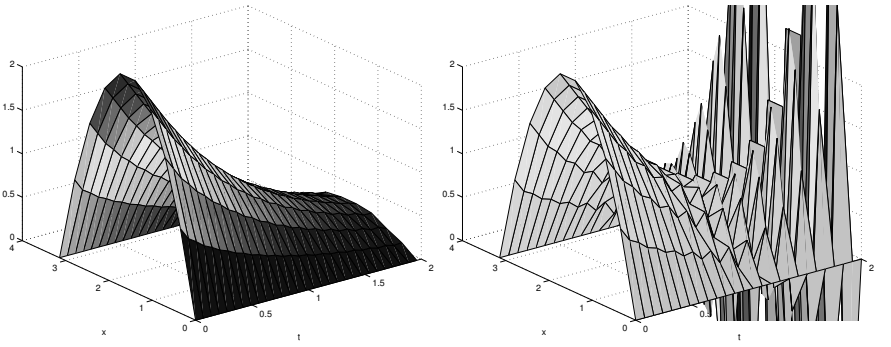

Abb. 1.3. Differenzengleichung mit $\Delta t = 0.07$ und $\Delta t = 0.08$

In Abbildung 1.2 ist für die Anfangsfunktion $u_0(x) := x(\pi - x)$ und alle $t \in [0, 2]$ ein Vergleich des Sinusansatzes (1.2) und des Differentialgleichungssystems (1.3) zu sehen, während Abbildung 1.3 die Ergebnisse der Differenzenmethode (1.4) mit $\Delta t = 0.07$ bzw. 0.08 zeigt.

Beispiel 1.7. Setzt man die Anfangsfunktion $u_0(x)$ auf ganz $\mathbb{R}$ so fort, dass eine ungerade Funktion der Periode 2π entsteht, so ist nach Standardergebnissen der mathematischen Physik die Funktion

$$u(t,x) = \sqrt{\pi/(2t)} \int_{-\infty}^{+\infty} u_0(s)\ \exp\left(-(x-s)^2/4t\right) ds \tag{1.5}$$

für alle $x \in \mathbb{R}$ und $t > 0$ eine in x periodische Lösung der Differentialgleichung (1.1), die für $t \to 0$ die Anfangswerte

$$\lim_{t \to 0} u(t,x) = u_0(x) \qquad \text{für alle } x \in \mathbb{R}$$

annimmt und $u(t, k\pi) = 0$ für alle $k \in \mathbb{Z}$ erfüllt. Mit einer hypothetischen Standardmethode, das Integral in (1.5) effizient auszuwerten, hat man ein weiteres Verfahren zur Lösung des primären mathematischen Problems.

Man sieht an diesen Beispielen deutlich, dass es viele Möglichkeiten geben kann, primäre mathematische Probleme in Ersatzprobleme umzuformen, die dann praktisch gelöst werden können. Die Auswahl wird davon abhängen, welche Standard-Rechenverfahren zur Verfügung stehen. Für das hier betrachtete Beispiel hat man die Wahl,

– das lineare Gleichungssystem (1.2),
– das Differentialgleichungssystem (1.3),
– die Rekursionsformel (1.4) oder
– die Integration (1.5)

mit den zur Verfügung stehenden Hilfsmitteln zu behandeln. Dieses Buch enthält Standardverfahren, die es erlauben, solche (Ersatz-)Probleme praktisch zu lösen; die eigentlichen Anwendungsprobleme und die zugehörigen primären mathematischen Probleme bleiben unberücksichtigt.

Man muss für jedes Ersatzproblem in Abhängigkeit von den verfügbaren Standardmethoden den *Rechenaufwand* ermitteln und den *Fehler* abschätzen; erst dann kann man eine sachgerechte Entscheidung zwischen den möglichen Standardverfahren fällen. Der nächste Abschnitt behandelt deshalb die wichtigsten Fehlerquellen bei der Lösung konkreter Probleme.

1.2 Fehler

Bei der Konstruktion praktischer Verfahren und bei der Vereinfachung von mathematischen Problemen treten zwei Arten von *Verfahrensfehlern* auf:

– *Abbruchfehler* entstehen beim Ersetzen eines infiniten Prozesses durch ein finites Verfahren (z.B. Partialsumme einer unendlichen Reihe) und
– *Diskretisierungsfehler* entstehen beim Ersetzen einer Funktion f durch endlich viele Zahlen, z.B. durch Funktionswerte $f(x_0), \ldots, f(x_n)$ oder Koeffizienten a_0, a_1, a_2 in $f(x) \approx a_0 + a_1 x + a_2 x^2$.

Sowohl die gegebenen (Ersatz-)Probleme als auch die Verfahren zu deren Lösung sind als Abbildungen zu verstehen, die zu jedem Satz möglicher *Eingangsdaten* eine Lösung des Problems angeben. Ungenaue Messwerte oder statistische Schwankungen ergeben oft Fehler in den Eingangsdaten (*Eingangsfehler*), die bei der weiteren Bearbeitung in Kauf genommen werden müssen; sie führen notwendig zu "falschen" Lösungen, aber es ist wünschenswert, dass kleine Eingangsfehler nur kleine Fehler im Ergebnis bewirken. Deshalb bezeichnet man ein (Ersatz-)Problem als *schlecht gestellt*, wenn die Lösung nicht stetig von den Eingangsdaten abhängt. Die *Störungstheorie* untersucht die Abhängigkeit der Lösung von Störungen in den Eingangsdaten; unter *Regularisierung* versteht man die Ersetzung eines schlecht gestellten Problems durch ein "gut gestelltes" Ersatzproblem, wobei ein zusätzlicher Verfahrensfehler in Kauf genommen werden muss.

Die bisher genannten Fehlerquellen sind *unabhängig* von der Rechenmethode, mit der das Ersatzproblem gelöst werden soll. Zusätzlich entstehen beim praktischen Rechnen mit reellen Zahlen weitere Fehler, die vom Lösungsverfahren und vom Rechenhilfsmittel abhängen:

- *Eingabefehler* entstehen bei der Rundung der (eventuell schon mit Eingangsfehlern behafteten) Eingabedaten auf maschinenkonforme Zahlen; selbst bei verschwindenden Eingangsfehlern und nachfolgender mathematisch exakter Rechnung bewirken sie im Endergebnis den
- *unvermeidbaren Fehler*, der aus dem Einfluss der Eingabefehler auf das Resultat besteht.
- *Rundungsfehler* können bei allen Rechenoperationen entstehen, sofern mit fester endlicher Stellenzahl und irgendeiner Rundung gerechnet wird.

In der hier verwendeten Terminologie sind Ein*gabe*fehler spezielle und maschinenabhängige Formen der Ein*gangs*fehler. Letztere umfassen auch die durch das Anwendungsproblem bedingten Fehler (z.B. Messfehler, statistische Unsicherheiten).

Definition 1.8. *Ist $\tilde{x} \in \mathbb{R}$ eine Näherung für $x \in \mathbb{R}$, so ist $|\tilde{x}-x|$ der* absolute *und $|(\tilde{x}-x)/x|$ im Falle $x \neq 0$ der* relative *Fehler.*

Beispiel 1.9. Die Exponentialfunktion

$$\exp(x) = \sum_{j=0}^{\infty} \frac{x^j}{j!}, \qquad x \in \mathbb{R}, \tag{1.6}$$

kann näherungsweise durch Auswerten der Partialsumme $P_n(x) = \sum_{j=0}^{n} x^j/j!$ berechnet werden. Für $x \leq 0$ erhält man den absoluten Abbruchfehler

$$|\exp(x) - P_n(x)| \leq |x|^{n+1}/(n+1)! \tag{1.7}$$

und für $x \geq 0$ beispielsweise

$$|\exp(x) - P_n(x)| \leq \exp(x)|x|^{n+1}/(n+1)! \tag{1.8}$$

In einer kleinen Umgebung der Null hat man also kleine absolute und relative Fehler.

Beispiel 1.10. Die *Taylor*-Formel

$$f(x) = \sum_{j=0}^{n} \frac{f^{(j)}(a)}{j!}(x-a)^j + \int_a^x \frac{(x-t)^n}{n!} f^{(n+1)}(t)dt$$

für eine $(n+1)$-mal stetig differenzierbare reelle Funktion f auf einer Umgebung von $a \in \mathbb{R}$ liefert bei Ersatz von f durch das nur von $n+1$ Koeffizienten abhängige Polynom

$$P_n(x) = \sum_{j=0}^{n} \frac{f^{(j)}(a)}{j!}(x-a)^j$$

eine Schranke für den zu erwartenden absoluten Diskretisierungsfehler

$$|f(x) - P_n(x)| \le \frac{|x-a|^{n+1}}{(n+1)!}|f^{(n+1)}(\xi)|$$

mit einem ξ zwischen a und x.

Die *reellen* Zahlen, in der Mathematik durch infinite Dezimalbrüche beschrieben, müssen in digitalen Rechenanlagen durch eine *endliche* Zahlenmenge ersetzt werden. Der entsprechende Datentyp heißt je nach Progammiersprache und Genauigkeit REAL, *float* oder *double* und besteht aus Gleitkommazahlen.

Definition 1.11. *Eine B-adische und m-stellige normalisierte* Gleitkommazahl *hat die Form $x = 0$ oder*

$$x = \pm B^e \sum_{k=-m}^{-1} x_k B^k, \qquad x_{-1} \neq 0,\ x_k \in \{0, 1, \ldots, B-1\}.$$

Man bezeichnet $e \in \mathbb{Z}$ als Exponent*, $B \ge 2$ als* Basis*, die x_k als* Ziffern *und $\sum_{k=-m}^{-1} x_k B^k$ als* Mantisse*.*

In Rechnern verwendet man in der Regel die Basis $B = 2$ und die Stellenzahl $m = 52$, um mit einem weiteren Vorzeichenbit und 11 Bits für die Exponentendarstellung insgesamt mit 64 Bits pro Zahl auszukommen. Dabei befolgt man die Norm IEEE754, die noch Raum für einige Sonderfälle hat:

- $\pm$*Inf* für $\pm\infty$ z.B. als Resultat der Division positiver oder negativer Zahlen durch Null, bei der Auswertung von log(0) oder bei Überschreitung des begrenzten Darstellungsbereichs für den Exponenten (*overflow*)
- *NaN* (*not a number*) für undefinierte Zahlen, z.B. bei Division von Null durch Null oder bei Aufruf der Logarithmusfunktion auf negativen Zahlen.

Aber die bei weitem wichtigste Eigenschaft von Gleitkommazahlen ist, eine feste *relative* Genauigkeit der Zahldarstellung zu garantieren. Zu jedem $x \in \mathbb{R}$ will man nämlich eine in der Maschine darstellbare "gerundete" Gleitkommazahl $\mathtt{rd}(x)$ haben, sodass der *relative Fehler* $|x - \mathtt{rd}(x)|/|x|$ für $x \neq 0$ durch eine feste Konstante $\mathtt{eps} > 0$ beschränkt ist. Das schreibt man auch als *Rundungsgesetz*

$$|x - \mathtt{rd}(x)| \le |x|\mathtt{eps} \qquad \text{für alle} \quad x \in \mathbb{R} \tag{1.9}$$

und unterstellt $\mathtt{rd}(0) = 0$, d.h. die Null muss exakt darstellbar sein.

Setzt man für $x \in \mathbb{R},\ x > 0$ eine infinite *B-adische Darstellung* der Form

$$x = \sum_{k=-\infty}^{n(x)} b_k B^k \tag{1.10}$$

mit Ziffern $b_k \in \{0, \ldots, B-1\}$ und einer Basis $B \geq 2,\ B \in \mathbb{N}$ und einer *führenden* Ziffer $b_{n(x)} \neq 0$ mit $n(x) \in \mathbb{Z}$ voraus, wobei ausgeschlossen wird, dass $b_k = B-1$ für alle $k \leq k_0$ mit irgendeinem $k_0 \in \mathbb{Z}$ gilt, so kann man x schreiben als

$$\begin{aligned} x &= B^{1+n(x)} \sum_{k=-\infty}^{-1} b_{n(x)+1+k} B^k \\ &= B^{1+n(x)} \sum_{k=-m}^{-1} b_{n(x)+1+k} B^k \; + \; B^{1+n(x)} \sum_{k=-\infty}^{-m-1} b_{n(x)+1+k} B^k \\ &= \qquad \mathtt{rd}(x) \qquad\qquad + \qquad x - \mathtt{rd}(x), \end{aligned} \tag{1.11}$$

wenn man nur die ersten m Ziffern von x als $\mathtt{rd}(x)$ beibehält.

Satz 1.12. *Definiert man zu jeder Zahl $x \in \mathbb{R}\backslash\{0\}$ eine gerundete Zahl $\mathtt{rd}(x)$ durch die Vorschrift, die führenden m Stellen der B-adischen Darstellung zu verwenden, so gilt das Rundungsgesetz (1.9) mit $\mathtt{eps} = B^{1-m}$.*

Beweis. Für $x > 0$ folgt aus (1.11) sofort

$$|x - \mathtt{rd}(x)| \leq B^{1+n(x)} \sum_{k=-\infty}^{-m-1} (B-1) B^k = B^{1+n(x)-m}.$$

Dies liefert mit $x \geq b_{n(x)} B^{n(x)} \geq B^{n(x)}$ dann

$$|x - \mathtt{rd}(x)| \leq |x| B^{1-m}.$$

Der analog zu behandelnde Fall negativer Zahlen ergibt schließlich die Behauptung. □

Folgerung 1.13. *Die in 64 Bit gespeicherten Gleitkommazahlen heutiger Rechner stellen wegen $B = 2$ und $m = 52$ alle reellen Zahlen mit einem maximalen relativen Fehler $\mathtt{eps} = 2^{-51} \approx 4.4409 \cdot 10^{-16}$ dar. Die sechzehnte Dezimalstelle ist also bis auf 5 Einheiten genau.*

Beispiel 1.14. Die obige Situation besteht für $B = 10$ und $m = 3$ darin, von jedem $x \in \mathbb{R}$ nur die führenden drei Dezimalstellen als $\mathtt{rd}(x)$ beizubehalten. Das bedeutet

$$\begin{aligned} \pi &= 3.141592\ldots = 10^1 \cdot 0.3141592\ldots \\ &= 10^1 \cdot 0.314 + 10^1 \cdot 0.0001592\ldots \\ &= \mathtt{rd}(\pi) + \pi - \mathtt{rd}(\pi) \end{aligned}$$

mit $n(\pi) = 1$.

Die Eingabe von Zahlen in Rechenanlagen erfolgt in der Regel durch die übliche dezimale Darstellung. Diese wird intern konvertiert in eine B-adische Gleitkommazahl. Der dabei auftretende Fehler (*Eingabefehler*) bewirkt, dass selbst dann, wenn absolut korrekt weitergerechnet wird, das wahre Ergebnis nicht berechnet werden kann, weil schon die internen Anfangsdaten verfälscht sind. Man hat daher *jedes* numerische Verfahren darauf zu untersuchen, wie stark sich die Eingabefehler auswirken.

Die Begrenztheit der Gleitkommazahlen hat zur Folge, dass die üblichen arithmetischen Operationen $+$, $-$, $\cdot$ und $/$ nicht immer exakt ausgeführt werden können. Ist $\circ$ eine der vier Standardoperationen, und sind x und y bereits Gleitkommazahlen, so kann man bestenfalls das Ergebnis $\mathtt{rd}(x \circ y)$ erwarten, d.h. die durch Runden des exakten Ergebnisses entstehende neue Gleitkommazahl. Dies halten die nach IEEE754 konstruierten Rechnerarithmetiken ein, und zwar auch bei der Auswertung von Standardfunktionen wie $\sin(x)$, $\mathrm{sqrt}(x) := \sqrt{x}$ oder $\exp(x)$.

Folgerung 1.15. *Die mit mindestens 64 Bit arbeitenden und nach IEEE754 konstruierten Rechnerarithmetiken führen alle* Einzeloperationen *auf Gleitkommazahlen mit einem maximalen relativen Fehler* $\mathtt{eps} = 2^{-51} \approx 4.4409 \cdot 10^{-16}$ *aus. Deshalb bezeichnet man* `eps` *auch als* Maschinengenauigkeit.

An dieser Stelle kann leicht der Fehlschluss entstehen, dass alle Ergebnisse von Rechenverfahren auf heutigen Computern die relative Genauigkeit `eps` hätten. Es ist in Folgerung 1.15 aber nur von *einzelnen* Operationen auf *Gleitkommazahlen* die Rede, keinesfalls von der Genauigkeit eines Endergebnisses nach 100 000 Rechenschritten. Bei jedem Einzelschritt kann nur ein maximaler relativer Fehler `eps` entstehen, aber bei 100 000 Rechenschritten wäre ja auch ein relativer Fehler der Größenordnung 100 000`eps` denkbar. Die Weiterverarbeitung fehlerhafter Daten führt im allgemeinen zur Vergrößerung der bisherigen Fehler, und das werden wir genauer zu untersuchen haben.

Definition 1.16. *Der* Rundungsfehler *eines numerischen* Verfahrens *ist der durch Gleitkommarechnung bewirkte Fehler des Endergebnisses bei exakten reellen Eingabedaten.*

Der gesamte Rundungsfehler eines Verfahrens entsteht zunächst durch Rundung der Eingabedaten auf Gleitkommazahlen. Hinzu kommen die Rundungsfehler der einzelnen Gleitkommaoperationen, aber der wichtigste Anteil entsteht durch die Fortpflanzung der Eingabefehler und der einzelnen Rundungsfehler durch alle nachfolgenden Operationen.

Deshalb betrachten wir die *Fehlerfortpflanzung* von Störungen $\Delta x = ((\Delta x)_1, \ldots, (\Delta x)_n)^T \in \mathbb{R}^n$ des Arguments $x = (x_1, \ldots, x_n)^T$ einer Abbildung $F : \mathbb{R}^n \to \mathbb{R}$ auf das Ergebnis $F(x + \Delta x)$ genauer. Ist F stetig differenzierbar, so folgt sofort durch Anwendung des Mittelwertsatzes auf die skalare Funktion $t \mapsto F(x + t\Delta x)$ die Abschätzung

$$|F(x+\Delta x)-F(x)| = \left|\sum_{j=1}^{n} \frac{\partial F(x+\widetilde{\tau}\Delta x)}{\partial x_j}(\Delta x)_j\right| \leq \max_{1\leq j\leq n}|(\Delta x)_j| \cdot \max_{0\leq\tau\leq 1}\sum_{j=1}^{n}\left|\frac{\partial F(x+\tau\Delta x)}{\partial x_j}\right|,$$

mit einem $\widetilde{\tau}\in(0,1)$, d.h. der Vergrößerungsfaktor des *absoluten* Fehlers ist im Wesentlichen durch die Ableitung bestimmt. Der *relative* Fehler pflanzt sich für $F(x)\neq 0$ und $x\neq 0$ gemäß

$$\frac{|F(x+\Delta x)-F(x)|}{|F(x)|} \leq \frac{\max_j|(\Delta x)_j|}{\max_j|x_j|} \cdot \max_{0\leq\tau\leq 1}\sum_{j=1}^{n}\left|\frac{\partial F(x+\tau\Delta x)}{\partial x_j}\right| \cdot \frac{\max_k|x_k|}{|F(x)|} \tag{1.12}$$

fort.

Definition 1.17. *Die* Kondition *eines Problems ist der im ungünstigsten Fall auftretende Vergrößerungsfaktor für den Einfluss von relativen Eingangsfehlern auf relative Resultatfehler.*

Deshalb ist der relative unvermeidbare Fehler abschätzbar durch die Kondition mal dem relativen Eingabefehler. Kennt man die Kondition eines Problems und eine Schranke für die relativen Eingangsfehler, so kann man den relativen Fehler des Ergebnisses abschätzen. In die Kondition von F geht nach (1.12) neben der Ableitung von F auch das Verhältnis von betragsmäßig größtem Eingabedatum zum Betrag des Resultats ein; man sollte deshalb stets vermeiden, kleine Ergebnisse aus großen Eingabedaten zu berechnen.

Definition 1.18. *Ist die Kondition eines Problems groß (i.Allg. diverse Zehnerpotenzen), so spricht man von einem* schlecht konditionierten *Problem.*

Bei gut konditionierten Problemen liegt der relative unvermeidbare Fehler in der Größenordnung des relativen Eingabefehlers, der wiederum nur von der relativen Genauigkeit des verwendeten Rechenhilfsmittels abhängt.

Bis hier ist stets von mathematischen (Ersatz-)Problemen gesprochen worden, ohne zu berücksichtigen, wie deren rechnerische Lösung erfolgt. Es wird sich im Beispiel 1.27 herausstellen, dass es keineswegs gleichgültig ist, wie eine Rechnung organisiert ist; bei gleichem Problem und theoretisch gleichen Resultaten bei gleichen Eingangsdaten können zwei verschiedene Verfahren völlig verschiedene Ergebnisse liefern, weil sie ein unterschiedliches Fehlerverhalten haben.

Zunächst wird das Fehlerverhalten einzelner Operationen studiert.

Satz 1.19. *Es seien $x,y\in\mathbb{R}\setminus\{0\}$ und φ sei eine der arithmetischen Operationen $+,\cdot$ und $/$. Die relativen Fehler*

$$\varepsilon_x := (\tilde{x}-x)/x,\ \ \varepsilon_y := (\tilde{y}-y)/y \qquad \textit{und}$$

$$\varepsilon_\varphi := (\varphi(\tilde{x},\tilde{y})-\varphi(x,y))/\varphi(x,y)$$

bezüglich zweier Näherungen $\tilde{x}$ von x und $\tilde{y}$ von y genügen den Formeln

$$\begin{aligned} \varepsilon_\varphi &\doteq \varepsilon_x + \varepsilon_y && \text{für } \varphi(x,y) = x \cdot y, \\ \varepsilon_\varphi &\doteq \varepsilon_x - \varepsilon_y && \text{für } \varphi(x,y) = x/y, \\ \varepsilon_\varphi &= \frac{x}{x+y}\,\varepsilon_x + \frac{y}{x+y}\,\varepsilon_y && \text{für } \varphi(x,y) = x+y \neq 0, \end{aligned} \tag{1.13}$$

wobei $\doteq$ bedeute, dass Produkte von Fehlern ignoriert werden.

Beweis. Wir beweisen von (1.13) nur die Fälle der Multiplikation und der Addition/Subtraktion, und zwar durch direkte Rechnung statt durch Anwendung von (1.12). Bei der Multiplikation folgt

$$\varepsilon_\varphi = \frac{\tilde{x}\tilde{y} - xy}{xy} = \frac{(\tilde{x}-x)\tilde{y} + x(\tilde{y}-y)}{xy} = \varepsilon_x \frac{\tilde{y}-y+y}{y} + \varepsilon_y = \varepsilon_x\varepsilon_y + \varepsilon_x + \varepsilon_y,$$

während bei der Addition

$$\varepsilon_\varphi = \frac{\tilde{x}+\tilde{y}-(x+y)}{x+y} = \frac{x\varepsilon_x + y\varepsilon_y}{x+y} = \frac{x}{x+y}\varepsilon_x + \frac{y}{x+y}\varepsilon_y$$

gilt. Die Division wird analog zur Multiplikation behandelt. □

Das Weglassen von Produkten von Fehlern ist eine Standardtechnik, die es erlaubt, komplizierte Fehlerausdrücke drastisch zu vereinfachen. Sind ε_i gewisse relative Fehler mit einer gemeinsamen Schranke $|\varepsilon_i| \leq \varepsilon$, so hat man oft Ausdrücke der Form

$$\prod_{i=1}^{n}(1+\varepsilon_i) \leq (1+\varepsilon)^n$$

abzuschätzen. Ist $n\varepsilon \leq 1/10$, so folgt

$$\begin{aligned} (1+\varepsilon)^n &\leq (\exp(\varepsilon))^n = \exp(n\varepsilon) \\ &\leq 1 + n\varepsilon + \frac{n^2\varepsilon^2}{2}(1 + n\varepsilon + n^2\varepsilon^2 + \ldots) \\ &\leq 1 + n\varepsilon + n\varepsilon \cdot \frac{0.1}{2}\frac{1}{1-0.1} \\ &\leq 1 + 1.06 \cdot n\varepsilon. \end{aligned}$$

In der Regel ist ε sehr klein und die Bedingung $n\varepsilon \leq 1/10$ unproblematisch. Dann kann man mit 1.06ε statt ε arbeiten und seelenruhig alle Ausdrücke der Form $\prod_{i=1}^{n}(1+\varepsilon_i)$ durch $1 + n\varepsilon$ a bschätzen, d.h. man ignoriert Produkte von Fehlern.

Folgerung 1.20. *Multiplikation und Division sowie die Addition von Zahlen gleichen Vorzeichens sind* gut konditioniert*, weil sie schlimmstenfalls eine Addition der Beträge der relativen Fehler der Eingabedaten bewirken. Dagegen ist die Subtraktion zweier fast gleicher Zahlen (d.h. die Addition $x+y$ im Falle $y \approx -x$)* schlecht konditioniert*, weil die Faktoren $x/(x+y)$ und $y/(x+y)$ nicht durch Eins beschränkt bleiben, sondern* sehr groß *werden können.*

Folgerung 1.21. *Der Verlust an relativer Genauigkeit bei der Differenzbildung fast gleicher Zahlen ist das schlimmste Fehlerphänomen im numerischen Rechnen. Es ist eine Konsequenz der* Fehlerfortpflanzung *und lässt sich nicht durch exakteres Ausführen der Differenzbildung beheben, wenn die Eingabedaten schon fehlerbehaftet sind. Es kann nur dringend empfohlen werden, die Bildung von Differenzen fast gleich großer Zahlen (*Auslöschung*) soweit wie möglich zu vermeiden.*

Beispiel 1.22. Subtrahiert man die sechsstelligen Zahlen

$$x = 0.344152 \qquad \text{und} \qquad y = 0.344135$$

voneinander, so folgt

$$x - y = 0.000017 = 0.17 \cdot 10^{-4}$$

mit nur zweistelliger Genauigkeit, weil sich die übereinstimmenden Ziffern "auslöschen". Enthält x einen (versteckten) relativen Fehler von nur 0.01 %, so bekommt $x - y$ einen relativen Fehler von 200 %. Dabei haben wir *exakt* gerechnet, in der Praxis können noch Rundungsfehler hinzukommen.

Beispiel 1.23. Bei der näherungsweisen Berechnung von Ableitungswerten

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}, \tag{1.14}$$

einer Funktion $f \in C^2(\mathbb{R})$ tritt für kleine h immer Auslöschung auf, wobei die Differenzbildung nach Satz 1.19 einen Fehlerfaktor

$$\frac{f(x)}{f(x+h) - f(x)} \approx \frac{f(x)}{hf'(x)}$$

bewirkt und man deshalb damit rechnen muss, dass ein relativer Fehler ε in der Berechnung der f-Werte schlimmstenfalls einen relativen Fehler

$$\frac{2\varepsilon f(x)}{hf'(x)}$$

zur Folge hat. Für kleine Werte von h oder $f'(x)$ liegt also ein schlecht konditioniertes Problem vor. Aus der *Taylor*-Formel folgt der Diskretisierungsfehler

$$|(f(x+h) - f(x))/h - f'(x)| \leq \frac{1}{2} h \max_{\xi} |f''(\xi)|,$$

sodass man (ohne Rundungseffekte und Eingabefehler für x und h) den relativen Gesamtfehler

$$\frac{2\varepsilon f(x)}{hf'(x)} + \frac{hf''(x)}{2f'(x)}$$

erwarten muss. Man ist in einer Zwickmühle: für kleine h ist die Auslöschung groß, und für große h ist der Diskretisierungsfehler groß. Minimiert man den Gesamtfehler als Funktion von h durch Nullsetzen der Ableitung nach h, so ergibt sich der günstigste h-Wert aus der Gleichung

$$h^2 = \varepsilon \frac{4f(x)}{f''(x)}.$$

Man sieht, dass bei Unkenntnis der Werte von f und f'' als Faustregel $h \approx \sqrt{\varepsilon}$ zu wählen ist, d.h. man sollte h keinesfalls kleiner als die *Wurzel* aus der relativen Genauigkeit der Funktionswerte wählen. Das Ergebnis hat bestenfalls die relative Genauigkeit

$$\sqrt{\varepsilon} \cdot 2\sqrt{|f(x)f''(x)|}/|f'(x)|.$$

Die Situation ist etwas günstiger, wenn man die symmetrische Form

$$f'(x) \approx (f(x+h) - f(x-h))/(2h) \tag{1.15}$$

verwendet, aber das ist Aufgabe 1.2 vorbehalten.

Beispiel 1.24. Für die Ableitung der Funktion $f(x) = \exp(x)$ im Punkt $x = 0$ ergeben die Formeln (1.14) und (1.15) die Werte aus Tabelle 1.1 bei einer relativen Rechengenauigkeit von 52 Binärstellen. Rechnet man mit unvernünftig kleinem h, so haben alle Funktionen scheinbar die Ableitung Null, weil $f(x+h)$ und $f(x)$ rechnerisch gleich werden! Es gehört zur Kunst des Rechnens auf Rechnern, solche katastrophalen Auslöschungsfehler zu vermeiden.

Beispiel 1.25. Es sei die Summe $s = \sum_{i=1}^{n} x_i$ von n reellen Zahlen $x_1, \ldots, x_n$ zu bilden. Führt man relative Fehler $\varepsilon_s,\ \varepsilon_1, \ldots, \varepsilon_n$ aus $[-\varepsilon, \varepsilon]$ von $s, x_1, \ldots, x_n$ ein, so ist der gestörte Wert $s(1+\varepsilon_s) \approx s$ das Ergebnis der Addition gestörter Werte $x_i(1+\varepsilon_i) \approx x_i$, was man auch als *Störungsgleichung*

$$s(1+\varepsilon_s) = \sum_{i=1}^{n} x_i(1+\varepsilon_i)$$

schreiben kann. Dann lässt sich die Kondition κ des Problems aus

$$\kappa = |\varepsilon_s|/|\varepsilon| = \left|\frac{1}{s}\sum_{i=1}^{n} \varepsilon_i x_i\right| /|\varepsilon| \le \frac{|\varepsilon|}{|s|}\ \sum_{i=1}^{n} |x_i|/|\varepsilon| = \frac{\sum_{i=1}^{n} |x_i|}{|s|}$$

ablesen; sie ist immer dann groß, wenn das Ergebnis s (durch Auslöschung) wesentlich kleiner ist als die Summe der Beträge der Summanden. Es gibt keine Probleme, wenn alle x_i gleiches Vorzeichen haben.

Tabelle 1.1. Numerische Differentiation

h	(1.14)	(1.15)
0.10000000000000000	1.051709180756477	1.001667500198441
0.01000000000000000	1.005016708416795	1.000016666749992
0.00100000000000000	1.000500166708385	1.000000166666681
0.00010000000000000	1.000050001667141	1.000000001666890
0.00001000000000000	1.000005000006965	1.000000000012102
0.00000100000000000	1.000000499962184	0.999999999973245
0.00000010000000000	1.000000049433680	0.999999999473644
0.00000001000000000	0.999999993922529	0.999999993922529
0.00000000100000000	1.000000082740371	1.000000027229220
0.00000000010000000	1.000000082740371	1.000000082740371
0.00000000001000000	1.000000082740371	1.000000082740371
0.00000000000100000	1.000088900582341	1.000033389431110
0.00000000000010000	0.999200722162641	0.999755833674953
0.00000000000001000	0.999200722162641	0.999200722162641
0.00000000000000100	1.110223024625157	1.054711873393899
0.00000000000000010	0.000000000000000	0.555111512312578
0.00000000000000001	0.000000000000000	0.000000000000000

Bei schlecht konditionierten Problemen wie der numerischen Differentiation (Beispiel 1.23) oder der Summierung großer Zahlen mit kleinem Ergebnis (Beispiel 1.25) kann kein *Verfahren* diesen Nachteil des *Problems* wettmachen. Dagegen kann die Lösung eines gut konditionierten *Problems* durch ein schlechtes *Verfahren* miserable Ergebnisse liefern.

Definition 1.26. *Ein Verfahren zur Lösung eines numerischen Problems heißt* gutartig, *wenn seine Kondition in der Größenordnung der Kondition des Problems liegt.*

Beispiel 1.27. Die Lösung $x = -p + \sqrt{p^2 + q}$ der quadratischen Gleichung

$$x^2 + 2px - q = 0 \;\text{ mit }\; p > 0 \;\text{ und }\; q > 0 \tag{1.16}$$

besitzt einen relativen Fehler ε_x, der von den relativen Fehlern ε_p und ε_q der Eingangsdaten abhängt. Die Gleichung (1.16) liefert

$$x^2(1+\varepsilon_x)^2 + 2px(1+\varepsilon_p)(1+\varepsilon_x) - q(1+\varepsilon_q) = 0,$$

und wenn man Produkte von Fehlern ignoriert, ergibt sich

$$2x^2\varepsilon_x + 2px(\varepsilon_p + \varepsilon_x) - q\varepsilon_q \doteq 0,$$

was sich nach ε_x auflösen lässt zu

$$\varepsilon_x \doteq \frac{1}{2}\frac{q\varepsilon_q - 2px\varepsilon_p}{x^2 + 2px} = \frac{q\varepsilon_q - 2px\varepsilon_p}{2q}$$

$$= \frac{1}{2}\varepsilon_q - \frac{px}{q}\varepsilon_p = \frac{1}{2}\varepsilon_q - \frac{p}{q}\frac{q}{p+\sqrt{p^2+q}}\varepsilon_p$$
$$= \frac{1}{2}\varepsilon_q - \frac{p}{p+\sqrt{p^2+q}}\varepsilon_p.$$

Falls $|\varepsilon_q| \le \varepsilon$ und $|\varepsilon_p| \le \varepsilon$ gilt, folgt die Abschätzung

$$|\varepsilon_x| \le \frac{1}{2}\,\varepsilon + \frac{1}{2}\,\varepsilon = \varepsilon$$

unabhängig von p und q. Das *Problem* ist also gut konditioniert.

Schlecht ist hingegen das Fehlerfortpflanzungsverhalten der üblichen Lösungsformel

$$x = -p + \sqrt{p^2+q}, \tag{1.17}$$

weil im Falle $p > 0$ und $q \approx 0$ starke Auslöschung eintritt. Die alternative Schreibweise

$$x = (-p + \sqrt{p^2+q})\frac{p+\sqrt{p^2+q}}{p+\sqrt{p^2+q}} = \frac{q}{p+\sqrt{p^2+q}} \tag{1.18}$$

bewirkt bei entsprechender Rechnung eine Vermeidung der Auslöschung. Auch ohne genaue Analyse ist schon klar, dass die zweite Form der ersten vorzuziehen ist.

Mit der relativen Genauigkeit ε_y des Zwischenergebnisses $y := \sqrt{p^2+q}$ führt die Formel (1.17) zu

$$\varepsilon_x \doteq \frac{y}{y-p}\varepsilon_y - \frac{p}{y-p}\varepsilon_p,$$
$$|\varepsilon_x| \le \frac{y+p}{y-p}\max(|\varepsilon_y|, |\varepsilon_p|)$$

und hat erwartungsgemäß schlechte Kondition, wenn

$$x = y - p = \sqrt{p^2+q} - p = q/(p+\sqrt{p^2+q})$$

klein wird. Die alternative Form (1.18) liefert bei schrittweiser Anwendung des Satzes 1.19 auf relative Eingangsfehler $\varepsilon_p,\ \varepsilon_q \in [-\varepsilon, \varepsilon]$ die Ergebnisse

$$|\varepsilon_{p^2}| \doteq |\varepsilon_p + \varepsilon_p| \le 2\varepsilon,$$
$$|\varepsilon_{p^2+q}| = |p^2\varepsilon_{p^2}/(p^2+q) + q\varepsilon_q/(p^2+q)| \le 2\varepsilon,$$
$$|\varepsilon_y| = \frac{1}{2}|\varepsilon_{p^2+q}| \le \varepsilon, \qquad \text{siehe Aufgabe 1.3,}$$
$$|\varepsilon_z| = |p\varepsilon_p/(p+y) + y\varepsilon_y/(p+y)| \le \varepsilon,$$
$$|\varepsilon_x| \doteq |\varepsilon_q - \varepsilon_z| \le 2\varepsilon.$$

Deshalb ist dieses Verfahren gutartig.

Es zeigt sich somit, dass man auf jeden Fall die Kondition des *Problems* und des *Verfahrens* abschätzen muss, wenn man Aussagen über die Brauchbarkeit eines Verfahrens machen will. Wir fassen zusammen:

In einer Rechnerarithmetik nach IEEE754 mit 64 Speicherbits gilt das Rundungsgesetz (1.9) mit $\texttt{eps} = 2^{-51} \approx 4.4409 \cdot 10^{-16}$ für jede arithmetische Einzeloperation auf Gleitkommazahlen. Bei jedem Rechenschritt entsteht also schlimmstenfalls ein neuer relativer Fehler der Größenordnung `eps`, der danach der Fehlerfortpflanzung unterliegt. Ignoriert man die Fehlerfortpflanzung, so entstehen durch k Eingangsfehler und die Rundungsfehler bei N Operationen insgesamt relative Fehler der Größenordnung $(N + k)$ `eps`, was bei moderaten Werten von N und k nicht allzu problematisch wäre. Große Zwischenergebnisse liefern aber auch große absolute Fehler, die sich dann bei kleinen Endergebnissen infolge der Auslöschung zu großen relativen Fehlern fortpflanzen, sofern Additionen oder Subtraktionen mit kleinen Endergebnissen auftreten. Um Fortpflanzung dieser Fehler zu vermeiden, sollten Zwischenergebnisse, die erheblich größer als die Endergebnisse sind, vermieden oder an das Ende des Verfahrens verschoben werden.

Folgerung 1.28. *Beim praktischen Rechnen auf Rechnern gilt:*

- *Man vermeide Auslöschungen.*
- *Betragsmäßig große Zwischenergebnisse sollten umgangen bzw. möglichst an den Schluss der Rechnung verschoben werden.*
- *Soweit möglich, untersuche man die Kondition des Problems und des Verfahrens. Stellt sich heraus, dass das Problem schlecht konditioniert ist, informiere man den Auftraggeber und versuche, ein anderes, besser konditioniertes Ersatzproblem zu finden. Stellt sich heraus, dass das Verfahren nicht gutartig ist, überlege man sich ein besseres durch Befolgung der beiden vorigen Regeln.*

Die dritte Regel ist oft nicht erfüllbar, weil die Konditionsanalyse schwieriger als die eigentliche Problemlösung sein kann. Dann sollte man eine Reihe von Testläufen des Verfahrens mit gestörten Eingabedaten machen. Wenn die Resultate unerwartet stark schwanken, liegt der Verdacht nahe, dass entweder das Problem schlecht konditioniert oder das Verfahren nicht gutartig ist.

Wenn weder eine Konditionsuntersuchung für Problem und Verfahren noch eine solche experimentelle *Sensitivitätsanalyse* gemacht wird, liegt ein Kunstfehler beim Rechnen auf Rechnern vor.

1.3 Landau-Symbole

Diese sind ein wichtiges Hilfsmittel zur quantitativen Beschreibung von Grenzprozessen.

Definition 1.29. *Für Abbildungen $f, g : \mathbb{R} \to \mathbb{R}$ oder $\mathbb{C}$ schreibt man*

$$f = \mathcal{O}(g) \qquad \text{für } x \to x_0,$$

falls es eine reelle Zahl $C > 0$ und eine Umgebung U von x_0 gibt, sodass

$$|f(x)| \le C|g(x)|$$

für alle $x \in U$ gilt. Entsprechend schreibt man $f = o(g)$ für $x \to x_0$, falls es zu jedem $\varepsilon > 0$ eine Umgebung U von x_0 gibt, sodass $|f(x)| \le \varepsilon|g(x)|$ für alle $x \in U$ gilt.

Im Falle von Folgen hat man als Umgebungen von $x_0 = \infty$ Mengen der Form $\{n \in \mathbb{N} : n \ge N\}$ zu nehmen:

Definition 1.30. *Für Folgen $\{a_n\}_{n\in\mathbb{N}}$ und $\{b_n\}_{n\in\mathbb{N}}$ in $\mathbb{R}$ oder $\mathbb{C}$ schreibt man $a_n = \mathcal{O}(b_n)$ für $n \to \infty$, falls es eine reelle Zahl $C > 0$ und eine natürliche Zahl N gibt, sodass $|a_n| \le C|b_n|$ für alle $n \ge N$ gilt. Man schreibt $a_n = o(b_n)$ für $n \to \infty$, falls es zu jedem $\varepsilon > 0$ eine natürliche Zahl N gibt, sodass $|a_n| \le \varepsilon|b_n|$ für alle $n \ge N$ gilt.*

Von großer praktischer Bedeutung ist, dass

– logarithmisches Wachstum schwächer ist als polynomiales,
– polynomiales Wachstum schwächer ist als exponentielles,
– exponentielles Wachstum schwächer ist als fakultatives.

Dies besagt der erste Teil des folgenden Satzes.

Satz 1.31. *(1) Für reelle Zahlen α, β, γ gilt für $n \to \infty$:*

$$\begin{aligned} (\log_\gamma n)^\beta &= o(n^\alpha) && \text{für alle } \alpha, \beta > 0,\ \gamma > 1, \\ n^\alpha &= o(\beta^n) && \text{für alle } \alpha > 0\ , \beta > 1, \\ \beta^n &= o(n!) && \text{für alle } \beta > 1. \end{aligned}$$

(2) Es gilt $e^{\frac{1}{x}} = o(x^n)$ für $x \to 0$ für jedes $n \in \mathbb{N}$.
(3) Die Aussage $f = o(1)$ für $x \to x_0$ ist gleichbedeutend mit der Aussage "f konvergiert gegen 0 für $x \to x_0$".
(4) Die Aussage $f = \mathcal{O}(1)$ für $x \to x_0$ ist gleichbedeutend mit der Aussage "f ist in einer Umgebung von x_0 beschränkt".
(5) Die Aussage $a_n = o(1)$ für $n \to \infty$ ist gleichbedeutend mit der Aussage "Die Folge $\{a_n\}$ ist eine Nullfolge".
(6) Die Aussage $a_n = \mathcal{O}(1)$ für $n \to \infty$ ist gleichbedeutend mit der Aussage "Die Folge $\{a_n\}$ ist beschränkt".

1.4 Elementare Rechentechniken

Die meisten Rechnungen werden wir, mathematisch gesehen, in den reellen Zahlen $\mathbb{R}$ ausführen, wobei es stets klar ist, dass in der Praxis Gleitkommazahlen nach IEEE754 benutzt werden. Komplexe Zahlen $z = x + iy$ kann man

rechentechnisch immer als Paare $(x, y) \in \mathbb{R}^2$ behandeln, und wenn es nicht nötig ist, zwischen $\mathbb{R}$ und $\mathbb{C}$ zu unterscheiden, benutzen wir das Symbol $\mathbb{K}$ für einen skalaren Grundkörper, der für $\mathbb{R}$ oder $\mathbb{C}$ steht.

Vektoren aus dem $\mathbb{K}^n$ werden in diesem Buch stets als *Spaltenvektoren* oder $(n \times 1)$-Matrizen verstanden; das hochgestellte Symbol "T" steht für die Transposition von Matrizen und Vektoren. Für $x \in \mathbb{K}^n$ ist daher $x^T = (x_1, \ldots, x_n)$ ein Zeilenvektor und $x = (x_1, \ldots, x_n)^T$ ein Spaltenvektor. Die *Komponenten* von Vektoren werden stets durch tiefgestellte Indizes angedeutet. Allerdings können tiefgestellte Indizes auch eine andere Bedeutung haben. Zum Beispiel werden wir die *Einheitsvektoren* mit $e_i \in \mathbb{K}^n$ bezeichnen; sie sind mit dem *Kronecker*-Symbol δ_{ij} durch

$$e_i^T e_j := \delta_{ij} := \left\{ \begin{array}{ll} 1, & \text{falls} \quad i = j \\ 0, & \text{sonst} \end{array} \right\}, \quad 1 \le i, j \le n,$$

definiert. Die Notation $A = (a_{ij}) \in \mathbb{K}^{m \times n}$ definiert eine $(m \times n)$-Matrix A mit Elementen a_{ij}, $1 \le i \le m$, $1 \le j \le n$, über $\mathbb{K}$. Dabei steht stets der Zeilenindex vorn. Das *Matrizenprodukt* wird sinngemäß auch auf Spaltenvektoren als $(n \times 1)$-Matrizen und auf Zeilenvektoren als $(1 \times n)$-Matrizen angewendet. Beispielsweise kann man die j-te Spalte einer Matrix A als Produkt Ae_j schreiben. Die i-te Zeile von A ist $e_i^T A$, und das Element a_{ij} ist als $e_i^T A e_j$ darstellbar. Für zwei Vektoren $u \in \mathbb{K}^m$, $v \in \mathbb{K}^n$ ist $uv^T := (u_i v_j) \in \mathbb{K}^{m \times n}$ eine $(m \times n)$-Matrix mit Elementen $u_i v_j$, das *dyadische* oder *äußere Produkt*. Die $(n \times n)$-Einheitsmatrix $E := E_n$ ist als die Summe der dyadischen Produkte $e_k e_k^T$ für $k = 1, \ldots, n$ zu schreiben. Aus $A = AE$ folgt dann auch die Darstellung

$$A = AE = A \sum_{k=1}^{n} e_k e_k^T = \sum_{k=1}^{n} (Ae_k) e_k^T \tag{1.19}$$

der Matrix A als "Summe ihrer Spalten".

Das praktische Rechnen mit reellen Zahlen wird normalerweise in Programmiersprachen wie C oder JAVA durch den Datentyp *double* und seine Standardoperationen ausgeführt. Aber wie arbeitet man mit Vektoren und Matrizen? Das wird einige Überlegungen aus der Informatik erfordern.

Seit den Anfängen der elektronischen Rechenanlagen verwendet man für Vektoren intern den indizierten Speicherzugriff und nutzt die wortweise linear adressierbare Struktur des Speichers des von Neumann-Rechners aus. Vektoren werden im effizientesten Idealfall also im Speicher durch lückenlos aneinandergereihte *double*-Zahlen dargestellt. Man nennt diese indizierten Datentypen *arrays*, während der Begriff *Vektor* in objektorientierten Sprachen wie JAVA für eine abstrahierte Klasse steht, die es erlaubt, mit Indexzugriff und dynamischer Speicherverwaltung auf geordnete Listen von Objekten zuzugreifen. Im numerischen Rechnen sind *arrays* immer vorzuziehen, weil Vektor-Klassen eine zusätzliche Dereferenzierung erfordern. Wir gehen im Folgenden immer davon aus, dass Vektoren als *arrays* gespeichert sind.

Weil bei heutigen Rechnern komplizierte hierarchische Speicherzugriffs- und Verarbeitungsmethoden (*Paging*, *Cache*, *Prefetch*, *Pipelining*) fest implementiert sind, sollten alle Zugriffe auf Vektoren oder *arrays datenlokal* ablaufen, d.h. immer auf im Speicher unmittelbar benachbarte Zahlen zugreifen.

Das lässt sich bei Vektoren relativ einfach machen, bei Matrizen aber nicht, denn der von-Neumann-Rechner hat keinen zweidimensionalen Speicher. Man muss Matrizen intern als Vektoren speichern, und das kann man entweder zeilen- oder spaltenweise tun. Eine Matrix $A = (a_{jk}) \in \mathbb{R}^{m\times n}$ kann man vektoriell entweder zeilenweise als

$$(a_{11}, a_{12}, \ldots, a_{1n}, a_{21}, a_{22}, \ldots, a_{2n}, \ldots, a_{m1}, a_{m2}, \ldots, a_{mn})$$

oder spaltenweise als

$$(a_{11}, a_{21}, \ldots, a_{m1}, a_{12}, a_{22}, \ldots, a_{m2}, \ldots, a_{1n}, a_{2n}, \ldots, a_{mn})$$

speichern. Aber schon bei der Matrixmultiplikation $C = A \cdot B$ sieht man das hier versteckte Problem: man muss die Zeilen von A mit den Spalten von B skalar multiplizieren, und das geht nur dann datenlokal und ohne Tricks, wenn man A zeilenweise und B spaltenweise speichert. Weil die einzelnen Programmierumgebungen aber die Speichertechnik für Matrizen fest definieren (in FORTRAN und MATLAB wird spaltenweise gespeichert, in C und JAVA zeilenweise), muss man zu mathematisch-informatischen Tricks greifen, die hier kurz erwähnt werden sollen. Dabei gehen wir davon aus, dass Vektoradditionen und Skalarprodukte $x^T y = y^T x = (x, y)_2$ sich problemlos berechnen lassen.

Zu berechnen sei der Vektor $z = Ax \in \mathbb{R}^m$ als Produkt einer $(m \times n)$-Matrix A mit einem Vektor $x \in \mathbb{R}^n$. Bei zeilenweiser Speicherung von A gibt es keine Probleme, weil man die Komponenten von Ax gemäß $e_k^T Ax = (e_k^T A)x$, $1 \le k \le m$, als Folge von Skalarprodukten von x mit den Zeilen $e_k^T A$ von A ausrechnen kann. Bei spaltenweiser Speicherung von A verwendet man (1.19), um die *column-sweep-Methode*

$$z = Ax = \sum_{k=1}^{n} (Ae_k e_k^T)x = \sum_{k=1}^{n} e_k^T x \cdot Ae_k = \sum_{k=1}^{n} x_k \cdot Ae_k$$

zu definieren, d.h. man summiert die Spalten von A auf, nachdem man sie jeweils mit den Faktoren $x_1, x_2, \ldots, x_n$ multipliziert hat. Vom theoretischen Aufwand her sind die beiden Formen gleich, auf konkreten Rechnern kann das Laufzeitverhalten aber sehr unterschiedlich sein, insbesondere dann, wenn der Speicherbedarf der Matrix den Umfang des Cache oder des physikalischen Hauptspeichers übersteigt.

Will man eine $(\ell \times m)$-Matrix $A = (a_{ik})$ mit einer $(m \times n)$-Matrix $B = (b_{kj})$ multiplizieren, so erfordert die naive Vorgehensweise eine zeilenweise

Speicherung von A und eine spaltenweise Speicherung von B. Man kann das Matrizenprodukt aber bei zeilenweiser Speicherung umschreiben in

$$e_i^T C = e_i^T AB = e_i^T A \sum_{k=1}^{m} e_k e_k^T B = \sum_{k=1}^{m} \underbrace{e_i^T A e_k}_{=a_{ik}} \cdot e_k^T B, \tag{1.20}$$

weil man B als Summe seiner Zeilen

$$B = \sum_{k=1}^{m} e_k e_k^T B$$

analog zu (1.19) darstellen kann. In (1.20) hat man dann eine Summation von skalierten Zeilen von B, um die Zeilen von C auszurechnen. Ganz analog geht das bei spaltenweiser Organisation:

$$Ce_j = ABe_j = \left(\sum_{k=1}^{m} Ae_k e_k^T \right) Be_j = \sum_{k=1}^{m} \underbrace{e_k^T Be_j}_{=b_{kj}} \cdot Ae_k,$$

d.h. die Spalten von C sind gewichtete Summen der Spalten von A.

In praktischen Anwendungen treten nicht selten gigantische Matrizen auf, die allerdings sehr viele Nullen enthalten. Man nennt solche Matrizen *dünn besetzt* oder engl. *sparse*. Man speichert dann die einzelnen Spalten oder Zeilen als dünn besetzte Vektoren, je nach zeilen- oder spaltenweiser Speichertechnik der Matrizen. Und von einem dünn besetzten Vektor $V \in double^N$ speichert man in einem *double-array* $v \in double^n$ nur die $n \ll N$ von Null verschiedenen Komponenten. Deren Indizes hat man dann anderswo zu speichern. Man könnte einfach die Indizes in ein weiteres *int-array* $I \in int^n$ der Länge n setzen und dann mit $v_j = V_{I(j)}$, $1 \le j \le n$, die von Null verschiedenen Komponenten von V durchlaufen. Der Zugriff auf eine einzelne Komponente V_k ist dann zwar nicht so einfach, tritt aber viel seltener auf als das Durchlaufen des ganzen Vektors. Um bei den Indizes Speicherplatz zu sparen, speichert man statt der Indizes in der Regel nur die *offsets* oder Indexsprünge $J(j) := I(j+1) - I(j)$ bis zum nächsten nicht verschwindenden Element. Wenn man den Index des ersten nicht verschwindenden Elements hat, kann man sich damit leicht durch den Vektor "durchhangeln" und hat stets Datenlokalität.

Es sollte nach diesen Bemerkungen klar sein, dass hocheffiziente Verfahren zum Rechnen mit großen Vektoren und Matrizen sehr sorgfältig konzipiert und implementiert sein müssen. Anfänger sollten die Finger davon lassen und sich auf bewährte Programmpakete[1] stützen. Grundlage ist BLAS (Basic Linear Algebra Subprograms)[2] in FORTRAN mit einem C-Interface. Das Projekt

[1] Siehe z.B. `http://www.netlib.org`

[2] `http://www.netlib.org/blas/faq.html`

ATLAS (Automatically Tuned Linear Algebra Software)[3] liefert optimierte Versionen für spezielle Architekturen. Programmpakete, die über die lineare Algebra hinausgehen, sind ohne Anspruch auf Vollständigkeit

- GSL (GNU Scientific Library)[4]: Eine numerische Freeware-Bibliothek in C und C++ unter der *GNU General Public License*.
- IMSL: Eine umfassende FORTRAN-Programmbibliothek mit vielen numerischen Verfahren in den Bereichen Algebra und Analysis.
- IMSL-C/MATH: Eine umfangreiche C-Funktionsbibliothek mit Verfahren in den Bereichen Algebra und Analysis.
- NAG: Eine umfassende Unterprogrammbibliothek für FORTRAN77, FORTRAN90 und C mit vielen numerischen Verfahren in den Bereichen Algebra und Analysis.
- Numerical Recipes: Eine Sammlung von Routinen aus Algebra und Analysis als C- und FORTRAN-Unterprogrammbibliothek.

Diese und andere kann man in Göttingen[5] und über die Rechenzentren vieler anderer Hochschulen abrufen.

Der Umgang mit den obigen Programmpaketen setzt aber das Verständnis ihrer mathematischen Grundlagen voraus, und dazu dient dieses Buch.

Für Projekte, die nicht an die Grenze der Leistungsfähigkeit von Computersystemen gehen, braucht man keine eigene Programmierung auf Ebene der Elemente von Vektoren und Matrizen. Man kann sich auf Programmsysteme wie MAPLE, MATHEMATICA, MATLAB oder MUPAD stützen, die eine eigene Kommandosprache haben, in der man mit Matrizen und Vektoren rechnen kann.

1.5 Aufgaben

1.1 Man schreibe in einer beliebigen Programmiersprache ein kleines Programm, das zu gegebenen Werten x und n die Partialsumme $P_n(x)$ von (1.6) berechnet und ausgibt. Dieses lasse man für verschiedene n und einige betragsmäßig große positive und negative x laufen und vergleiche die Ergebnisse mit den Fehlerabschätzungen (1.7) und (1.8).

1.2 Wie in Beispiel 1.23 untersuche man die Näherungsformel (1.15).

1.3 Man zeige, dass $\varphi(x) = \sqrt{x}$ die Kondition 1/2 hat (bei Weglassen von Produkten von Fehlern).

1.4 Wie kann man

$$\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

[3] `http://sourceforge.net/projects/math-atlas/`
[4] `http://www.gnu.org/software/gsl/`
[5] `http://www.gwdg.de/service/software/software-rz/sw_numerisch.html`

für $x \approx 0$ unter Vermeidung von Auslöschungen berechnen?

1.5 Man zeige, dass $\varepsilon = \frac{1}{2}B^{1-m}$ erreichbar ist, wenn man die letzte der m B-adischen Stellen geeignet auf- oder abrundet. Man gebe eine präzise Formulierung dieser Rundungsstrategie an!

1.6 Man gebe eine im Dezimal-Gleitkommasystem ($B = 10,\ m \geq 1$) exakt darstellbare Zahl an, die in keinem binären Gleitkommasystem ($B = 2^k, m \geq 1$) exakt darstellbar ist.

1.7 Berechnen Sie die Darstellung von $(3417)_{10}$ im Binär-, im Oktal- und im Hexadezimalsystem. Verwenden Sie für das Hexadezimalsystem die Zeichen: $0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F$.

1.8 Berechnen Sie

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

und

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

für $x = 20, x = 50$ und $x = 100$ in dreistelliger dezimaler Gleitkommaarithmetik. Geben Sie jeweils den relativen Fehler an.

1.9 Durch Umformung der entsprechenden Ausdrücke beseitige man die subtraktive Auslöschung bei

a) $\sqrt{1+x^2} - \sqrt{1-x^2}$, x nahe bei 0

b) $\frac{1}{\sqrt{1+x}-\sqrt{1-x}}$, für große x

c) $(1+x^2)^2 - (1-x^2)^2$, x nahe bei 0.

1.10 Sei $A \in \mathbb{R}^{n\times n}$ eine Matrix und $x \in \mathbb{R}^n$ ein Vektor. Welche Komplexitäten, gemessen als Anzahl der arithmetischen Operationen, haben folgende Operationen?

– Vektor-Vektor Multiplikation: $x^T x$
– Vektor-Matrix Multiplikation: Ax
– p-te Potenz einer Matrix: A^p wobei $p \in \mathbb{N}$.

Man drücke das Ergebnis in Abhängigkeit von n durch Landau–Symbole aus.

1.11 Seien $\{a_n\}, \{b_n\}$ zwei Folgen reeller Zahlen. Beweisen Sie folgende Aussagen:

– Ist $a_n = o(b_n)$ für $n \to \infty$, so gilt auch $a_n = \mathcal{O}(b_n)$ für $n \to \infty$.
– Eine Folge $\{a_n\}$ ist genau dann beschränkt, wenn $a_n = \mathcal{O}(1)$ für $n \to \infty$.
– Die Folge $\{a_n\}$ konvergiert genau dann gegen Null, wenn $a_n = o(1)$ für $n \to \infty$.

1.12 Man zeige durch ein Beispiel: Aus $f(x) = o(x^\alpha)$ für $x \to 0$ und ein $\alpha \in \mathbb{R}$ folgt nicht $f(x) = \mathcal{O}(x^{\alpha+1})$ für $x \to 0$.