Notiz zur Genauigkeit der Arithmetik in Rechenanlagen

Elektron. Rechenanl. 22 (1980), H. 5, S. 243–244
Manuskripteingang: 17. April 1980

von S. M. RUMP
Institut für Angewandte Mathematik, Universität Karlsruhe

Die folgende Notiz zur Genauigkeit der Arithmetik in Rechenanlagen ist ein Appell an den Benutzer. Sie entstand aus der sich beinahe täglich wiederholenden Erfahrung, daß auch elektronische Rechenanlagen fehlerhafte Ergebnisse liefern. Und zwar nicht durch einen Fehler des Benutzers verursacht, sondern vielmehr eine fehlerhafte Arithmetik der Maschine, einen fehlerhaften Compiler u. v. m. Ich halte diesen Appell für um so dringender notwendig, da sich immer mehr beinahe blind auf Ergebnisse elektronischer Rechenanlagen verlassen wird. Es wird gar die Tatsache, ein Ergebnis auf einer elektronischen Rechenanlage berechnet zu haben, als Beweis für die Richtigkeit dieses Ergebnisses angesehen.

Daß dem keineswegs so ist, wird im Folgenden an den Erfahrungen mit einer spezifischen Rechenanlage demonstriert. Es sei ausdrücklich darauf hingewiesen, daß diese Anlage keinen Einzelfall sondern vielmehr den Regelfall repräsentiert.

Über Rundungsfehler in modernen Großrechenanlagen und deren Folgen ist schon viel geschrieben worden. Viele glaubten, Geheimrezepte geben zu können und fügten damit ihrer eigenen Sache vielleicht mehr Schaden als Nutzen zu. Es sind auch genügend Beispiele gegeben worden, wo eben durch Rundungsfehler tragisch falsche Ergebnisse errechnet wurden, durch die Millionenschäden verursacht wurden. Doch solche Unstimmigkeiten pflegt man lieber totzuschweigen. Genauso fortdauernd wurde die Forderung nach einer sauberen Arithmetik gestellt. Doch die Computer-Hersteller scheinen sich hartnäckig zu weigern, auch nur ihre Arithmetiken sauber zu beschreiben, geschweige denn mathematische Eigenschaften anzugeben.

An dieser Stelle sei über ein paar persönliche Erfahrungen mit einer weitverbreiteten Großrechenanlage berichtet. Die Unstimmigkeiten fangen schon bei so einfachen Dingen in FORTRAN an wie dem Vergleich von Zahlen. Ist a bzw. da größer als die Hälfte der dem Absolutbetrag nach größte einfachlange bzw. doppeltlange Maschinenzahl m bzw. dm, so führt die Anweisung IF (da.GT.$-da$).. zu einem Überlauf, während bei IF (a.GT.$-a$) nichts passiert. Es wird nämlich im doppeltlangen Fall $da-(-da) = 2da > dm$ mit 0 verglichen, d.h., soweit kommt es gar nicht mehr. Im einfachlangen Fall wird der Vergleich richtig durchgeführt. Manche Compiler ziehen es vor, aus Geschwindigkeitsgründen den Überlauf gar nicht erst bekanntzugeben, sondern kommentarlos weiterzurechnen. In diesem Fall ist die Folge irgendwelcher Unsinn. Ebensooft wird die Division durch Null kommentarlos zur Kenntnis genommen ohne Reaktion seitens des Compilers, so daß der beliebte Abbruch $a = a/(1-1)$ (z. B. wenn man einen Dump erzeugen möchte) bei diesen Compilern gar nicht zum Abbruch führt. Wo wir schon bei Vergleichen sind, auch mit kleinen Zahlen gibt es Schwierigkeiten. Ist e der kleinste zulässige Exponent einer Gleitkommazahl, so ist der Vergleich von $b = 1.0_{10}$e und $c = 1.1_{10}$e mit Vorsicht zu genießen. Der Teufel steckt im Detail: so sind die Anweisungen IF(a.LT.b) GOTO 10 und IF($a-b$) 10,, nicht gleichwertig. Denn wo im ersten Fall die Aussage $a < b$ noch als wahr eingestuft wird, ist im zweiten Fall auf einmal $a = b$, da $a-b$ auf dem Rechner gleich Null wird.

Nun wird man vom Rechner nichts Unmögliches verlangen. Gegen die Berechnung von ⅓ ist nun einmal im Dualsystem kein Kraut gewachsen. Und $(10^{20}+1) - 10^{20}$ ist auf dem Rechner nun mal 0 und nicht 1. Im Umgang mit Fehleranalysen wird eine relative Rundungsfehlereinheit definiert. Rechnet man also mit 9 Dezimalstellen ⅓ aus, so sollte der Fehler der Zahl, die vom Rechner geliefert wird, höchstens einen Fehler von 10^{-9} haben (bei Rundung zur nächstgelegenen Gleitkommazahl sogar höchstens $0.5 \cdot 10^{9}$). Eine vernünftige Forderung, oder was würden Sie sagen wenn das Ergebnis 0.333333332 wäre? Was der Division recht ist, sollte der Subtraktion billig sein. Man braucht ja nur die Ziffern untereinander zu schreiben, abziehen etc., kein Problem. Was würden Sie jedoch sagen, wenn auf einmal zwei minus eins gleich zwei wäre? Unmöglich, oder doch?! Die Meinung wandelt sich jedoch bei der Berechnung von

```
  134217728.0
- 134217727.0
```

Die Zahlen sind exakt darstellbar auf unserem Rechner (*ohne* Konvertierungsfehler). Doch, siehe da, das Ergebnis ist 2 und nicht 1, wie der unvoreingenommene Leser vermuten könnte und nicht nur FORTRAN, auch in ALGOL, und BASIC und SIMULA und ...

Zugegeben, die Beispiele wurden auf einer bestimmten Rechenanlage gerechnet. Doch kennen Sie die Schwachstellen Ihrer Maschine??

P.S.: Insbesondere das letzte, exorbitante Beispiel bewog mich, diese Notiz zu schreiben. Der Grund für den katastrophalen Fehler ist einfach. In Dualdarstellung lautet die Aufgabe

```
  100 ... 0
- 11 ... 11
```

Auf der Maschine wird nun die letzte 1 des Subtrahenden abgeschnitten, das Ergebnis ist folglich im Dualsystem 10, also 2 im Zehnersystem. Aber genauso einfach ist die Misere zu beheben, nämlich indem der Rest 1 vom Ergebnis abgezogen wird, das (richtige) Ergebnis ist 1. Man beachte, daß es sich hier um einen echten Maschinenfehler handelt. Die Arithmetik ist schlichtweg falsch, der relative Fehler des errechneten Ergebnisses in bezug auf die exakte Lösung 1 ist 100%! Der Fehler ist streng zu unterscheiden von Compilerfehlern. So steht bei der einfachen Anweisung

$a := 134217727.0;$

in SIMULA in der Gleitkommavariablen a nicht 134217727.0 (wie man geneigt ist zu glauben), sondern fatalerweise die Zahl

132120583.0.

Und dies obwohl die Zahl 134217727.0 *ohne* Konvertierungsfehler, d.h. exakt auf der Rechenanlage darstellbar ist. Gar nicht auszudenken, was solche Fehler für Folgen haben können. Das ist wohl auch der Grund, warum keine Garantie für die Ergebnisse übernommen wird, die Rechenanlagen berechnen und sämtliche Gewährleistungsansprüche ausdrücklich ausgeschlossen sind. Die Frage bleibt, was zu tun ist (oder was geschähe mit einem Taschenrechner, der

$1\,000\,000 - 999\,999 = 2$

berechnet?).

Der obige Maschinenfehler ist übrigens nicht der einzige; hinter dem Beispiel steckt eine ganze Fülle von exorbitanten Rechenfehlern, wobei jeweils beide Operanden exakt, d.h. ohne Konvertierungsfehler darstellbar sind. So hat auf unserer Rechenanlage im wesentlichen *jede* Addition und Subtraktion einen Fehler größer als die relative Rundungsfehlereinheit, bei der der Exponent des Ergebnisses ungleich dem Maximum der Exponenten der Operanden ist!

Hinter dem Beispiel steckt die Forderung nach einer mathematisch sauber definierten Arithmetik. Da eine reelle Zahl nun einmal *i.a.* auf der Rechenanlage nicht exakt darstellbar ist, wird eine Abbildung $\square$ der reellen Zahlen $\mathbf{R}$ in die „Rechnerzahlen" T definiert $\square : \mathbf{R} \to T$. Sie gibt an, wie eine reelle Zahl zu einer Gleitkommazahl gerundet wird und heißt daher Rundung. Trivialerweise sollten Rechenzahlen auf sich selbst abgebildet werden, also

$\square(t) = t$ für $t \in T$.

Es ist zu unterscheiden zwischen

den reellen Operationen $+, -, \cdot, / : \mathbf{R} \to \mathbf{R}$ und
den „Rechneroperationen" $\oplus, \ominus, \odot, \oslash : T \to T$.

Bei der Definition der Arithmetik wird man bestrebt sein, einen Homomorphismus weitgehendst anzustreben. Daher scheint die Definition

$a \circledast b := \square(a * b)$ für $a, b \in T, * \in \{+, -, \cdot, /\}$

vernünftig. Daraus folgt insbesondere, was wieder trivial erscheint, daß für $a * b \in T$ auch $a \circledast b = a * b$ gilt, d.h. liegt $a * b$ das Verknüpfungsergebnis zweier Rechnerzahlen a und b sogar in T, ist dies auch das Ergebnis der Rechneroperation $a \circledast b$. Gerade diese Forderung ist aber auf unserer Rechenanlage nicht erfüllt. Für die Fülle von Folgerungen, die mit sauber beschriebenen Arithmetiken möglich werden, sei auf *Kulisch* [2; 3] verwiesen. Besonders hingewiesen sei auf die dadurch gegebene Möglichkeit, bewiesene Fehlerschranken zu erhalten (siehe [1; 4]). Mit den Programmpaketen LGL und LEIG (siehe [5]) werden auf unserer Rechenanlage trotz Rundungsfehler Fehlerschranken für die Lösung eines linearen Gleichungssystems und für die Eigenwerte und Eigenvektoren beliebiger reeller Matrizen *bewiesen*. (Dies ist einer der wenigen Algorithmen, die mathematisch-exakt bewiesene Dezimalstellen der Lösung ausgeben.) Das wird insbesondere dadurch möglich, da für spezielle Probleme kleine Assembler-Programme geschrieben wurden, die die Schwachstellen der Arithmetik umgehen. Diese Programme liefern nur mathematisch exakte Lösungen, auf die Sie sich verlassen können. Für diese Sicherheit ist ein kleiner Aufpreis zu zahlen. Doch wie schnell wiegen 15 bewiesene Stellen die Vielzahl von Kontrollen an Algorithmus und Ergebnis auf, die bisher notwendig waren. Die Ergebnisse von LGL und LEIG sind tatsächlich auf mindestens 15 Stellen genau und benötigen 85 Sekunden für die Lösung eines 100×100-vollbesetzten, beliebigen, reellen Gleichungssystems oder Eigenproblems einer ebenso vollbesetzten, beliebigen, reellen Martrix. Und wenn dies nicht eintreten sollte heißt das genau, daß im Modell ein Fehler steckt oder die Daten zu empfindlich sind. Dann ist das Problem ohne besondere Vorkehrungen nicht besser lösbar. D.h., eine zu ungenaue, nicht befriedigende Lösung ist *nicht* dem Verfahren anzulasten, sondern die Rechengenauigkeit ist gemessen an der schlechten Kondition des Problems unzureichend. D.h., es muß entweder mit höherer Genauigkeit gerechnet werden oder die Problemstellung ist zu überprüfen. Ein Gleitkommaalgorithmus liefert jedoch trotzdem ein Ergebnis. Und ist die Mantissenlänge 9 Dezimalstellen werden auch 9 Dezimalstellen ausgedruckt und eine hohe Genauigkeit vorgegaukelt.

Die obigen Beispiele machen deutlich, was bei hunderttausenden Operationen und einer schwachen Arithmetik passieren kann. Für denjenigen, der die obigen Beispiele auf seiner Anlage rechnet und befriedigt feststellt, daß kein Fehler auftrat soll das kein Grund sein, sich die Hände zu reiben und an die Perfektheit seiner Rechenanlage zu glauben. Im Gegenteil soll diese Notiz ein Denkanstoß sein, sich nicht blind auf Rechenergebnisse zu verlassen.

Literatur

[1] *Kaucher, E.*, und *Rump, S. M.*: Generalized Iteration Methods for Bounds of the Solution of Fixed Point Operator-Exquations. Computing 24, 131–137 (1980).

[2] *Kulisch, U.*: Grundlagen des numerischen Rechnens. Bibl. Inst., Mannheim, Wien, Zürich, 1976.

[3] *Kulisch, U.*, und *Miranker, W.*: Computer Arithmetic in Theory and Practice. Erscheint bei Academic Press (1980).

[4] *Rump, S. M.*: Kleine, exakte Fehlerschranken für die Lösung linearer Gleichungssysteme. ZAMM, Band 61, Heft 4, T 6660 (1980).

[5] *Rump, S. M.*: Exakte Fehlerschranken für Eigenwerte und Eigenvektoren. ZAMM, Band 61, Heft 5, T 6716 (1980).