3 Numerisches Rechnen

3.1 Zahlen und ihre Darstellung

Grundlage der Analysis bilden die reellen Zahlen. Wir sind heute daran gewöhnt, eine reelle Zahl im Dezimalsystem als (unendlichen) Dezimalbruch darzustellen. Grundsätzlich kann jedoch als Basis statt der Zahl 10 jede natürliche Zahl $g \geq 2$ gewählt werden.

Sei $g \geq 2$ eine natürliche Zahl und $x \neq 0$ eine reelle Zahl. Dann gibt es eine Darstellung der Gestalt

$$x \;=\; \sigma \cdot g^N \cdot \sum_{\nu=1}^{\infty} x_\nu g^{-\nu} \tag{6}$$

mit $\sigma \in \{-1, +1\}$ (*Vorzeichen*), $N \in \mathbb{Z}$ (*Exponent*) und $x_\nu \in \{0, 1, 2, \ldots, g-1\}$ (*Ziffern*). Man nennt g die *Basis* des Zahlensystems.

Diese Darstellung ist eindeutig, wenn man zusätzlich noch verlangt, dass $x_1 \neq 0$ gilt und dass es zu jedem $n \in \mathbb{N}$ ein $\nu \geq n$ gibt mit $x_\nu \neq g-1$. Wir sprechen dann von der *normalisierten Darstellung* und schreiben (6) kürzer in der Form

$$x \;=\; \sigma 0.x_1x_2x_3\cdots\; g^N \tag{7}$$

Ohne diese Zusatzvoraussetzung hätte im Zehnersystem (d.h. $g = 10$) die reelle Zahl 1 die Darstellungen

$$\begin{aligned} 1 &= 0.1000... \cdot 10^1 \quad , \\ 1 &= 0.9999... \cdot 10^0 \quad , \\ 1 &= 0.0100... \cdot 10^2 \quad . \end{aligned}$$

Häufig verwendete Basen sind:

Name des Sytems	Basis g	Ziffern
Dualsystem	2	0,1
Dezimalsystem	10	0,1,2,3,4,5,6,7,8,9
Hexadezimalsystem	16	0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

Beispiel: Die reelle Zahl $x = 11.1875$ hat folgende normalisierte Darstellungen

dezimal:	$0.111875 \cdot 10^2$
dual:	$0.10110011 \cdot 2^4$
hexadezimal:	$0.B3 \cdot 16^1$

In Computern wird nicht das Dezimalsystem, sondern das Dualsystem verwendet. Dargestellt werden können dann nur die Null und die Zahlen der Form

$$x \;=\; \sigma \cdot 2^N \cdot \sum_{\nu=1}^{t} x_\nu 2^{-\nu} \;, \qquad x_\nu \in \{0,1\},\; x_1 = 1 \tag{8}$$

mit einem festen $t \in \mathbb{N}$ und $N^- \leq N \leq N^+,\ N^-, N^+ \in \mathbb{Z}$. Dabei hängen $t,\ N^-$ und N^+ von der konkreten Rechenanlage bzw. von den verwendeten Programmpaketen ab. Diese Zahlen heißen *Maschinenzahlen*. Die Zahl

$$m \ := \ \sum_{\nu=1}^{t} x_\nu 2^{-\nu}$$

heißt die *Mantisse* von x und t die *Mantissenlänge*. Daneben bezeichnen wir σ als *Vorzeichen* und N als *Exponent* der Zahl x. Man nennt Zahlen in der Dartellung (8) auch *Gleitkommazahlen*.
Matlab verwendet für die Darstellung von reellen Zahlen 64 Bit (= 8 Byte); diese teilen sich auf in

Vorzeichen	1 Bit
Exponent	11 Bit
Mantisse	52 Bit

3.2 Operationen mit Gleitkommazahlen

Die Menge M der Maschinenzahlen (bei festem $t,\ N^-,\ N^+$) ist natürlich endlich. Man spricht hier auch von einem diskreten Raum. Eine reelle Zahl x muss in der Regel durch eine Maschinenzahl $\bar{x}$ ersetzt (*approximiert*) werden. Diesen Prozess nennt man *Runden*; es entsteht ein Fehler. Dabei unterscheiden wir zwei Fehlerarten:

Definition: Es sei $x \in \mathbb{R}$ und $\bar{x}$ eine Näherung für x.
(i) $x - \bar{x}$ heißt *absoluter Fehler*.
(ii) Für $x \neq 0$ heißt $\frac{x-\bar{x}}{x}$ der *relative Fehler*.

Wir definieren folgende **Rundungsvorschrift:**
Sei $g \geq 2$ gerade (nur diese Fälle sind von praktischer Bedeutung), $t \in \mathbb{N}$ (fest vorgegebene Mantissenlänge) und $x \in \mathbb{R} \setminus \{0\}$ mit

$$x \ = \ \sigma \cdot g^N \cdot \sum_{\nu=1}^{\infty} x_\nu g^{-\nu} \quad (N^- \leq N \leq N^+).$$

Dann setzt man

$$\mathrm{rd}(x) \ := \ \begin{cases} \sigma \cdot g^N \cdot \sum\limits_{\nu=1}^{t} x_\nu g^{-\nu} \quad , & \text{falls } x_{t+1} < \frac{g}{2} \quad \text{(abrunden)}, \\ \sigma \cdot g^N \cdot \left(\sum\limits_{\nu=1}^{t} x_\nu g^{-\nu} + g^{-t} \right) \ , & \text{falls } x_{t+1} \geq \frac{g}{2} \quad \text{(aufrunden)}. \end{cases}$$

$\mathrm{rd}(x)$ heißt der auf t *Stellen gerundete Wert von* x. (Beachte: rd hängt auch von t ab; korrekter wäre deshalb $\mathrm{rd}_t(x)$. Wir verzichten darauf, da wir stets mit einer fest vorgegebenen Mantissenlänge arbeiten.)

Satz: (o.B., vgl. Hämmerlin-Hoffmann)
(i) Für den absoluten Fehler gilt: $|\mathrm{rd}(x) - x| \ \leq \ 0.5 g^{N-t}$
(ii) Für den relativen Fehler gilt: $\left| \frac{\mathrm{rd}(x) - x}{x} \right| \ \leq \ 0.5 g^{-t+1}$

Definition: Die Zahl $\tau := 0.5 g^{-t+1}$ heißt die *relative Rechengenauigkeit* der t-stelligen

Gleitkomma-Arithmetik.

Setzt man $\varepsilon := \dfrac{\mathrm{rd}(x) - x}{x}$, so gilt

$$\mathrm{rd}(x) \;=\; x(1+\varepsilon) \quad \text{mit } |\varepsilon| \le \tau$$

In Matlab ist die relative Rechengenauigkeit $\tau = 2^{-52}$. Diesen Wert erhält man durch Eingabe von `eps`.

Verknüpfung von Gleitkommazahlen
Es bezeichne nun $\diamond$ eine der Rechenoperationen $+, -, *, /$. Die Verknüpfung zweier Maschinenzahlen x und y durch eine dieser Operationen ist i.A. keine Maschinenzahl mehr; es muss gerundet werden. Das Ergebnis ist also

$$\mathrm{rd}(x \diamond y) \;=\; (x \diamond y)(1+\varepsilon) \quad \text{mit } |\varepsilon| \le \tau \ .$$

Beispiel: Sei $g := 10,\ t := 3$

$$\begin{aligned} &x = 0.123{\cdot}10^6\,, \quad y = 0.456{\cdot}10^2 \quad , \\ &x + y = 0.1230456{\cdot}10^6 \quad , \\ &\mathrm{rd}(x+y) = 0.123{\cdot}10^6 \quad . \end{aligned}$$

Sind nun x und y keine Maschinenzahlen, so müssen diese zunächst in Maschinenzahlen umgewandelt werden, bevor die Verknüpfung ausgeführt wird:

$$\begin{aligned} \mathrm{rd}(x) &= x(1+\varepsilon_1) \quad \text{mit } |\varepsilon_1| \le \tau \quad , \\ \mathrm{rd}(y) &= y(1+\varepsilon_2) \quad \text{mit } |\varepsilon_2| \le \tau \quad . \end{aligned}$$

In einem ersten Schritt untersuchen wir nun, wie sich Rundungsfehler bei der Addition auswirken. Wird zusätzlich unterstellt, dass die Addition der gerundeten Zahlen auf dem Rechner exakt ausgeführt wird, so erhält man

$$\mathrm{rd}(x) + \mathrm{rd}(y) \;=\; x + x\varepsilon_1 \;+\; y + y\varepsilon_2$$

und somit für den relativen Fehler

$$\frac{\mathrm{rd}(x) + \mathrm{rd}(y) - (x+y)}{x+y} \;=\; \frac{x}{x+y}\,\varepsilon_1 \;+\; \frac{y}{x+y}\,\varepsilon_2 \;=:\; \delta \ .$$

Bei exakter Rechnung bewirken die relativen Fehler ε_1 bzw. ε_2 bei den Eingangsdaten x und y im Ergebnis einen relativen Fehler δ. Diesen Fehler bezeichnet man auch als *Fortpflanzungsfehler*. Falls x und y dasselbe Vorzeichen haben, so folgt sofort

$$|\delta| \;\le\; |\varepsilon_1| + |\varepsilon_2| \;\le\; 2\tau \quad .$$

Problematisch ist dagegen die Situation $x \approx -y$. Dann kann der relative Fehler bei der Addition sehr groß werden (trotz kleiner Fehler ε_1 und ε_2).

Fazit: Die Subtraktion zweier ungefähr gleich großer Zahlen ist **numerisch instabil**: kleine Fehler in den Summanden können zu einem großen Fehler im Ergebnis führen.

Nun betrachten wir die Situation, dass nach der Rechenoperation des Ergebnis zu einer Maschinenzahl gerundet werden muss. Dann erhält man

$$\begin{aligned} \mathrm{rd}(\mathrm{rd}(x) \diamond \mathrm{rd}(y)) &= (\mathrm{rd}(x) \diamond \mathrm{rd}(y))(1+\varepsilon_3) \quad \text{mit } |\varepsilon_3| \leq \tau \\ &= (x(1+\varepsilon_1) \diamond y(1+\varepsilon_2))(1+\varepsilon_3) \\ &= x \diamond y \ + \ F \quad . \end{aligned}$$

Verwendet man z.B. die Addition als Verknüpfung, so ergibt sich für den absoluten Fehler die Abschätzung (Beweis als Übung)

$$\begin{aligned} |F| &\leq |x|(\tau + \tau(1+\tau)) \ + \ |y|(\tau + \tau(1+\tau)) \\ &= (|x| + |y|)(\tau^2 + 2\tau) \quad . \end{aligned}$$

Die Untersuchung des Rundungsfehlers bei der Auswertung arithmetischer Ausdrücke nennt man *Rundungsfehleranalyse*. Sie wird bei größeren Ausdrücken sehr kompliziert.

3.3 Algorithmen

Eine Reihe (unendliche Summe) kann man im Computer nicht exakt berechnen, sondern muss zuerst durch eine endliche Summe ersetzt werden. Desweiteren ist dem Rechner auch mitzuteilen, in welcher Reihenfolge die Summe ausgewertet werden soll. Dies führt uns auf den Begriff des *Algorithmus*.

Definition: Ein *Algorithmus* ist eine Vorschrift, die aus einer Menge eindeutiger Regeln besteht. Diese spezifizieren eine endliche Aufeinanderfolge von Operationen, so dass deren Ausführung die Lösung eines Problems aus einer bestimmten Problemklasse liefert.

Ein Algorithmus ist also durch die folgenden 3 Punkte gekennzeichnet:

- Bestimmtheit: Jeder Rechenschritt ist eindeutig festgelegt. Jede mögliche Situation wird erfasst, und es muss genau spezifiziert werden, welcher Rechenschritt auszuführen ist.
- Endlichkeit: Der Rechenvorgang wird nach endlich vielen Schritten beendet.
- Allgemeingültigkeit: Die Rechenvorschrift soll auf eine ganze Problemklasse anwendbar sein, wobei die Lösung der verschiedenen Probleme der Klasse lediglich eine Änderung der Eingabe erfordert.

Komplexität von Algorithmen
Zur Lösung desselben Problems kann es verschiedene Algorithmen geben. Ein mögliches Vergleichskriterium ist dann der Rechenaufwand. Mit diesem Themenkreis beschäftigt sich die *Komplexitätstheorie*.

Definition: Sei A ein Algorithmus zur Lösung eines Problems mit $n \in \mathbb{N}$ Eingabedaten. Die Abbildung $T_A : \mathbb{N} \to \mathbb{N}$, die jeder Anzahl von Eingabedaten die Anzahl der Grundoperationen beim Ablauf des Algorithmus zuordnet, heißt *Komplexität* von A.

Die Komplexität gibt den Rechenaufwand an. Da eine Punktoperation ($*$, $/$) wesentlich mehr Aufwand erfordert als eine Strichoperation ($+$, $-$), werden bei Komplexitätsbetrachtungen häufig auch nur die Punktoperationen berücksichtigt.

Zur Beschreibung des Verhaltens der Komplexitätsfunktion verwendet man meistens das *Landau-Symbol* $\mathcal{O}$.

Definition: Es seien $f, g : D \subset \mathbb{R} \to \mathbb{R}$ zwei Funktionen mit $g(x) \neq 0$ für alle $x \in D$. f heißt *von der Ordnung "groß O" von g für* $x \to x_0$, wenn es eine Konstante $C > 0$ und ein $\delta > 0$ gibt, so dass für alle $x \in \mathrm{U}_\delta(x_0)$ mit $x \neq x_0$ die Ungleichung

$$\left|\frac{f(x)}{g(x)}\right| \leq C$$

gilt. Man verwendet dafür die Schreibweise $f(x) = \mathcal{O}(g(x))$ für $x \to x_0$.

Beispiel: Sei $f : \mathbb{N} \to \mathbb{N}$, $f(n) := \frac{1}{2}n(n+3)$. Dann gilt

$$f(n) = \mathcal{O}(n^2) \quad \text{für } n \to \infty$$

(d.h. $g(n) := n^2$), denn

$$\frac{f(n)}{n^2} = \frac{\frac{1}{2}n^2 + \frac{3}{2}n}{n^2} = \frac{1}{2} + \frac{3}{2n} \leq 1 \quad \text{für alle } n \geq 3 \ .$$

Hier kann man also $C = 1$ wählen.

Eigenschaften des Landau-Symbols

1. $f(x) = K \cdot \mathcal{O}(g(x))$ für ein $K \in \mathbb{R} \Rightarrow f(x) = \mathcal{O}(g(x))$
2. $f(x) = \mathcal{O}(g(x))$ und $g(x) = \mathcal{O}(h(x)) \Rightarrow f(x) = \mathcal{O}(h(x))$
3. $f_1(x) = \mathcal{O}(g_1(x))$ und $f_2(x) = \mathcal{O}(g_2(x)) \Rightarrow f_1(x) \cdot f_2(x) = \mathcal{O}(g_1(x) \cdot g_2(x))$

Beispiel zur Berechnung der Komplexität:
Gegeben sei ein Polynom vom Grad n:

$$p(t) := \sum_{i=0}^{n} a_i t^i \tag{9}$$

und $\xi \in \mathbb{R}$. Man bestimme den Funktionswert $p(\xi)$.

Der naive Algorithmus zur Polynomauswertung lautet:
Eingabe: $n \in \mathbb{N}$, $(a_0, a_1, \ldots, a_n) \in \mathbb{R}^{n+1}$, $\xi \in \mathbb{R}$.
Rechenvorschrift:

$p := a_0$
Für $i = 1, \ldots, n$:
  $b := a_i$
  Für $j = 1, \ldots, i$:
    $b := b * \xi$
  {Ende der j-Schleife}
  $p := p + b$
{Ende der i-Schleife}

Ausgabe: p.

Man berechnet also

$$a_0 + a_1 \cdot \xi + a_2 \cdot \xi \cdot \xi + a_3 \cdot \xi \cdot \xi \cdot \xi + \cdots$$

Dieser Algorithmus hat bei Zählung von Additionen und Multiplikationen die Komplexität

$$T_A(n) \;=\; \frac{1}{2}n(n+3) \;=\; \mathcal{O}(n^2) \quad (\text{für } n \to \infty).$$

(nämlich n Additionen und

$$\sum_{i=1}^{n} i \;=\; \frac{1}{2}n(n+1)$$

Multiplikationen.)

3.4 Das Hornerschema

Wir geben nun einen Algorithmus mit kleinerer Komplexität an (zur Berechnung eines Funktionswertes eines Polynoms). Gegeben sei wieder die Situation aus (9):

$$p(t) \;=\; \sum_{i=0}^{n} a_i t^i \;= \sum_{i=0}^{n} a_i^{(0)} t^i \qquad .$$

Setze

$a_n^{(1)} := a_n^{(0)}$
Für $i = 1, \ldots, n$:
    $a_{n-i}^{(1)} := a_{n-i}^{(0)} + a_{n+1-i}^{(1)} * \xi$

Behauptung: Hier gilt $p(\xi) = a_0^{(1)}$ (Beweis als Übungsaufgabe).

Der Rechenaufwand beträgt in diesem Fall n Multiplikationen und n Additionen:

$$T_A(n) \;=\; 2n \;=\; \mathcal{O}(n) \quad (\text{für } n \to \infty).$$

Für die Handrechnung benutzt man folgendes Schema (*Hornerschema*):

$a_n^{(0)}$	$a_{n-1}^{(0)}$	$a_{n-2}^{(0)}$	$\ldots$	$a_1^{(0)}$	$a_0^{(0)}$	
–	$+a_n^{(1)}\xi$	$+a_{n-1}^{(1)}\xi$	$\ldots$	$+a_2^{(1)}\xi$	$+a_1^{(1)}\xi$	
$a_n^{(1)}$	$a_{n-1}^{(1)}$	$a_{n-2}^{(1)}$	$\ldots$	$a_1^{(1)}$	$a_0^{(1)}$	$= p(\xi)$

Beispiel: Es sei $p(t) := t^5 - 5t^3 + 4t + 1$ und $\xi := 2$.

1	0	–5	0	4	1	
–	2	4	–2	–4	0	
1	2	–1	–2	0	1	$= p(2)$

Wir können nun das Hornerschema mehrfach anwenden und erhalten dann

$a_n^{(0)}$	$a_{n-1}^{(0)}$	$a_{n-2}^{(0)}$	$\dots$	$a_1^{(0)}$	$a_0^{(0)}$	
$-$	$+a_n^{(1)}\xi$	$+a_{n-1}^{(1)}\xi$	$\dots$	$+a_2^{(1)}\xi$	$+a_1^{(1)}\xi$	
$a_n^{(1)}$	$a_{n-1}^{(1)}$	$a_{n-2}^{(1)}$	$\dots$	$a_1^{(1)}$	$a_0^{(1)}$	$= p(\xi)$
$-$	$+a_n^{(2)}\xi$	$+a_{n-1}^{(2)}\xi$	$\dots$	$+a_2^{(2)}\xi$		
$a_n^{(2)}$	$a_{n-1}^{(2)}$	$a_{n-2}^{(2)}$	$\dots$	$a_1^{(2)}$	$= p'(\xi)$	
$\vdots$	$\vdots$	$\vdots$	$\vdots$			
$\vdots$	$\vdots$	$\vdots$				
$a_n^{(n-1)}$	$a_{n-1}^{(n-1)}$	$a_{n-2}^{(n-1)}$				
$-$	$+a_n^{(n)}\xi$					
$a_n^{(n)}$	$a_{n-1}^{(n)}$					
$-$						
$a_n^{(n+1)}$						

Dieses Tableau wird als *vollständiges Hornerschema* bezeichnet.

Es gilt die Beziehung

$$a_\nu^{(\nu+1)} = \frac{p^{(\nu)}(\xi)}{\nu!} \qquad (\nu = 0, \dots, n)$$

(Beweis als Übungsaufgabe).

Anmerkung: Mit dem vollständigen Hornerschema können wir von einem Polynom die Taylor-Entwicklung um den Punkt ξ berechnen, ohne die Ableitungen explizit zu bestimmen.

Beispiel: Bestimmen Sie die Taylor-Entwicklung von

$$p(x) = 3x^4 + 2x - 10$$

um den Punkt $x_0 = -1$.

Hier erhalten wir das folgende vollständige Hornerschema

3	0	0	2	-10	
$-$	-3	3	-3	1	
3	-3	3	-1	-9	$= p(-1)$
$-$	-3	6	-9		
3	-6	9	-10	$= p'(-1)$	
$-$	-3	9			
3	-9	18	$= \frac{p^{(2)}(-1)}{2!}$		
$-$	-3				
3	-12	$= \frac{p^{(3)}(-1)}{3!}$			
$-$					
3	$= \frac{p^{(4)}(-1)}{4!}$				

Damit ergibt sich die Taylor-Entwicklung

$$p(x) = 3(x+1)^4 - 12(x+1)^3 + 18(x+1)^2 - 10(x+1) - 9 \, .$$

Literatur: Hämmerlin; Hoffmann: Numerische Mathematik, 4. Auflage, Springer 1994.