Garantiert richtige Ergebnisse auf Rechenanlagen

Siegfried M. Rump
TU Hamburg

0. Einleitung. Mikrocomputer, Minicomputer, elektronische Rechenanlagen werden in immer stärkerem Maße in wichtigen, ja lebenswichtigen Prozessen des täglichen Lebens eingesetzt. Insbesondere Regelprozesse sind kaum mehr denkbar ohne elektronische Unterstützung.

Während eines solchen Prozesses laufen Algorithmen ab, letztendlich numerische Verfahren, die Ergebnisse gemäß einer Rechenvorschrift erzeugen. Bei der zunehmenden Komplexität der Systeme wachsen gelegentlich Zweifel, ob die erzielten Rechenergebnisse immer richtig sind oder aber durch äußere oder innere Einflüsse verfälscht werden können.

Im vorliegenden Aufsatz werden fünf verschiedene Aspekte der Sicherheit von Rechenergebnissen betrachtet. Bei jedem einzelnen dieser Aspekte wurden in jüngster Zeit in Hamburg ganz erhebliche Fortschritte erzielt, die jeder für sich genommen die Sicherheit von Rechenergebnissen wesentlich verbessern. Es wird aufgezeigt werden, wie die neuen Resultate Programme von der Entwicklung bis hin zu den Ergebnissen nachhaltig beeinflussen und so zu deutlich effektiverem Arbeiten und zu einer neuen Qualität der Sicherheit von Rechenergebnissen führen.

1. Numerische Rechenfehler. Trotz der zunehmenden Verbreitung von Rechenhilfsmitteln ist bei vielen Rechneranwendern eine gewisse Rechnergläubigkeit anzutreffen. Was der Rechner als Ergebnis liefert, wird quasi als mathematisch bewiesen akzeptiert. Häufig löst man großes Erstaunen aus, wenn man Rechnerbenutzer darauf hinweist, daß selbst bei einfachen Rechnungen mit nur wenigen Operationen völlig falsche Ergebnisse produziert werden können, und daß dies bei den heute verbreiteten Rechentechniken völlig zwangsläufig ist.

Für Spezialisten ist dies eine altbekannte Tatsache, und diese verfügen auch über die Erfahrung, falsche Rechenergebnisse als solche zu entlar-

ven. Der nicht spezialisierte Anwender ist dem Ergebnis der Rechenanlage mehr oder weniger hilflos ausgeliefert.

In jüngerer Zeit wurden Verfahren entwickelt (siehe [3, 4]), die zu einer numerischen Problemstellung das Ergebnis nicht nur approximieren, sondern mit Sicherheit richtig berechnen. Das Ergebnis wird hierbei entgegen herkömmlichen numerischen Verfahren in Schranken eingeschlossen. Der Algorithmus zeigt, ja quasi beweist, daß das gegebene Problem mit Sicherheit lösbar ist, daß eine Lösung also existiert, und zwar mit Sicherheit und sogar eindeutig innerhalb der berechneten Schranken. Diese Algorithmen werden Einschließungsalgorithmen genannt.

Die berechneten Schranken sind äußerst scharf. Im allgemeinen unterscheiden sich linke und rechte Grenze nur in der letzten Ziffer des verwendeten Formats, sie sind also praktisch maximal genau. Diese Genauigkeit geht weit über die im Ingenieurbereich notwendige Genauigkeit hinaus.

Die neuen Verfahren liefern prinzipiell nur richtige Ergebnisse. Dies ist mathematisch bewiesen. Es sind Verfahren entwickelt zur Lösung linearer Gleichungssysteme, zur Berechnung von Polynomnullstellen, Auswertung arithmetischer Ausdrücke, Lösung linearer und nichtlinearer Optimierungsprobleme, allgemeiner nichtlinearer Gleichungssysteme, spärlich besetzter Matrizen und viele andere mehr. Die Algorithmen wurden auf verschiedenen Rechnern implementiert, und es gibt eine Reihe kommerzieller Implementierungen, unter anderem von der Firma IBM unter dem Namen ACRITH (siehe [1]).

2. Datendarstellbarkeit. Üblicherweise sind die Daten eines Problems auf dem Rechner nicht exakt darstellbar. Diese Spezialisten wiederum altbekannte Tatsache führt in der Praxis immer wieder zu erheblichen Problemen.

Elektronische Rechenanlagen arbeiten üblicherweise im Binär- oder Hexadezimalsystem. Diese Zahlsysteme stellen Gleitpunktzahlen in computergerechter Art und Weise dar, während wir Menschen üblicherweise im Dezimalsystem rechnen. Durch den Übergang vom Dezimalsystem in das Rechnerformat entstehen sogenannte Konvertierungsfehler. Beispielsweise ist die Dezimalzahl 0.1, also ein Zehntel oder 10 Prozent, auf fast allen üblichen Rechenanlagen nicht exakt darstellbar. Der Konversionsfehler ist zwar gering, kann aber in bestimmten Fällen zu deutlichen Veränderungen des Ergebnisses führen (im Gegensatz dazu arbeiten Taschenrechner üblicherweise im Dezimalsystem, hier gibt es also nur Konvertierungsfehler mangels Stellenzahl wie etwa bei der Kreiszahl π).

Es ist also (sic!) auf heute üblichen Rechenanlagen (einschließlich PC's) ein Unterschied, ob mit 0,1 multipliziert wird oder durch 10 dividiert wird. Mit anderen Worten, der Rechner löst durch die Konvertierungsfehler ein anderes Problem, als der Benutzer eigentlich meint.

Neben dieser Schwierigkeit, daß man exakt vorgegebene Daten nicht ohne Fehler in den Rechner hereinbekommt, tritt es sehr häufig auf, daß die Daten eines Problems nicht exakt gegeben sind oder aber nicht exakt ermittelt werden können. Man denke etwa an Meßwerte, Preise, Lieferzeiten etc. Es wäre wünschenswert, diese nicht exakt gegebenen oder zu ermittelnden Daten mit der ihr eigenen Toleranz in den Rechner eingeben zu können und diese Toleranz algorithmisch verarbeiten zu können. Dies ist möglich.

Es wurden Algorithmen entwickelt, die es erlauben, für alle Eingabedaten individuelle Toleranzen zu spezifizieren. Die Algorithmen berechnen dann automatisch die Menge aller Lösungen für beliebig innerhalb der Toleranzen gewählte Eingabedaten. Diese Einschließungsalgorithmen berechnen für die Menge aller Lösungen wiederum Schranken, und diese Schranken sind mit Sicherheit richtig. Das Konvertierungsproblem und das Problem toleranzbehafteter Daten werden durch diese Algorithmen also gelöst.

Die Algorithmen wurden ebenfalls auf verschiedenen Rechnern implementiert, und es gibt eine Reihe kommerzieller Implementierungen, unter anderem von der Firma IBM unter dem Namen ACRITH (siehe [1]).

3. Sensitivitätsanalyse. Die Sensitivität eines Problems ist die Abhängigkeit der Lösung von den Eingabedaten. Ein besonders sensitives Problem, das heißt ein Problem, dessen Lösung bei geringfügiger Änderung der Eingabedaten stark variiert, kann als instabil bezeichnet werden. Die Information, wie sensitiv ein Problem ist, ist also von vitaler Bedeutung für den Anwender.

Diese Sensitivität eines Problems kann automatisch berechnet werden. Die unter den obigen Punkten 1 und 2 beschriebenen Verfahren erlauben einerseits, die Daten eines Problems mit Toleranzen zu versehen, und andererseits, für die Menge aller entstehenden Lösungen garantierte und richtige Schranken zu berechnen.

In allerjüngster Zeit ist in Hamburg hier ein neuer Durchbruch gelungen. Es wurden Verfahren entwickelt, die die entstehende Lösungsmenge nicht nur einschließen, das heißt von außen einschließen, sondern ebenfalls garantiert richtige Inneneinschließungen berechnen. Der Benutzer bekommt damit eine garantiert richtige Information darüber geliefert, wie sensitiv sein Problem ist. Und zwar wie sensitiv es mindestens ist,

und wie sensitiv es höchstens ist, und das für jede einzelne Lösungskomponente. Die Mindest- und Höchstsensitivität liegen wiederum unmittelbar beieinander.

Das bedeutet für den Benutzer, daß er direkt an der Lösung ablesen kann, wie empfindlich jede einzelne Lösungskomponente von den Eingabedaten abhängt. Das hat natürlich unmittelbare praktische Bedeutung.

Die Lösung eines Problems kann jetzt mit der Berechnung der Empfindlichkeit gekoppelt werden. Über die mit Sicherheit richtige Lösung des Originalproblems hinaus wird gleichzeitig die Sensitivität und damit die Korrektheit der Problemstellung überprüft, und zwar garantiert richtig. Eine zu große Empfindlichkeit der Lösung wird sofort erkannt, und es können Gegenmaßnahmen wie etwa die Überprüfung der Problemstellung, der Daten usw. getroffen werden.

Die Empfindlichkeit wird mit der Berechnung der Lösung des Problems praktisch kostenlos mitgeliefert. Der enorme Gewinn sei anhand linearer Optimierungsprobleme erläutert.

Herkömmliche Verfahren nähern die Lösung eines Optimierungsproblems an, Toleranzen sind nur mit großer Mühe und erheblichem Aufwand eingeb- bzw. berechenbar. Die neuen Verfahren bieten zwei grundlegende Vorteile:

a) Toleranzen sind problemlos eingeb- und berechenbar.
b) Es werden alle möglichen Optimallösungen innerhalb der Eingabetoleranzen berechnet. Der Benutzer kann nach schwer formulierbaren oder subjektiven Kriterien eine passende Lösung auswählen.

Vorteile für den Benutzer sind, daß einerseits die Toleranzen überhaupt berücksichtigt werden und andererseits mehrere Optimallösungen geliefert werden, die alle praktisch gleich gut sind und zwischen denen ausgewählt werden kann. Dadurch wird eine Flexibilität erst möglich (siehe [2]).

Die Rechenzeit wird gegenüber herkömmlichen Methoden, die im allgemeinen nur einen Teil der Optimallösungen approximieren, deutlich verringert. Das kommt daher, daß etwa beim Monte-Carlo-Verfahren ein und dasselbe Problem sehr häufig gelöst werden muß, während die neuen Verfahren das Problem nur einmal mit Toleranzen lösen.

4. Korrekte Implementierungen. Die mathematische Richtigkeit der erzielten Lösung durch einen Algorithmus ist natürlich nur dann gewährleistet, wenn der Algorithmus korrekt implementiert wurde. Das ist eine triviale Feststellung, die für jeden Algorithmus zutrifft. Die neuen Einschließungsalgorithmen bergen in sich einen Mechanismus, der die Rich-

tigkeit der Implementierung ganz wesentlich wahrscheinlicher und in gewisser Weise überprüfbar macht.

Bei einem herkömmlichen numerischen Algorithmus wird in der Regel jedes Ergebnis akzeptiert, das in der Nähe der Lösung liegt. Unterscheidet sich ein Ergebnis also nur geringfügig von der korrekten Lösung und ist ursächlich hierfür ein Programmierfehler verantwortlich, wird dies im Normalfall nicht erkannt werden. Bei den neuen Einschließungsalgorithmen ist dies anders. Bei fehlerhafter Implementierung unterscheidet sich die Einschließung geringfügig von der korrekten Lösung, mit anderen Worten, die korrekte Lösung wird nicht mehr eingeschlossen; sie liegt, wenn auch nur geringfügig, außerhalb der Schranken. Das ist mathematisch falsch und folgerichtig muß die Implementierung des Algorithmus fehlerhaft sein.

Die großen Erfahrungen bei den diversen Implementierungen der Einschließungsalgorithmen haben gezeigt, daß Programmierfehler auf diese Art und Weise sehr schnell entdeckt werden. Das zeigt sich auch darin, daß in den Implementierungen, die etliche hunderttausend lines of code umfassen, noch kein einziger Fehler gefunden wurde. Die Richtigkeit der Implementierung wird so durch die Struktur der Algorithmen gewährleistet.

5. Spezifikation vs. Programm. Vor der Implementierung eines Algorithmus steht seine Spezifikation. Es wurde ein neuartiges Programmierwerkzeug entwickelt, das eine Spezifikation in mathematischen Formeln unmittelbar verarbeiten und ausführen kann. Das heißt, bereits die Spezifikation ist ein ausführbares Programm, dessen Ergebnis mit dem Ergebnis des dann in der Zielsprache programmierten Algorithmus übereinstimmen muß, und zwar bitweise übereinstimmen muß. Entsprechende Vergleiche können also ohne Aufwand maschinell ausgeführt werden.

Die Spezifikation als ausführbares Programm ist natürlich langsamer in der Ausführungszeit als das endgültige Programm, aber schnell genug, um auch große Testserien durchzuführen.

Durch die Möglichkeit, gegen die Spezifikation Bit für Bit, und zwar automatisch, zu testen, erreichen die so beschriebenen Algorithmen einen extrem hohen Sicherheitsstandard. Die Spezifikation ist sehr kurz und übersichtlich, und diese Vorzüge werden durch den Testmechanismus unmittelbar auf die zu erstellenden Programme übertragen.

Dieses Programmierwerkzeug (CALCULUS für IBM /370 Anlagen, ABACUS für DOS- und UNIX-Rechner) ist besonders als Entwicklungswerkzeug äußerst wertvoll. Es erlaubt, in kurzer, prägnanter und vor allem mathematisch üblicher Notation Programme zu schreiben. Die enthaltenen

Operationen sind äußerst mächtig. Häufig kann man in einer Zeile hinschreiben, wozu sonst ein ausgewachsenes Programm notwendig ist. Ein paar Beispiele hierzu.

Ein gewisses Maß für die Sensitivität einer Matrix ist die Konditionszahl. Diese kann als Quotient aus größtem und kleinstem Singulärwert berechnet werden. Ein "Programm" hierfür sieht in CALCULUS oder ABACUS wie folgt aus:

```
A = hilb(5); s = svd(A); max(s)/min(s)
```

Es wird nacheinander eine Hilbert-5x5-Matrix erzeugt, die Singulärwertzerlegung (singular value decomposition) berechnet, gespeichert und der maximale dividiert durch den minimalen Singulärwert (vorausgesetzt die Matrix ist nicht singulär) berechnet. Es ergibt sich schnell die Ausgabe

```
4.76607E+005
```

Dieser Wert ist eine Näherung, die durch numerische Rechenfehler verfälscht sein könnte. Ein mit Sicherheit richtiges Rechenergebnis wird immer dann berechnet, wenn mit Toleranzen gearbeitet wird. Die folgende Zeile etwa

```
A = hilb(5); s = ival svd(A); max(s)/min(s)
```

führt die Singulärwertzerlegung von A mit garantierten Schranken durch und schließt den Quotient aus maximalem und minimalem Singulärwert in garantierte Schranken ein. In diesem Fall ist obiges, rein numerisch erzieltes Ergebnis in allen angegebenen Stellen richtig.

Selbstverständlich können in CALCULUS oder ABACUS Programme mit üblicher Parameterübergabe usw. geschrieben werden. Als Beispiel sei die numerische Überprüfung des Satzes von Cayley-Hamilton hingeschrieben. Der Satz von Cayley-Hamilton besagt, daß die Null-Matrix entsteht, wenn eine Matrix in ihr charakteristisches Polynom eingesetzt wird. Dieser Satz kann beispielhaft für eine Matrix "numerisch überprüft" werden, indem das charakteristische Polynom einer Matrix berechnet wird, die Matrix eingesetzt wird und eine Norm der Ergebnismatrix angezeigt wird. Dies wird durch folgendes Programm ausgeführt:

```
module Cayley_Hamilton (A);
  <m,n> = size(A);
  if m <> n then {display('A nicht quadratisch'); return};
  P = charpoly(A); C = P(n+1)*Id(A);
  for i = n:1:-1 do C = C*A+P(i)*Id(A);
  norm(C)
```

Im Programm wird zunächst abgefragt, ob die Matrix A quadratisch ist. Dazu wird die Größe von A, das heißt Anzahl der Zeilen und Spalten, in m und n abgespeichert. Dann wird das charakteristische Polynom P von A und das Ergebnis C mit einer Matrix initialisiert, die gleich einer Ein-

heitsmatrix der Größe von A (die nicht wirklich gespeichert wird) multipliziert mit dem führenden Koeffizienten des charakteristischen Polynoms ist. Id(A) bedeutet eine Einheitsmatrix der Größe von A. Dann wird das charakteristische Polynom nach dem Horner-Schema berechnet und in der letzten Zeile der Wert, die Norm von C ausgegeben.

Im Programm kommen mehrere Matrix-Additionen und -Multiplikationen vor, es muß das charakteristische Polynom berechnet werden, die Norm einer Matrix berechnet werden und einiges mehr. Das Programm bleibt sehr übersichtlich, ist in wenigen Minuten geschrieben und sofort für jede Matrix (reell, komplex, mit Toleranzen behaftet) ausführbar. Natürlich wird durch diese Vorgehensweise kein mathematischer Beweis des Satzes von Cayley-Hamilton erbracht. Es kann aber sehr wohl durch diese Vorgehensweise eine gewisse Evidenz für eine Vermutung erreicht werden, die dann durch den mathematischen Beweis erhärtet werden muß.

Die Eleganz des Programmierwerkzeugs CALCULUS oder ABACUS erlaubt es, ein solches Programm in Minutenschnelle hinzuschreiben und auszuführen. In herkömmlicher Programmiersprache verfaßt, würde das Schreiben eines solchen Programms ein Vielfaches der Zeit in Anspruch nehmen, ganz abgesehen von allfälligen Schwierigkeiten mit file handling, Betriebssystem und vielem anderen mehr.

Soll ein entsprechendes Produktionsprogramm geschrieben werden, kann das Produktionsprogramm leicht aus der Vorlage entwickelt werden und sehr einfach und effizient gegen die Spezifikation getestet werden.

6. Zusammenfassung und Ausblick. Jeder einzelne der oben zusammengestellten Aspekte 1 bis 5 bringt einen deutlichen Sicherheitsgewinn, die fünf Punkte zusammengenommen heben die so erstellte Software auf einen neuen Qualitätsstandard. Die Programmierwerkzeuge CALCULUS und ABACUS erlauben die prägnante Formulierung von Algorithmen. Diese Algorithmen können als ausführbare Spezifikation für Produktionsprogramme dienen und leisten bei der täglichen Entwicklungsarbeit unschätzbare Dienste. Darüber hinaus wird durch die mächtigen Operationen eine nachfolgende Vektorisierung oder Parallelisierung der Algorithmen wesentlich unterstützt: Wird ein Algorithmus zunächst in einer herkömmlichen Programmiersprache implementiert und nachher durch einen automatisch vektorisierenden oder parallelisierenden Compiler übersetzt, muß der Compiler die inhärente mathematische Struktur aus den Programmstrukturen wieder herausfiltern, was häufig sehr schwierig oder kaum möglich ist. Durch die mathematische Schreibweise und die mächtigen Operatoren entfällt das Auseinanderdividieren und Zusammenfügen der Strukturen, eine Vektorisierung oder Parallelisierung kann den Anweisungen unmittelbar entnommen werden.

Bei der Entwicklung von Algorithmen besteht meines Erachtens ein regelrechter Applikationsstau: Viele Ideen liegen brach, und eine Implementierung auf dem Rechner wird nicht vorgenommen, weil eine derartige Implementierung viel zu zeitaufwendig ist und ein potentieller Nutzen im vorhinein nicht abgeschätzt werden kann. Durch die Möglichkeit, einen Algorithmus rasch und problemlos, vor allem ohne den lästigen Spezifikationsteil, implementieren zu können, wird dieser Stau aufgelöst. Es ergibt sich hieraus eine neue Qualität der Entwicklung und Forschung.

Die DOS- und UNIX-Version des Programmiersystems ABACUS (IBM PCs oder kompatibel und Workstations) ist demnächst in Hamburg verfügbar.

[1] ACRITH, High-Accuracy Arithmetic Subroutine Library, Program Description and User's Guide, IBM Publications, Document Number SC 33-6164-3 (1986).

[2] Jansson, Ch.: A Self-validating Method for Solving Linear Programming Problems with Interval Input Data, Computing Supplementum, 6, 1988.

[3] Rump, S. M.: Solving Algebraic Problems with High Accuracy, in: A New Approach to Scientific Computation, eds. U. Kulisch and W.L. Miranker, Academic Press, 51 - 120 (1981).

[4] Rump, S. M.: New Results on Verified Inclusions, in: Accurate Scientific Computations, eds. W. L. Miranker and R. Toupin, Springer Lecture Notes in Computer Science 235, 39 Seiten (1986).

[5] Genaues Rechnen, Broschüre zur Rechnerarithmetik, IBM Fachbereich Lehre und Forschung, 1986.

Prof. Dr. Siegfried M. Rump
Informatik III
Technische Universität Hamburg
Eißendorfer Straße 38
2100 Hamburg 90
West Germany