19.) C. Temperton: 'Fast Fourier Transforms on the Cyber 205'
Vortrag beim Nato-Workshop Jülich, (1983)

20.) U. Trottenberg: 'On the SUPRENUM-Conception'
Parallel Computing 85, Proceedings of the Second International Conference on Parallel Computing, Berlin 1985, (1986), 97-105

21.) H.H. Wang: 'On Vectorizing the Fast Fourier Transform'
BIT 20, (1980), 233-243

22.) H.H. Wang: 'A parallel method for tridiagonal equations'
ACM Trans. on Math. Software, 7-2, (1981), 170-183

RECHNERARITHMETIK UND DIE BEHANDLUNG ALGEBRAISCHER PROBLEME

Ulrich Kulisch
(Institut für Angewandte Mathematik der Universität Karlsruhe)

Siegfried M. Rump
(IBM Deutschland GmbH, Forschung und Entwicklung, Böblingen)

Einleitung

Rechenanlagen sind ursprünglich einmal für umfangreiche Berechnungen im technisch-wissenschaftlichen Bereich entwickelt und erfunden worden. Heute ist eine Rechenanlage eine Universalmaschine. D.h. sie wird in den verschiedensten Wissensgebieten eingesetzt, wie beispielsweise im Bankwesen, in Spielautomaten, als Platzbuchungssystem, zur Verkehrskontrolle, bei der medizinischen Diagnose, zur Sprachübersetzung oder zur Inventarüberwachung. Wegen dieser vielfältigen Anwendungen übersieht man häufig die zentrale Rolle, die der Rechner beim technisch-wissenschaftlichen Rechnen auch heute noch immer spielt.

Berechnungen spielen sich in Rechenanlagen in zwei unterschiedlichen Zahlsystemen ab. Das eine besteht aus einer beschränkten Teilmenge der ganzen Zahlen, das andere sind die sogenannten Gleitkommazahlen. Für Anwendungen im Bereich des nichtnumerischen Rechnens, von den im ersten Abschnitt einige genannt wurden, benötigt man ausschließlich das System der ganzen Zahlen. Solange der Bereich der darstellbaren ganzen Zahlen nicht überschritten wird, verlaufen Rechnungen in diesem Zahlbereich aus mathematischer Sicht - solange der Rechner technisch intakt ist - fehlerfrei. Wegen der großen Zahl derartiger Anwendungen wird der Rechner heute vielfach für ein perfektes mathematisches Werkzeug gehalten. Was der Rechner liefert, so glaubt man, ist korrekt wie ein mathematisch bewiesener Satz.

Für den Bereich des technisch-wissenschaftlichen Rechnens trifft dies leider nicht zu. Diesen Bereich wollen wir hier näher betrachten. Gleitkommazahlen und Gleitkommaoperationen können die reellen Zahlen und deren Verknüpfungen nur approximieren. Ein Hauptärgernis besteht hier in der Tatsache, daß der Rechenfehler nur mühsam und häufig gar nicht beherrschbar ist. Man wird hier mit einer scheinbar paradoxen Situation konfrontiert. Viele Rechenanlagen führen die Gleitkommaoperationen heute mit maximaler Genauigkeit aus. D.h. das Ergebnis einer solchen Verknüpfung ist die dem korrekten Verknüpfungsergebnis im Sinne der reellen Zahlen nächst gelegene Gleitkommazahl. Nichtsdestoweniger kann das Ergebnis einer Rechnung, welche aus mehreren Verknüpfungen zusammengesetzt ist, auch nach nur wenigen Operationen bereits völlig falsch sein. Die Berechnung der folgenden Summe möge dies illustrieren:

$$10^{50} + 511 - 10^{50} + 10^{35} - 812 - 10^{35} = -301 .$$

Praktisch alle auf dem Markt befindlichen Rechenanlagen liefern bei der Be rechnung der Summe in der angegebenen Reihenfolge das falsche Ergebnis Nul da die üblichen Gleitkommaformate den großen Zahlbereich der auftretenden Summanden nicht erfassen können.

Die ersten Universalrechner im heutigen Sinne sind Anfang der fünfziger Ja re auf den Markt gekommen. Die damit verbundene enorme Steigerung der Möglichkeit Rechenoperationen auszuführen, der Eintritt ins Computerzeitalter wurde als eine Art Revolution empfunden. Was war geschehen? Zur Zeit der elektromechanischen Tischrechner galt die Faustregel, daß eine geübte Person in der Lage ist, einigermaßen zuverlässig 1000 Rechenoperationen pro T auszuführen. Dies sind größenordnungsmäßig etwa $0.1 = 10^{-1}$ Rechenoperation in der Sekunde. Die ersten elektronischen Rechenanlagen Anfang der fünfzig Jahre waren demgegenüber in der Lage, größenordnungsmäßig $100 = 10^{2}$ Gleitkommaoperationen in der Sekunde auszuführen. Dies bedeutete eine gigantisc Geschwindigkeitssteigerung um den Faktor 1000 ($=10^{3}$) gegenüber ihren unmit telbaren elektromechanischen Vorgängern.

Seit dieser Zeit hat sich eine stille, aber noch viel größere Computerrevo lution vollzogen. Die schnellsten heute existierenden Rechner sind in der Lage, größenordnungsmäßig 1 Milliarde = 10^{9} Gleitkommaoperationen in der S kunde auszuführen. Dies ist noch einmal eine Geschwindigkeitssteigerung um den Faktor 10^{7} gegenüber den elektronischen Rechenanlagen der frühen fünfziger Jahre. Daraus muß man den Schluß ziehen, daß die eigentliche Compute revolution nach Aufkommen der ersten elektronischen Rechenanlagen stattgefunden hat. Die nochmalige Geschwindigkeitssteigerung um den Faktor 10^{7} b deutet nicht nur einen quantitativen, sondern auch einen qualitativen Spru

Es ist schwierig, die enorme Rechenleistung einer Großrechenanlage unserer Tage zu erfassen. Sie sei daher an einem Beispiel illustriert. Auf der Erd leben heute ca. 5×10^{9} (= 5 Milliarden) Menschen. Wenn wir jedes menschli che Wesen (Männer, Frauen und Kinder) mit einem elektromechanischen Tischrechner oder einem einfachen elektronischen Taschenrechner ausstatten, könn ten diese, wenn sie alle rechnen, gerade die Leistung einer der schnellste Rechenanlagen unserer Tage erbringen ($5 \times 10^{9} \times 0{,}1 = 5 \times 10^{8}$).

Mechanische Tischrechner oder einfache elektronische Taschenrechner erlauben eine gewisse Kontrolle des Rechenvorganges durch den Benutzer. Der Benutzer gibt jede Rechenoperation selbst ein und nimmt jedes Zwischenergebnis selbst entgegen. Durch sein Verständnis dessen was abläuft, kontrolliert er den Rechenprozeß. Vom Gebrauch des Rechenschiebers oder eines Taschenrechners ist diese Art der Kontrolle einer maschinell unterstützten Rechnung auch heute noch vielen vertraut.

Mit dem Aufkommen der elektronischen Rechenanlagen war diese interaktive Art der Fehlerkontrolle einer Rechnung durch den Benutzer nicht mehr praktikabel. Die enorme Geschwindigkeitssteigerung verlangte die Entwicklung systematischerer Methoden zur Fehlerkontrolle. Die für die ersten elektronischen Rechenanlagen entwickelten Methoden basieren auf Fehlerabschätzungen jeder einzelnen Gleitkommaoperation. Da der Rechner in der Lage ist, eine große Anzahl von Operationen auszuführen, muß der Benutzer eine große Anzahl von Fehlerschätzungen vornehmen. Darüberhinaus muß er studieren, wie sich diese Fehler in komplizierten Algorithmen fortpflanzen.

Bei den heute erzielbaren Rechengeschwindigkeiten von einer Milliarde Operationen in der Sekunde, ist auch diese Vorgehensweise kaum mehr praktikabel. Mangels einer Alternative wird häufig gar keine Fehleranalyse durchgeführt. Damit wird das Rechnen zu einem systematischen, häufig aussichtslosen Experimentieren. Wenn in einem komplizierten Rechenprozeß Teilrechnungen der obigen Art anfallen, ist in der Regel die gesamte Rechnung wertlos. Das Problem der Fehleranalyse entpuppt sich damit als ähnlich schwierig oder vielleicht noch schwieriger als der Rechenprozeß selbst.

Nun sind Rechenanlagen einmal dazu erfunden worden, komplizierte Aufgaben dem Menschen abzunehmen. Es ist daher nur naheliegend, zu versuchen, auch den Prozeß der Fehleranalyse einer numerischen Rechnung selbst wieder dem Rechner zu übertragen.

In Anbetracht dieser Entwicklung wurde Ende der sechziger Jahre vom ersten der Autoren dieses Beitrages die Vorstellung entwickelt, daß die arithmetische Basis des Gleitkommarechnens gegenüber früheren Ansätzen noch einmal wesentlich verbreitert werden muß, wenn man bei der Frage der Rechengenauigkeit und Rechenkontrolle weiterkommen will. Verantwortlich für das Auftreten von Rechenfehlern sind die bei jeder einzelnen Operation ausgeführ-

ten Rundungen. Ziel einer Basisverbreiterung des wissenschaftlichen Rechnens mußte es demnach sein, die Anzahl der in einem Rechenprozeß ausgeführten Rundungen drastisch herunterzudrücken. Als ein wichtiger Schritt in dieser Richtung wurde der Versuch unternommen, in den üblichen linearen Räumen der Mathematik, wie reelle und komplexe Zahlen, Vektoren und Matrizen, sowie den zugehörigen Intervallräumen die arithmetischen Verknüpfungen nicht wie bisher aus einzelnen Gleitkommaoperationen aufzubauen, sondern mit maximaler Genauigkeit im Rechner direkt zu erzeugen. Die Rechnerarchitektur muß dazu natürlich so ausgelegt werden, daß dies möglich ist.

So wurde zunächst eine allgemeine Theorie der Rechnerarithmetik entwickelt, welche diese Idee systematisch verfolgt. Die einschlägigen Untersuchungen waren Mitte der siebziger Jahre abgeschlossen. Sie haben in zwei Buchveröffentlichungen ihren Niederschlag gefunden:
[12], [15].

Die neue Theorie der Rechnerarithmetik gipfelt in einer neuartigen Definition der arithmetischen Verknüpfungen in Rechenanlagen. Eines der für die Praxis relevanten Hauptergebnisse dieser Untersuchungen besteht in dem Nachweis, daß es möglich ist, alle in den oben aufgeführten linearen Räumen und zugehörigen Intervallräumen auftretenden inneren und äußeren Verknüpfungen mit maximaler Genauigkeit zu berechnen, wenn der Rechner in der Lage ist, Skalarprodukte zweier Vektoren beliebiger Länge maximal genau auszuführen. Dem Skalarprodukt zweier Vektoren kommt demnach eine ganz zentrale Bedeutung zu.

Die neue Arithmetik beeinflußt auch die Entwicklung von Programmiersprachen. Es sind Erweiterungen der Programmiersprachen PASCAL und FORTRAN ausgearbeitet worden, welche das Programmieren im Bereich technisch-wissenschaftlicher Anwendungen erheblich vereinfachen und zuverlässiger machen [4], [33], [37].

Während der Jahre 1977 bis 1979 wurde am Institut von Professor Kulisch mit Unterstützung des BMFT und der Nixdorf Computer AG ein erster Rechner aufgebaut, welcher vollständig mit der neuen Arithmetik ausgestattet ist. Es zeigt sich, daß ein so gebauter Rechner in der Lage ist, bei praktisch allen Grundaufgaben der Numerik die Lösung mit maximaler Genauigkeit in Schranken einzuschließen. Die neue Arithmetik erlaubt darüberhinaus eine anschließende

Verifikation der Lösung, d.h. den rechnerischen Nachweis der Existenz und Eindeutigkeit der Lösung des Problems innerhalb der berechneten Schranken. Ist diese nicht gegeben, so stellt der Rechner dies fest und informiert hierüber den Benutzer. Der Rechner informiert den Benutzer auch dann, wenn er mittels des vorgegebenen Algorithmus oder des verwendeten Zahlenformates die Lösung nicht finden kann. Dies sind Ergebnisse, welche mit herkömmlicher Gleitkommaarithmetik nicht erzielt worden sind und auch nicht erzielt werden können. Die aufzuwendende Rechenzeit ist stets von der gleichen Größenordnung wie diejenige, welche auch bei Ausführung eines herkömmlichen Gleitkomma-Näherungsalgorithmus aufzuwenden wäre. Die neue Arithmetik eröffnet der Numerik neue Dimensionen im Hinblick auf eine automatisierte Fehlerkontrolle bzw. eine Nachkorrektur von Rechenergebnissen durch den Rechner selbst bis hin zu maximaler Genauigkeit und sogar noch darüber hinaus. Numerisches Rechnen wird mit diesem Werkzeug in vieler Hinsicht von dem Niveau einer experimentellen auf das einer mathematischen Wissenschaft angehoben.

1 Die Räume des numerischen Rechnens

Wir beginnen mit einer Auflistung derjenigen Räume, welche beim numerischen Rechnen mit Rechenanlagen auftreten.
Numerische Algorithmen werden in der Regel definiert und hergeleitet im Raum R der reellen Zahlen, der Vektoren VR oder der Matrizen MR über den reellen Zahlen. Daneben treten gelegentlich auch die entsprechenden komplexen Räume ¢, V¢ und M¢ auf. All diese Räume sind geordnet bezüglich der Ordnungsrelation $\leq$. Diese wird in den Produkträumen komponentenweise erklärt. Mittels dieser Ordnungsrelation lassen sich Intervalle bilden. Das Rechnen mit Mengen über den zunächst genannten Räumen wird notwendig für die später beschriebenen Algorithmen mit verifizierten Ergebnissen. Als Mengen, in denen Operationen auf dem Rechner effizient implementierbar sind, kommen z.B. Intervalle in Betracht. Numerische Algorithmen werden so gelegentlich auch für Intervalle über den bereits genannten Räumen formuliert. Wenn wir die Menge der Intervalle über einer geordneten Menge $\{M, \leq\}$ mit IM bezeichnen, entstehen so die Räume IR, IVR, IMR und I¢, IV¢ und IM¢. Siehe dazu die zweite Spalte in der Figur 1.

Die in diesen Räumen gegebenen Algorithmen lassen sich nun aus verschiedenen Gründen auf dem Rechner nicht ausführen. Das Rechnen mit reellen Zahlen approximiert man daher in einem Teilsystem S (single precision), in welchem die Verknüpfungen stets einfach und schnell ausführbar sind. Auf Rechenanlagen dienen dazu sogenannte Gleitkommasysteme mit einer festen Anzahl von Ziffern in der Mantisse. Erst dann, wenn die gewünschte Genauigkeit des Ergebnisses anders nicht gewährleistet werden kann, geht man zu umfassenderen Systemen D (double precision) mit der Eigenschaft $R \supset D \supset S$ über. Ausgehend von S können wir nun entsprechend Vektoren (n-tupel), Matrizen (n^2-tupel), Komplexifizierungen (Paare), Vektoren und Matrizen solcher Paare, sowie die entsprechenden Räume der Intervalle über diesen Mengen bilden. Wir kommen so zu folgenden Mengen: S, VS, MS, ¢S, V¢S, M¢S, IS, IVS, IMS, I¢S, IV¢S, IMCS, sowie den entsprechenden Räumen über D. Siehe dazu die 3. und 4. Spalte in Figur 1. Unter Rechenarithmetik wollen wir hier alle in den Räumen der 3. und 4. Spalte der Figur 1 zu erklärenden inneren und äußeren Verknüpfungen verstehen.

		R	⊃	D	⊃	S
		VR	⊃	VD	⊃	VS
		MR	⊃	MD	⊃	MS
PR	⊃	IR	⊃	ID	⊃	IS
PVR	⊃	IVR	⊃	IVD	⊃	IVS
PMR	⊃	IMR	⊃	IMD	⊃	IMS
		¢	⊃	¢D	⊃	¢S
		V¢	⊃	V¢D	⊃	V¢S
		M¢	⊃	M¢D	⊃	M¢S
P¢	⊃	I¢	⊃	I¢D	⊃	I¢S
PV¢	⊃	IV¢	⊃	IV¢D	⊃	IV¢S
PM¢	⊃	IM¢	⊃	IM¢D	⊃	IM¢S

- Figur 1 -

Die Anzahl dieser Verknüpfungen ist größer als man vielleicht erwartet. Eine komplexe Matrix läßt sich beispielsweise wieder mit einer komplexen Matrix multiplizieren, aber auch mit einem komplexen Vektor oder einer komplexen Zahl darüber hinaus aber auch mit einer reellen Matrix, einem reellen Vektor oder einer reellen Zahl oder einer ganzzahligen Matrix, einem ganzzahligen Vektor oder einer ganzen Zahl.

Im Sinne des Typkonzeptes in Programmiersprachen sind dies alles verschiedene Verknüpfungen. Zählt man allein die in der letzten Spalte der Figur 1 zu erklärenden inneren und äußeren Verknüpfungen, so erhält man eine Zahl von einigen Hundert.

2 Herkömmliche Definition der Rechnerarithmetik

Die herkömmliche Definition der Rechnerarithmetik unterscheidet sich ganz wesentlich von der im folgenden anzugebenden. Herkömmlicherweise versteht man unter Rechnerarithmetik nur die Verknüpfungen in den Mengen D und S. Von den einigen Hundert Verknüpfungen in der Spalte unter S stellen herkömmliche Rechenanlagen nur 4, nämlich die Addition, Subtraktion, Multiplikation und Division von Gleitkommazahlen zur Verfügung. Alle anderen Verknüpfungen muß der Benutzer im Bedarfsfalle selbst programmieren. Es geschieht dies in Form von Prozeduren. Jede in einem Algorithmus auftretende von den vier Grundrechnungsarten verschiedene Verknüpfung verlangt dann einen eigenen Prozeduraufruf. Diese Vorgehensweise ist umständlich, zeitraubend und vor allem viel zu ungenau.

Bei der Schaffung von Programmiersprachen (ALGOL und FORTRAN) in den fünfziger Jahren galt es als allgemeiner Konsens, daß man die Arithmetik nicht verbindlich festlegt, sondern ihre Realisierung dem Hersteller überläßt. Dies hat zur Folge, daß der Benutzer im allgemeinen nicht weiß, was passiert, wenn er in einem Algorithmus +, -, · oder / schreibt. Zwei Rechenmaschinen verschiedener Hersteller unterscheiden sich so häufig in der Arithmetik. Die Numerik ist folglich nicht in der Lage, auf allgemein verbindlichen Grundannahmen über die Arithmetik aufzubauen. Alles was man tun kann, besteht darin, die Grundannahmen, welche bei der Fehleranalyse numerischer Algorithmen benutzt werden, als Ersatzdefinition der Arithmetik anzusehen. Betrachten wir zunächst

2.1 Die Grundverknüpfungen

Die Fehleranalyse numerischer Algorithmen arbeitet weitgehend mit folgenden Annahmen über die Arithmetik (Wilkinson). Sofern kein Überlauf und kein Unterlauf auftritt, gilt für das gerundete Bild $\square a$ einer Zahl a :

$$\square a = a(1 - \varepsilon) \quad \text{mit} \quad |\varepsilon| < \varepsilon^* . \tag{1}$$

Für die Maschinenverknüpfungen $\boxed{*}$,$* \in \{+,-,\cdot,/\}$ gilt

$$a \boxed{*} b = (a * b)(1 - \varepsilon) \quad \text{mit} \quad |\varepsilon| < \varepsilon^* . \tag{2}$$

D.h. der relative Fehler der Rundung bzw. jeder Verknüpfung ist dem Betra nach kleiner als eine Maschinenkonstante ε^*:

$$|\square a - a| < \varepsilon^* |a| \tag{3}$$

$$|a \boxed{*} b - a * b| < \varepsilon^* |a * b| \tag{4}$$

Die Konstante ε^* besagt, daß der bei der Rundung wie auch bei jeder Verknüpfung begangene Fehler sich höchstens auf die letzte mitgeführte Ziffer de Mantisse auswirkt (relative Rundungsfehlereinheit).

Mit diesen Fehlerformeln gelingt es in der Numerik, einige Algorithmen au ihre Genauigkeit hin zu untersuchen. Tatsache aber ist, daß alle so gewon nenen Fehlerabschätzungen nicht dazu benutzt werden können, zuverlässige Schranken für die berechnete Lösung daraus abzuleiten. Vor der Formel (1) te nämlich stehen für alle $a \in R$ und vor der Formel (2) für alle a, $b \in S$ sofern kein Unterlauf und kein Überlauf auftritt. Für viele der auf dem M befindlichen Großrechenanlagen aber gelten (1) und (2) nicht strikt in di Sinne.

Subtrahiert man beispielsweise auf einer an vielen Hochschulen installier ten Großrechenanlage die beiden Zahlen a = 134 217 728.0 und b = 134 217 so erhält man in allen Sprachen ALGOL, FORTRAN, PASCAL usw. als Ergebnis und nicht 1 [25]. Der relative Fehler ist also 1 und nicht 10^{-8}, was nach (2) zu erwarten gewesen wäre. Was passiert auf der Rechenanlage? Die beid Zahlen lauten im Dualsystem $a = 0.100...0 * 2^{28}$ und $b = 0.11 \ldots 1 * 2^{27}$. Sie sind in der Rechenanlage ohne Rundung exakt darstellbar. Vor der Sub traktion gleicht die Rechenanlage den Exponenten der Zahl b an, d.h. die 2. Zahl wird um eine Stelle nach rechts verschoben. Das Ergebnis erhält m dann so:

0.1 0 0 ... 0	$*2^{28}$
0.0 1 1 ... 1	$1*2^{28}$
0.0 0 0 ... 0	$1*2^{28}$

(5)

Auf der realen Rechenanlage aber wird die letzte 1 des Subtrahenden abgeschnitten. Als Ergebnis der Subtraktion erhält man folglich 2 anstelle von 1.

Ähnliche Beispiele lassen sich auch für andere Großrechenanlagen anführen. Der Effekt ist offenbar unabhängig von der Mantissenlänge, was die Punkte in (5) andeuten sollen.

Führt man bei einer Rechenanlage, welche mit n ziffriger Mantisse arbeitet, die Subtraktion in einem Akkumulator der Länge n aus, so gibt es größenordnungsmäßig b^n [1] Subtraktionen, für die (2) nicht gilt.

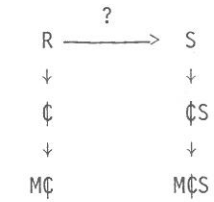
Was soll eine Fehleranalyse, welche auf (2) beruht, wenn schon die Grundformeln derart durchlöchert sind? Die konsequente Folgerung aus dieser Misere besteht darin, daß man beim Bau der Arithmetik deren Eigenschaften durch mathematische Forderungen erklärt, welche vom Hersteller einzuhalten sind.

2.2 Höhere arithmetische Verknüpfungen

Um die Betrachtungen nicht unnötig zu verkomplizieren, beschränken wir uns auf einen kleinen Ausschnitt aus Figur 1, die Räume R, ¢ und M¢, sowie deren auf einer Rechenanlage darstellbaren Teilmenge S, ¢S und M¢S; siehe dazu die Figur 2. Komplexe Matrizen spielen bei vielen numerischen Anwendungen, etwa der schnellen Fourier Transformation eine Rolle. Mit den Verknüpfungen der reellen Zahlen R werden nach bekannten Formeln die Verknüpfungen für die komplexen Zahlen ¢ erklärt und mit diesen wiederum die Verknüpfungen für die komplexen Matrizen M¢ mittels der Skalarproduktformel bzw. der komponentenweisen Addition. Nach denselben Formeln erklärt man nun auch bei herkömmlichen Rechenanlagen die Verknüpfungen für die jeweiligen auf der Rechenmaschine darstellbaren Teilmengen $S \subset R$, $¢S \subset ¢$ und $M¢S \subset M¢$. Dabei nimmt man an, daß die Verknüpfungen in S einigermaßen ordentlich erklärt sind und etwa der Formel (2) genügen. Die folgende Figur veranschaulicht diesen Zusammenhang:

---

[1] b ist die Basis des verwendeten Zahlsystems.



(6)

- Figur 2 -

Das Skalarprodukt etwa in MCS wird zurückgeführt auf die Verknüpfungen in CS und diese wiederum auf diejenigen in S. Mittels der mit den komplexen Zahlen $\alpha = \alpha_1 + i\alpha_2$ und $\beta = \beta_1 + i\beta_2$, sowie den komplexen Matrizen $a = (a_{ij})$ und $b = (b_{ij})$ gebildeten Produktformeln

$$\alpha \cdot \beta = (\alpha_1\beta_1 - \alpha_2\beta_2,\ \alpha_1\beta_2 + \alpha_2\beta_1)$$

$$a \cdot b = \left(\sum_{k=1}^{n} a_{ik} \cdot b_{kj}\right)$$

rechnet man leicht nach, daß bei einer Matrix mit nur 100 Zeilen und Spalten ca. 800 Ungenauigkeitsfaktoren ε_i benötigt werden, um den Fehler der einzelnen Komponenten der Produktmatrix zu beschreiben, für die gesamte Produktmatrix also ca. 8 Millionen Ungenauigkeitsfaktoren ε_i. Eine Fehleranalyse bei großen Algorithmen gestaltet sich dementsprechend schwierig und führt in der Regel auf viel zu grobe und unrealistische Schranken.

Wir werden unten eine Definition der arithmetischen Verknüpfungen in M¢S angeben, welche das Auftreten derart vieler Ungenauigkeitsfaktoren bei einer einzigen Multiplikation in M¢S vermeidet. Was hier für die Zeilen R, ¢ und M¢ gesagt wurde, gilt entsprechend auch für alle anderen Zeilen der Figur 1.

Bei der hier betrachteten, herkömmlichen Definition der Rechnerarithmetik werden alle Verknüpfungen in der Spalte unter S in Figur 1 an die Verknüpfungen in S angehängt. Wir sprechen daher von der senkrechten Methode. Eine unklare Definition der Eigenschaften der Arithmetik in S setzt sich dadurch in alle darunter liegenden Räume der Figur 1 fort.

2.3 Fehleranalyse bei numerischen Algorithmen

Die traditionelle Fehleranalyse numerischer Algorithmen unterscheidet die beiden Formen der "Vorwärts-Analyse" und der "Rückwärts-Analyse".

Die natürliche Weise Rechenungenauigkeiten zu betrachten, ist die Vorwärts-Analyse. Dabei fragt man einfach wie weit das errechnete Resultat vom exakten Resultat abweicht. In der herkömmlichen Numerik hat es sich nahezu als ein Axiom erhärtet, daß die Vorwärts-Analyse numerischer Algorithmen im allgemeinen zu schwierig bzw. nicht möglich ist.

Statt dessen betreibt man die Rückwärts-Analyse. Bei ihr wird das errechnete Ergebnis interpretiert als das exakte Ergebnis zu veränderten Eingangsdaten. Je kleiner die Abweichung von den Eingangsdaten ausfällt, umso größer ist die Genauigkeit des berechneten Ergebnisses. Die Rückwärts-Analyse hat durchaus Erfolge aufzuweisen. Dennoch haftet ihr der Makel an, daß sie im Grunde nur Ersatz ist für eine im allgemeinen nicht ausführbare Vorwärts-Analyse. Darüber hinaus kann man sich auch auf ihre Ergebnisse nicht streng verlassen, da auch die Rückwärts-Analyse mit den im allgemeinen nicht strikt realisierten Formeln (1) und (2) arbeitet.

Die unten angegebene Definition einer Rechnerarithmetik erlaubt es hingegen, auf Grund eines ziemlich allgemein verwendbaren Prinzipes sehr genaue Schranken für die Lösung eines gegebenen Problems mit dem Rechner selbst zu berechnen. Im allgemeinen lassen sich damit auch scharfe Schranken für die Lösung von Problemen mit ungenauen Ausgangsdaten ermitteln.

Das Bedürfnis nach einer mittels mathematischer Methoden für spezielle Algorithmen durchgeführten Vorwärts- oder Rückwärts-Analyse erscheint damit als sekundär. Da eine mathematische Theorie immer eine ganze Klasse von Problemen betrachten muß, werden die vom Rechner mittels der individuellen Daten errechneten Schranken für die Lösung des Problems i.a. ohnehin auch schärfer ausfallen.

3 Die neue Definition der Rechnerarithmetik mittels Semimorphismen

Versuche, den Begriff Rechnerarithmetik zu präzisieren und zu formalisieren wurden schon des öfteren unternommen. Man orientierte sich dabei stets am Übergang von den reellen Zahlen zu den Gleitkommazahlen (1. Zeile der Figur 1). Die reellen Zahlen aber besitzen sehr viele spezielle Eigenschaften, so daß es schwierig sein dürfte, allein an diesem Modell die mathematisch tragenden Eigenschaften herauszufinden. Eine allgemeine Theorie läßt sich zudem schwerlich an einem einzigen Modell entwickeln, man muß nach weiteren Modellen Ausschau halten. Diese bieten sich an in der Gesamtheit der Zeilen der Figur 1.

3.1 Eigenschaften von Semimorphismen

Wir wollen jetzt die Verknüpfungen in den Räumen der 3. und 4. Spalte der Figur 1 nach einem allgemeinen Prinzip erklären. Zunächst können wir feststellen, daß die Verknüpfungen in einigen Räumen der 2. Spalte der Figur 1 wohl bekannt sind. Es sind dies die Räume R, VR, MR, ¢, V¢ und M¢. Es bezeichne zunächst M eine dieser Mengen und * eine darin erklärte Verknüpfung. Dann können wir auch in der Potenzmenge PM von M, das ist die Menge aller Teilmengen von M, eine Verknüpfung * erklären durch die Vorschrift

$$\bigwedge_{A,B \in PM} A*B := \{a * b \mid a \in A \wedge b \in B\} . \qquad (7)$$

Mittels dieser Definition lassen sich alle Verknüpfungen in M in die zugeordnete Potenzmenge PM übertragen. Wir können damit feststellen, daß in den in Figur 1 aufgeführten Mengen die Verknüpfungen bekannt sind jeweils in dem Element ganz links einer jeden Zeile. Das allgemeine Prinzip besteht nun darin, ausgehend von diesen Verknüpfungen auch in den rechts davon stehenden Teilmengen Verknüpfungen nach einer geeigneten Vorschrift zu definieren.

Dazu bezeichne M jetzt irgendeine Menge der Figur 1, in der Verknüpfungen bekannt sind, und N die unmittelbar rechts davon stehende Teilmenge der Figur 1. Für jede Operation * in M erklären wir eine Operation ⊛ in N nach der Vorschrift:

$$(RG) \quad \bigwedge_{a,b \in N} \; a \circledast b := \square (a * b) \qquad \text{für alle } * \in \{+,-,\cdot,/\} .$$

Dazu bezeichnet $\square : M \longrightarrow N$ eine monotone und antisymmetrische Projektion von M in N, welche wir als Rundung bezeichnen. Diese Abbildung hat folgende Eigenschaften

$$(R1) \quad \bigwedge_{a \in N} \; \square a = a \qquad \text{(Rundung)}$$

$$(R2) \quad \bigwedge_{a,b \in M} \; (a \leqq b \Longrightarrow \square a \leqq \square b) \qquad \text{(Monotonie)}$$

$$(R4) \quad \bigwedge_{a \in M} \; \square (-a) = - \square a \qquad \text{(Antisymmetrie)}$$

Wenn man vor die Aufgabe gestellt wird, eine gegebene algebraische Struktur durch eine andere zu ersetzen, anzunähern oder zu approximieren, so wird man zunächst versuchen, auf bewährte mathematische Abbildungseigenschaften wie Isomorphismus oder Homomorphismus zurückzugreifen. In allen Zeilen der Figur 1 treten aber als unmittelbare Nachbarn Mengen auf, die nicht gleich mächtig sind und zwischen nicht gleichmächtigen Mengen kann es keinen Isomorphismus geben. Durch einfache Beispiele oder auch durch einen mathematischen Satz [24] läßt sich darüberhinaus zeigen, daß sich auch Homomorphismen auf sinnvolle Weise nicht einrichten lassen. Man kann jetzt versuchen, die Abbildungseigenschaften des Homomorphismus weiter abzuschwächen. Die oben aufgeführten Abbildungseigenschaften (RG) bis (R4) lassen sich als notwendige Bedingungen an einen Homomorphismus zwischen geordneten algebraischen Strukturen in M und N ableiten. Wir bezeichnen eine Abbildung, welche die Eigenschaften (RG), (R1), (R2) und (R4) besitzt, daher als einen Semimorphismus. Man zeigt nun leicht, daß ein Semimorphismus von einem Semimorphismus wieder ein Semimorphismus ist. Mit einem Semimorphismus geht man gewissermaßen so nahe an einen Homomorphismus heran wie möglich. Auch alle in Figur 1 auftretenden äußeren Verknüpfungen (Skalar mal Vektor, Matrix mal Vektor usw.) erklären wir durch entsprechende Semimorphismen.

Im Falle der Intervallräume der Figur 1 verlangen wir von der Rundung □ darüberhinaus noch die Eigenschaft

$$(R3) \quad \bigwedge_{a \in M} \; a \leqq \square a . \qquad \text{(nach oben gerichtet)}$$

In diesem Falle bezeichnet $\leqq$ die Inklusion $\subseteq$.

Man mache sich an dieser Stelle deutlich den Unterschied zwischen der herkömmlichen und der neuen Definition der Rechnerarithmetik klar. Bei der letzteren werden die Verknüpfungen in einer in der Figur 1 auftretenden Teilmenge direkt erklärt anhand der Verknüpfungen in der unmittelbar links daneben stehenden Grundmenge mittels der Vorschriften des Semimorphismus. Da wir oben bereits gesehen haben, daß die Verknüpfungen in den Mengen ganz links einer jeden Zeile der Figur 1 wohl definiert sind, sind sie damit in jeder in der Figur 1 auftretenden Menge erklärt. Die Verknüpfungen in M¢S (vergleiche dazu die Figur 3) werden also beispielsweise direkt definiert anhand der Verknüpfungen in M¢ und nicht auf dem Umwege über ¢, R, S, ¢S nach M¢S, wie im Falle der vertikalen Methode.

$$\begin{array}{ccc} R & \longrightarrow & S \\ \downarrow & & \\ ¢ & \longrightarrow & ¢S \\ \downarrow & & \\ M¢ & \longrightarrow & M¢S \end{array}$$

- Figur 3 -

Bei der Definition der arithmetischen Verknüpfungen in einer Teilmenge $N \subseteq$ der Figur 1 mittels Semimorphismus erhält man in allen Fällen aufgrund der Formeln (RG), (R1), (R2) maximale Genauigkeit in dem Sinne, daß zwischen de Ergebnis einer Verknüpfung und seiner Approximation kein weiteres Element aus N liegt.

Darüberhinaus lassen sich im Falle des Überganges von R nach S (1. Zeile de Figur 1) die obigen Fehlerformeln (1) und (2) jetzt als mathematischer Satz beweisen [12], [15]. Diese Fehlerformeln gelten jetzt aber auch für alle ar deren Zeilen der Figur 1, also beispielsweise auch für die Verknüpfungen re eller oder komplexer Gleitkommamatrizen:

$$\bigwedge_{a \in M} |\square a - a| < \varepsilon^* |a|$$

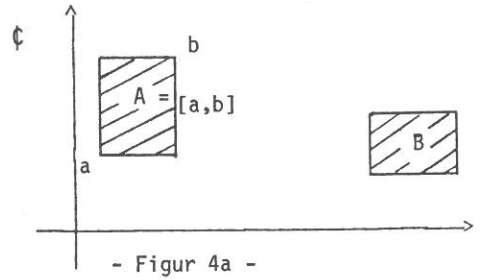
$$\bigwedge_{a,b \in N} |a \circledast b - a * b| < \varepsilon^* |a * b|$$

mit einem einzigen skalaren und zwar dem gleichen ε^* wie in (3) und (4). Die Vereinfachung gegenüber den oben abgezählten 8 Millionen ε_i bei einem einfachen Produkt zweier komplexer Matrizen der Dimension 100 ist also offensichtlich. Fehlerabschätzungen für komplexere Algorithmen, wie etwa die Matrixinvertierung gestalten sich dementsprechend einfacher und fallen wesentlich schärfer aus [6].

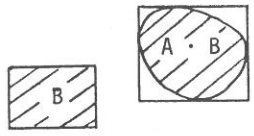
Wir wollen jetzt noch kurz auf die Herleitung und Implementierung von Semimorphismen zu sprechen kommen.

3.2 Zur Herleitung von Semimorphismen

Wir haben oben schon erwähnt, daß sich die wesentlichen Eigenschaften des Semimorphismus als notwendige Bedingungen an einen Homomorphismus gewinnen lassen. Es gibt aber noch weitere Möglichkeiten, diese Eigenschaften herzuleiten. An einigen Modellen der Figur 1 kann man sie direkt anschaulich ablesen. Betrachten wir etwa den Übergang von der Potenzmenge der komplexen Zahlen P¢ zu den Intervallen über den komplexen Zahlen I¢. Ein Intervall [a,b] zwischen zwei vergleichbaren komplexen Zahlen $a \leqq b$ ist ein Rechteck in der komplexen Zahlenebene (Figur 4a). Wenn man zwei komplexe Intervalle A und B im Sinne der Vorschrift (7) miteinander multipliziert, erhält man i.a. nicht wieder ein Intervall, sondern ein Element $A \cdot B$ der Potenzmenge P¢.



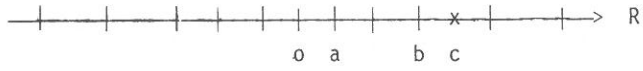
- Figur 4a -



- Figur 4b -

Als Ergebnis einer Intervallverknüpfung aber will man wieder ein Intervall bekommen. Es bietet sich daher an, das Produkt $A * B$ auf das kleinste einschließende Intervall (Figur 4 b) abzubilden. Man kann sich nun unmittelbar anschaulich davon überzeugen, daß diese Abbildung $\square : P¢ \longrightarrow$ alle Eigenschaften (R1,2,3,4) und (RG) eines Semimorphismus besitzt. Die nungsrelation ist hier natürlich die Inklusion $\subseteq$. Wenn die Menge $A \cdot B$ reits ein Intervall ist, bewirkt die Abbildung $\square$ keine Änderung (R1). We man die Menge $A \cdot B$ aufbläht, wird auch das kleinste einhüllende achsenp lele Viereck vergrößert (R2). (R3) ist unmittelbar klar und (R4) ergibt aus Symmetrieüberlegungen. Nach unserer Konstruktion erfüllt das Verknüp ergebnis ferner die Formel (RG) $A \circledast B = \square (A * B)$.

Man kann diese Eigenschaften nun auch am Modell der ersten Zeile der Figur 1 veranschaulichen. Will man zwei Gleitkommazahlen a und b addieren, ist die korrekte Summe $c = a + b$ i.a. nicht wieder eine Gleitkommazahl (Figur 5). Um wieder eine Gleitkommazahl zu erhalten, rundet man das Erg nis jetzt noch in das Raster der Gleitkommazahlen.



Dieser Rundungsprozeß erfüllt nun ebenfalls wieder die Eigenschaften (R1 (R2) und (R4). Die Ordnungsrelation ist hier das $\leqq$. Die Verknüpfung erfüllt die Formel (RG).

Natürlich kommt man mit der Anschauung nicht bei allen Modellen der Figu durch. Dies ist auch gar nicht notwendig; denn seine eigentliche Begründ erhält das Abbildungsprinzip des Semimorphismus dadurch, daß es gewisse volle Strukturen, sogenannte geordnete bzw. schwach geordnete Ringoide u Vektoide invariant läßt. Auf diese Eigenschaften kann aber hier nicht we eingegangen werden. Siehe dazu [12], [15].

3.3 Implementierung von Semimorphismen

Es stellt sich natürlich die Frage, ob sich die Regeln des Semimorphismus auch auf Rechenanlagen in allen Zeilen der Figur 1 durch schnelle Algorithmen implementieren lassen. Zunächst sieht es so aus, als ob eine Implementierung der Vorschrift (RG) schon im Falle der ersten Zeile der Figur 1 unmöglich wäre, da das Resultat a * b auf der Rechenanlage nicht in allen Fällen darstellbar sein wird (sonst bräuchte man es ja nicht zu approximieren). Ist beispielsweise a von der Größenordnung 10^{50} und b von der Größenordnung 10^{-50}, so bräuchte man zur Darstellung der Summe a + b etwa 100 Dezimalstellen und auch die größten Rechenanlagen verfügen über keine so langen Register. Man kann jedoch zeigen, daß in allen Fällen, in denen das Verknüpfungsergebnis a * b auf der Rechenanlage nicht darstellbar ist, ein geeignetes Ersatzergebnis $a \tilde{*} b$ angegeben werden kann, welches darstellbar und effektiv berechenbar ist und die Eigenschaft $\square (a * b) = \square (a \tilde{*} b)$ hat. Dies kann zur Definition von $a \boxed{*} b$ verwendet werden nach der Formel

$$\bigwedge_{a,b \in S} a \boxed{*} b := \square (a \tilde{*} b) = \square (a * b) .$$

Der Nachweis dieser Aussage wird in allen nichttrivialen Fällen durch detaillierte Angabe der betreffenden Algorithmen (Fallunterscheidungen) gegeben [12], [15] .

Bei sorgfältiger Implementierung dieser Algorithmen fallen leicht auch die beiden monotonen gerichteten Rundungen ∇ und Δ , welche erklärt sind durch (R1), (R2) und

(R3) $$\bigwedge_{a \in R} \nabla a \leqq a \qquad\qquad \bigwedge_{a \in R} a \leqq \Delta a ,$$

sowie die mittels dieser Verknüpfungen nach (RG) erklärten Verknüpfungen $\boxed{*}$, $\boxed{\tilde{*}}$, $* \in \{+,-,\cdot,/\}$, an. Die Rundungen ∇, Δ und die Verknüpfungen $\boxed{*}$ und $\boxed{\tilde{*}}$ werden für eine saubere Fehleranalyse in der Numerik benötigt. Sie sind mit den Verknüpfungen für Intervalle über S äquivalent. Eine genaue Analyse zeigt, daß mit diesen Algorithmen die Implementierung von Semimorphismen möglich ist in den Zeilen 1, 2 und 4 und 5 der Figur 1 .

Eine Lücke bleibt zunächst in der Zeile 3. Schwierigkeiten bereitet hier vo allem die Multiplikation von Gleitkommamatrizen $a = (a_{ij})$ und $b = (b_{ij})$. Di Formel (RG) verlangt hier eine Implementierung der Vorschrift

$$a \boxed{\cdot} b := \square (a \cdot b) = \square \left(\sum_{k=1}^{n} a_{ik} \cdot b_{kj} \right) .$$

Dabei treten in der Summe ganz rechts die Addition und Multiplikation reeller Zahlen auf. Die Rundung darf nur am Schluß ein einziges Mal (komponente weise) ausgeführt werden. Berücksichtigt man, daß hier a_{ij} und b_{ij} Gleitkommazahlen sind, und daß die Produkte $a_{ik} \cdot b_{kj}$ in einem doppeltlangen Akk mulator korrekt dargestellt werden können, so reduziert sich das Problem au eine Realisierung der Formel

$$c = \square \left(\sum_{k=1}^{n} c_k \right) \qquad (8)$$

Darin sind die c_i, $i = 1(1)n$, doppeltlange Gleitkommazahlen und c ist eine einfachlange Gleitkommazahl. Das Summenzeichen in (8) beschreibt die korrel te Addition für reelle Zahlen. Da man natürlich in der Lage sein will, Matrizen beliebiger Dimension zu multiplizieren, verlangt die Vorschrift (8) Algorithmen, welche in der Lage sind, beliebig lange Summen von Gleitkomma zahlen bis auf eine Einheit der letzten Ziffer genau auszuführen. Ein solcher Algorithmus wurde zunächst in [13] angegeben. Ein sehr eleganter weit rer Algorithmus findet sich in [2] . Beide Algorithmen sind auch in [15] b schrieben. Inzwischen sind weitere Algorithmen zur Lösung dieser Fragestel lung angegeben worden. Man überlegt sich leicht, daß mit diesen Algorithme das Problem der Implementierung von Semimorphismen auch gelöst ist für die Zeilen 3, 7, 8 und 9 der Figur 1.

Die Frage der Implementierung bleibt damit zunächst noch offen in den Zeil 6, 10, 11 und 12 der Figur 1. Hier begegnet man abermals einer neuen Probl matik, welche noch kurz geschildert werden soll.

Die Verknüpfungen in IMR (Zeile 6 der Figur 1) wurden erklärt mittels der Formel

$$(RG) \qquad \bigwedge_{A,B \in IMR} A \circledast B := \square (A * B) \quad \text{für alle} \quad * \in \{+,-,\cdot\} .$$

Darin bezeichnet * das Verknüpfungszeichen in PMR, welches nach (7) erklärt ist. Die Verknüpfungen in der Potenzmenge aber sind prinzipiell nicht ausführbar.

Unabhängig von der Menge IMR kann man aber die Menge MIR, das sind Matrizen, deren Komponenten Intervalle sind, betrachten. In MIR lassen sich ebenfalls Verknüpfungen einführen:

$$A \oplus B := (A_{ij} \boxplus B_{ij}) ,$$

$$A = (A_{ij}), \bigwedge B=(B_{ij}) \in MIR$$

$$A \odot B := (\square (\sum_{k=1}^{n} A_{ik} \boxdot B_{kj}))$$

Dabei werden auf der rechten Seite die ausführbaren Verknüpfungen in IR verwendet. Bereits in [10] wurde gezeigt, daß die Verknüpfungen in IMR und MIR bezüglich der Abbildung

$$([a_{ij}{}^1, a_{ij}{}^2]) \longleftrightarrow [(a_{ij}{}^1), (a_{ij}{}^2)]$$

isomorph sind.

In [21], [22] wurde dann nachgewiesen, daß entsprechende Isomorphismen auch gelten, wenn man in den Räumen der letzten drei Zeilen der Figur 1 das I nach rechts durchschiebt:

$$\begin{aligned} I\text{¢} &\longleftrightarrow \text{¢}IR , \\ IV\text{¢} &\longleftrightarrow V\text{¢}IR , \\ IM\text{¢} &\longleftrightarrow M\text{¢}IR . \end{aligned}$$

Es sei hier darauf verzichtet, die entsprechenden Abbildungen anzugeben. Di Verhältnisse sind natürlich komplizierter.

In [14] wurde schließlich gezeigt, daß diese Isomorphien erhalten bleiben, wenn man mittels Semimorphismen in die betreffenden Teilmengen der 3. und 4 Spalte der Figur 1 fortschreitet. Es gilt also beispielsweise

$$\begin{aligned} IMS &\longleftrightarrow MIS , \\ I\text{¢}S &\longleftrightarrow \text{¢}IS , \\ IV\text{¢}S &\longleftrightarrow V\text{¢}IS , \\ IM\text{¢}S &\longleftrightarrow M\text{¢}IS . \end{aligned}$$

Der Nachweis dieser Isomorphien muß natürlich unter Einschränkung auf die i den betreffenden Teilmengen vorliegenden schwachen Strukturen geführt werde welche zu diesem Zwecke sorgfältigst analysiert werden müssen. Der Nachweis von Isomorphien für die betreffenden äußeren Verknüpfungen erfolgt analog.

Mittels dieser Isomorphien kann man dann zeigen, daß ein Algorithmus zur op timalen Berechnung von Skalarprodukten auch dazu benutzt werden kann, um di Verknüpfungen in den Mengen der Zeilen 6, 10, 11 und 12 mittels Semimorphis men auf optimale Weise zu implementieren. Die Algorithmen zur optimalen Berechnung von Skalarprodukten, welche ursprünglich für die Zeile 3 entwickel wurden, gewährleisten demnach auch die Implementierung von Semimorphismen i den Zeilen 6 bis 12. Siehe dazu insbesondere das Kapitel 6 in [15].

4 Rechnerarithmetik und Programmiersprachen

Herkömmliche Programmiersprachen wie etwa ALGOL, FORTRAN, PASCAL, PL/1 usw. kennen als Elementarverknüpfungen nur die integer und die real Verknüpfunge Alle anderen Verknüpfungen werden darauf zurückgeführt. Dies hat ein umstän liches und vor allem zeitraubendes Hantieren mit höheren numerischen Einheiten, wie Vektoren, Matrizen, komplexen Zahlen, komplexen Vektoren und Matrizen, sowie den Intervallen über allen genannten Räumen zur Folge.

Wir betrachten als Beispiel etwa die Multiplikation zweier n-reihiger Matrizen $a = (a_{ij})$ und $b = (b_{ij})$. Sie erfolgt in allen genannten Sprachen mittels der Laufanweisung. Wir notieren von den drei nötigen Schleifen nur die innerste mit einer Summationsvariablen s :

```
.................................
for k := 1 to n do
    s := s + a [i,k] * b [k,j] ;        (11)
.................................
```

Was muß der Rechner tun, wenn er einmal diese innerste Schleife durchläuft? Er sucht zunächst die Anfangsadresse des Feldes a . Wir bezeichnen diese einfach mit a . Dann berechnet er die Adresse der Komponente a [i,k] . Wenn die Matrix zeilenweise gespeichert ist, geschieht dies in folgender Weise:

$$a + (i-1) \cdot n + k \ .$$

Anschließend berechnet er die Adresse der Komponente von b :

$$b + (k-1) \cdot n + j \ .$$

Um das eigentlich gewünschte Produkt a [i,k] * b [k,j] ausführen zu können, muß der Rechner also zwei Suchprozesse ausführen und dazu noch zwei Multiplikationen und vier Additionen. Bei großen Matrizen handelt es sich bei i, n, k und j keineswegs um kleine Zahlen.

Beim nächsten Durchlauf der innersten Schleife werden dann aufgrund der ungeschickten Notation die Adressen von a [i,k+1] und b [k+1,j] berechnet. Dies geschieht wiederum mit zwei Suchprozessen, zwei Multiplikationen und vier Additionen.

Dabei steht die Komponente a [i,k+1] in der auf die Komponente a [i,k] folgenden Speicherzelle. Die Komponente b [k+1,j] steht n Plätze weiter als b [k,j]. Mit einem Zählprozeß und einer einfachen Addition ließen sich also die Operanden für den nächsten Term des Skalarproduktes finden.

Die hier angesprochene Problematik ist den Compilerbauern natürlich bekannt und zu dem Stichwort Indexfortschaltung gibt es umfangreiche Untersuchunge Praktisch schlagen sich diese häufig in einem zusätzlichen Übersetzungslau zur Optimierung der Laufanweisung nieder. Tatsache aber ist, daß hier ein imaginäres Problem gelöst wird, welches bei geeigneterer Notation überhaup nicht entsteht.

Die herkömmlichen Programmiersprachen erlauben nur im Falle der ersten Zei le der Figur 1 die arithmetischen Verknüpfungen mittels der in der Mathema tik üblichen Operationszeichen +,-,•,/ anzusprechen. Alle Verknüpfungen in den darunter liegenden Zeilen müssen durch Prozeduren bereitgestellt und häufig vom Benutzer selbst formuliert werden. Treten in einem Ausdruck für Matrizen oder komplexe Zahlen mehrere Verknüpfungen auf, so erfordert jed einen eigenen Prozeduraufruf. Dieses Vorgehen hat lange und unübersichtlic Programme zur Folge.

Ein einfacher Ausweg aus diesen Schwierigkeiten stellt sich unmittelbar ei wenn man in Programmiersprachen alle Verknüpfungen für höhere numerische E heiten wie reelle oder komplexe Zahlen, -Vektoren und -Matrizen, sowie auc die Verknüpfung für Intervalle über diesen Räumen wie in der Mathematik üb lich mittels Operatoren notiert, also z.B. die Zuweisung des Produktes zwe Matrizen a und b an eine Matrix x in der Weise

$$x := a \cdot b \ . \qquad (12)$$

Dabei muß man sich a und b als Variable vom Typ real matrix vereinbart de ken. Eine Zuweisung eines arithmetischen Ausdruckes für Matrizen würde da etwa so aussehen:

$$z := (a * x + b) * y + c \ .$$

Man mache sich den Unterschied klar, indem man das Problem für die Berech nung des auf der rechten Seite stehenden Ausdruckes einmal in herkömmlich Weise mittels Laufanweisung und Prozeduraufruf niederschreibt.

Durch eine solche Operatornotation werden die Programme kürzer und übersi licher. Sie sind leichter zu lesen und zu schreiben und folglich auch le ter auszutesten. Sie werden dadurch auch zuverlässiger. Ein anderer ganz sentlicher Vorteil kommt aber noch hinzu:

Bei der Schreibweise (12) wird nicht die sonst übliche vertikale (ungenaue) Definition der Matrixverknüpfungen benutzt, sondern die vertikale Methode unter Zuhilfenahme des genauen Skalarprodukts. Der Fehler einer Matrixoperation kann also durch eine einzige Größe ε^* abgeschätzt werden.
Bei der Schreibweise (12) werden die Komponenten der Matrizen a und b nicht mehr genannt. Folglich brauchen deren Adressen auch nicht berechnet zu werden. Die Multiplikation wird mittels einer internen Skalarproduktroutine ausgeführt, welche den einfachen Zähl- und Additionsprozeß automatisch ausführt. Ferner braucht nach den einzelnen Additionen und Multiplikationen im Skalarprodukt weder eine Normalisierung noch eine Rundung ausgeführt zu werden, was ebenfalls zur Beschleunigung beiträgt. Für die Verknüpfungen aller oben genannten höheren numerischen Einheiten lassen sich dadurch zum Teil ganz beträchtliche Geschwindigkeitssteigerungen erzielen.
Abschließend sei noch erwähnt, daß die Definition der arithmetischen Verknüpfungen mittels Semimorphismus eine Operatornotation wie in (12) praktisch erzwingt. Eine Definition der Matrixmultiplikation mittels dreier Laufanweisungen wie in (11) würde ja wieder auf die herkömmliche Methode hinauslaufen.

5 Realisierung und Ausblick

Eine vollständige Implementierung der Arithmetik, wie sie in Figur 1 skizziert wurde, mittels Semimorphismen wurde am Institut des ersten Autors erstmals durchgeführt. Der Implementierung lag zunächst ein Mikroprozessor Z80 zugrunde. Als Basissprache wurde PASCAL verwendet. Um vor allem die im Abschnitt D. geschilderten Schwierigkeiten zu vermeiden, wurde die Sprache PASCAL um ein allgemeines Operatorkonzept, ähnlich wie es in ALGOL-68 vorgesehen war, erweitert. Die erweiterte Sprache wurde PASCAL-SC (PASCAL for Scientific Computation) genannt. [1)]
Die höheren Arithmetikroutinen sind bereits in diesem Operatorkonzept formuliert. Es ergeben sich ganz überraschende Effekte. Obwohl wesentlich grössere Sorgfalt als üblich bei der Implementierung der arithmetischen Verknüpfungen mittels Semimorphismen aufgewendet werden muß, sind die Verknüpfungen von Zeile 3 an schneller, als wenn sie auf herkömmliche Weise im Basis-PASCAL ausgeführt werden. Beispielsweise ist die Matrixmultiplikation mittels Semimorphismus und Operatorkonzept etwa doppelt so schnell, als wenn sie im herkömmlichen PASCAL programmiert und ausgeführt wird.

---

1) Der Compiler wurde an der Universität Kaiserslautern (Professor Dr. H.-W. Wippermann) implementiert.

Die Auswirkungen der neuen Arithmetik in der Numerik sind enorm und noch kaum zu übersehen. Der Rechner ist nicht länger ein Werkzeug des Experimentierens, sondern wird zu einem Instrument der Mathematik. Binnen kurzer Zeit war es möglich, auf dem neuen Rechnersystem Programmpakete zu entwickeln für die Lösung linearer Gleichungssysteme, die Invertierung von Matrizen, Eigenwertprobleme, Nullstellen von Polynomen, nichtlineare Gleichungssysteme, Auswertung beliebiger arithmetischer Ausdrücke (mathematischer Funktionen), Optimierungsprobleme, numerische Quadratur, Anfangs- und Randwertaufgaben bei gewöhnlichen Differentialgleichungen, iterative Behandlung großer Gleichungssysteme usw. Derartige Routinen werden im folgenden beschrieben. Dabei wird bei 12-stelliger dezimaler Rechnung die Lösung jeweils auf mindestens 11 Stellen genau in Schranken eingeschlossen. Die numerischen Algorithmen beweisen zudem die Existenz und Eindeutigkeit der Lösung innerhalb der berechneten Schranken. Ist diese nicht gegeben (z.B. bei einer singulären Matrix) oder ist ein spezielles Problem so schlecht konditioniert, daß die Lösung mit 12-stelliger Arithmetik nicht berechnet werden kann (es geht dann auch nicht mit üblicher Gleitkommarechnung!), so stellt der Algorithmus dies fest und teilt es dem Benutzer mit. Dies alles sind Ergebnisse, welche mit der herkömmlichen Rechnerarithmetik nicht erreicht worden sind. Sie können auch dann nicht erreicht werden, wenn man die Arithmetik in der 1. Zeile der Figur 1 sorgfältigst mittels eines Semimorphismus definiert. Eine Gruppe, vor allem amerikanische Wissenschaftler, hat gerade im Rahmen der IEEE Computer Society in dieser Richtung einen Standard geschaffen. Ganz sicherlich ist dies ein Schritt in die richtige Richtung. Man muß sich aber fragen, warum man nicht gleich den hier noch einmal unterbreiteten, älteren und wesentlich weitergehenden Vorschlag übernimmt, welcher den eigentlichen Durchbruch in der Numerik erst ermöglicht.

Nachträglich betrachtet, erscheint die ursprüngliche Zustimmung, Festlegung und Implementierung der Rechnerarithmetik dem Hersteller zu überlassen, als ein mathematischer Schildbürgerstreich. Wenn man die Genauigkeit numerische Algorithmen kontrollieren will, und dieser Wunsch ist sicherlich legitim, so muß man zunächst einmal die Genauigkeit der einfachsten Algorithmen, und da sind eben die Algorithmen für die arithmetischen Verknüpfungen unter Kontrolle halten und diese genau und präzise erklären. Diese einfache notwendige Bedingung erweist sich in vielen Fällen dann auch bereits als hinreichend um das Dunkel der Rechenungenauigkeit auch bei komplizierteren Algorithmen aufzuhellen, wie das Fenster in dem Rathaus von Schilda.

6 Schlecht konditionierte Probleme

Bevor wir in eine Diskussion über Sicherheit von Ergebnissen einsteigen, soll zunächst die Notwendigkeit anhand einiger Beispiele gezeigt werden.

In einer Rechenanlage können nur endlich viele Zahlen verarbeitet werden, insbesondere nur Zahlen mit einer endlichen Mantissenlänge. Bei der Darstellung oder Konvertierung des gestellten Problems auf dem Rechner wird ein Konvertierungsfehler begangen. Kommt in dem Problem etwa die Zahl π vor und ein Rechner hat 8 Dezimalstellen, wird π durch die π am nächsten gelegene Maschinenzahl approximiert:

3.1415927 statt 3.1415926535...

Verwendet der Rechner Dualzahlen, tritt zusätzlich noch das Problem der Konvertierungsfehler auf. Dann ist nämlich

die Dezimalzahl 0.1

auf dem Rechner nicht darstellbar, da 0.1 im Zehnersystem eine unendliche Dualzahl 0.00011001..ist. Bei der Eingabe des Problems wird also bereits ein (Eingabe-) Konvertierungsfehler begangen.

Eine Operation auf dem Rechner kann i.a. ebenfalls nicht exakt ausgeführt werden. Am besten approximiert man das exakte, reelle Ergebnis bestmöglich, also etwa

1/6 = 0.16666667.

Realisiert man diese intuitive Definition einer Rechnerarithmetik strikt, resultiert dies zwangsläufig in der vorstehenden gegebenen Definition der Rechnerarithmetik. Offensichtlich sind aber selbst bei wenigen Operationen beliebig große Fehler möglich. Etwa bei der Berechnung von

(13) $$10^8 + 7 - 10^8 - 2$$

erhält man als Ergebnis auf einem 8-stelligen Rechner (bei Ausführung von links nach rechts)

- 2 statt des richtigen Ergebnisses +5 .

Auf einem Rechner eines bekannten deutschen Versandhauses berechnet man gar

(14) $$10^8 - 99\,999\,999 = 10 .$$

Der in (13) begangene Fehler ist prinzipieller Natur. Der Fehler in (14) läßt sich hingegen beheben. Es ist klar, daß, wenn die Arithmetik nicht vernünftigen Gesetzen genügt, man auch keine gesicherten Ergebnisse erwarten kann.

Der Term "sichere Ergebnisse" verdient es, etwas genauer betrachtet zu werden. Setzen wir zunächst voraus, daß alle Daten des Problems exakt auf dem Rechner darstellbar sind und betrachten eine lineare Kurvenanpassung nach der Methode der kleinsten Quadrate. Gegeben sei die Wertetabelle

(15)

y	99999	100000	100001
x	a-1	a	a+1

Für verschiedene Werte von a soll ein "curve fitting" durchgeführt werden. Im vorliegenden Fall scheint das besonders einfach, da alle drei Punkte bereits auf einer Geraden liegen. Die beste Kurvenanpassung ist nach bekannten Formeln gegeben durch

$$y = mx + b \quad \text{mit } m = \frac{\Sigma x_i y_i - \frac{1}{n} \Sigma x_i \, \Sigma y_i}{\Sigma x_i^2 - \frac{1}{n} (\Sigma x_i)^2} \quad \text{und } b = \frac{1}{n} \Sigma y_i - \frac{m}{n} \Sigma x_i \, .$$

Die nachfolgenden Ergebnisse wurden mit 12-stelliger (!) Dezimalarithmetik berechnet. Dabei spielen keine Konvertierungsfehler und keine Arithmetikfehler eine Rolle und es wird eine maximal genaue Arithmetik verwendet.

a	x = a + 2		x = a+10^5		x = a-10^5	
	exakt	*glk*	exakt	*glk*	exakt	*glk*
5201456	100002	9999.9	200000	90000	0	110000
2568741	100002	100000.02	200000	102000	0	98000
5201478	100002	100000	200000	100000	0	100000

Tabelle 1. Extrapolationswerte für verschiedene x

Nach Durchführung der Kurvenanpassung (die exakt möglich ist mit m = 1, b = 100000 - a) wurde auf x = a + 2, x = a+10^5 und x = a-10^5 extrapoliert. Das ist eine Extrapolation relativ zu a von weniger als 5% für alle Werte von a. Unter *glk* sind die mit 12-stelliger Gleitkomma-Rechnung erzielten Werte aufgeführt. Diese haben mit der wahren Lösung so gut wie nichts zu tun. In der dritten Zeile ist das Resultat sogar eine Konstante.

Im nächsten Beispiel wurde ein vom Institut für Angewandte Mathematik der Universität Karlsruhe entwickelter Rechner benutzt. Er verfügt insbesondere über eine Gleitpunkt-Arithmetik mit maximaler Genauigkeit. Trotzdem ist bei der Verwendung von reinen Gleitpunkt-Algorithmen Vorsicht geboten, wie das folgende Beispiel zeigt:

$$(16) \qquad \begin{aligned} 941664x - 665857y &= 1 \\ 665857x - 470832y &= 0 \end{aligned}$$

Die mit einem Standard-Gleitpunktalgorithmus in 12-stelliger Rechnung erzielten Näherungen sind

$$\begin{aligned} x &= 166666.666667 \\ y &= 235702.260396 \end{aligned}$$

Die wahre Lösung lautet

$$\begin{aligned} x &= 470832.0 \\ y &= 665857.0 \end{aligned}$$

Schwieriger werden die Probleme, wenn die Eingabedaten nicht exakt auf dem Rechner darstellbar sind oder wenn die Eingabedaten gar noch mit Fehlern behaftet sind. Hier kommen wir zur Frage, wann ein Ergebnis richtig heißen soll und wann nicht. Es gibt zwei Möglichkeiten, richtige Ergebnisse auszugeben. Entweder, man gibt Schranken an, zwischen denen die wahre Lösung (mit Sicherheit) liegt, oder, man gibt nur so viele Stellen an, wie garantiert werden können etwa mit der Vereinbarung, daß die wahre Lösung mit genau diesen Stellen beginnt. Der letzte Vorschlag gibt (im Positiven) immer ein Ergebnis aus, das zu klein ist. Möchte man dieses vermeiden, vereinbart man, daß das richtige Ergebnis mit genau diesen Stellen oder mit diesen Stellen und die letzte Stelle um eins vermindert beginnt (bzw. im Negativen um eins erhöht). Demnach wäre etwa in der ersten Darstellung

$$1/6 = [0.16666666,\ 0.16666667], \quad \text{d.h. } 0.16666666 \leqq 1/6 \leqq 0.16666667$$

In der zuletzt beschriebenen Darstellung wäre

$$\text{sowohl } 1/6 = 0.16666666 \text{ als auch } 1/6 = 0.16666667$$

richtig. Interessanter (aber auch schwieriger) wird es bei mehr Operationen oder bei mit Fehlern behafteten Eingabedaten.

Von einer Pyramide sei die mittlere Höhe der 207 Stufen 71 cm. Dieser Meßwert sei mit einem Fehler von $\pm$ 0.25 cm behaftet. Dann liegt offenbar die Gesamthöhe der Pyramide

zwischen 146.4525 m und 147.4875 cm.

Das richtige Ergebnis muß also etwa [146.4, 147.5] oder

147 m

lauten. Es wäre jedoch in dem eben beschriebenen Sinne *falsch*, die Gesamthöhe zu $207 \cdot 0.71 = 146.97$ m anzugeben.

Die Crux der heutigen Rechner (Taschenrechner, Personal Computer wie Großrechenanlagen) ist, daß ein Ergebnis auf so viele Stellen ausgegeben wird, wie die verwendete Genauigkeit beträgt, also 8, 16 oder noch mehr Dezimalstellen. Diese hohe Anzahl von ausgegebenen Stellen für ein (Näherungs-) Ergebnis suggeriert eine hohe Genauigkeit (wie überhaupt digitale Angaben eine Exaktheit suggerieren, z.B. gegenüber analogen Anzeigen).

Ein im Sinne der Angabe von Schranken richtiges Ergebnis wäre natürlich auch [-1000000, + 1000000]. An Solcherlei sind wir aber nicht interessiert. Zur Definition der Richtigkeit gehört also auch die *Schärfe* der Ergebnisse.

Sind die Eingabedaten mit Fehlern behaftet, können sich kleine Fehler (z.B. Meßfehler) unter Umständen sehr stark auf die Lösung auswirken. Betrachten wir etwa

$$(17) \qquad \begin{aligned} 100000x + 99999y &= b_1 \\ 99999x + 99998y &= b_2 \ . \end{aligned}$$

Die rechte Seite sei $(b_1, b_2) = (200000, 200000)$ gemessen mit einem Fehler einer Einheit der 5. Stelle, also

$$\begin{aligned} b_1 &\in [199990,\ 200010] \\ b_2 &\in [199990,\ 200010] \end{aligned}$$

In der rechten Seite enthalten ist u.a. $b_1 = 199999$ und $b_2 = 199997$, was die Lösung x=y=1 ergibt. Bei den gegebenen Grenzen für b_1 und b_2 gilt jedoch

$$\begin{aligned} -1800000 &\leq x \leq 2200000 \\ -2200000 &\leq y \leq 1800000 \ . \end{aligned}$$

Diese Grenzen sind scharf, also nicht verbesserbar.

D.h., liefert ein Instrument Meßwerte für b_1, b_2 mit einem Fehler einer Einheit der 5ten Stelle, so können die Ergebnisse des linearen Gleichungssystems (17) zwischen rund minus 2 Millionen und plus 2 Millionen liegen. Um Ergebnisse mit zwei gesicherten Stellen zu erhalten, muß die rechte Seite mit einem Fehler von höchstens einer Einheit in der 8ten Stelle gemessen werden.

Es besteht die Möglichkeit, die rechte Seite leicht zu ändern und zu versuchen, aus der Änderung der Lösung auf die Abhängigkeit der Lösung von den Daten zu schließen. Gleichwohl ist auch hier große Vorsicht geboten. Berechnet man etwa die Lösung des obigen Gleichungssystems (17) nacheinander für rechte Seiten

$$\begin{pmatrix} 200000 \\ 200000 \end{pmatrix}, \quad \begin{pmatrix} 199990 \\ 199990 \end{pmatrix}, \quad \begin{pmatrix} 200010 \\ 200010 \end{pmatrix},$$

so erhält man als Lösungen nacheinander

$$(18) \quad \begin{pmatrix} 200000 \\ -200000 \end{pmatrix}, \quad \begin{pmatrix} 199990 \\ -199990 \end{pmatrix}, \quad \begin{pmatrix} 200010 \\ -200010 \end{pmatrix}.$$

Aus diesen scheinbar schönen Ergebnissen zu schließen, das Problem sei gutartig, wäre jedoch (wie gezeigt) völlig falsch.

Verändert man die Daten des Problems irgendwie und die Lösung ändert sich immer von etwa gleicher Größenordnung, heißt das Problem gut konditioniert. Ist die Änderung der Lösung stark überproportional wie im obigen Beispiel, heißt es schlecht konditioniert. Die Information über die Kondition eines Problems ist von hoher Wichtigkeit. Sie hat direkte Auswirkung auf die Auslegung (Belastbarkeit) von Komponenten, Genauigkeitsanforderungen an Meßwertgeber etc. und damit wichtige finanzielle Bedeutung. U.U. kann sich auch ergeben, daß einzelne Komponenten praktisch keinen Einfluß besitzen.

Im folgenden sollen Methoden skizziert werden, die sichere Ergebnisse liefern für alle Klassen von Eingabedaten (darstellbar, nicht darstellbar und fehlerbehaftet) für eine große Anzahl mathematischer Probleme. Die im Hintergrund stehende Mathematik wird dabei bewußt anschaulich dargestellt, so daß die folgenden Ausführungen auch ohne tiefen mathematischen Hintergrund nachvollzogen werden können.

Mathematische Hintergründe können in [23], [24], [30], [31] nachgelesen werden.

7 Algorithmen für Grundaufgaben der Numerischen Mathematik

Wir betrachten zunächst das Problem der Lösung linearer Gleichungssysteme. Für eine nxn Matrix A und einen Vektor b mit n Komponenten schreiben wir zunächst

das Gleichungssystem $Ax = b$

um und suchen

die Nullstelle von $Ax - b = 0$.

Nun wird auf Rechnern die Suche nach einer Nullstelle häufig umgeformt in eine

Fixpunktgleichung $x = x - (Ax-b)$.

Zur numerischen Anwendung schreibt man eine solche Gleichung als

Iteration $x^{k+1} := x^k - (Ax^k-b)$.

Die Anwendung und Auswertung einer solchen Iteration läßt sich auf dem Rechner besonders vorteilhaft durchführen. Es sind auch mathematische Kriterien bekannt, wann eine solche Iteration konvergiert und wann nicht. Im vorliegenden Fall wird die Iteration im allgemeinen *nicht* konvergieren.
Daher wird eine solche Iteration umgeschrieben in eine bessere, nämlich das

Newton-Verfahren $x^{k+1} := x^k - A^{-1} \cdot (Ax^k - b)$.

In dieser Form ist das Verfahren natürlich noch nicht brauchbar. Denn wäre die Inverse A^{-1} von A bekannt, könnte man die Lösung x von $Ax = b$ sofort ausrechnen zu $x = A^{-1} \cdot b$. Nun kann man versuchen, das Newton-Verfahren anzunähern, indem nicht die exakte Inverse sondern eine "Näherungs"-Inverse R von A verwendet wird. Man gelangt so zum

(19) vereinfachten Newton-Verfahren $x^{k+1} := x^k - R \cdot (Ax^k - b)$.

Das Verfahren heißt *vereinfachtes* Newton-Verfahren, weil in jedem Iterationsschritt die gleiche Näherung R verwendet wird und nicht in jedem Iterationsschritt eine neue (verbesserte) Näherungsinverse R_i von A verwendet wird. Das Verfahren wird auch als "Residuen-Iteration" bezeichnet. Was bis jetzt dasteht ist alles bekannt, nichts Neues. Was jedoch ebenso bekannt ist, sind die auftretenden Probleme: Wann "konvergiert" eine Iteration? Welches Abbruchkriterium ist zu verwenden? Ist die Matrix des Gleichungssystems nicht singulär? Wie sicher sind die Ergebnisse? und andere mehr. Wir werden später durch Beispiele zeigen, daß keine dieser Fragen zufriedenstellend gelöst war.

Um zu bewiesenen, zu sicheren Ergebnissen zu gelangen, muß ein mathematischer Satz vorliegen, dessen Voraussetzungen auf dem Rechner überprüfbar sind um dessen Aussagen anwenden zu können. Die Grundlage für einen solchen Satz ist der Brouwer'sche Fixpunktsatz:

> Brouwer: Gegeben sei eine stetige Funktion $f:\mathbb{R}^n \to \mathbb{R}^n$, die eine nichtleere, konvexe, abgeschlossene, beschränkte Menge $X \subseteq \mathbb{R}^n$ in sich abbildet. Dann hat f in X mindestens einen Fixpunkt.

Um diesen Satz anwenden zu können, benötigen wir also eine stetige Funktion f und eine adäquate Menge X. Betrachten wir obige Fixpunktiteration, so definiert man naheliegend

$$f:\mathbb{R}^n \to \mathbb{R}^n \text{ mit } f(x) := x - R \cdot (Ax - b).$$

Findet man dann ein $X \subseteq \mathbb{R}^n$ mit $f:X \to X$, dann ist bereits die Existenz eines $\bar{x} \in X$ mit $f(\bar{x}) = \bar{x}$ bewiesen. Dann gilt also

$$\bar{x} = \bar{x} - R(A\bar{x} - b) \qquad R \cdot (A\bar{x} - b) = 0.$$

Es sind also zwei Probleme zu lösen:

1) Es muß $f : X \longrightarrow X$ auf dem Rechner nachgeprüft werden
2) Es muß die Nicht-Singularität von R bewiesen werden, um zu einer Lösung des linearen Gleichungssystems zu kommen.

Gehen wir zunächst das erste Problem an. Mathematisch gesehen kann man in (19) die Näherung x^k durch die Menge X ersetzen und jede Operation durch die entsprechende Potenzmengenoperation:

$$(20) \qquad X - R \cdot (A \cdot X - b).$$

Eine Potenzmengenoperation $C * D$ mit $C,D \subseteq \mathbb{P}\mathbb{R}^n$ ist dabei definiert als

$$C * D := \{x * y \mid x \in C \wedge y \in D\} \quad \text{für } * \in \{+,-,\cdot,/\}.$$

Nun kann man zeigen, daß $f : X \longrightarrow X$ mittels (20) i.a. nicht realisierbar ist. Denn es gilt im allgemeinen

$$X - R(AX - b) \not\subseteq X .$$

Ein Kunstgriff rettet die Situation; wir schreiben statt (20):

$$(21) \qquad R \cdot b + \{I - R \cdot A\} \cdot X.$$

Hierbei ist I die nxn Einheitsmatrix. (21) ist äquivalent zu (19), wie man durch Ausrechnen bestätigt. Jetzt kommt die Menge X nur noch einmal vor, der Wertebereich von (21) wird also nicht mehr überschätzt. Denn das doppelte Auftreten von X in (20) bewirkt, daß jedesmal die ganze Menge X eingesetzt wird:

$$X - R(AX - b) = \{x_1 - R(Ax_2 - b) \mid x_1, x_2 \in X\}.$$

Hierbei sind x_1 und x_2 unabhängig voneinander, obwohl $x_1 = x_2$ angenommen werden könnte. Dieser Mißstand ist mit (21) beseitigt.

Zu einer Lösung unseres ersten Problems ist man aber erst gelangt, wenn die Menge X auf dem Rechner darstellbar und abgespeichert ist und die Operationen $+,-,\cdot,/$ auf dem Rechner ausführbar sind. Betrachten wir dazu als spezielle Teilmengen X des $\mathbb{R}^n$ "Intervallvektoren".

Ein Bereich oder Intervall innerhalb der reellen Zahlen ist die Menge aller Zahlen, die zwischen zwei Grenzen liegt:

$$[-1,3] := \{x \in \mathbb{R} \mid -1 \leq x \leq 3\}.$$

Intervalle wurden bereits im ersten Abschnitt als Lösungsbereiche angesprochen. Mit solchen Bereichen kann man rechnen, und zwar mit der anfangs definierten Rechnerarithmetik. Wichtig für uns ist die Feststellung, daß

a) Bereiche durch Angabe der linken und rechten Grenze auf dem Rechner darstellbar sind und, daß

b) alle Operationen zwischen Bereichen auf dem Rechner ausführbar sind (siehe [12], [15]).

Entsprechend definieren wir Intervallvektoren als die Menge aller Vektoren, die zwischen zwei Grenzen liegen:

$$[\begin{pmatrix}-1\\1\\2\end{pmatrix},\begin{pmatrix}0\\4\\2\end{pmatrix}] := \{v \in \mathbb{R}^3 \mid \begin{pmatrix}-1\\1\\2\end{pmatrix} \leqq v \leqq \begin{pmatrix}0\\4\\2\end{pmatrix}\} .$$

Hierbei ist $\leqq$ komponentenweise, also als für alle Komponenten gültig zu verstehen. Es gilt also etwa

$$\begin{pmatrix}-0.5\\4\\2\end{pmatrix} \in [\begin{pmatrix}-1\\1\\2\end{pmatrix},\begin{pmatrix}0\\4\\2\end{pmatrix}] \text{ , aber } \begin{pmatrix}0\\5\\2\end{pmatrix} \notin [\begin{pmatrix}-1\\1\\2\end{pmatrix},\begin{pmatrix}0\\4\\2\end{pmatrix}].$$

Offenbar kann man Intervallvektoren also auf dem Rechner darstellen und abspeichern. Die Ausführung von Operationen ist auf dem Rechner möglich, wie in den ersten Abschnitten dieser Abhandlung erläutert.

Ebenso kann man Intervallmatrizen als die Menge aller Matrizen definieren, die zwischen zwei Grenzen liegen. Wieder ist herausragend, daß alle Operationen (auch zwischen Intervallmatrizen und -vektoren) auf dem Rechner (schnell) ausführbar sind. Die Details, mathematische Theorie und Algorithmen sind in ([12], [15]) nachzulesen.

Die Bedingung $f : X \to X$ oder

$$(22) \qquad R \cdot b + \{I - R \cdot A\} \cdot X \subseteq X$$

kann demnach auf dem Rechner nachgeprüft werden, indem alle Operationen durch die entsprechenden Bereichsoperationen ersetzt werden. Dabei ist es wichtig, die Berechnung von I-RA durch Verwendung des genauen Skalarproduktes als

eine Operation zu betrachten und entsprechend mit einer Rundung auszuführen. Unser erstes Problem des Nachprüfens der Bedingung $f: X \longrightarrow X$ auf de Rechner ist damit gelöst.

Kommen wir zu dem zweiten Problem, des Beweises der Nicht-Singularität von (diese wurde benötigt, um vom Fixpunkt der Funktion f zur Nullstelle des Gleichungssystems zu kommen). Der Beweis der Nicht-Singularität einer Matri ist mathematisch ein schwieriges und aufwendiges Problem; in der Praxis ein Gleitpunkt-Algorithmus war er bisher nicht möglich. Um so erstaunlicher ist es, daß dieser Beweis sich nahtlos in das Bisherige einfügt, und zwar ohne zusätzlichen Aufwand! Statt (22) schreiben wir:

$$(22') \qquad R \cdot b + \{I - RA\} \cdot X \subseteq \overset{\circ}{X} .$$

Hierbei bedeutet $\overset{\circ}{X}$ das "Innere" von X. Auf dem Rechner prüft man für zwei B reiche $X := [x_1,x_2]$ und $Y := [y_1,y_2]$

$$X \subseteq Y \quad <===> \quad y_1 \leqq x_1 \quad \text{und} \quad x_2 \leqq y_2 .$$

Entsprechend ist

$$X \subseteq \overset{\circ}{Y} \quad <===> \quad y_1 < x_1 \quad \text{und} \quad x_2 < y_2 .$$

Der Unterschied zwischen (22) und (22') besteht auf dem Rechner also nur da rin, daß man $\leqq$ durch $<$ ersetzt.

Schauen wir uns kurz den Beweis der Nicht-Singularität von R an. Wir beweisen dazu zunächst die Nicht-Singularität von A, also die Existenz und Einde tigkeit einer Lösung von $Ax = b$ im $\mathbb{R}^n$. Mit der Regularität von R wird dann gezeigt, daß diese Lösung in X liegen muß.

Mit (22') gilt insbesondere

$$R \cdot b + \{I - RA\} \cdot X \subseteq \overset{\circ}{X} \subseteq X .$$

Nach dem Fixpunktsatz von Brouwer besitzt $f: \mathbb{R}^n \longrightarrow \mathbb{R}^n$ mit

$$f(x) := x - R(Ax - b) = Rb + (I - RA)x$$

also einen Fixpunkt $\hat{x} \in X$. Also gilt wie wir bereits gesehen haben

$$f(\hat{x}) = \hat{x} - R(A\hat{x} - b) = \hat{x} \Longrightarrow R(A\hat{x} - b) = 0 .$$

Angenommen, $A \cdot y = 0$ für ein $y \in \mathbb{R}^n$. Für ein beliebiges $\lambda \in \mathbb{R}$ gilt dann:

$$f(\hat{x} + \lambda y) = \hat{x} + \lambda \cdot y - R(A(\hat{x}+\lambda y) - b) = \hat{x}+\lambda y - \lambda \cdot RAy = \hat{x}+\lambda y ,$$

also jedes $\hat{x} + \lambda y$ ist Fixpunkt von f. Wäre $y \neq 0$, so gäbe es ein $\lambda \in \mathbb{R}$, so daß $\hat{x} + \lambda y$ auf dem Rand von X liegt. Das wäre aber ein Widerspruch zu $f(X) \subseteq \overset{\circ}{X}$ wegen $f(\hat{x} + \lambda y) = \hat{x} + \lambda y \in f(X)$. Also ist $y = 0$ der einzige Vektor aus $\mathbb{R}^n$ mit $Ay = 0$, d.h. A ist nicht singulär.

Bleibt noch zu zeigen, daß $\hat{x}$ in X liegt, also R nicht singulär ist. Denn angenommen, es sei $y \in \mathbb{R}^n$ mit $Ry = 0$. Dann existiert $A^{-1}y$ wegen A regulär und für beliebiges $\lambda \in \mathbb{R}$ gilt

$$f(\hat{x} + \lambda A^{-1}y) = \hat{x}+\lambda A^{-1}y - R(A(\hat{x}+\lambda A^{-1}y)-b) = \hat{x}+\lambda A^{-1}y+\lambda Ry = \hat{x}+\lambda A^{-1}y ,$$

also jedes $\hat{x} + \lambda A^{-1}y$ ist Fixpunkt von f. Wäre $y \neq 0$, so auch $A^{-1}y \neq 0$ und es gäbe ein $\lambda \in \mathbb{R}$, so daß $\hat{x} + \lambda A^{-1}y$ auf dem Rand von X liegt. Das wäre aber wieder ein Widerspruch zu $f(X) \subseteq \overset{\circ}{X}$ wegen $f(\hat{x} + \lambda A^{-1}y) = \hat{x} + \lambda A^{-1}y \in f(X)$. Also folgt $y = 0$ und damit die Nicht-Singularität von R. □

Insgesamt gilt also:

Satz: Seien A,R nxn-Matrizen und $b \in \mathbb{R}^n$. Gilt dann für einen Intervallvektor X (mit n Komponenten)

$$(23) \qquad R \cdot b + \{I - R \cdot A\} \cdot X \subseteq \overset{\circ}{X},$$

dann
- gibt es eine Lösung $\hat{x}$ von $Ax = b$
- ist diese Lösung eindeutig bestimmt und
- die Lösung $\hat{x}$ liegt in X .

Mit diesem Satz ist ein einfaches Verfahren gegeben, für eine gegebene Menge X auf dem Rechner nachzuprüfen (zu beweisen), ob X die Lösung von $Ax = b$ enthält und gleichzeitig noch die Eindeutigkeit der Lösung zu beweisen.

Schließlich bleibt das Problem, ein geeignetes X zu finden das (23) erfüll Dazu nimmt man eine Näherungslösung $\tilde{x}$ von $Ax - b = 0$ und "legt einen klein Intervallvektor darum", indem in der letzten Mantissenstelle eins abgezoge und dazu addiert wird. Nun kann es passieren, daß die Bedingung (22) für d Start-X nicht erfüllt ist. In diesem (allerdings seltenen) Fall wird iteri

$$(24) \qquad \underline{\text{repeat}}\ X^{k+1} := R \cdot b + \{I - R \cdot A\} \cdot X^k\ \underline{\text{until}}\ X^{k+1} \subseteq \overset{\circ}{X}^k.$$

Bis jetzt haben wir ein mathematisch hinreichendes Kriterium angegeben: wenn Bedingung (23) erfüllt ist, dann gelten die Folgerungen des Satzes. Die Frage ist was passiert, wenn (23) nicht erfüllt werden kann oder wie dieser Fall zu bewerten ist. Anders ausgedrückt inwieweit ist Bedingung (2 notwendig für die Konvergenz des Gleitkomma-Verfahrens bzw. unter welchen zusätzlichen Voraussetzungen? Betrachten wir die Iteration (24). Ohne auf mathematische Hintergründe einzugehen sei angegeben, daß man die Iteration (24) so umschreiben kann, daß dann und nur dann, eine Einschließung berech net wird, wenn der Spektralradius von I - RA kleiner als 1 ist. Das ist ei bestmögliches Ergebnis (für Details siehe [31]).

Der entscheidende Fortschritt ist, daß nur gesicherte Information an den B nutzer weitergegeben wird. Entweder die Lösung oder (wenn das Gleichungs- system nicht lösbar ist oder die Rechengenauigkeit nicht ausreicht) eine e sprechende Meldung.

Praktische Erfahrungen haben gezeigt, daß (24) fast immer für k = 1 erfüllt ist, in sehr seltenen Fällen wird einmal k = 3. Im unten angegebenen Algori mus wurde k = 10 noch zugelassen. Das bedeutet keinen signifikanten Mehraut wand, da die Rechenzeit für einen Schritt in (24) nur n^2 beträgt. In jeden Falle sind alle Ergebnisse sicher.

Mit diesen Vorbereitungen können wir bereits einen Algorithmus angeben:

(1) Berechne eine Näherungsinverse R mit einem geeigneten Algorithmus

(2) Berechne eine Einschließung $X^0 := z := R \cdot b$ mit maximaler Genauigkeit

(3) Berechne eine Einschließung $B := I - RA$ mit maximaler Genauigkeit

(4) repeat $X^{k+1} := z + B \diamond X^k$ until ($X^{k+1} \subseteq \overset{\circ}{X}^k$ or $k > 10$)

(5) if $X^{k+1} \subseteq \overset{\circ}{X}^k$ then { Das Gleichungssystem $Ax = b$ ist lösbar;
Die Lösung $\hat{x}$ ist eindeutig bestimmt;
es gilt $\hat{x} \in X^{k+1}$ }
else { Das Gleichungssystem ist singulär oder
mit der verwendeten Rechengenauigkeit mit
diesem Verfahren nicht lösbar }

Algorithmus 1. Lineare Gleichungssysteme

In Schritt (1) ist irgendein Gleitkomma-Algorithmus verwendbar, etwa Gauß oder Gauß-Jordan. Es sei nochmals betont, daß an die Genauigkeit von R keine Voraussetzungen geknüpft werden; durch die Inklusion $X^{k+1} \subseteq \overset{\circ}{X}^k$ wird die Nicht-Singularität von R vom Rechner bewiesen. In den Schritten (2), (3) und (4) des Algorithmus 1 ist wesentlich, daß alle Verknüpfungen nach dem Prinzip des Semimorphismus ausgeführt werden. Man erhält dann maximale Genauigkeit bei der Auswertung der zu berechnenden Ausdrücke. Die betreffenden Verknüpfungen und Genaueres zur Implementierung sind in [0], [24], und [30] detailliert ausgeführt.

Für diese einfache Version eines Algorithmus für lineare Gleichungssysteme gibt es zahlreiche Verbesserungen (siehe [24], [30], [31]). Hier sei nur angeführt, daß es wesentlich besser ist nicht die Lösung $\hat{x}$ selbst, sondern die Differenz zu einer Näherungslösung $\tilde{x}$ einzuschließen. In diesem Fall schreibt sich Schritt (2) in Algorithmus 1:

(2) $\tilde{x} := R \cdot b$; $X^0 := z := -R(A\tilde{x} - b)$ intervallarithmetisch.

Die Lösung $\hat{x}$ von $Ax - b = 0$ liegt dann in $\tilde{x} + X^{k+1}$.

Wir wollen einige praktische Ergebnisse des vorgestellten Algorithmus angeben und auf numerische Schwierigkeiten bestimmter Gleitpunktverfahren hinweisen.

Zunächst wurde mit einer weitverbreiteten Anlage gerechnet, die mit 8 1/2 Dezimalstellen arbeitet. Alle Eingabedaten sind exakt, d.h. ohne Konvertierungsfehler in der Anlage darstellbar. Als erstes betrachten wir folgendes lineare Gleichungssystem (siehe [24]):

$$
\begin{array}{rrrrrrrr}
(25) & -8392848 \cdot x & & -3566221 \cdot y & & -3799934 \cdot z & = & -15759003 \\
 & 1699109 \cdot x & & +3679519 \cdot y & & +2370515 \cdot z & = & 7749143 \\
 & -6693739 \cdot x & & +113298 \cdot y & & -1429419 \cdot z & = & -8009860 \ .
\end{array}
$$

Offenbar ist die Summe von erster und zweiter Zeile gleich der dritten Zeile, das Gleichungssystem also singulär. Es wurde auf (25) der Gleitkomma-Gauß-Algorithmus mit Spaltenpivotisierung angewandt (*das* Verfahren für lineare Gleichungssysteme) und anschließend in Gleitkomma die Residueniteration (19). Die Residueniteration, natürlich doppeltlang gerechnet, stand nach einem Schritt, also $x^1 = x^0$. Dies sollte ein sicheres Zeichen sein für beste Kondition. In Wirklichkeit ist aber das Gleichungssystem singulär, hat also die schlechteste Kondition, die überhaupt möglich ist. Das heißt: Ändert man eine Komponente der Matrix oder der rechten Seite um einen beliebig kleinen Wert, ändert sich die Lösung beliebig viel oder das Gleichungssystem wird sogar unlösbar! Der oben vorgestellte neue Algorithmus gibt hier die Meldung aus, daß das System nicht lösbar ist.

Eine bekannte Technik ist, die Daten ein klein wenig abzuändern und aus der Änderung der Lösung auf die Kondition des Problems zu schließen. Wie wir an Beispiel (17) gesehen haben, ist diese Methode äußerst gefährlich. Nach den Ergebnissen (18) sieht (17) sehr gutmütig aus. Der vorgestellte neue Algorithmus gibt für Problem (17) als Lösung aus

$$x \in [-1800000,\ 2200000]$$
$$y \in [-2200000,\ 1800000]$$

Wie wir gesehen haben ist das das Bestmögliche, wenn b_1 und b_2 in den angegebenen Grenzen variieren. Die Lösung ist hier natürlich kein Punkt mehr, sondern eine Menge. Bei herkömmlichen Verfahren besteht nicht die Möglichkeit für einzelne Komponenten ganze Bereiche einzusetzen.

In dem vorgestellten neuen Algorithmus gibt es die Möglichkeit, statt "Punktdaten" auch fehlerbehaftete Koeffizienten einzugeben (siehe [30]). Der Algorithmus berechnet dann tatsächlich die Menge aller möglichen Lösungen, und zwar in scharfe Schranken eingeschlossen. Wie wir in Beispiel (17) gesehen haben, kann diese Einschließung sehr groß werden (bei schlecht konditionierter Matrix). Das heißt dann aber nichts anderes, als daß derartige extreme Unterschiede in der Lösung tatsächlich vorkommen. Denn wenn die Daten einmal mit Fehlern behaftet sind, kann die Toleranz jeder einzelnen Komponente ja in jeder Richtung angenommen werden, oder auf einmal immer zu kleine Werte annehmen oder wie auch immer. D.h., an den Ergebnissen des neuen Algorithmus kann die maximale Toleranz der Daten (Meßwertgeber etc.) für eine Mindestgenauigkeit der Lösung abgelesen werden.

Als nächstes betrachten wir die

(26) Hilbert 7x7 Matrix.

Nach der Definition lautet die ij-te Komponente

$$H_{ij} := (i+j-1)^{-1} .$$

Um die Matrix überhaupt exakt speichern zu können wurde mit dem kleinsten gemeinsamen Vielfachen aller Nenner multipliziert. Dadurch bleibt die Kondition gleich, alle Komponenten sind jedoch ganzzahlig und exakt darstellbar ohne Konvertierungsfehler. Die rechte Seite wird zu (1,1,1,1,1,1,1) vorgegeben. Es ergibt sich

Glk-Gauß	Fehler	neuer Algorithmus	Fehler
11.658203	67%	7	0
-522.74219	56%	-336	0
5583.7500	48%	3780	0
-23826.750	42%	-16800	0
47546.500	37%	34650	0
-44423.000	34%	-33264	0
15679.250	31%	12012	0

Wie man sieht liegt der *minmale* Fehler der Gleitkomma-Näherung bei 31% (wohingegen er bei 8 1/2-stelliger Rechnung bei 0.000005% liegen sollte).

Zufälligerweise sind die Lösungen ganzzahlig, so daß der neue Algorithmus sogar die exakten Lösungen ausgibt. Ist das nicht der Fall, ist das Ergebnis für 1/3 etwa

$$0.3333333^{4}_{3} .$$

Übrigens wurde Schritt (5) im neuen Algorithmus genau einmal ausgeführt. Für Spezialisten noch die Information: Die Summennorm von I - RA ist 1.7 im Beispiel (26), so daß von der Abschätzung her die Residueniteration (19) *nicht konvergieren* dürfte. Bedingung (22') stellt also ein schärferes Kriterium für die Konvergenz dar als Normabschätzungen.
Die größte Hilbert-Matrix, die in 12-stelliger Mantisse noch exakt darstellbar ist, ist die

(27) Hilbert 15x15 Matrix.

Ein Gleichungssystem mit dieser Matrix (Konditionszahl $\tilde{}10^{22}$) und rechter Seite (1,2,3,4,5,6,7,8,7,6,5,4,3,2,1) wurde auf einem 12-stelligen Rechner gelöst. Nach 2 Iterationen in Schritt (5) des neuen Algorithmus kam das Ergebnis:

Glk-Gauß	neuer Algorithmus
-0.00000370144248083	-0.0057899925470^{3}_{4}
0.000390101713089	1.144151753^{3}_{2}
-0.00993527792450	-56.54193589^{1}_{2}
0.10510498172	1227.3136752^{2}_{1}
-0.556103731082	-14632.195852^{0}_{1}
1.52452075214	107653.53043^{5}_{4}
-1.76642775931	-523134.22538^{7}_{8}
-0.959186415565	1748857.8047^{6}_{5}
4.90243939872	-4114391.8270^{1}_{2}
-4.09472893569	6869251.6291^{2}_{1}
-0.796632647673	-8093808.1800^{7}_{8}
1.88744082265	6579048.8554^{6}_{5}
1.12694074067	-3510155.1269^{5}_{6}
-2.02683453639	1106162.4229^{6}_{5}
0.663018130949	-156024.600^{400}_{399}

Für viele Komponenten stimmt nicht einmal mehr das Vorzeichen der Gleitkomma-Näherung. Programmiert man den neuen Algorithmus mit den in [12], [15] angegebenen Algorithmen für die Arithmetik auf dem erwähnten Z 80-Rechner, ist der Aufwand der Dreifache gegenüber dem Gleitkomma-Gauß-Algorithmus, und zwar unabhängig von der Anzahl der Unbekannten. Das mag im Moment viel erscheinen, jedoch

- sind alle Ergebnisse sicher,
- sind keine Kontrollrechnungen notwendig
- ist der Algorithmus auch Laien zugänglich,
- wird nur mit einfacher Genauigkeit gerechnet und
- wird sehr hohe Genauigkeit erzielt.

8 Anwendungen

Mit Algorithmus 1 sind eine Reihe weiterer Probleme gelöst, nämlich

- Matrixinversion
- Beweis der Nicht-Singularität einer Matrix
- positive (Semi-) Definitheit einer Matrix.

In jedem Fall ist das Ergebnis sicher, bewiesenermaßen richtig. Sämtliche Verfahren (wie auch die folgenden) gelten entsprechend für komplexe Daten.

Eine weitere Anwendung wären etwa Eigenwert/Eigenvektor-Probleme, Nullstellen von Polynomen, Optimierung u.ä. Wir geben hier zunächst einen allgemeinen Algorithmus an für nicht-lineare Gleichungssysteme. Zur Anwendung auf die oben genannten speziellen Teilprobleme läßt sich dieser Algorithmus natürlich ebenfalls spezialisieren und wesentlich verbessern. Wir benötigen dazu folgenden Satz:

Satz. Sei $f: \mathbb{R}^n \to \mathbb{R}^n$ eine stetig differenzierbare Funktion und R eine beliebige nxn Matrix. Gilt für einen Vektor $\tilde{x} \in \mathbb{R}^n$ und einen Intervallvektor X mit $\tilde{x} \in X$

$$\tilde{x} - R \cdot f(\tilde{x}) + \{I - R \cdot f'(X)\} \cdot (X - \tilde{x}) \subseteq \overset{\circ}{X} , \tag{28}$$

dann

- ist die Gleichung $f(x) = 0$ lösbar,
- gibt es ein $\hat{x} \in X$ mit $f(\hat{x}) = 0$,
- ist $\hat{x}$ in X eindeutig bestimmt.

In Formel (28) ist die Auswertung von $f(\tilde{x})$ und $f'(X)$ ein Problem. Prinzipiell ist hier jede Methode brauchbar, die die Werte von $f(\tilde{x})$ und $(f'_1(x_1), \ldots, f'_n(x_n))$ für alle $x_i \in X$ (sicher) einschließt. Natürlich ist man an ei möglichst scharfen Einschließung interessiert. Ersetzt man in der Auswertung von f und f' alle Operationen durch die entsprechenden Intervalloperationen, ist die Einschließung zwar gewährleistet, jedoch unter Umständen nicht scharf.

Betrachten wir als Beispiel

$$100x^4 - y^4 + 2y^2 \quad \text{für } x = 328776 \text{ und } y = 1039681 . \tag{29}$$

Auf einem 12-stelligen Rechner ergibt sich hier als

Gleitpunkt-Näherung $-2\,000\,000\,000\,000$,

und bei Ersetzen der Gleitpunkt-Operationen durch die entsprechenden Intervall-Operationen

naive Intervallrechnung $[-2\,000\,000\,000\,000, +2\,000\,000\,000\,000]$.

Diese Information ist zwar richtig, doch wenig signifikant. Daher wurde ein neuer Algorithmus entwickelt, der den Wert beliebiger arithmetischer Ausdrücke mit maximaler Genauigkeit berechnet. Dabei ist die neue Arithmetik unerläßlich (siehe [12], [15]). Mit maximaler Genauigkeit heißt hierbei: bis auf die letzte Dezimalstelle genau in Schranken eingeschlossen. In Beispiel (29) berechnet dieser neue Algorithmus den Wert der Formel zu

$$[1,1] .$$

D.h. der Wert von (29) ist exakt 1 (und nicht minus 2 Billionen). Die Rechenzeit dieses neuen Algorithmus ist dabei etwa gleich der Zeit die benötigt wird, um den Ausdruck gleitpunktmäßig auszuwerten, vorausgesetzt, die Gleitpunkt-Näherung ist nur in etwa angenähert der tatsächlichen Lösung. Ist das nicht der Fall, so

1) bemerkt dies der neue Algorithmus und

2) berechnet in einem nächsten Lauf den genauen Wert des Ausdrucks.

Im Prinzip werden die gemachten Fehler erkannt, berechnet und verbessert. Im Beispiel (29) etwa ist die Rechenzeit gerade die doppelte gegenüber der der Gleitpunkt-Rechnung mit der Näherung -2 000 000 000 000 für das exakte Ergebnis +1.
Im folgenden werden "Gleitpunkt-Ergebnisse" den Ergebnissen "der neuen Algorithmen mit sauberer Arithmetik" gegenübergestellt. Mit Gleitpunkt-Ergebnissen sind dabei immer Ergebnisse gängiger Gleitpunkt-Algorithmen gemeint (also etwa Gauß für lineare Gleichungssysteme). Die Ergebnisse der neuen Algorithmen bezeichnen das Ergebnis der jeweils für die gegebene Problemklasse neu entwickelten Algorithmen unter Verwendung der neuen Arithmetik. Dabei wird für die Berechnung von Ausdrücken auch der eben beschriebene "Formelauswerter" verwendet.

Kehren wir zu allgemeinen, nicht-linearen Gleichungssystemen zurück. Hier geschieht also die Auswertung von $f(\tilde{x})$ und $f'(X)$ in (28) mit dem Formelauswerter. R ist eine Näherungsinverse von $f'(\tilde{x})$. Es sei wieder betont, daß keinerlei Information über das nicht-lineare Gleichungssystem (Lösbarkeit etc.), über $\tilde{x}$ oder die Matrix R (Nicht-Singularität) dem Benutzer bekannt zu sein braucht. Er, der Benutzer, gibt sein Problem dem Rechner und dieser beweist alles Notwendige selbständig. D.h. die Algorithmen arbeiten völlig automatisch ohne zusätzliches know how und ohne Mehraufwand des Benutzers.

Für den Algorithmus zur Lösung nicht-linearer Gleichungssysteme gibt es viele spezielle Anwendungen. Zum Beispiel ist ein Polynom p(x) ja ein nicht-lineares Gleichungs"system" mit einer Gleichung. Formel (28) schreibt sich für Polynome p(x)

$$\tilde{x} - p(\tilde{x})/p'(\tilde{x}) + \{1-p'(X)/p(\tilde{x})\} \cdot (X - \tilde{x}) \subseteq X \ .$$

Hierbei werden die Funktionsauswertungen natürlich wieder von unserem Formelauswerter übernommen. Was passiert, wenn man dies nicht tut sondern gewöhnliche Gleitpunktrechnung verwendet veranschaulicht folgendes Beispiel:

$$(30) \quad p(x) = 543339720\,x^3 - 768398401\,x^2 - 1086679440\,x + 1536796802.$$

Gegenübergestellt werden in der nachstehenden Tabelle erstens das Horner-Schema (der übliche Algorithmus zur Polynomauswertung) mit gewöhnlicher Gleitpunkt-Arithmetik und zweitens das neue Verfahren zur Polynomauswertung (ein Spezialfall des Formelauserters). Das Beispiel wurde gerechnet auf ein 12-stelligen Rechner:

x	p(x) Gleitpunkt-Horner	p(x) neuer Algorithmus
1.41421356238	0.01	$0.00000000000007327192471_{7}^{8}$
1.41421356100	-0.01	$+0.0000000028974613436_{8}^{9}$

Das erste Beispiel bedeutet also etwa

$$7.32719247117_{10}\text{-}14 < p(1.41421356238) < 7.32719247118_{10}\text{-}14 \ .$$

Das ist das schärfste Ergebnis, welches mit 12-stelliger Rechnung erzielt werden kann.

Im zweiten Beispiel stimmt nicht einmal mehr das Vorzeichen der Gleitpunkt-Näherung. Bei der Nullstellensuche werden gerade die Punkte mit kleinem Funktionswert gesucht.
Was bei der Nullstellensuche bei Polynomen passieren kann, erhellt folgendes Beispiel:

$$(31) \quad p(x) = 67872320568\,x^3 - 95985956257\,x^2 - 135744641136\,x + 1919719125$$

Auf dieses Polynom wurde auf einem 12-stelligen Rechner das Newton-Verfahren angewandt mit dem Startwert $x^0 := 2$. Alle Koeffizienten sind in 12-stelliger Mantisse exakt darstellbar. Die Polynomwerte wurden nach dem Horner-Schema mittels Gleitpunktrechnung ermittelt:

1.73024785661	2.698_{10}-01
1.57979152125	1.505_{10}-01
1.49923019011	8.056_{10}-02
1.45733317058	4.190_{10}-02
1.43593403289	2.140_{10}-02
1.42511502231	1.082_{10}-02
1.41967473598	5.440_{10}-03
1.41694677731	2.728_{10}-03
1.41558082832	1.366_{10}-03
1.41489735833	6.835_{10}-04
1.41455549913	3.419_{10}-04
1.41438453509	1.710_{10}-04
1.41429903606	8.550_{10}-05
1.41425628589	4.275_{10}-05
1.41423488841	2.140_{10}-05
1.41422414110	1.075_{10}-05
1.41421847839	5.663_{10}-06
1.41421582935	2.649_{10}-06
1.41421353154	2.298_{10}-06
1.41421353154	0
1.41421353154	0
1.41421353154	0
1.41421353154	0
1.41421353154	0
...	...

In obenstehender Tabelle sind links die Iterierten x^k und rechts die Differenz zweier aufeinanderfolgender Iterierter $x^k - x^{k+1}$ angezeigt. Diese Differenz ist immer positiv (oder Null), d.h. die Iteration ist monoton fallend. Außerdem fällt die Differenz selbst monoton, d.h. zwei aufeinanderfolgende Iterierte haben immer kleineren Abstand ("konvergieren"). Schließlich wird die Differenz Null, d.h. die Iteration hat einen Fixpunkt. Mathematisch würde daraus folgen:

$$x^k = x^{k+1} = x^k - \frac{p(x^k)}{p'(x^k)} \quad ==> \quad \frac{p(x^k)}{p'(x^k)} = 0 \quad ==> p(x^k) = 0 \ .$$

D.h. das Polynom p aus (31) hätte eine Nullstelle:

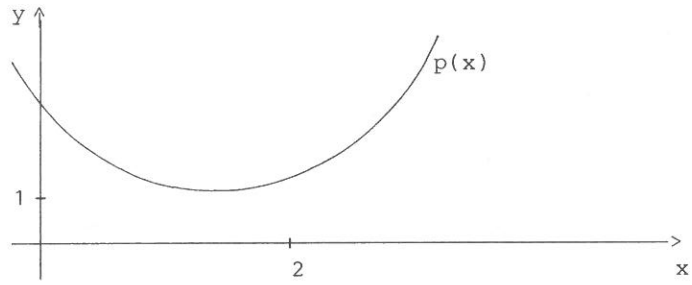
$$p(1.41421353154) = 0 \ .$$

Wir schreiben im Konjunktiv, denn "$x^k = x^{k+1}$" gilt ja nur nach einer Gleitpunktrechnung, eine mathematische Schlußfolgerung wie oben ist also nicht zulässig.

Bei der obigen Gleitpunkt-Iteration deutet alles auf "Konvergenz" und daher auf eine Nullstelle bei x = 1.41421353154 hin. Das wäre nicht passiert, hätte man p(x) und p'(x) nicht mit gewöhnlicher Gleitpunktrechnung sondern mit dem neuen Algorithmus zur Auswertung von p(x) mit maximaler Genauigkeit berechnet. Wendet man gleich den neuen Algorithmus zur Einschließung von Nullstellen von Polynomen an, so erhält man die Antwort:

"Bei x = 2 kann keine Nullstelle gefunden werden".

Tatsächlich sieht der Graph von p(x) wie folgt aus:



Es gibt also gar keine Nullstelle, der Gleitpunkt-Algorithmus hat nur eine Nullstelle vorgetäuscht. Mit dem neuen Algorithmus und der sauberen Arithmetik berechnet man übrigens

$$1.00018250381 < p(1.41421353154) < 1.00018250382 \ .$$

Nach der Gleitpunkt-Rechnung sollte der Wert an dieser Stelle gleich Null sein. Wie man aus dem Graph erkennt, ist das Minimum für $x \geq 0$ etwa 1, von Nullstelle kann also gar keine Rede sein.

Der Mathematiker sieht den Zahlen der Iteration natürlich an, daß hier ein schlecht konditioniertes Problem vorliegt. Die neuen Algorithmen sollen aber helfen zu vermeiden, daß solche Zahlenkolonnen überhaupt angeschaut werden müssen und daß ein Benutzer durch die gleitpunktmäßig erzielte Aussage "p(1.41421353154) ist Fixpunkt" nicht in die Irre geleitet wird.

Kommen wir zur Berechnung der Eigenwerte und Eigenvektoren einer (beliebigen) Matrix. Das Eigenwert/Eigenvektor-Problem kann als nicht-lineares Gleichungssystem geschrieben werden:

$$Ax - \lambda x = 0$$
$$e_k'x - \zeta = 0$$

für eine Nomierungskonstante ζ .
Der obige Satz für nicht-lineare Systeme ist wieder anwendbar. Besonders hier lassen sich gegenüber dem allgemeinen Satz wieder erhebliche Verbesserungen angeben (siehe [30]). Insbesondere ist es möglich, die Eindeutigkeit eines Eigenwerts in einem Bereich zu zeigen und/oder zu zeigen, daß in einem Bereich mit Sicherheit kein Eigenwert liegt. Das ist besonders für Eigenschwingungen hochinteressant. Ein schlecht konditioniertes Beispiel ist etwa

(32) die Hilbert 7x7 Matrix.

Es wurde ein Standard-Gleitpunkt-Algorithmus (aus einer Bibliothek) zur Berechnung von Näherungen für Eigenwerte und Eigenvektoren angewandt. Für einen Eigenwert ergab sich (bei 8 1/2 stelliger Rechnung):

Gleitpunkt-Näherung	neuer Algorithmus
$-2.1705692_{10}-3$	$+1.2590613\,{}^{1}_{0}\cdot 10^{-3}$

In der rechten Spalte ist das Ergebnis des neuen Algorithmus (ebenfalls mit 8 1/2 stelliger Rechnung) mit sauberer Arithmetik angegeben. Das Ergebnis ist zu lesen als $1.25906130 < 1000\,\lambda < 1.25906131$. Von der gleitpunktmäßig errechneten Näherung ist nicht einmal mehr das Vorzeichen richtig. Der neue Algorithmus unter Verwendung der neuen Arithmetik gibt 8 sichere Stellen aus.

Weiter ergibt sich eine direkte Anwendung auf Optimierungsprobleme. Und zwar kann für

lineare Optimierung,
quadratische Optimierung und
konvexe Optimierung

von einer Näherungslösung *bewiesen* werden, ob sie wirklich optimal ist oder nicht.

Betrachten wir folgendes Beispiel:

Eine Erdölraffinerie steht vor dem Problem, wieviel Erdöl sie im Durchschnitt jährlich verkaufen soll und wieviel lagern, also etwa x Mill. Barrel verkaufen, y Mill. Barrel lagern. Da Lagern Verluste bringt, soll mit geeigneten Konstanten α,β gelten

$$\alpha x - \beta y \geq 0.$$

(33) Andererseits entsteht durch Lieferung und Verträge eine Lieferungsverpflichtung, eine Reserve soll bleiben, d.h. es darf nicht zu viel verkauft werden:

$$\gamma x - \delta y \leq \varepsilon.$$

Schließlich soll der Gewinn maximiert werden, also

$$\zeta x - \eta y = \text{Max!}$$

Das Beispiel wurde mit Konstanten $(\alpha,\beta,\gamma,\delta,\varepsilon,\zeta,\eta) = (470831.9, 665857, 665857, 941664.2, 1.02, 19975710000, 28249914000)$ auf einem 12-stelligen (!) Rechner bearbeitet. Dieser ermittelte den

(Gleitkomma) maximalen Gewinn zu 3.06 Milliarden DM.

In Wirklichkeit ist

(neuer Algorithmus) maximaler Gewinn 5.65 Milliarden DM.

Der neue Algorithmus mit der neuen Arithmetik beweist zum einen, daß das Ergebnis mit Sicherheit dem maximalen Gewinn entspricht und berechnet zum zweiten diesen maximalen Gewinn auf 12 sichere Stellen genau.

Die Crux gerade bei Optimierungsproblemen ist, daß ein Ergebnis zwar meist den Nebenbedingungen genügt, es allerdings keineswegs auch die optimale Lösung sein muß. Der neue Algorithmus beweist die Optimalität, alle Ergebnisse sind sicher.

Kommen wir noch einmal zur Berechnung von arithmetischen Ausdrücken zurück. Dieses grundsätzliche Problem kann nicht genug betont werden.

Denn wenn nicht einmal die Auswertung einer einfachen Formel ein richtiges Ergebnis liefert, was kann dann ein Algorithmus noch bringen, in dem ja solche Auswertungen laufend vorkommen. Betrachten wir noch ein Beispiel:

(34) $x^4 - 2x^2 - 1.96y^4$ für $x = 10081$ und $y = 8520$

ergab auf einem 16-stelligen Rechner (!) mit

Gleitkomma-Rechnung 0

statt dessen erhält man mit der neuen Arithmetik auf einem 12-stelligen Rechner mit dem

neuen Algorithmus das korrekte Ergebnis -1.

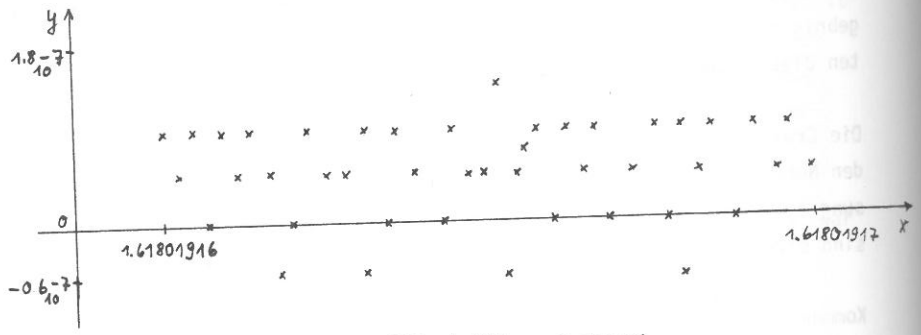
Dieses Beispiel spricht für sich.

Tatsächlich lassen sich solche Beispiele in Fülle angeben. Betrachten wir etwa das Polynom

(35) $-22300658x^3 + 1083557822x^2 - 1753426039x + 945804881$

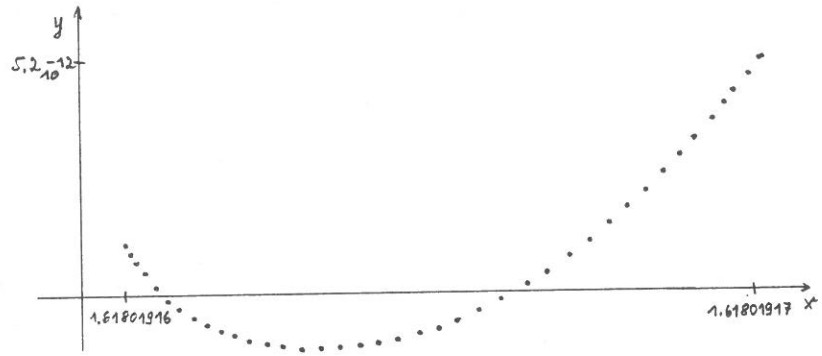
in dem Bereich B := 1.61801916...1.61801917.

Zunächst wurden die Funktionswerte von (35) im Bereich B mit dem Horner-Schema und gewöhnlicher Gleitpunkt-Arithmetik auf einer 16-stelligen, weitverbreiteten Großrechenanlage berechnet. Es ergab sich folgendes Bild:



Gleitpunkt-Hornerschema (16-stelliger Rechner)

Dann wurden die Funktionswerte von (35) im Bereich B auf einem 12-stelligen Rechner mit dem neuen Algorithmus und der neuen Arithmetik berechnet. Jetzt ergab sich folgendes Bild:



neuer Algorithmus (12 stelliger Rechner)

Zum einen waren sämtliche Funktionswerte im zweiten Bild mit Sicherheit auf 12 Dezimalstellen richtig. Zweitens waren von allen Werten im ersten Bild in keinem Fall nicht einmal eine Stelle richtig (gerechnet auf einem 16-stelligem Rechner!). Und setzte man drittens den maximalen Funktionswert im Bereich B im zweiten Bild (also den wahren maximalen Funktionswert) zu

einem Centimeter

fest, so entspricht der maximale Funktionswert im Bereich B im ersten Bild (16-stellige Gleitpunktrechnung) der Höhe des

Empire State Building .

Neue Algorithmen wie die bereits besprochenen wurden auch für große, schwachbesetzte Gleichungssysteme entwickelt. Wie in allen neuen Algorithmen sind auch hier die Ergebnisse in jedem Fall sicher.

Schließlich ergibt sich bei Anwendung des obigen Satzes auch ein Algorithmus für nicht-lineare Gleichungssysteme. Die vielen Beispiele oben sind ja alles Spezialfälle von nicht-linearen Gleichungssystemen, so daß leicht ausmalbar ist, was bei voller Allgemeinheit alles passieren kann und passiert. Zum Beispiel kann eine Formelauswertung direkt als nicht-lineares Gleichungssystem geschrieben werden, wie etwa Beispiel (29):

$$\begin{aligned} (36) \qquad & x - 328776 = 0 \\ & y - 1039681 = 0 \\ & 100x^4 - y^4 + 2y^2 - z = 0 \end{aligned}$$

Ein solches Gleichungssystem *kann* selbst auf einem 20-stelligen Rechner *nicht* gelöst werden, da auf einem solchen Rechner selbst wenn man die richtige Lösung (328776, 1039681,1) einsetzt, nicht 0 herauskommt. Wie wir gesehen haben, ist das Ergebnis von (36) auf einem 12-stelligen Rechner für

$$(37) \qquad (x,y,z) = (328776,\ 1039681,\ -2000000000000)$$

genau (0,0,0). (37) müßte also eine Lösung sein! Der neue Algorithmus findet selbst auf einem 8 1/2-stelligen Rechner die Lösung von (36) und schließt sie optimal ein. In diesem Fall sogar exakt, da die Lösung ganzzahlig in 8 1/2 Stellen exakt darstellbar ist.

Der neue Algorithmus für nicht-lineare Gleichungssysteme wurde auf verschiedenen Rechnern implementiert und getestet. Die Rechenzeit beträgt $2n^3$ plus n^2 Funktionsauswertungen bei vollbesetzter (!) Jacobimatrix. Die Elemente der Jacobimatrix werden automatisch numerisch (nicht symbolisch) berechnet ohne Hinzutun des Benutzers. Es wurden nicht-lineare Systeme bis zur obersten Hauptspeichergrenze (ohne Auslagerung) gelöst ($\tilde{=}$ 400 x 400). In jedem Beispiel unterschieden sich linke und rechte Grenze der Ergebnisbereiche nur um wenige Bit.

9 Schlußbetrachtung

Es wurden neuartige Algorithmen vorgestellt für verschiedenste Aufgabenbereiche in der Numerik. Die Algorithmen geben nur bewiesenermaßen richtige Ergebnisse aus. Die Ergebnisse sind von hoher Genauigkeit, d.h. auf einem 12-stelligen Rechner sind die Ergebnisse i.a. auf wenigstens 11 Stellen garantiert. Der wesentliche Fortschritt liegt neben der hohen Genauigkeit

der Ergebnisse in der Tatsache, daß jedes Ergebnis *mit Sicherheit* richtig ist.

Die Rechenzeit für alle vorgestellten neuen Algorithmen ist von der Größenordnung wie die gängiger Gleitpunktalgorithmen. Die neuen Algorithmen sind sogar oft schneller als herkömmliche Gleitpunktalgorithmen, da sie mit geringerer Stellenzahl bereits arbeiten. In die Reihe der Beispiele sei eingefügt:

$$\begin{aligned} (38) \qquad & 37639840x - 46099201y = 0 \\ & 29180479x - 35738642y = -1 \end{aligned}$$

Der eingebaute Standardalgorithmus einer weltweit verbreiteten Großrechenanlage mit Mantissenlänge 16 Dezimalstellen liefert als Gleitpunkt-Näherung

$$\begin{aligned} x &= 28869851.52297299 \\ y &= 23572135.06039856\ . \end{aligned}$$

Unser neuer Algorithmus mit der neuen Arithmetik findet auf einem 12-stelligen (!) Rechner die exakte Lösung von (38):

$$\begin{aligned} x &= 46099201.0 \\ y &= 37639840.0 \end{aligned}$$

Die mit dem 16-stelligen Rechner erzielten Näherungen sind in keiner einzigen Dezimale richtig. Abgesehen davon ist ein Vergleich der Algorithmen, Ergebnisse oder Rechenzeiten gar nicht zulässig, da Gleitkomma-Algorithmen nur Näherungen liefern, die neuen Algorithmen dagegen nur bewiesenermaßen richtige Ergebnisse.

Man betrachte bei allen Beispielen die Größenordnungen: Bereits so wenige Operationen können einen derartigen Unsinn produzieren.
Es können auch mit Fehlern behaftete Daten behandelt werden. Die Lösung ist in diesem Fall gleich der Menge aller Lösungen, die bei irgendeiner Kombination der Eingabedaten entsteht, es werden also sämtliche Kombinationen von Abweichungen der Daten betrachtet. Diese Lösung wird von den neuen Algorithmen scharf und mit Sicherheit eingeschlossen. Durch die Möglichkeit, ungenaue Daten eingeben zu können ist es auch möglich, maximale Toleranzen von Meßgebern, Materialkonstanten etc. zu ermitteln, also die empfindlichen

Stellen eines Problems aufzudecken. Schließlich können auch Sicherheitsgebiete festgestellt werden etwa durch den Beweis der Nicht-Existenz einer Eigenschwingung in einem Bereich.
Die Firma IBM hat als erste die neue Arithmetik in ihre Rechenanlagen übernommen. Eine große Anzahl von Algorithmen wird unter dem Namen ACRITH als Programmprodukt vertrieben [35], [36]. Nach einem ersten und zweiten Release in den Jahren 1983 und 1984 ist im Oktober 1985 ein weiterer, umfangreicher, dritter Release angekündigt worden.

Es folgt eine Demonstration einiger Algorithmen wie sie auf dem erwähnten Z80-Rechner implementiert sind. Eine ausführliche Beschreibung der verwendeten Programmiersprache PASCAL-SC und die Implementierung auf IBM-PC wird in einem demnächst erscheinenden Buch [37] erfolgen. Dort sind zwei Disketten für den IBM-PC beigelegt mit dem PASCAL-SC Compiler, einem Syntax prüfenden Editor und den Programmen für die neuen Algorithmen.

```
+++++++++  Dies ist eine kurze Demonstration des Systems PASCAL-SC.
+++++++++
+++++++++  Der Rechner verfuegt ueber eine genaue Gleitpunkt-Arithmetik und
+++++++++  ein genaues Skalarprodukt. Dies gestattet es, nicht nur Naeherungs-
+++++++++  loesungen eines Problems zu berechnen, sondern es koennen sogar
+++++++++  Bereiche angegeben werden, in denen sich genau eine Loesung des
+++++++++  gestellten Problems befindet. Und dies fuer lineare Gleichungs-
+++++++++  systeme ebenso wie fuer Eigenwerte/Eigenvektoren, Nullstellen von
+++++++++  Polynomen, Loesung von Differentialgleichungen etc. Dabei wird die
+++++++++  Existenz und Eindeutigkeit der Loesung in den ausgegebenen
+++++++++  Schranken vom Rechner vollautomatisch bewiesen ohne jedes Hinzutun
+++++++++  seitens des Benutzers.
+++++++++
+++++++++  Die Spracherweiterung PASCAL-SC (Pascal for Scientific Computation)
+++++++++  gestattet es, die Programme in einfacher und uebersichtlicher Form
+++++++++  zu entwerfen und zu schreiben.
+++++++++
+++++++++  Nachfolgend bezeichnen die Zeilen, die mit ++++++++++ beginnen,
+++++++++  einen Kommentar, die Zeilen die mit * beginnen, eine Eingabe vom
+++++++++  vom Terminal aus.
+++++++++
+++++++++  Intervalle werden jeweils bis zur ersten differierenden Stelle
+++++++++  gedruckt. Somit kann die Guete einer Einschliessung bereits optisch
+++++++++  an der Laenge (Anzahl der Stellen) der linken und rechten Grenze
+++++++++  erkannt werden.
```



```
++++++++++  Intervall-Newton-Verfahren in PASCAL-SC

BEGIN (* HAUPTPROGRAMM *)
  WRITELN ('ANFANGSINTERVALL EINGEBEN. ETWA [1,1.5]');
  IREAD( INPUT,X1 );
  REPEAT
    XO := X1;
    XM := MITTE XO;
    X1 := ( XM - FKT(XM)/ABL(XO) ) ** XO;
    IWRITE( OUTPUT,X1 );  WRITELN
  UNTIL XO = X1
END.


++++++++++  Intervall-Newton-Verfahren in gewoehnlichem PASCAL

(* H A U P T P R O G R A M M *)
  BEGIN WRITELN('  ANFANGSINTERVALL EINGEBEN  ');
    READI(INPUT,X1);
    REPEAT XO:=X1;
      MITTE(XO,XM);
      FKT(XM,YM);
      ABL(XO,YO);
      DIVI(YM,YO,Y);
      SUBI(XM,Y,Z);
      DS(Z,XO,X1);
      WRITEI(OUTPUT,X1);
      WRITELN
    UNTIL EQ(XO,X1)
  END.


++++++++++  Ausfuehrung des Intervall-Newton-Verfahrens fuer
++++++++++  F(X) = X * (X ** 9 - 1) - 1

ANFANGSINTERVALL EINGEBEN. ETWA [1,1.5]

* [1,1.6]

[             1.0E+00,             1.3E+00]
[             1.0E+00,             1.2E+00]
[            1.07E+00,            1.09E+00]
[          1.0755E+00,          1.0761E+00]
[       1.0757659E+00,       1.0757662E+00]
[  1.07576606608E+00,  1.07576606609E+00]


++++++++++  Bei der Programmierung des Intervall-Newton-Verfahrens in gewoehn
++++++++++  lichem Pascal muessen die verwendeten Unterprogramme READI, MITTE
++++++++++  DIVI, SUBI, DS und WRITEI vom Benutzer programmiert werden. Das
++++++++++  sind etwa 180 lines of code. Bei der Programmierung in PASCAL-SC
++++++++++  sind alle Prozeduren vorprogrammiert und zudem in der leicht
++++++++++  lesbaren Operator-Schreibweise aufrufbar.
```

```
++++++++++  Ausfuehrung eines kombinierten Gleitpunkt-Intervall-Newton-Verfahre
++++++++++  fuer die gleiche Funktion wie oben  F(X) = X * (X ** 9 - 1) - 1


ANFANGSWERT EINGEBEN. ETWA 1.0

* 1.3

 1.30000000000E+00
 1.19065781455E+00
 1.11558133052E+00
 1.08181447152E+00
 1.07592376424E+00
 1.07576617573E+00
[  1.07576606608E+00,  1.07576606609E+00]



++++++++++  Hier wird zunaechst gleitpunktmaessig iteriert und im letzten
++++++++++  Schritt erst eine Intervall-Iteration angesetzt.
++++++++++
++++++++++  In jedem Fall wird vom Rechner automatisch bewiesen, dass im
++++++++++  Ergebnisbereich genau eine Nullstelle der Ausgangsfunktion liegt.



 ++++++++++  Aufruf des Algorithmus zur genauen Berechnung von Skalarprodukten.
 ++++++++++
 ++++++++++  Der Algorithmus berechnet das Ergebnis nicht nur genau sondern au
 ++++++++++  schneller, als der in gewoehnlichem PASCAL programmierte Algorith-
 ++++++++++  mus. Und zwar fuer jedes Skalarprodukt, auch bei noch so grosser
 ++++++++++  Ausloeschung.



 Bitte Dimension eingeben :

 * 3

 Bitte X[i], Y[i] eingeben  (i=1..DIM) :

 * 1000000.00  1000000.00
 * 1.00  1.00
 * 1000000.00  -1000000.00


 Ergebnis des genauen Skalarprodukts :

 SUMME =  1.0000000000000E+00


 Ergebnis des Skalarprodukts programmiert in gewoehnlichem PASCAL :

 SUMME =  0.0000000000000E+00
```



```
++++++++++  Algorithmus zur Berechnung der Inversen einer Matrix.
++++++++++
++++++++++  Das Programm invertiert eine reelle Matrix A. Das Ergebnis is
++++++++++  eine Intervall-Matrix, die die Inverse von A bewiesenermassen
++++++++++  enthaelt wobei die Nicht-Singularitaet der Matrix gleichzeitig
++++++++++  durch den Rechner bewiesen wird.


Das Programm LINV berechnet Schranken fuer die Inverse einer Matrix.
Geben Sie einen der folgenden Buchstaben ein:

S  wenn Sie selbst die Matrix eingeben wollen.
Z  wenn die Matrix zufaellig erzeugt werden soll.
H  wenn eine Hilbertmatrix erzeugt werden soll.
Q  wenn Sie aufhoeren wollen.
L  schaltet die Ausgabe der Zwischenergebnisse ein.
N  schaltet die Ausgabe der Zwischenergebnisse ab.
R  gibt nur die gleitpunktmaessig berechnete Naeherung als
   Zwischenergebnis aus.
I  gibt diese Information aus.

* S

Welche Dimension?

* 2

Geben Sie die Matrix zeilenweise ein. Die Dimension ist 2.

* 1 1
* 9 9

1. Gleitpunktmaessig berechnete Inverse:

-9.999999999999E+11 1.111111111111E+11
 1.000000000000E+12-1.111111111111E+11


2. Berechnung der Inversen durch neues Verfahren:

Gemessen an der Kondition der Matrix ist die Rechengenauigkeit
nicht ausreichend, d.h. bei dem Problem ist grosse Vorsicht geboten.


Geben Sie H,S,Z oder Q ein. Die Beschreibung erhalten Sie durch I.

* S

Welche Dimension?

* 2

Geben Sie die Matrix zeilenweise ein. Die Dimension ist 2.

* 941664 665857
* 665857 470832
```

```
++++++++++  Algorithmus fuer lineare Gleichungssysteme.
++++++++++
++++++++++  Das Programm berechnet eine bewiesene Einschliessung der Loesung
++++++++++  eines linearen Gleichungssystems.



Das Programm berechnet Schranken fuer die Loesung eines
linearen Gleichungssystems.
Geben Sie bitte einen der folgenden Buchstaben ein:

S  wenn Sie selbst die Matrix und rechte Seite eingeben wollen.
Z  wenn Matrix und rechte Seite zufaellig erzeugt werden sollen.
H  wenn eine Hilbertmatrix erzeugt werden soll. Die rechte Seite koennen.
   Sie dann selbst eingeben.
Q  wenn Sie aufhoeren wollen.
L  schaltet die Ausgabe von Zwischenergebnissen ein.
N  schaltet die Ausgabe von Zwischenergebnissen ab.
R  gibt nur die gleitpunktmaessig berechnete Naeherung als
   Zwischenergebnis aus.
I  gibt diese Information aus.

* S

Geben Sie bitte die Anzahl der Gleichungen und die Anzahl der Unbekannten ein

* 3 2

Geben Sie die Matrix zeilenweise ein und dann die rechte Seite.
Einzugeben sind 3 Gleichungen in 2 Unbekannten.

* 665857 -941664
* 470832 -665857
* 470833 -665857
* 1
* 0
* 665858


1. Gleitpunktmaessig berechnete Naeherung :

     1       6.65858261006E+05
     2       4.70832085775E+05
2. Durch den neuen Algorithmus wurde bewiesen, dass die Matrix maximalen Rang
hat und dass die beste Approximation garantiert in folgenden Bereich liegt :

     1    [  6.65858000000E+05,  6.65858000001E+05]
     2    [  4.70832707107E+05,  4.70832707108E+05]


Geben Sie H,S,Z oder Q ein. Die Beschreibung erhalten Sie durch I.

* Q


++++++++++  Offensichtlich muss eine gleitpunktmaessig berechnete Naeherung
++++++++++  nicht notwendig viel mit der wahren Loesung zu tun haben, auch
++++++++++  wenn ein so bewaehrter Algorithmus wie der Gauss'sche verwendet
++++++++++  wird mit optimaler Arithmetik.
++++++++++
++++++++++  Der vorliegende Algorithmus berechnet genaue Schranken fuer die
++++++++++  Loesung linearer Gleichungssysteme, und zwar auch fuer ueber-
++++++++++  oder unterbestimmte Systeme (mehr Gleichungen bzw. mehr Unbe-
++++++++++  kannte). Im Fall ueberbestimmter Systeme wird die Loesung mit
++++++++++  kleinstem Residuum, im Fall unterbestimmter Systeme die Loesung
++++++++++  kleinster Laenge bestimmt.
++++++++++
++++++++++  In jedem Fall wird durch den Rechner bewiesen, dass die Matrix
++++++++++  des gegebenen Gleichungssystems maximalen Rang hat und zwar
++++++++++  automatisch ohne ein Hinzutun des Benutzers.
++++++++++
```



```
1. Gleitpunktmaessig berechnete Inverse:

-1.66666666667E+05 2.35702260396E+05
 2.35702260396E+05-3.33333333333E+05


2. Berechnung der Inversen durch neues Verfahren:

Durch den Algorithmus wurde bewiesen, dass die Matrix nicht singulaer ist
und dass die  Inverse garantiert in folgendem Bereich liegt:

 1-te Spalte:
[ -4.70832000000E+05, -4.70832000000E+05]
[  6.65857000000E+05,  6.65857000000E+05]

 2-te Spalte:
[  6.65857000000E+05,  6.65857000000E+05]
[ -9.41664000000E+05, -9.41664000000E+05]


Geben Sie H,S,Z oder Q ein. Die Beschreibung erhalten Sie durch I.

* Q


++++++++++  Die gleitpunktmaessige Naeherung der Inversen ist voellig falsch
++++++++++  bzw. es wird die Inverse einer singulaeren Matrix berechnet. Die
++++++++++  berechnete Einschliessung ist bis auf die letzte Dezimale genau.
```

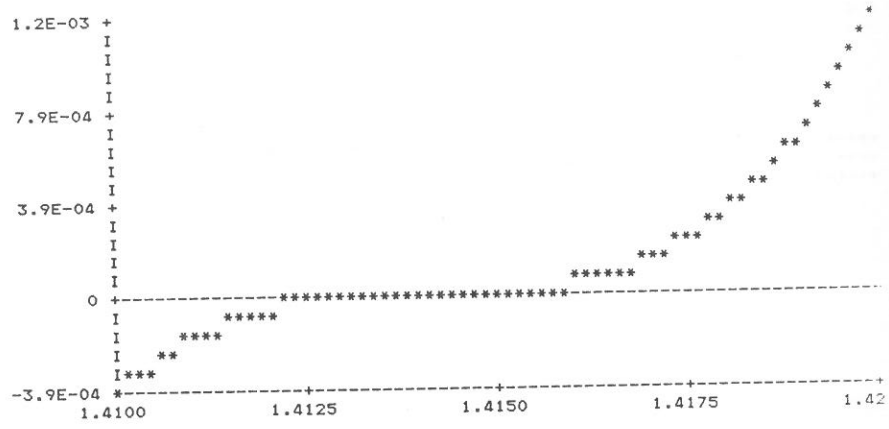
++++++++++ Zeichnung des Polynoms 2030 x**4 - 5741 x**3 - x**2 + 11482 x
++++++++++ - 8118 .
++++++++++ Die Auswertung der Polynomwerte erfolgt mit dem Gleitpunkt-Horner-
++++++++++ Schema.
++++++++++
++++++++++ Trotz der best-moeglichen Gleitpunktarithmetik werden die Funk-
++++++++++ tionswerte durch Ausloeschungen, Rundungsfehler etc. voellig ver-
++++++++++ faelscht.

Geben Sie den Grad des Polynoms und dann die Koeffizienten in
aufsteigender Reihenfolge ein.

* 4
* -8118 11482 -1 -5741 2030

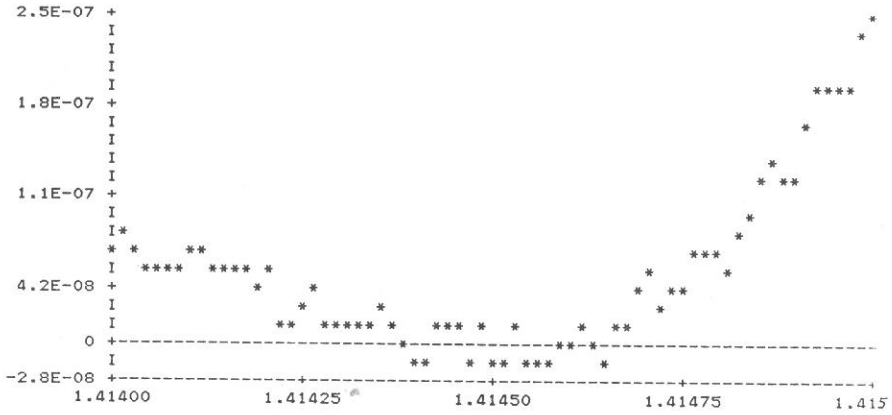
In welchem Bereich soll das Polynom gezeichnet werden?

* 1.41 1.42



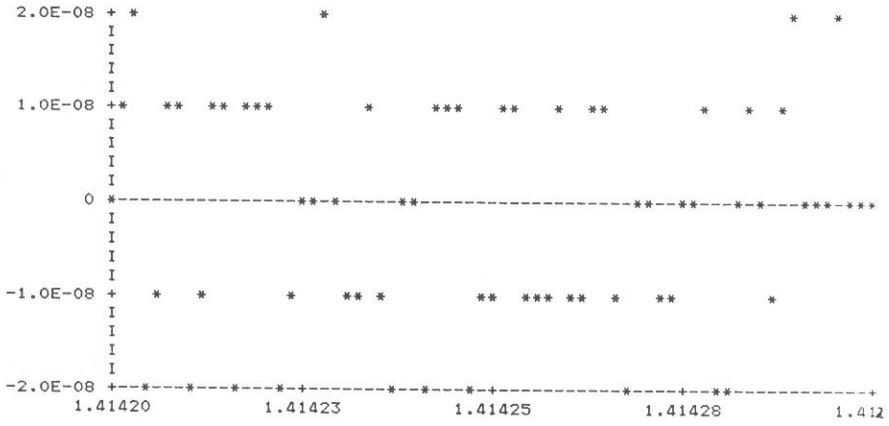
Geben Sie entweder einen neuen Bereich ein oder P oder Q.

* 1.414 1.415



Geben Sie entweder einen neuen Bereich ein oder P oder Q.

* 1.4142 1.4143



Geben Sie entweder einen neuen Bereich ein oder P oder Q.

* Q

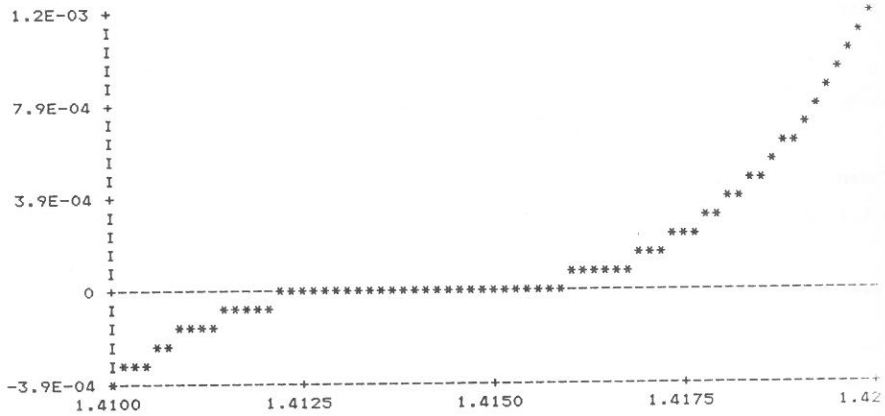
++++++++++ Zeichnung desselben Polynoms wie oben in denselben Bereichen.
++++++++++
++++++++++ Die Berechnung der Polynomwerte erfolgt jetzt mit einem neuen Algo
++++++++++ rithmus unter Verwendung der genauen Arithmetik.
++++++++++ Alle Funktionswerte werden bis auf 12 Dezimalen genau berechnet.

Geben Sie den Grad des Polynoms und dann die Koeffizienten
in aufsteigender Reihenfolge ein.

* 4
* -8118 11482 -1 -5741 2030

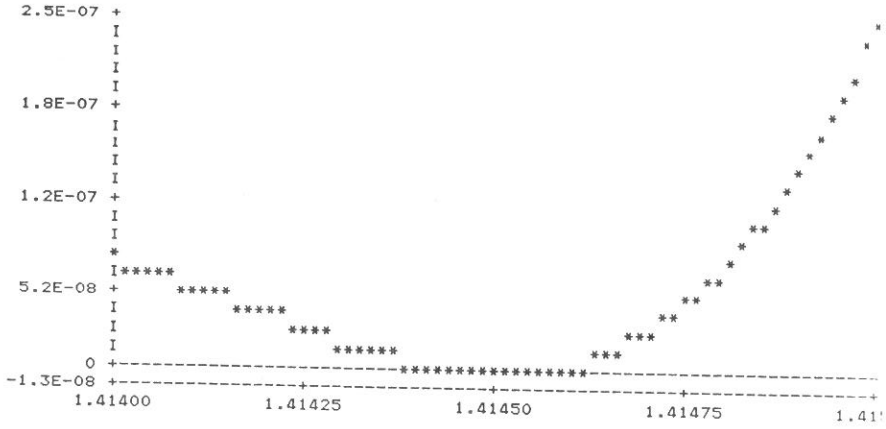
In welchem Bereich soll das Polynom dargestellt werden?

* 1.41 1.42



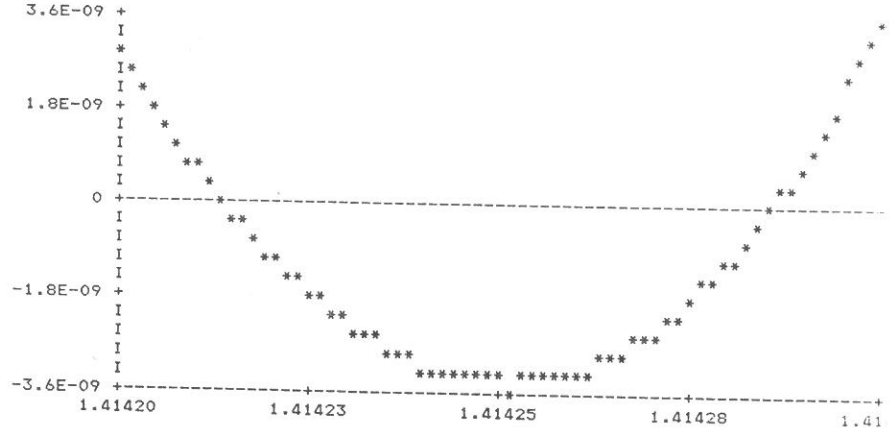
Geben Sie entweder einen neuen Bereich ein oder P oder Q.

* 1.414 1.415



Geben Sie entweder einen neuen Bereich ein oder P oder Q.

* 1.4142 1.4143



Geben Sie entweder einen neuen Bereich ein oder P oder Q.

*Q

++++++++++ Algorithmus zur Berechnung des Werts einer beliebigen aus +,-,*,
++++++++++ /,(,) gebildeten Formel fuer eine Variablenbelegung. Das Programm
++++++++++ stellt in gewisser Weise die Fortfuehrung des Algorithmus zur
++++++++++ genauen Berechnung von Skalarprodukten dar.

Geben Sie bitte eine Formel ein.

* X**4 + 2Y**2 - 4Y**4

Geben Sie bitte einen der folgenden Buchstaben ein :

F fuer eine neue Formel.
E fuer eine Einschliessung des Werts der Formel.
L um die Zwischenergebnisse zu sehen.
N um keine Zwischenergebnisse zu sehen.
D um die momentan bearbeitete Formel zu sehen.
Q um aufzuhoeren.
I fuer diese Information.

* E

Geben Sie die Werte der Variablen ein.

Wert fuer X :
* 665857

Wert fuer Y :
* 470832

Reelle Naeherung : -3.00000000000E+12
Naive Intervallrechnung : [-3.0E+12, 5.0E+12]

Neues, genaues Verfahren : [1.00000000000E+00, 1.00000000000E+00]

Geben Sie E,F oder Q ein. Die Information erhalten Sie durch I.

* Q

++++++++++ Offensichtlich kann bereits bei einfachsten Formeln das Ergebnis
++++++++++ durch Ausloeschung beliebig verfaelscht werden. Der neue Algorith
++++++++++ liefert alle Funktionswerte bis auf die letzte Dezimale genau.

++++++++++ Algorithmus zur bewiesenen Loesung eines nicht-linearen Gleichungs
++++++++++ systems.
++++++++++
++++++++++ Die einzelnen Gleichungen duerfen beliebig aus +,-,*,/,(,) zusamme
++++++++++ gesetzt sein. Der Algorithmus gibt einen Loesungsbereich aus, in d
++++++++++ bewiesenermassen genau eine Loesung des nicht-linearen Gleichungs-
++++++++++ systems ist.

Dies ist ein Programm zur bewiesenen Einschliessung der Loesung
eines nicht-linearen Gleichungssystems.

Geben Sie bitte die Dimension des Systems ein.

* 4

Geben Sie bitte die 1-te Gleichung des nicht-linearen Systems ein.

* A+B+D-1

Geben Sie bitte die 2-te Gleichung des nicht-linearen Systems ein.

* B-0.6D/C

Geben Sie bitte die 3-te Gleichung des nicht-linearen Systems ein.

* C+D-1

Geben Sie bitte die 4-te Gleichung des nicht-linearen Systems ein.

* D-0.3AC

Die Dimension des nicht-linearen Gleichungssystems ist 4.
Geben Sie bitte Anfangswerte der Variablen ein.

Wert fuer A :
* 1
Wert fuer B :
* 1
Wert fuer C :
* 1
Wert fuer D :
* 1

Die Dimension des nicht-linearen Gleichungssystems ist 4.
Das System lautet :

  A+B+D-1 = 0
 B-0.6D/C = 0
    C+D-1 = 0
  D-0.3AC = 0

Momentane Naeherung fuer A : 1.00000000000E+00
Momentane Naeherung fuer B : 1.00000000000E+00
Momentane Naeherung fuer C : 1.00000000000E+00
Momentane Naeherung fuer D : 1.00000000000E+00

```
Geben Sie bitte einen der folgenden Buchstaben ein :

G  zur Durchfuehrung eines Schritts einer Gleitpunkt-Newton-Iteration.
L  um die Zwischenergebnisse zu sehen.
N  um keine Zwischenergebnisse zu sehen.
P  um eine neue Gleitpunktnaeherung einzugeben.
E  fuer eine bewiesene Einschliessung der Loesung des Systems.
D  um das momentan bearbeitete Problem zu sehen.
C  fuer Kommentare der Arbeitsgaenge.
S  um ein neues nicht-lineares System einzugeben.
Q  um aufzuhoeren.
I  fuer diese Information.


Geben Sie bitte E,G,S oder Q ein. Die Information erhalten Sie durch I.

* E


Das nicht-lineare Gleichungssystem ist bewiesenermassen loesbar.

Durch den Algorithmus wurde bewiesen, dass die Loesung
mit Sicherheit in folgendem Bereich liegt :

[  7.00322250679E-01,  7.00322250680E-01]
[  1.26058005122E-01,  1.26058005123E-01]
[  8.26380255801E-01,  8.26380255802E-01]
[  1.73619744198E-01,  1.73619744199E-01]

Ausser der Existenz der Loesung wurde durch den Algorithmus darueberhinaus
sogar die Eindeutigkeit der Loesung in diesem Bereich bewiesen !

Geben Sie bitte E,G,S oder Q ein. Die Information erhalten Sie durch I.

* Q
```



```
++++++++++  Die Loesung der nachfolgenden Randwertaufgabe ist y = cos x


Y" + Y = 0

 Y(0) = 1
Y'(0) = 0


Zur vorliegenden Randwertaufgabe existiert genau eine Loesung Y(X)
und diese liegt in dem folgenden Funktionsstreifen V(X):

 V(X) =  + V[ 0]*X↑ 0 + V[ 1]*X↑ 1 + V[ 2]*X↑ 2 + V[ 3]*X↑ 3 + V[ 4]*X↑ 4 +
           V[ 5]*X↑ 5 + V[ 6]*X↑ 6 + V[ 7]*X↑ 7 + V[ 8]*X↑ 8 + V[ 9]*X↑ 9 +
           V[10]*X↑10 + V[11]*X↑11 + V[12]*X↑12

Mit den Intervallkoeffizienten :

V[ 0] = [  9.99999999999E-01,  1.00000000001E+00]
V[ 1] = [           -1.0E-99,            1.0E-99]
V[ 2] = [ -5.00000000001E-01, -4.99999999999E-01]
V[ 3] = [           -2.0E-99,            2.0E-99]
V[ 4] = [  4.16666666606E-02,  4.16666666607E-02]
V[ 5] = [           -2.0E-99,            2.0E-99]
V[ 6] = [ -1.38888886079E-03, -1.38888886078E-03]
V[ 7] = [           -2.0E-99,            2.0E-99]
V[ 8] = [  2.48015230275E-05,  2.48015230276E-05]
V[ 9] = [           -2.0E-99,            2.0E-99]
V[10] = [ -2.75495951432E-07, -2.75495951430E-07]
V[11] = [           -2.0E-99,            2.0E-99]
V[12] = [  2.04071075133E-09,  2.04071075135E-09]


Wertetabelle fuer X = -1(0.1)1 :
V(-1.00) = [   5.4030230586E-01,   5.4030230588E-01]
V(-0.90) = [   6.2160996826E-01,   6.2160996829E-01]
V(-0.80) = [   6.9670670934E-01,   6.9670670936E-01]
V(-0.70) = [   7.6484218728E-01,   7.6484218730E-01]
V(-0.60) = [   8.2533561490E-01,   8.2533561493E-01]
V(-0.50) = [   8.7758256188E-01,   8.7758256191E-01]
V(-0.40) = [   9.2106099400E-01,   9.2106099402E-01]
V(-0.30) = [   9.5533648912E-01,   9.5533648914E-01]
V(-0.20) = [   9.8006657784E-01,   9.8006657786E-01]
V(-0.10) = [   9.9500416527E-01,   9.9500416529E-01]
V( 0.00) = [  9.99999999999E-01,  1.00000000001E+00]
V( 0.10) = [   9.9500416527E-01,   9.9500416529E-01]
V( 0.20) = [   9.8006657784E-01,   9.8006657786E-01]
V( 0.30) = [   9.5533648912E-01,   9.5533648914E-01]
V( 0.40) = [   9.2106099400E-01,   9.2106099402E-01]
V( 0.50) = [   8.7758256188E-01,   8.7758256191E-01]
V( 0.60) = [   8.2533561490E-01,   8.2533561493E-01]
V( 0.70) = [   7.6484218728E-01,   7.6484218730E-01]
V( 0.80) = [   6.9670670934E-01,   6.9670670936E-01]
V( 0.90) = [   6.2160996826E-01,   6.2160996829E-01]
V( 1.00) = [   5.4030230586E-01,   5.4030230588E-01]
```

Literatur

[0] Böhm, H.: Berechnung von Polynomnullstellen und Auswertung arithmetischer Ausdrücke mit garantierter, maximaler Genauigkeit. Dissertation, Universität Karlsruhe (1983)

[1] Bohlender, G.: Floating-point computation of functions with maximum accuracy. IEEE Trans. Comput. C-26, No. 7, 621-632, (1977)

[2] Bohlender, G.: Genaue Summation von Gleitkommazahlen, Computing Supp. 1, 21-32, (1977)

[3] Bohlender G.: Genaue Berechnung mehrfacher Summen, Produkte und Wurzeln von Gleitkommazahlen und allgemeine Arithmetik in Höheren Programmiersprachen, Dissertation, Universität Karlsruhe, 1978

[4] Bohlender, G. Kaucher, E., Klatte, R., Kulisch, U., Miranker, W.L., Ullrich, Ch. und Wolff v. Gudenberg, J.: FORTRAN for contemporary numerical computation, Report RC 8348, IBM Thomas J. Watson Research Center, 1980

[5] Grüner, K.: Fehlerschranken für lineare Gleichungssysteme, Computing Suppl. 1, 47-55, (1977)

[6] Grüner, K.: Allgemeine Rechnerarithmetik und deren Implementierung, Dissertation, Universität Karlsruhe, 1979

[7] Kaucher, E., Rump, S.M.: Generalized iteration methods for bounds of the solution of fixed point operatior equations, Computing 24, 131-137 (1980)

[8] Knuth, D.: "The art of Computer Programming", Vol. 2, Addison-Wesley. Reading, Massachusetts, 1962

[9] Kulisch, U.: An axiomatic approach to rounded computations, TS Report No. 1020. Mathematics Research Center, University of Wisconsin, Madison, Wisconsin, 1969 und Numer. Math. 19, 1-17, (1971)

[10] Kulisch, U.: Interval arithmetic over completely ordered ringoids, TS Report No. 1105, Mathematics Research Center, University of Wisconsin, Madison, Wisconsin, 1970

[11] Kulisch, U.: Implementation and formalization of floating-point arithmetics. Report RC 4608, IBM Thomas J. Watson Research Center, 1973, und C. Caratheodory Symp., 1973, 328-369

[12] Kulisch, U.: Grundlagen des Numerischen Rechnens - Mathematische Begründung der Rechnerarithmetik, Reihe Informatik, Band 19, Wissenschaftsverlag des Bibliographischen Instituts Mannheim, 1976

[13] Kulisch, U., Bohlender, G.: Formalization and implementation of Floating-point matrix operations, Computing 16, 239-261, (1976)

[14] Kulisch, U., Miranker, W.L.: Arithmetic operations in interval spaces, Report RC 7681, IBM Thomas J. Watson Research Center, 1979; Computing Supp. 2, 51-67, (1980)

[15] Kulisch, U., Miranker, W.L.: Computer Arithmetic in Theory and Practice, Academic Press, 1980

[16] Kulisch, U.: Numerisches Rechnen, wie es ist und wie es sein könnte, Jahrbuch Überblicke Mathematik 1981, Bibliographisches Institut, 1981

[17] Lortz, B.: Eine Langzahlarithmetik mit optimaler einseitiger Rundung, Dissertation, Universität Karlsruhe, 1971

[18] Moore, R.E.: "Interval Analysis", Prentice Hall, Englewood Cliffs, New Jersey, 1966

[19] Reinsch, Ch.: Die Behandlung von Rundungsfehlern in der Numerischen Analysis, "Jahrbuch Überblicke Mathematik 1979", Wissenschaftsverlag des Bibliographischen Instituts Mannheim, 43-62, (1979)

[20] Rutishauser, H.: Eine Axiomatik des numerischen Rechnens und ihre Anwendung auf den Quotienten-Differenzen-Algorithmus, "Vorlesungen über Numerische Mathematik", Band 2, pp. 179-221. Birkhäuser-Verlag Basel, (1976)

[21] Ullrich, Ch.: Rundungsinvariante Strukturen mit äußeren Verknüpfung Dissertation, Universität Karlsruhe, 1972

[22] Ullrich, Ch.: Über die beim numerischen Rechnen mit komplexen Zahle und Intervallen vorliegenden mathematischen Strukturen, Computing 1 51-65, (1975)

[23] Rump, S.M., Kaucher, E.: Small bounds for the solution of systems o linear equations, Computing Suppl. 2, 157-164 (1980)

[24] Rump, S.M.: Kleine Fehlerschranken bei Matrixproblemen, Dissertatio Universität Karlsruhe, 1980

[25] Rump, S.M.: Notiz zur Genauigkeit der Arithmetik in Rechenanlagen, Elektronische Rechenanlagen 22 (1980), H.5, S. 243-244

[26] Wippermann, H.-W.: Implementierung eines ALGOL-60 Systems mit Schrankenzahlen, Electron. Datenverarb. 10, 189-194, (1968)

[27] Wolff von Gudenberg, J.: Berechnung maximal genauer Standardfunktionen mit einfacher Mantissenlänge, Elektron. Rechenanlagen 26 H. 5, 230-238 (1984)

[28] Wolff von Gudenberg, J.: Einbettung allgemeiner Rechnerarithmetik in PASCAL mittels eines Operatorkonzeptes und Implementierung der Standardfunktionen mit optimaler Genauigkeit, Dissertation, Universität Karlsruhe, (1980)

[29] Rump, S.M.: Wie zuverlässig sind die Ergebnisse unserer Rechenanlagen, Jahrbuch Überblicke Mathematik, Bibliographisches Institut 163-168 (1983)

[30] Rump, S.M.: Solving algebraic problems with high accuracy, in [33], 51-120 (1983)

[31] Rump, S.M.: New Results on Verified Inclusion, Springer Lecture Notes, to appear

[32] Kulisch, U.und Ullrich, Ch. (Hrsg.): Wissenschaftliches Rechnen und Programmiersprachen, B.G. Teubner Stuttgart (1982)

[33] Kulisch, U. and Miranker, W.L. (Eds): A New Approach to Scientific Computation, Academic Press (1983)

[34] Kulisch, U. and Miranker, W.L.: The arithmetic of the digital computer - a new approach, SIAM-Review, March 1986

[35] High-Acciracy Arithmetic, Subroutine Library, General Information Manual, IBM Program Number 5664-185, (1984)

[36] High Accuracy Arithmetic, Subroutine Library, Program Description and User's Guide, IBM Program Number 5664-185, (1984)

[37] PASCAL-SC Manual and System Disks, A PASCAL-Extension for Scientific Computation, Teubner-John Wiley (1986)