Ergänzung zum Kapitel: Numerische Fehler und Grenzen

Als Ausgangspunkt der Diskussion betrachten wir ein einfaches Problem: wir möchten eine quadratische Gleichung der Form

$$0 = x^2 + px + q \tag{1}$$

lösen. Die Lösungen können wir leicht mit Hilfe quadratischer Ergänzung konstruieren ("pq-Formel"):

$$x_{\pm} = -\frac{p}{2} \pm \sqrt{\frac{p^2}{4} - q} \tag{2}$$

Die Umsetzung in eine Matlab-Funktion könnte wie folgt aussehen:

```
function [ xp xm ] = SolveEq( p,q )
% Naive Loesung einer quadratischen Gleichung der
% Form x^2 + p*x + q = 0 mit pq-Formel
%    Input: p,q
%    Output: xp, xm = -p/2 +/- sqrt(p^2/4-q)
xp = -p/2 + sqrt(p^2/4-q);
xm = -p/2 - sqrt(p^2/4-q);
end
```

Um die Funktion zu testen, wollen wir unterschiedliche q-Werte betrachten und die berechnete Lösung überprüfen, indem wir die vermeintliche Lösung in die ursprüngliche Gleichung einsetzen. Nachstehend ist ein kleines Matlab-Programm einschließlich der Ausgabe gezeigt:

```
% Wende pq-Formel aus SolveEq() auf unterschiedliche
% q Werte an, studiere Qualitaet der Loesung
% Warum genuegt es q zu variieren?
clc
clear
p=1;
disp('Loese quadratische Gleichung fuer p = 1 und unterschiedliche q')
fprintf('q              xm          xm^2+p*xm+q      xp          xp^2+p*xp + q\n'
fprintf('-------------------------------------------------------------------\n')
for i=1:16
    q=10^(-i);
[xp,xm] = SolveEq(p,q);
fprintf('%5.0e %20.15e %10.1e %20.15e %10.1e\n',q, xm, xm^2+p*xm+q, xp,...
    xp^2+p*xp+q)
end
```

```
Loese quadratische Gleichung fuer p = 1 und unterschiedliche q
q            xm        xm^2+p*xm+q     xp         xp^2+p*xp + q
-----------------------------------------------------------
1e-01 -8.872983346207417e-01    2.8e-17 -1.127016653792583e-01    1.4e-17
1e-02 -9.898979485566356e-01   -8.7e-18 -1.010205144336440e-02   -2.1e-17
1e-03 -9.989989979949860e-01    1.1e-16 -1.001002005014018e-03    2.4e-17
1e-04 -9.998999899979994e-01   -1.0e-16 -1.000100020004946e-04    5.6e-18
1e-05 -9.999899998999979e-01   -6.6e-17 -1.000010000201668e-05   -1.7e-17
1e-06 -9.999989999990000e-01   -2.9e-17 -1.000001000006634e-06   -4.6e-18
1e-07 -9.999998999999899e-01   -5.8e-17 -1.000000099948828e-07    5.1e-18
1e-08 -9.999999899999998e-01   -5.0e-17 -1.000000010575874e-08   -5.8e-18
1e-09 -9.999999989999999e-01   -8.3e-17 -1.000000027229220e-09   -2.6e-17
1e-10 -9.999999999000000e-01   -8.3e-18 -1.000000082740371e-10   -8.3e-18
1e-11 -9.999999999900000e-01   -8.3e-19 -1.000000082740371e-11   -8.3e-19
1e-12 -9.999999999989999e-01   -8.9e-17 -1.000033389431110e-12   -3.3e-17
1e-13 -9.999999999999000e-01   -3.1e-17 -1.000310945187266e-13   -3.1e-17
1e-14 -9.999999999999900e-01    8.0e-18 -9.992007221626409e-15    8.0e-18
1e-15 -9.999999999999990e-01    8.0e-19 -9.992007221626409e-16    8.0e-19
1e-16 -9.999999999999999e-01   -1.1e-17 -1.110223024625157e-16   -1.1e-17
```

Betrachten wir das Ergebnis so stellen wir zunächst fest, dass wir im Unterschied zur naiven Erwartung keine 0 finden, wenn die Lösung in den ursprünglichen Ausdruck eingesetzt wird. Für x_- finden wir, dass x_- von der Größenordnung 1 ist. Setzten wir diese Werte in die Ausgangsgleichung ein so ist das Ergebnis typischerweise um $\sim$16 Größenordnungen kleiner. Aufgrund der beschränkten Rechengenauigkeit können wir dies im Vergleich zur 1 als 0 interpretieren. Ein detailierteres Verständnis erhält man folgendermaßen. Im Computer wird Fließkomma-Arithmetik verwendet. In der Fließkommadarstellung stellen wir ganzzahligen Anteil und Nachkommarest wie folgt dar:

$$\begin{aligned}(12345,678)_{10} &= 1\times 10^4 + 2\times 10^3 + 3\times 10^2 + 4\times 10^1 + 5\times 10^0 \\ &+ 6\times 10^{-1} + 7\times 10^{-2} + 8\times 10^{-3} \end{aligned} \tag{3}$$

bzw. allgemein

$$\begin{aligned}(b_3b_2b_1b_0, b_{-1}b_{-2}b_{-3}\ldots)_\beta &= b_3\times\beta^3 + b_2\times\beta^2 + b_1\times\beta^1 + b_0\times\beta^0 \\ &+ b_{-1}\times\beta^{-1} + b_{-2}\times\beta^{-2} + b_{-3}\times\beta^{-3} + b_{-4}\times\beta^{-4}\end{aligned} \tag{4}$$

dabei ist die Basis β frei wählbar. Man kann natürlich leicht zwischen den unterschiedlichen Darstellungen umrechnen, z.B. findet man

$$(47,11)_{10} = (101111,0001110000101\ldots)_2 \tag{5}$$

Im Alltag verwenden wir typischerweise das Dezimalsystem also $\beta = 10$. Im Computer ist es vorteilhaft im Zweiersystem zu arbeiten, also $\beta = 2$ zu verwenden. In der normalisierten Exponentialschreibweise macht man die Größenordnung explizit indem man die Größenordnung abfaktorisiert z.B.:

$$(b_1 b_0, b_{-1} b_{-2} \ldots)_\beta = \underbrace{(+b_1 \times \beta^{-4} + b_0 \times \beta^{-5} + \ldots)}_{q} \times \beta^5 = q \times \beta^m \qquad (6)$$

q bezeichnet man dann als Mantisse und m als Exponenten. Bzw. im Beispiel:

$$(0,4711)_{10} \times 10^2 = (0,1011110001110000101\ldots)_2 \times 2^6 \qquad (7)$$

Normalisiert heißt in diesem Kontext, dass q keinen ganzzahligen Anteil besitzt. Ansonsten wäre die Darstellung nicht eindeutig da z.B.

$$(a_1\beta^0 + a_2\beta^{-1} + \ldots)_\beta \times \beta^4 = (a_1\beta^1 + a_2\beta^0 + \ldots)_\beta \times \beta^3 \qquad (8)$$

gilt. Man versteht leicht warum das Binärsystem für den Computer vorteilhaft ist: Die Koeffizienten in der Mantisse können nur die Werte 0 oder 1 annehmen. Dies ist genau durch ein Bit darstellbar. Da man im Computer nur endlich viel Speicherplatz zur Verfügung hat muß man eine Entscheidnung treffen wie viel Speicherplatz für die Mantisse und wie viel Speicherplatz für den Exponenten verwendet werden soll. Im Industriestandard IEEE 754 ist für einfach und doppelt genaue Fließkommazahlen die Aufteilung wie folgt festgelegt: Im Fall von einfach genau-

Type	Vorzeichen	Mantisse	Exponent	Summe	Bereich
einfach genau	1 Bit	23 Bit	8 Bit	32Bit=4Byte	$10^{-38} - 10^{38}$
doppelt genau	1 Bit	52 Bit	11 Bit	64Bit=8Byte	$10^{-308} - 10^{308}$

Tabelle 1: Fließkommastandard IEEE 754 (1985)

en Zahlen kann man für den Exponenten $m = -127 \ldots 128$ haben. Das entspricht grob dem Bereich $2^{-127} = 0.5 \times 10^{-38} - 2^{128} = 3 \times 10^{38}$. Für doppelt genaue Zahlen hat man mit 11 Bit für den Exponenten die Möglichkeit $2^{11} = 2048$ verschiedene Zahlen darzustellen. Aufgeteilt in $m = -1023 \ldots 1024$ ergibt sich der Bereich $10^{-308} - 10^{308}$. Wichtig ist noch, dass aufgrund des endlichen Speicherplatzes nur endlich viele Zahlen exakt dargestellt werden können. Alle anderen Zahlen müssen durch Rundung genähert werden. Beispielsweise ist $(0.1)_{10}$ im Binärsystem periodisch und somit nicht exakt darstellbar. Die dadurch bedingte Unsicherheit schlägt sich in der letzten Binärstelle nieder. Die 52(53) Binärstellen entsprechen 15-16 Dezimalstellen. Rechnet man mit der Basis β und nimmt als letzte Stelle β^{-n} so ist offenbar der relative Rundungsfehler von der Größenordnung $\beta^{-n}/2$. Für $\beta = 2$ und $n = 52$ ergibt sich ein relativer Fehler von 10^{-16}, eben

die erwähnten 15-16 Stellen im Dezimalsystem. (Die 15-16 Dezimalstellen kann man auch wie folgt verstehen: Man braucht $\log_2(10) \approx 3.3$ Binärstellen um eine Dezimalstelle zu kodieren, $53/3.3 \approx 16$). Zurück zum Beispiel: Offenbar kommt es beim Einsetzen der Lösung x_- zu einer Kürzung. Es werden Zahlen gleicher Größenordnung subtrahiert—wir erwarten schließlich 0 zu erhalten. Werden 15-16 Dezimalstellen gekürzt, dann reicht die Rechenengenauigkeit nicht aus, um das Ergebnis anzugeben. Dies tritt im Beispiel auf. Die Einsetzung liefert einen Wert der um 16 Größenordnungen kleiner ist und somit als Null interpretiert werden kann. Würde eine höhere Rechengenauigkeit zur Verfügung stehen wäre das Ergebnis entsprechend kleiner aber immer noch im allgemeinen ungleich Null. Quantitativ hat man: Tritt bei einer Subtraktion eine Kürzung um n Dezimalstellen auf also

$$x = A - B \text{ mit } |x|/A = O(10^n) \tag{9}$$

wobei A und B vergleichbare Größenordnung haben sollen, so gilt für den relativen Fehler von x:

$$|\delta x|/|x| \approx \frac{\varepsilon A + \varepsilon B}{|x|} \approx 2\varepsilon \frac{A}{|x|} = 2\varepsilon 10^n, \tag{10}$$

wobei ε der relative Fehler von A und B ist. Für $\varepsilon \approx 10^{-16}$ ist der relative Fehler von x bei einer Kürzung um 16 Stellen $(n = 16)$ dann gerade von der Größenordnung 1!
Was passiert im Falle von x_+? Die Einsetzung liefert im wesentlichen Zahlen von der gleichen Größenordnung wie x_+. Der berechnete Wert für x_+ ist damit offenbar keine Lösung der quadratischen Gleichung. Was ist passiert? Schauen wir uns die Formel zur Berchnung von x_+ an so sehen wir, dass für sehr kleine q die Wurzel im wesentlichen $p/2$ liefert. Bei der Subtraktion reicht dann die Genauigkeit nicht mehr aus um die Differenz mit der erforderlichen Genauigkeit zu berechnen. Rechnet man mit höherer Genauiggkeit (z.B. 4-fache Genauigkeit "quadrupel precision") so verschiebt sich das Problem nur zu kleineren q. Eine Alternative wäre die Wurzel als Taylorreihe zu entwickeln und die Kürzung analytisch vorzunehmen. Eleganter ist auszunutzen, dass aus

$$x^2 + px + q = (x - x_-)(x - x_+) \tag{11}$$

die Relation

$$q = x_+ x_- \tag{12}$$

folgt. Kennt man x_- gut, so kann dann x_+ mit

$$x_+ = q/x_- \tag{13}$$

berechnet werden. Die Modifikation des Programmes liefert dann:

```
Loese quadratische Gleichung fuer p = 1 und unterschiedliche q
q            xm        xm^2+p*xm+q     xp         xp^2+p*xp + q
-----------------------------------------------------------------
1e-001 -8.872983346207417e-001   2.8e-017 -1.127016653792583e-001   0.0e+000
1e-002 -9.898979485566356e-001  -8.7e-018 -1.010205144336438e-002   0.0e+000
1e-003 -9.989989979949860e-001   1.1e-016 -1.001002005014042e-003   0.0e+000
1e-004 -9.998999899979995e-001  -1.0e-016 -1.000100020005002e-004   0.0e+000
1e-005 -9.999899998999979e-001  -6.6e-017 -1.000010000200005e-005   0.0e+000
1e-006 -9.999989999990000e-001  -2.9e-017 -1.000001000002000e-006   0.0e+000
1e-007 -9.999998999999900e-001  -5.8e-017 -1.000000100000020e-007  -1.3e-023
1e-008 -9.999999899999998e-001  -5.0e-017 -1.000000010000000e-008  -1.7e-024
1e-009 -9.999999989999999e-001  -8.3e-017 -1.000000001000000e-009  -2.1e-025
1e-010 -9.999999999000000e-001  -8.3e-018 -1.000000000100000e-010   0.0e+000
1e-011 -9.999999999900000e-001  -8.3e-019 -1.000000000010000e-011   0.0e+000
1e-012 -9.999999999989999e-001  -8.9e-017 -1.000000000001000e-012   0.0e+000
1e-013 -9.999999999999000e-001  -3.1e-017 -1.000000000000100e-013  -1.3e-029
1e-014 -9.999999999999900e-001   8.0e-018 -1.000000000000010e-014   0.0e+000
1e-015 -9.999999999999990e-001   8.0e-019 -1.000000000000001e-015   0.0e+000
1e-016 -9.999999999999999e-001  -1.1e-017 -1.000000000000000e-016   0.0e+000
```

Die endliche Genauigkeit muß bei der Formulierung und Implementierung der Algorithmen berücksichtigt werden.