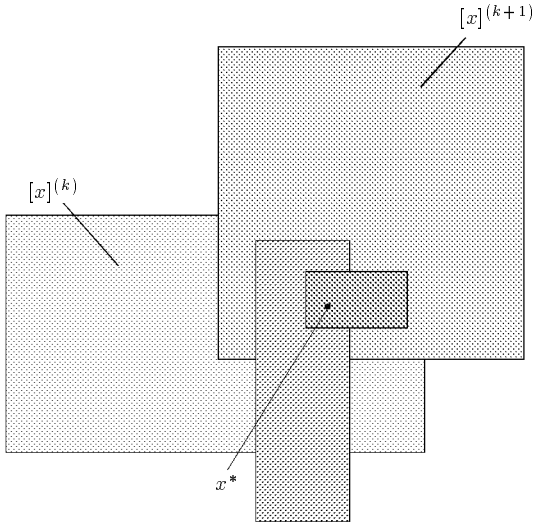
Institut für
Angewandte
Mathematik

Universität Karlsruhe (TH)
D-76128 Karlsruhe

Memorandum
über
Computer, Arithmetik und Numerik

Ulrich Kulisch

Forschungsschwerpunkt
Computerarithmetik,
Intervallrechnung und
Numerische Algorithmen mit
Ergebnisverifikation



Bericht 01/1996

Impressum

Herausgeber:	Institut für Angewandte Mathematik Lehrstuhl Prof. Dr. Ulrich Kulisch Universität Karlsruhe (TH) D-76128 Karlsruhe
Redaktion:	Dr. Dietmar Ratz

Internet-Zugriff

Die Berichte sind in elektronischer Form erhältlich über

`ftp://iamk4515.mathematik.uni-karlsruhe.de`

im Verzeichnis: `/pub/documents/reports`

oder über die World Wide Web Seiten des Instituts

`http://www.uni-karlsruhe.de/~iam`

Autoren-Kontaktadresse

Rückfragen zum Inhalt dieses Berichts bitte an

Vorname Name
Institut für Angewandte Mathematik
Universität Karlsruhe (TH)
D-76128 Karlsruhe

E-Mail: Ulrich.Kulisch@math.uni-karlsruhe.de

Memorandum
über
Computer, Arithmetik und Numerik

Ulrich Kulisch

Inhaltsverzeichnis

Zusammenfassung

Memorandum über Computer, Arithmetik und Numerik: Die heute verfügbare Rechnertechnologie und die damit erzielbaren Rechengeschwindigkeiten erlauben und erfordern eine Erweiterung des arithmetischen Repertoires eines Rechners. Die Genauigkeitsanforderungen an die elementaren Gleitkommaverknüpfungen lassen sich auch auf die häufigsten numerischen Datentypen wie Vektoren, Matrizen, komplexe Zahlen und Intervalle über diesen Typen ausdehnen. Es wurde ein VLSI-Vektorarithmetik-Koprozessor mit integriertem PCI-Interface entwickelt, welcher diese Operationen zur Verfügung stellt. Die Erweiterungen können mit moderaten Hardwarekosten und ohne Geschwindigkeitseinbußen realisiert werden. Eine klare und saubere Definition der arithmetischen Verknüpfungen erleichtert bei vielen Rechenprozessen eine Analyse der berechneten Ergebnisse.

Sprach- und Programmierunterstützungen stehen in Form der XSC-Sprachen zur Verfügung. Dabei greifen die Operatoren für die verschiedenen Datentypen der Sprache direkt auf die Operationen des Chips zu. Durch Operatorüberladung werden zudem eine reelle Langzahlarithmetik als Feld von reals und eine Langzahlintervallarithmetik sowie eine Differentiationsarithmetik zur Verfügung gestellt und in das Laufzeitsystem des Compilers verlagert. D. h. Ableitungen, Taylorkoeffizienten, Gradienten, Jacobi- und Hessematrizen oder Einschließungen all dieser Größen können direkt durch Austausch des Datentyps der Operanden aus dem entsprechenden Ausdruck heraus berechnet werden. Für viele Standardprobleme der Numerik wurden Programmpakete entwickelt, bei denen der Computer die Korrektheit der berechneten Ergebnisse verifiziert und die Existenz und Eindeutigkeit der Lösung innerhalb der berechneten Schranken nachweist. Für eine große Anzahl von Anwendungen wurden damit bereits Techniken und Programme entwickelt, bei denen Rechner selbsttätig die Qualität und Zuverlässigkeit der berechneten Ergebnisse überprüft.

Für viele Anwendungen ist es notwendig mit dem Computer unter Verwendung von Gleitkommaarithmetik strenge mathematische Aussagen zu erzielen. Dies ist z. B. entscheidend bei vielen Simulationsprozessen (Fusionsreaktor) oder bei der mathematischen Modellbildung. Hier muß der Benutzer in der Lage sein, zwischen vom Rechner erzeugten Artefakten und echten Reaktionen des Modells zu unterscheiden. Das Modell kann nur dann systematisch weiterentwickelt werden, wenn Rechenfehler praktisch ausgeschlossen werden können. Durch automatische Ergebnisverifikation wird das numerische Rechnen zu einer echten Teildisziplin der Mathematik.

Abstract

Memorandum on Computers, Arithmetics and Numerics: Advances in computer technology are now so profound that the arithmetic capability and repertoire of computers can and should be expanded. The quality of the elementary floating-point operations should be extended to the most frequent numerical data types or mathematical spaces of computation (vectors, matrices, complex numbers and intervals over these types). A VLSI co-processor chip with integrated PCI-interface has been developed which provides these operations. The expanded capability is gained at modest hardware cost and does not implicate a performance penalty. Language and programming support (the -XSC languages) are available. There, the chip's functionality is directly coupled to the operator symbols for the corresponding data types. By operator overloading a long real arithmetic (array of reals) and long interval arithmetic as well as automatic differentiation arithmetic become part of the runtime system of the compiler. I. e. derivatives, Taylor- coefficients, gradients, Jacobian and Hessian matrices or enclosures of these are directly computed out of the expression by a simple type change of the operands. Techniques are now available so that with this expanded capability, the computer itself can be used to appraise the quality and the reliability of the computed results over a wide range of applications. Program packages for many standard problems of Numerical Analysis have been developed where the computer itself verifies the correctness of the computed result and proves existence and uniqueness of the solution within the computed bounds.

Many applications require that rigorous mathematics can be done with the computer using floating-point. As an example, this is essential in simulation runs (fusion reactor) or mathematical modeling where the user has to distinguish between computational artifacts and genuine reactions of the model. The model can only be developed systematically if errors entering into the computation can be essentially excluded. Automatic result verification re-integrates digital computing into real mathematics.

Konrad Zuse,

dem Pionier der Datenverarbeitung,

zum 85. Geburtstag gewidmet.

Vorbemerkung: Die folgende Abhandlung beschäftigt sich mit einem komplexen Gegenstand. Die behandelten Themen sind stark miteinander verflochten. Der Inhalt der einzelnen Abschnitte läßt sich daher nicht strikt voneinander trennen. Die verschiedenen Abschnitte ergänzen sich vielmehr gegenseitig. Sie können weitgehend in beliebiger Reihenfolge gelesen werden.

1 Einleitung

Gleitkommaarithmetik ist die schnelle Art, Rechnungen im naturwissenschaftlich-technischen Bereich auszuführen. Die verwendeten Gleitkomma-Zahlenformate stellen heute einen großen Zahlenbereich zur Verfügung, so daß lästige Skalierungen im allgemeinen vermieden werden. Gleitkommaarithmetik wird seit Jahrzehnten erfolgreich verwendet, und es gibt große Programmpakete auch für äußerst komplexe Anwendungen. Durch umfangreiche Tests an Standardproblemen „nach allen Regeln der Kunst“ und unter Umständen durch Bereitstellung von speziellen Lösungsvarianten für bekannte kritische Fälle wird ein hoher Grad an Robustheit erlangt. Dennoch kann auch ein völlig korrekt implementierter Algorithmus sehr ungenaue oder sogar völlig falsche Ergebnisse liefern. Die berechneten Daten einer langen Gleitkommarechnung enthalten im allgemeinen keinerlei Hinweis darauf, ob die Rechnung zu einem brauchbaren Ergebnis geführt hat oder nicht. Es stellt sich die Frage: Kann man die Situation verbessern? In vielen Fällen lautet die Antwort auf diese Frage heute ja.

Traditionell liegt die Kontrolle des Rechenfehlers in der Verantwortung des Benutzers. Bei noch relativ langsamen Rechnern war dies auch durchaus zu vertreten. Die Rechengeschwindigkeit hat heute aber Größenordnungen erreicht (viele Millionen Gleitkommaoperationen in der Sekunde), bei denen eine Fehleranalyse im herkömmlichen Sinne durch den Benutzer praktisch nicht mehr möglich ist. Hinzu kommt, daß von seiten der Hersteller immer wieder versucht wird, die Rechengeschwindigkeit zu Lasten der mathematischen Präzision zu steigern. Für serielle Rechner entwickelte Algorithmen werden Parallelisierungen unterworfen, ohne sich Gedanken darüber zu machen, ob

und inwieweit dies die Ergebnisgenauigkeit beeinflußt. Bei herkömmlichen Vektorrechnern beispielsweise stellt die Hardware eine Reihe „zusammengesetzter Operationen“, wie „Akkumuliere eine Reihe von Zahlen oder Produkten“, bereit, welche besonders schnell ausgeführt werden. Durch die dabei verwendete Technik wird die Reihenfolge der Additionen verändert, was zu falschen Ergebnissen führen kann. Eine Fehleranalyse von seiten des Benutzers müßte zunächst die Hardware einer solchen Anlage einer genauen Analyse unterziehen, was praktisch nicht möglich ist.

Man trifft immer wieder Benutzer, welche solche Tricks zur Beschleunigung der Ausführungszeiten einer Rechnung sogar begrüßen und fordern mit dem Argument „ich rechne lieber schnell als genau“. Solange schlechte Arithmetik schnell ist und genaue in Software simuliert werden muß, ist da durchaus etwas Wahres dran. Heute aber gilt diese Alternative nicht mehr. In gleicher Technologie kann man auch alle zusammengesetzten Operationen von Vektorrechnern mathematisch völlig einwandfrei mindestens gleich schnell, wenn nicht sogar noch schneller, bereitstellen als alles, was üblicherweise zur Verfügung steht.

Der vorliegende Beitrag versucht, eine Bestandsaufnahme zu geben und Orientierung zu vermitteln. Bei der heute verfügbaren Rechnertechnologie und den damit erzielbaren Rechengeschwindigkeiten scheint eine Umorientierung erforderlich zu sein. Gesichtspunkten wie einfache und übersichtliche Programmierbarkeit, hohes Abstraktionsniveau bei den Programmiersprachen, Zuverlässigkeit berechneter Ergebnisse, saubere Definition der Rechnerarithmetik und automatische Kontrolle des Rechenfehlers durch den Rechner selbst bis zu einem gewissen Grade sollten größere Aufmerksamkeit gewidmet und höhere Priorität eingeräumt werden. Bezüglich all dieser Gesichtspunkte sind in den letzten Jahrzehnten große Fortschritte erzielt worden. Die Hersteller werden diese aber nur in ihre Anlagen übernehmen, wenn sie von den Benutzern auch gefordert werden, was natürlich voraussetzt, daß sie diesen bekannt sind. Letztlich muß es das Ziel sein, den Rechner bezüglich all seiner Komponenten wie Architektur, Arithmetik, Programmiersprache und Compiler, Algorithmen und Numerik so weiterzuentwickeln, daß man nicht nur Näherungsrechnung, sondern echte Mathematik damit betreiben kann, in dem Sinne, daß genaue Schranken für die Lösung von Problemen berechnet werden oder sogar Existenz- und Eindeutigkeitsaussagen mit dem Rechner nachgewiesen werden können. Die heute verfügbare, sehr leistungsfähige Rechnertechnologie kommt einem hier entgegen. Viele Anwendungen werden überhaupt erst durch Fortschritte auf den genannten Gebieten erschlossen. Bei vielen Anwendungen ist die mathematische Qualität und Zuverlässigkeit der berechneten Ergebnisse für den Erfolg entscheidend. Als Beispiel sei die mathematische Modellbildung erwähnt. Die häufig mittels heuristischer Methoden entwickelten Modelle können nur dann systematisch weiterentwickelt werden, wenn der Rechenfehler weitestgehend ausgeschlossen werden kann. In einem numerischen Expertensystem entscheidet die automatische Ergebniskontrolle und damit der Rechner selbst darüber, ob ein anderer Algorithmus aufgerufen werden muß oder nicht.

Anhand einiger Beispiele soll die Notwendigkeit von Fortschritten auf den genannten Gebieten illustriert werden.

1.1 Berechnung einer Turbine mit hoher Drehzahl

Es soll eine große und schwere Turbine, etwa für ein Dampf- oder Wasserkraftwerk, gebaut werden. Wegen der großen Masse und ihrer teuren und präzisen Bauweise verbietet sich hier ein umfangreiches Experimentieren mit Hardwaremodellen. Die Auslegung der Turbine, ihre Dimensionierung und Schwingungseigenschaften müssen vor ihrem Bau berechnet werden. Das Anfahren einer Turbine ist im allgemeinen mit Temperaturänderungen verbunden und darf daher nicht zu schnell erfolgen. Im Betrieb führt eine solche Turbine Schwingungen um ihre Rotationsachse aus. Hier kommt es darauf an, die Eigenfrequenzen, bei denen Resonanz auftreten kann, genau zu kennen. Ein Betrieb in der Nähe dieser Eigenfrequenzen ist unbedingt zu vermeiden, da das Aggregat dann explosionsartig zu Bruch geht. Beim Anfahren der Turbine wird man daher in der Nähe der Eigenfrequenzen rasch und dazwischen langsamer beschleunigen. Eine genaue Kenntnis der Eigenfrequenzen ist daher erforderlich.

In einem praktischen Anwendungsfall ergab eine herkömmliche Näherungsrechnung bei niedrigen Drehzahlen nur wenige Eigenfrequenzen. Von einer gewissen Drehzahl an traten hingegen gehäuft Eigenfrequenzen auf. Will man auch in diesen Drehzahlbereich hinein beschleunigen und hat vielleicht den Verdacht, daß die Rechnung hier nicht ganz in Ordnung ist, so wird man herkömmlicherweise das Problem in diesem Bereich mit mathematischen Abschätzungen genauer untersuchen.

Durch Berechnung mit hochgenauen, verifizierenden Methoden konnte im vorliegenden Fall rein rechnerisch nachgewiesen werden, daß auch bei höheren Drehzahlen tatsächlich nur wenige diskrete Eigenfrequenzen vorliegen. (S. dazu das Bild 1. Die Schnittpunkte der rot gezeichneten Kurve mit der x-Achse beschreiben die Eigenfrequenzen.) Mathematische Zusatzarbeit ist in diesem Fall nicht mehr erforderlich.

1.2 Das Doppelpendel

Das Doppelpendel wird durch seine beiden Massen, die Pendellängen und die Ablenkungswinkel charakterisiert. Das Schwingungsverhalten wird durch zwei nichtlineare gewöhnliche Differentialgleichungen zweiter Ordnung beschrieben. Wir integrieren diese Differentialgleichungen mit zwei unterschiedlichen numerischen Lösern, wobei wir in beiden Fällen mit identischen Anfangsauslenkungen beginnen. Bild 2 zeigt die berechneten Lösungskurven für verschiedene Anfangsbedingungen. Die Ergebnisse jeder der beiden Routinen sehen plausibel aus. Verifizierung durch Visualisierung, wie man dies gelegentlich nennt, sagt, daß man wahrscheinlich die richtige Lösung berechnet hat. Hält man aber die beiden Rechnungen nach gewissen festen Zeitpunkten an, so stellt man unterschiedliche Pendelstellungen fest. Die berechneten Lösungen stimmen also nicht überein, und von da ab unterscheiden sie sich tatsächlich wesentlich. Es stellt sich also die Frage: Welches ist die richtige Lösung oder sind etwa beide berechneten Lösungen falsch?

Typischerweise fängt ein Benutzer einer Rechenanlage in einer solchen Situation an, mit weiteren Rechenläufen zu experimentieren, um ein Gefühl dafür zu entwickeln, welche der berechneten Lösungen wohl die richtige sein mag. Vielleicht gelingt es ihm auch, durch mathematische Abschätzungen herauszufinden, welche von beiden Berechnungen

die wahre Lösung besser beschreibt. In unserem Fall ist die Entscheidung einfach; denn die Schwingungskurve des rot gezeichneten Pendels wurde mit einem verifizierenden Löser berechnet, welcher die wahre Lösung bis auf zehn Stellen genau in Schranken einschließt. Zusätzliche mathematische Abschätzungen werden dadurch überflüssig.

1.3 Berechnung der Prozeßparameter von Gasen

Für viele technische Anwendungen in den Ingenieurwissenschaften, insbesondere in der Thermodynamik und der Gaskinetik interessiert man sich für die sogenannten Transport- oder Prozeßparameter von Gasen; das sind im wesentlichen die Viskosität, die Wärmeleitfähigkeit und die Diffusion. Für technische Anwendungen werden die Prozeßparameter sehr genau benötigt. Sie werden herkömmlicherweise experimentell durch Messungen ermittelt, was für jedes einzelne Gas an Personal und Material extrem aufwendig und kostspielig ist.

Die Prozeßparameter werden beispielsweise zur Ermittlung der Energieeffizienz eines Gases als Kühlmittel in Klimaanlagen, Wärmepumpen und Kühlschränken benötigt. Wenn es gelänge, ein Kühlmittel zu entwickeln, welches 10 Prozent weniger Energie verbraucht oder auch nur geeignet ist, das derzeit verwendete, für die Ozonschicht schädliche Kühlmittel Fluor-Chlor-Kohlenwasserstoff (FCKW) zu ersetzen, wäre dies von unermeßlichem wirtschaftlichem Nutzen. Man möchte daher gerne mit sehr vielen Gasen experimentieren. Dabei stößt man sehr schnell an finanzielle Grenzen.

Es ist daher naheliegend zu versuchen, die Prozeßparameter zu berechnen. Dazu gibt es seit Jahrzehnten formelmäßige mathematische Beschreibungen im Rahmen einer etablierten Theorie. Wiederholte Berechnungsversuche mit herkömmlichen Methoden gelten bei Fachleuten aber als derart unzuverlässig, daß es nicht sinnvoll erscheint, technische Geräte danach zu bauen. Dies hat zwei Gründe:

In den formelmäßigen Zusammenhang gehen die sogenannten zwischenmolekularen Potentiale von Gasen ein. Die Moleküle eines Gases verhalten sich nicht wie starre Kugeln, die sich erst bei Berührung gegenseitig beeinflussen. Man hat sich die Molekülstöße vielmehr als Wechselwirkung von die Moleküle umgebenden Kraftfeldern vorzustellen. Die Moleküle besitzen Potentiale. Diese Felder werden bereits in großen Abständen wirksam. Bei größeren Abständen ziehen sich die Moleküle an, und sie stoßen sich ab, wenn die zwischenmolekularen Abstände gering sind. Für die zwischenmolekularen Potentiale gibt es viele verschiedene formelmäßige Ansätze, und man ist nicht sicher, welcher davon den Sachverhalt richtig beschreibt.

Um diesen zu ermitteln, muß man die Prozeßparameter berechnen und versuchen, durch Variation des Modellansatzes bei einigen Gasen Rechen- und Meßergebnisse in Übereinstimmung zu bringen. Wenn dies gelingt, ist die Berechnung der Prozeßparameter für eine große Anzahl von Gasen möglich. Hier tut sich aber noch eine zweite Schwierigkeit auf.

Die Formeln zur Berechnung der Prozeßparameter enthalten dreifache singuläre Integrale, welche nichtlinear ineinander geschachtelt sind. Um genügend Genauigkeit im Endergebnis zu erhalten, muß insbesondere das innerste Integral sehr oft (viele tausend Mal) ausgewertet werden. Auch dabei muß die Genauigkeit kontrolliert werden. Dies

erfolgt üblicherweise durch Abschätzung des Restgliedes der verwendeten Quadraturformel. Wegen der großen Häufigkeit, mit der dies erforderlich ist, kann dies nicht mehr von Hand ausgeführt werden. Es muß vielmehr dem Rechner übertragen werden. Das Restglied wird dabei intervallmäßig ausgewertet. Dadurch kann die Quadratur adaptiv gesteuert werden, d. h. an Stellen, an denen der Integrand ruhig verläuft, wird mit großer Schrittweite und an unruhigen Stellen mit kleiner Schrittweite integriert.

Es ist praktisch unmöglich, ein Programm, welches den hier skizzierten Lösungsweg beschreibt und nachvollzieht, in einer der verbreiteten üblichen Programmiersprachen FORTRAN, ALGOL, PASCAL, MODULA, BASIC oder C unter Verwendung von einfacher Gleitkommaarithmetik zu erstellen. Dazu sind wesentlich ausdrucksstärkere Programmierumgebungen wie die später noch zu beschreibenden XSC-Sprachen und eine wesentlich leistungsfähigere Rechnerarithmetik notwendig, welche die Programmierung massiv vereinfachen und dadurch überhaupt erst ermöglichen. Mit diesem Kenntnisstand gerät eine Lösung des geschilderten Problems erstmals in Reichweite. Die Kosten, welche für die Entwicklung eines solchen Programmes bei Verwendung geeigneter Werkzeuge aufgewendet werden müssen, sind u. U. geringer als diejenigen, welche für die experimentelle Ermittlung der Prozeßparameter für ein einziges Gas anfallen.

1.4 Dynamische Systeme

Wir betrachten ein einfaches nichtlineares Iterationsverfahren, wie es heute häufig bei der Behandlung chaotischer Systeme vorkommt, die sogenannte *logistische Gleichung*:

$$x_{n+1} := 3,75 * x_n * (1 - x_n), \quad n = 0, 1, 2 \ldots \tag{1}$$

Solche Systeme haben die Eigenschaft, daß sie sehr empfindlich sind gegenüber Änderungen in den Daten. (1) ist eine Iterationsvorschrift für reelle Zahlen. Wir berechnen die durch (1) definierte Folge in Gleitkommaarithmetik und beginnen mit dem einziffrigen Anfangswert $x_0 = 0,5$. In jedem Iterationsschritt vergrößert sich die Anzahl der Ziffern und sehr rasch sind alle Ziffern unserer 16-stelligen Gleitkommamantisse erfaßt. Wenn wir jetzt mit Gleitkommaarithmetik weiterrechnen und nach jeder Operation in eine Gleitkommazahl runden, verlassen wir die durch x_0 definierte Iterationsfolge. Man könnte den jeweils berechneten Näherungswert $\tilde{x_\nu}$ als Startwert einer neuen Iteration interpretieren. Aber auch diese durch $\tilde{x_\nu}$ definierte Folge wird bereits im nächsten Iterationsschritt wieder verlassen. Die in Gleitkommaarithmetik tatsächlich berechneten Werte springen in jedem Schritt von einer Iterationsfolge auf eine andere. Wenn die verschiedenen Folgen sich stark voneinander unterscheiden, berechnet man in Gleitkommaarithmetik eine Folge, welche mit der durch x_0 definierten Folge nicht viel zu tun hat.

Die zweite Spalte in Bild 3 zeigt die in Gleitkommaarithmetik in doppelter Genauigkeit berechneten „Näherungswerte“, während die vierte Spalte hochgenaue Einschließungen der exakten Werte enthält. Die Werte der dritten Spalte wurden mit Intervallarithmetik ebenfalls in doppelter Genauigkeit berechnet. Sie zeigen drastisch wie rasch bei Rechnung in Gleitkommaarithmetik (zweite Spalte) die Anzahl der gültigen Ziffern abnimmt.

Man könnte auch in jedem Iterationsschritt einen Korrekturprozeß draufsetzen, welcher die Genauigkeit immer bis auf eine gewisse Anzahl von Ziffern oder volle Gleitkommagenauigkeit verbessert. Eine andere Möglichkeit besteht darin, mit dynamischer Genauigkeit zu arbeiten und immer alle gültigen Ziffern in der Rechnung mitzuführen. Der Rechenaufwand kann dabei aber sehr schnell explodieren.

Bild 4 zeigt ein ähnliches Beispiel (Barry Martin). In Gleitkommaarithmetik wird eine fraktale Punktfolge (x_n, y_n) nach folgender Vorschrift berechnet[1]:

$$\begin{aligned} &x_{n+1} := C * x_n + S * y_n - sin(-3 * x_n - 36) \\ &y_{n+1} := 12 - S * x_n + C * y_n, \\ &x_0 := 0, \;\; y_0 := 0, \qquad C := \cos \pi/3, \;\; S := \sin \pi/3. \end{aligned} \tag{2}$$

Es werden (x_n, y_n) ausgegeben. Nach jeweils 2000 Iterationen wird die Farbe geändert. Die Farbfolge ist zyklisch: rot, fuchsin, braun, hellgrau, dunkelgrau, hellblau, hellgrün, helles türkis, hellrot, helles fuchsin und gelb. Die Figur zeigt das Ergebnis zweier identischer Rechnungen mit Gleitkommaarithmetik in einfacher (linkes Bild) und doppelter Genauigkeit (rechtes Bild). Bei den oberen Bildern wurden jeweils 28000 und bei den unteren 45000 Punkte berechnet (G. Bohlender). Die Bilder sind offensichtlich recht unterschiedlich. Wenn man nur an schönen Bildern interessiert ist, mag man sich darüber freuen. Wie im vorigen Beispiel haben aber beide Bilder nicht viel mit dem wahren Verlauf der durch $x_0 := 0$ und $y_0 := 0$ definierten Folge zu tun.

Ähnliche Verhältnisse liegen auch bei dynamischen Systemen von Differentialgleichungen vor. Auch hier gibt es zu jedem Anfangswert bei Rechnung im Reellen eine eindeutig bestimmte Lösung. Im allgemeinen existiert keine Darstellung der Lösung in geschlossener Form durch mathematische Formeln. Eine Berechnung der Lösung mit reellen Zahlen ist ebenfalls nicht möglich. Bei Rechnung mit einem Näherungsverfahren in Gleitkommaarithmetik springt die „Näherung“ im allgemeinen in jedem Schritt von einer Lösungskurve auf eine andere. Wenn sich die Lösungskurven stark voneinander unterscheiden, kommt dabei unter Umständen etwas heraus, was mit der wahren, gesuchten Lösung nichts mehr zu tun hat.

Letzteres gilt insbesondere wieder für chaotische Lösungen dynamischer Systeme. Auch diese sind sehr empfindlich gegenüber Änderungen in den Daten. Bei numerischer Näherungsrechnung erfolgt zwangsläufig in jedem Schritt eine Änderung der Daten. Chaotisches Verhalten einer mittels einer Näherungsrechnung berechneten Bahnkurve kann daher durchaus durch die Numerik verursacht sein. Nur eine verifizierte Berechnung der Lösungskurve kann hier weiterhelfen. Eine mit einem solchen Löser berechnete Bahnkurve und ein damit gezeichnetes Bild sind korrekt wie ein mathematisch bewiesener Satz. Im Grunde können numerisch nur mit solchen Methoden Aussagen über chaotisches Verhalten von Lösungen dynamischer Systeme gewonnen werden. Natürlich sind auch die dadurch gegebenen Möglichkeiten durch die verfügbare Rechenleistung begrenzt.

Erfreulicherweise sind nicht alle Lösungen von Differentialgleichungen derart empfindlich gegen auch nur kleinen Änderungen in den Daten. Sonst wäre die zuverlässige Berechnung von Satellitenbahnen mit numerischen Näherungsmethoden nicht möglich.

[1] Im gedruckten Original steht hier versehentlich eine Formel, die nicht zu Bild 4 gehört!

Bereits im Jahre 1801 ist es dem Mathematiker C. F. Gauß mit wesentlich schwächeren Rechenhilfsmitteln als sie uns heute zur Verfügung stehen gelungen, aus nur wenigen Meßwerten der Bahnkurve des Planetoiden Ceres dessen Bahnkurve vollständig zu berechnen und sein Wiederauftauchen hinter der Sonne vorherzusagen.

Oft wird argumentiert, daß eine Rechnung nicht hochgenau ausgeführt zu werden braucht, da man an nur wenigen Ziffern des Ergebnisses interessiert ist oder da bei der gegebenen Fragestellung überhaupt nur wenige Ziffern der gesuchten Lösung berechenbar sind. Bei kleinen und einfachen Berechnungen mag dies richtig sein. Bei der bei heutigen Rechnungen aber üblicherweise ausgeführten riesigen Anzahl von Gleitkommaoperationen ist dies mit Sicherheit falsch. Die Algorithmen werden im allgemeinen im Raum der reellen Zahlen hergeleitet und ihre Eigenschaften werden unter dieser Voraussetzung nachgewiesen. Bei Ausführung eines Algorithmus in Gleitkommaarithmetik liegen diese Voraussetzungen nicht vor. Die aufgeführten Beispiele zeigen, daß viele, während einer Rechnung mitgeführte Ziffern nicht automatisch viele gültige Ziffern im Ergebnis zur Folge haben oder gar garantieren. Bei umfangreichen Rechnungen ist daher eine hohe Rechengenauigkeit in jedem Fall angeraten. Die Beispiele zeigen sogar, daß man gelegentlich gar nicht genau genug rechnen kann. Bei arithmetischen Ausdrücken, welche nur die vier Grundrechenoperationen enthalten, kann man zeigen, daß die Ergebnisgenauigkeit zunimmt, wenn man die Anzahl der in der Rechnung mitgeführten Ziffern vergrößert. Bei komplizierteren Funktionen nimmt sie zumindest nicht ab. In jedem Fall müssen die Rechner so weiterentwickelt werden, daß sie in kritischen Fällen selbst feststellen können, ob die Rechnung verunglückt ist oder nicht.

1.5 Einsatz von Supercomputern

Kann man Supercomputer für Probleme, wie sie in den voranstehenden Abschnitten beschrieben wurden, einsetzen?

Vektorprozessoren und Parallelverarbeitung bilden die Bausteine von Supercomputern.

Sogenannte Massivparallelrechner erreichen ihre hohe Rechenleistung durch eine große Anzahl parallel arbeitender, relativ preiswerter Prozessoren. Im einfachsten Fall sind es Prozessoren, welche auch in Personal Computern oder Workstations verwendet werden. Bei den Parallelrechnern hat es sich eingebürgert, bei der Programmierung Genauigkeitsfragen weitgehend außer acht zu lassen. Man tut so, als käme es nur darauf an, die Rechenoperationen geschickt auf viele Prozessoren zu verteilen, als läge nur ein organisatorisches Problem vor. Letztlich entartet die Numerik dadurch zu einer experimentellen Wissenschaft. Einen Mathematiker müßte dies eigentlich erschrecken. Notwendig ist diese Einstellung keineswegs. Um Abhilfe zu schaffen, müßte man zunächst den einzelnen Rechnerknoten arithmetisch und programmiersprachlich weiterentwickeln.

Vektorprozessoren erreichen ihre hohe Rechenleistung durch Verwendung besonders schneller Technologien und durch Pipelining (Fließbandverarbeitung) von Vektoroperationen. Daher kommt auch ihr Name. Bei den Vektorrechnern besteht die Problematik darin, daß die besonders schnellen, sogenannten Vektoroperationen heute generell nach einem falschen Konstruktionsprinzip implementiert sind. Akkumulationen von Zahlen

und Produkten werden in Gleitkommaarithmetik in Fließbandverarbeitung ausgeführt. Dadurch wird die Reihenfolge der Additionen vertauscht, was zu erheblichen Genauigkeitseinbrüchen und zu völlig falschen Ergebnissen führen kann. Eine Fehleranalyse von seiten des Benutzers wird dadurch völlig unmöglich. Auch dies ist keine Denknotwendigkeit. Hersteller, auf diese Problematik angesprochen, antworten schlicht mit der Feststellung, daß diese Effekte bekannt seien. Wenn man sie vermeiden wolle, müsse man die Vektoreinheit abschalten. Dabei wäre auch hier Abhilfe leicht möglich. In Festkommaarithmetik lassen sich diese Operationen immer richtig und zudem auch noch schneller ausführen.

Damit beantwortet sich auch die eingangs gestellte Frage. Will man mit einem Vektorrechner die Ergebnisgenauigkeit kontrollieren, so muß man die Vektoreinheit abschalten und korrekte Vektoroperationen über die ganzzahlige Arithmetik simulieren. Zudem muß eine leistungsfähige Programmierumgebung implementiert werden, mit der man diese Operationen einfach ansprechen kann. Beides erfordert einen erheblichen Personaleinsatz und natürlich auch Zeit. Durch die Softwaresimulation der Arithmetik wird der Rechner wesentlich langsamer. Wegen der schnellen verwendeten Technologie ist er wahrscheinlich immer noch schneller als eine Workstation. Bis alles einsatzbereit ist, ist der Rechner aber in der Regel technologisch veraltet.

Um das Interesse an Supercomputern zu stimulieren, hat man in den USA die „GRAND CHALLENGES“ und in Japan das „REAL WORLD COMPUTING“ erfunden. Darunter werden Probleme zusammengefaßt, welche riesige Rechenleistung und enorme Speicherkapazität erfordern, wie beispielsweise Wettervorhersage, Modellierung des Klimas, Berechnung turbulenter Strömungen, Modellierung der Supraleitung, Strömung der Ozeane, Quantum Chromodynamik und sogar solche Probleme, bei denen das zu lösende Problem nicht eindeutig und vollständig beschrieben ist. Man spricht daher heute schon vielerorts vom TERAFLOPS-Computer (1 TERAFLOPS $= 10^{12}$ floating-point operations per second) mit zehn Giga Words ($= 10^{10}$ Worte) Speicher.

Auf Mathematiker wirkt so etwas faszinierend. Man beschäftigt sich gerne schon heute mit Problemen, welche auch die Rechner von morgen auslasten können. Bei den heute verwendeten Lösungsmethoden sind dies vor allem Probleme bei partiellen Differentialgleichungen, welche auf riesige Gleichungssysteme führen. Natürlich ist es auch wichtig, sich mit solchen Problemen zu beschäftigen, um unter Umständen rechtzeitig Algorithmen und Erfahrungen für die Rechner von morgen zu entwickeln. Die Größe eines Problems darf aber nicht unbedingt mit seiner Bedeutung gleichgesetzt werden.

Schon bei der seinerzeitigen Explosion der Rechenleistung von den mechanischen zu den ersten elektronischen Rechnern in den fünfziger Jahren hat man davon geträumt, einmal das Wetter vorausberechnen zu können. Seither ist die Rechenleistung in geradezu gigantischem Ausmaß gesteigert worden. Eine zuverlässige Berechnung des Wetters etwa über 48 Stunden mit heutigen Methoden liegt noch immer außerhalb der Reichweite der heutigen Supercomputer.

In den vergangenen Jahrzehnten waren es vor allem die vielen mittleren und kleinen Probleme, die man tatsächlich lösen konnte, welche zum technischen Fortschritt beigetragen haben. Mittels Satellitenaufnahmen und fortgeschrittener Kommunikationstechnik ist die Wettervorhersage heute um ein Vielfaches besser und zuverlässiger

als vor 40 Jahren. Auch zu diesem Fortschritt haben die Rechner beigetragen, indem viele kleine und mittlere Probleme tatsächlich gelöst wurden.

Eine schnelle Workstation bringt heute durchaus die Leistung eines Supercomputers der frühen achtziger Jahre. Damals gab es weltweit vielleicht 100 solcher Rechner. Sie waren damit ausgewählten Spezialisten vorbehalten. Sie waren in erster Linie schnell und verfügten über ein relativ einfaches Betriebssystem. Heute gibt es viele Millionen Personal Computer und Workstations der Spitzenklasse. Um diese vielen Rechner breiten Bevölkerungsschichten zugänglich zu machen, hat man sehr viel der enormen Rechenleistung dazu verwandt, benutzerfreundliche Bedienungsoberflächen zu schaffen. Man schätzt, daß heute etwa 80 Prozent der Rechenleistung für den Bedienungskomfort verbraucht werden und nur etwa 10 bis 20 Prozent für die Programmausführung zur Verfügung stehen. Manch ein Benutzer wundert sich darüber, wenn ein theoretisch viel schnelleres Programm in der Praxis gar nicht so viel schneller läuft. Die Beschleunigung wirkt sich unter Umständen eben nur auf die 10 bis 20 Prozent aus.

Dies sollte als Hinweis darauf verstanden werden, daß bei den heutigen Rechnern endlich auch im Hinblick auf die Numerik die Prioritäten anders gesetzt werden können und sogar müssen, nämlich weg von dem alleinigen Schielen nach höheren Ausführungsgeschwindigkeiten und hin zu einer wesentlich leistungsfähigeren Arithmetik, anspruchsvolleren Programmierumgebungen, mehr Sicherheit in den Ergebnissen bis zu automatischer Genauigkeitskontrolle und Verifikation von Ergebnissen. Aus Sicht der Numerik hat die Entwicklung der Rechner mit den enormen, durch die Technologie bedingten Fortschritten nicht mitgehalten.

Ein amerikanischer Kollege illustriert das weit verbreitete, ausschließliche und bedingungslose Trachten nach höherer Rechengeschwindigkeit gelegentlich etwas drastisch durch die folgende kleine Geschichte: Ein Junge kommt aus der Schule und erzählt dem Vater, daß er im Rechenwettbewerb die Bronzemedaille gewonnen habe. Der Vater staunt und möchte gerne wissen, wie er dies gemacht hat. Der Junge antwortet: „Der Lehrer hat gefragt: Was ist drei mal sieben? Da habe ich ganz schnell 19 gesagt. Hätte ich zwanzig gesagt, so hätte ich die Silbermedaille gewonnen."

1.6 Die generelle Problematik des einfachen Gleitkommarechnens

Eine wesentliche Problematik des herkömmlichen Gleitkommarechnens besteht darin, daß die berechneten Ergebnisse richtig, ungenau oder auch völlig falsch sein können. Die Rechnung liefert i. a. keinerlei Hinweis darauf, welcher der drei Fälle vorliegt. Bei den hohen Rechengeschwindigkeiten ist eine Fehlerabschätzung von Hand heute in der Regel unmöglich. Wie problematisch einfaches Gleitkommarechnen sein kann, ist immer wieder demonstriert worden. Man kann aber nicht oft genug daran erinnern und erinnert werden. Es seien daher auch hier noch einmal einige Beispiele aufgeführt.

1.61 Berechnet man auf einem herkömmlichen Computer in doppelter Genauigkeit (double precision) die theoretisch gleichwertigen Ausdrücke

$$\begin{array}{l} 10^{20} + 17 - 10 + 130 - 10^{20} \\ 10^{20} + 17 - 10^{20} - 10 + 130 \\ 10^{20} - 10^{20} + 17 - 10 + 130 \\ 10^{20} + 17 + 130 - 10^{20} - 10 \end{array}$$

so erhält man der Reihe nach die Werte $0, 120, 137$ und -10! Muß dies heute noch so sein? Wir werden später sehen, daß man mit einer etwas anderen Technik auch bei Änderung der Summationsreihenfolge ohne weiteres immer den richtigen Wert erhalten kann.

1.62 Wir betrachten ein einfaches lineares Gleichungssystem $A * x = b$ mit

$$A = (a_{ij}) = \begin{pmatrix} 64919121 & -159018721 \\ 41869520,5 & -102558961 \end{pmatrix} \quad \text{und} \quad b = (b_i) = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

Die Lösung berechnet sich nach der einfachen Formel:

$$\begin{array}{l} x_1 = a_{22}/(a_{11} * a_{22} - a_{12} * a_{21}) \\ x_2 = -a_{21}/(a_{11} * a_{22} - a_{12} * a_{21}). \end{array}$$

In konventioneller Gleitkommaarithmetik (double precision) erhält man für die Lösung die „Näherungswerte“:

$$\tilde{x_1} = 102558961 \text{ und } \tilde{x_2} = 41869520,5.$$

Die korrekte Lösung lautet hingegen:

$$x_1 = 205117922 \text{ und } x_2 = 83739041.$$

Nach nur vier Gleitkommaoperationen ist keine einzige Ziffer der Gleitkommanäherung $\tilde{x_1}, \tilde{x_2}$ mehr korrekt.

1.63 Wir betrachten den einfachen arithmetischen Ausdruck

$$f = 333,75 * b^6 + a^2(11a^2 * b^2 - b^6 - 121 * b^4 - 2) + 5,5 * b^8 + a/(2 * b)$$

und berechnen den Wert für $a = 77617,0$ und $b = 33096,0$.

Keines der Daten enthält mehr als fünf von null verschiedene Ziffern. Es sollte also eigentlich kein Problem vorliegen. Man erhält auf einer Workstation in single, double und extended precision (nach dem IEEE-Arithmetik-Standard) immer den gleichen Wert $f = +1,172603....$ Es gibt auch Rechner, auf denen man Werte von der Größenordnung 10^{17} oder 10^{19} bekommt. Mittels Intervallrechnung findet man, daß der mathematisch korrekte Wert von f in dem Intervall $f = -0,827396059946821_4^3$ liegt. Bei den berechneten Näherungswerten stimmt also nicht einmal das Vorzeichen.

1.64 Neben linearen Gleichungssystemen sind in der Numerik und in den Anwendungen sehr häufig Werte von Polynomen zu berechnen. Es sei daher auch hierzu noch ein Beispiel angeführt. Wir betrachten ein Polynom dritten Grades in der Nähe einer Nullstelle.

$$p(x) = 223200658x^3 - 1083557822x^2 + 1753426039x - 945804881$$
$$= ((223200658x - 1083557822)x + 1753426039)x - 945804881$$

Die Zick-Zack Kurve in Bild 5 verbindet diejenigen Werte, welche mit dem bekannten, in der Numerik zur Polynomauswertung üblicherweise empfohlenen Hornerschema (untere Zeile in obiger Polynomdarstellung) berechnet wurden. Durch die Anwendung der Gleitkommaarithmetik akkumulieren sich dabei die Rundungsfehler so, daß in dem angegebenen Bereich von x ungenaue Werte für $p(x)$ berechnet werden. Die glatte Linie zeigt demgegenüber den wahren Verlauf des Polynoms. Sie ist mit einem Verfahren berechnet worden, bei dem der Rechner die Polynomwerte hochgenau in Schranken einschließt. Die Ergebnisgenauigkeit wird dabei vom Rechner selbst kontrolliert.

Natürlich sind diese Beispiele konstruiert, um auf wenig Papier kritische Situationen aufzuzeigen. Aber wer kann schon übersehen, was bei einer Rechnung von einer halben Stunde auf einem Rechner, welcher 100 Millionen Gleitkommaoperationen in der Sekunde ausführt, für Konstellationen in den Zwischenergebnissen auftreten? Die einfachen Iterationen (1) und (2) sehen auf den ersten Blick auch nicht kritisch aus.

Von C. F. Gauß stammt die Bemerkung: *„Der Mangel an mathematischer Bildung gibt sich durch nichts so auffallend zu erkennen, wie durch maßlose Schärfe im Zahlenrechnen.“* Gelegentlich wird versucht, mit dieser Aussage die Verwendung einer mathematisch nicht sauber definierten Rechnerarithmetik zu rechtfertigen. Natürlich kann man auch heute noch manche Rechnung durch mathematische Überlegungen stark vereinfachen. Bei der Leistungsfähigkeit der heute verfügbaren Rechnertechnologie und -hardware aber hätte der Mathematicorum Princeps C. F. Gauß sich sicherlich nicht 40 Jahre lang damit zufriedengegeben, den Computer nur als Rechenknecht (number cruncher) zu verwenden. Er hätte längst damit begonnen, den Rechner auch im Gleitkommabereich so weiterzuentwickeln, daß man auch viele, für eine Rechnung nützliche mathematische Überlegungen damit erledigen kann. Eine mathematisch sauber und präzise definierte Rechnerarithmetik ist eine Grundvoraussetzung dafür. Eine Entwicklung in diese Richtung ist längst überfällig.

Zum Schluß dieses einleitenden Abschnittes sei noch erwähnt, daß alles, was in den folgenden Abschnitten im Hinblick auf leistungsfähigere Arithmetik, entsprechende Hardwareprozessoren, anspruchsvolle Programmierumgebungen, Problemlöseroutinen mit automatischer Ergebnisverifikation, usw. andeutungsweise angesprochen wird, bereits existiert, in der Literatur ausführlich beschrieben ist und auch vorgeführt und benutzt werden kann, s. Literaturhinweise. Gute und schlechte Arithmetik sind bei der heutigen Technologie gleich teuer. Da sollte man eigentlich nur eine mathematisch sauber definierte und implementierte Arithmetik bereitstellen und verwenden.

2 Die Entwicklung der Rechnerhardware

Computer werden immer schneller. Schnelle Rechner und die darin verwendeten Prozessoren kommen heute fast ausschließlich aus den USA und aus Japan. Diese beiden Feststellungen mögen bereits Grund genug sein, einmal eine Bestandsaufnahme zu versuchen und einige Fragen zu beantworten: Warum ist dies so? Wo stehen wir? Wie

wird es vielleicht weitergehen? Gibt es hierzulande und/oder in Europa Beiträge zur Weiterentwicklung des Computers?

Die Vorgeschichte ist rasch erzählt. Nach heutiger Erkenntnis hat der Tübinger Professor Wilhelm Schickard um das Jahr 1623 die erste Vier-Spezies-Rechenmaschine überhaupt erfunden und entwickelt. Gottfried Wilhelm Leibniz gilt als der Entdecker des Dualsystems. 1672 hat er die Staffelwalze erfunden und damit einen Rechner gebaut. Auch das Sprossenrad, ein anderes mechanisches Schaltelement, geht auf ihn zurück. Der erste programmgesteuerte Rechner wurde 1935 von Konrad Zuse entworfen und 1937 auch gebaut. 1941 folgte die erste programmgesteuerte Rechenanlage der Welt, welche rein im Dualsystem arbeitete und bereits in Gleitkomma rechnete. In den Jahren 1945/46 hat Konrad Zuse mit dem Plankalkül die erste algorithmische Programmiersprache, welche bereits die bedingte Anweisung enthielt, entwickelt. Nach dem Kriege folgten hierzulande respektable Rechnerentwicklungen an deutschen Hochschulen und Forschungsinstituten [2], an der TH München, der TH Dresden und am Max-Planck-Institut in Göttingen. Der Funke ist dann auch auf die Industrie übergesprungen und es kam zu kommerziellen Rechnerentwicklungen bei SIEMENS, TELEFUNKEN, SEL und ZUSE. Auch in anderen europäischen Ländern (England, Frankreich, Holland, Italien usw.) gab es entsprechende Entwicklungen. Einer der ersten Vier-Bit Mikroprozessoren ist 1972 von der Firma OLYMPIA entwickelt worden.

Diese Tradition und Entwicklung ist in der Folgezeit abgerissen. Heute gibt es in Europa praktisch keine eigenständige und konkurrenzfähige Rechnerentwicklung. Es werden jährlich für einige zig Milliarden DM Rechner beschafft. Man beschränkt sich ausschließlich auf die Nutzung von Rechnern.

Wenn man fragt, wann die große alte Tradition abgerissen ist, dann kommt man ziemlich genau auf das Jahr 1970. Damals gab es noch einige europäische Rechner auf dem Weltmarkt, aber noch keinen japanischen Rechner. Um das Jahr 1970 wurde an einigen deutschen und europäischen Hochschulen die Informatik als Studiengang eingerichtet. Dies ist ein merkwürdiges, vielleicht aber doch nur äußerliches Zusammentreffen. Vielleicht hat man geglaubt, daß mit der Aufnahme einer systematischen Ausbildung auf diesem Gebiet die Rechner von selbst entstehen werden. Dies ist aber offenbar nicht eingetreten. Rechnerentwicklung und der dazugehörige technologische Fortschritt müssen wohl auch politisch gewollt sein. Hinter den entsprechenden Entwicklungen in den USA und in Japan stand jedenfalls zu allen Zeiten auch ein entsprechender politischer Druck und Wille.

Wo stehen wir und wie wird es vielleicht weitergehen? Früher, vor etwa 60 Jahren, galt die Erfahrungsregel, daß eine geübte Person in der Lage ist, mit einem elektromechanischen Tischrechner etwa 1000 Rechenoperationen an einem Arbeitstag einigermaßen zuverlässig auszuführen. Dies entspricht etwa der Leistung, die man heute mit einem einfachen elektronischen Taschenrechner erbringen würde. Die ersten elektronischen Rechner Mitte der 50er Jahre waren dann in der Lage, etwa $100 = 10^2$ Rechenoperationen in der Sekunde auszuführen. Dies entspricht grob einer Geschwindigkeitssteigerung um den Faktor $1000 = 10^3$. Schnelle Rechner sind heute in der Lage, 10^9 Rechenoperationen in der Sekunde auszuführen. Ein Vergleich der Zahlen 10^3 und 10^9 zeigt, daß sich die eigentliche Computerrevolution **nach** der Entwicklung der ersten elektronischen Rechenanlagen vollzogen hat.

Die Rechenleistung 10^9 Rechenoperationen in der Sekunde ist sicherlich schwer vorstellbar. Ein Vergleich ist daher vielleicht angebracht. Auf der Welt leben heute noch weniger als 10 Milliarden Menschen. Wenn man jeden Erdenbürger Rechenoperationen mit einem einfachen elektronischen Taschenrechner ausführen läßt, so erbringen diese alle zusammen in etwa die Rechenleistung eines der zur Zeit schnellsten Rechners. Dieser ist aber in der Lage auch noch in der Nacht und an den Wochenenden zu arbeiten.

Wenn man die Rechenleistung der jeweils schnellsten Rechner von 1955 bis heute linear interpoliert, so kommt man zu dem Ergebnis, daß die Rechenleistung des einzelnen Rechners alle 10 Jahre etwa um den Faktor 100 gesteigert werden konnte. Tatsächlich ist die Entwicklung auch in etwa so verlaufen.

Die jeweils schnellsten Rechner, die Supercomputer, sind und waren immer etwas für die Spezialisten. Den Normalbürger interessiert vielleicht mehr die Entwicklung bei den Mikroprozessoren, den Personal Computern und den Workstations. Es sollen daher auch hierzu summarisch einige Aussagen gemacht werden. Darin läßt sich vielleicht auch erkennen, wie die Entwicklung vielleicht weitergehen wird.

Die ersten Vier-Bit Mikroprozessoren sind 1971/72 erschienen (der bereits oben erwähnte Prozessor von OLYMPIA, welcher später als Fairchild-Prozessor auf den Markt gebracht wurde oder der 4004 von INTEL). Sie enthielten etwa 2000 Transistoren. Der Autor dieser Zeilen hat Anfang des Jahres 1973 einen dringenden Brief an den damaligen Forschungsminister gerichtet mit dem Hinweis, der Entwicklung der Mikroelektronik und der Mikroprozessoren besondere Aufmerksamkeit zu widmen und diese verstärkt zu fördern. In dem Brief wurde besonders auf die völlig problemlose Technologie und den großen volkswirtschaftlichen Nutzen hingewiesen, welcher sich aus einer solchen Entwicklung ergeben könnte. Als Ergebnis entsprechender Beratungen wurde festgestellt, daß es zur Förderung der Mikroelektronik noch zu früh sei. Man wollte die Forschung damals nach sozialen Gesichtspunkten ausrichten. Was das bedeutet, hat niemand gesagt.

Die Leistung und Technologie der Mikroprozessoren ist seither massiv vorangetrieben worden. Allein seit 1984 hat sich die Rechenleistung der Mikroprozessoren in jedem Jahr verdoppelt. Sie ist damit in zehn Jahren um den Faktor 1000 angestiegen. Im Vergleich mit den Supercomputern sind die Mikroprozessoren also dabei aufzuholen. Eine schnelle Workstation bringt heute die Leistung eines Supercomputers der frühen 80er Jahre. Diese enorme Leistungssteigerung wurde erreicht durch kürzere Weglängen und Steigerung der Gatterzahl auf dem Chip, durch Steigerung der Taktrate, aber auch durch Fortschritte in der Prozessorarchitektur [13]. All diese Dinge sind stark miteinander verwoben und lassen sich nicht strikt voneinander trennen.

Aus heutiger Sicht scheint die Entwicklung damit noch keineswegs am Ende zu sein. Natürlich wird intensiv auf dem Gebiet völlig anderer Technologien experimentiert und geforscht. Aber auch die heutige Silicium-Technologie scheint noch lange nicht erschöpft zu sein. Unter Berücksichtigung dessen, was in den Labors der führenden Firmen (AT&T, IBM, TOSHIBA, NEC, HITACHI) heute bereits funktioniert, kann man davon ausgehen, daß die Rechenleistung in den kommenden Jahren weiter in etwa 18 Monaten verdoppelt werden kann. Dies entspricht noch einmal einer Steigerung um den Faktor 1000 in den nächsten 15 Jahren.

Dem Autor sei als Mathematiker der Hinweis erlaubt, daß diese enormen Steigerungen der Rechenleistung nicht nur technologischer Art sind. Der Rechner kann heute nur noch unter Verwendung von Rechnern weiterentwickelt werden. Hier haben natürlich zu allen Zeiten auch die Mathematiker wesentliche Beiträge geleistet, bei der Verbesserung der Entwurfswerkzeuge, der verwendeten Algorithmen, der Verbesserung und Weiterentwicklung der Rechnerarchitektur usw.

Natürlich bemüht man sich darüber hinaus auch auf der Seite der Mathematik ständig um Erhöhung der Rechenleistung etwa durch Entwicklung besserer und schnellerer Algorithmen und es ist vielleicht interessant zu wissen, was auf dieser, von der Technologie völlig unabhägigen Ebene, an Geschwindigkeitssteigerungen in etwa erreicht werden konnte. Bild 6 gibt hierzu im Original eine Graphik aus einer Veröffentlichung des US Office of Science and Technology Policy wieder. Die Abbildung zeigt, daß in den 20 Jahren von 1970 bis 1990 die erreichten Leistungssteigerungen bei der Bearbeitung wissenschaftlicher Aufgaben durch Computer im Methoden- und Softwarebereich etwa von der gleichen Größenordnung waren, wie die bei Supercomputern in der Hardwareentwicklung begründeten Leistungssteigerungen. Fairerweise muß man dazu aber sagen, daß sich die Graphik nicht generell auf die Leistungssteigerung numerischer Algorithmen bezieht, sondern speziell auf die Behandlung großer linearer Gleichungssysteme, wie sie bei der Behandlung von partiellen Differentialgleichungen typischerweise auftreten. Es führen keineswegs alle numerischen Probleme auf derartige Gleichungssysteme. Bei der großen Klasse mittlerer Probleme hat man es in der Regel mit geringeren Leistungssteigerungen durch Algorithmen zu tun.

Die Leistungssteigerung durch die Hardware um den Faktor 1000 in nur zehn Jahren für **alle** Algorithmen bei den Mikroprozessoren, mit denen wir täglich umgehen, relativiert natürlich etwas die Aussage der oberen Graphik in Bild 6. Aber selbst eine Leistungssteigerung um den Faktor zehn bei einem speziellen Algorithmus durch mathematische Maßnahmen ist ja schon etwas, da sie immer noch dazu kommt und es macht natürlich einen Unterschied, ob ein Programm zehn Stunden oder nur eine Stunde läuft. Dennoch, eine projektierte Geschwindigkeitssteigerung bei den Mikroprozessoren um den Faktor eine Million in nur 25 Jahren allein durch die Hardware legt es nahe, auch in der Mathematik die Prioritäten einmal anders zu setzen, beispielsweise auf höheres Abstraktionsniveau bei den Programmiersprachen, eine komfortablere Arithmetik, Kontrolle der Rechengenauigkeit und Zuverlässigkeit von Ergebnissen durch den Computer, Existenz und Eindeutigkeitsnachweis berechneter Lösungen durch den Computer usw.

Wie eingangs bereits erwähnt, hat man sich in Europa bedauerlicherweise vollständig in die Rolle des Nutzers von Rechenanlagen, d. h. im Hinblick auf das wissenschaftliche Rechnen auf die Entwicklung mathematisch-numerischer Verfahren und Algorithmen und die Bearbeitung von Anwendungen zurückgezogen bzw. abdrängen lassen. Hierin liegt eine Schwierigkeit und eine Gefahr, welche nicht übersehen werden darf. Der Algorithmus wird nämlich in einer Programmiersprache formuliert. Diese wird von einem Compiler oder Interpreter übersetzt. Alles wird von einem komplizierten Betriebssystem organisiert und schließlich von der Hardware in einer speziellen Technologie ausgeführt. Der Entwickler guter und schneller Algorithmen muß stets das ganze Spektrum im Auge haben und nicht nur die mathematische Oberfläche. Wenn er

im Markt erfolgreich sein will, muß er sogar wissen, was sich bei der nächsten Prozessorgeneration an den unteren Schichten ändert. Wenn er dies erst in seine Überlegungen und Algorithmen einbezieht, wenn der Prozessor bereits im Markt ist, sind diese u. U. zwei bis drei Jahre zu spät ausgereift. Dann interessiert sich niemand mehr dafür, da die nächste Prozessorgeneration bereits ins Haus steht. Ein junger Wissenschaftler, welcher exzellente hardwarenahe Algorithmen entwickelt, merkt u. U. erst nach vielen Jahren, daß er damit keinen Erfolg hat, weil er der Entwicklung an der Spitze immer um zwei bis drei Jahre hinterher rennt.

Ganz sicherlich brauchen wir Supercomputer, um die Probleme von morgen lösen zu können. Es ist auch wichtig, daß man sich in Europa etwa an der Entwicklung großer Parallelrechner beteiligt. Hier zeigt sich aber eine besondere Schwierigkeit, welche daher rührt, daß es keine eigene Prozessorentwicklung mehr gibt. Es ist sicherlich problematisch, große Parallelrechner bauen zu wollen mit Prozessoren, welche man im Markt kaufen kann. Wenn man Rechnerarchitektur und zugehörige Programmentwicklung betreibt, wird es immer Probleme geben, wo man erkennt, daß man die wirklich optimale Lösung nur realisieren kann, wenn man sie, sei es auch nur geringfügig, prozessorseitig unterstützt. Eine Konkurrenzentwicklung läßt sich letztlich nur auf der Prozessorseite überholen. Mit Ziegelsteinen kann man bekanntlich viele schöne Häuser bauen, auch einen hohen Kirchturm. Bei einem Wolkenkratzer aber wird man nicht mit Ziegelsteinen, sondern mit anderen, individuell ausgewählten und gefertigten Materialien bauen. Wer an der Prozessorentwicklung nicht mehr teilnimmt und nicht einen starken, wirklich kooperationsfreudigen Partner zur Seite hat, der ist auch bei der Rechnerarchitektur und der Entwicklung großer Parallelsysteme in einer schwachen Position.

3 Die Arithmetik von Rechenanlagen

> *A disappointing feature is the failure of the numerical analysts to influence computer hardware and software in the way they should.*
>
> *Now there is a regrettable tendency for numerical analysts to opt out of any responsibility for the design of the arithmetic facilities and a failure to influence the more basic features of software.*
>
> *It is often said that the use of computers for scientific work represents a small part of the market and numerical analysts have resigned themselves to accepting facilities „designed" for other purposes and making the best of them.*
>
> J. H. Wilkinson: Turing Lecture 1970, J.ACM 18 (1971), 146

Zu Deutsch: „Ein enttäuschender Gesichtspunkt besteht darin, daß die Numeriker es unterlassen, die Entwicklung der Computerhardware und -software so zu beeinflussen, wie sie es eigentlich sollten.

Unter den Numerikern beobachtet man heute eine bedauerliche Neigung, sich jeglicher Verantwortung für die Realisierung der Arithmetik im Computer zu entziehen, sowie ein Versäumnis, die mehr grundlegenden Merkmale der Software zu beeinflussen.

Oft wird gesagt, daß die Verwendung des Computers zum wissenschaftlichen Rechnen nur einen kleinen Teil des Marktes ausmacht und die Numeriker sich damit abgefunden haben, einen Gegenstand zu benutzen und das Beste daraus zu machen, welcher eigentlich für andere Zwecke gebaut wurde."

Auf Europa bezogen gilt dieser Ausspruch von J. Wilkinson heute mehr denn je; denn hier hat man sich ausschließlich in die Rolle des Nutzers von Rechenanlagen zurückgezogen bzw. abdrängen lassen. Dies schränkt natürlich die Möglichkeit, die Computerarithmetik und die mehr grundlegenden Eigenschaften der Software, die Basissoftware, zu beeinflussen, wesentlich ein bzw. macht dies überhaupt unmöglich. Wo beschäftigen sich hierzulande noch Mathematiker mit Rechnerarithmetik, Programmiersprachen, Übersetzungstechniken und Basissoftware von Rechenanlagen und versuchen diese zu beeinflussen? Der Ausspruch von J. Wilkinson beinhaltet indirekt aber auch Kritik an der entstehenden Software. Könnte diese nicht den Anwendungen besser angepaßt sein, wenn man den mehr grundlegenden Fragen, auch der Entwicklung des Computers und der Basissoftware, stärker nachgehen würde?

Wir wollen auch hier eine kurze Bestandsaufnahme vornehmen und die Fragen diskutieren: Wo stehen wir? Wie könnte oder sollte es weitergehen? Nutzen wir die Möglichkeiten der zur Verfügung stehenden Technologie auch wirklich aus?

Von dem Mathematiker L. Kronecker (1821 bis 1891) stammt die Bemerkung: *„Die ganzen Zahlen hat der liebe Gott gemacht, alles andere ist Menschenwerk"*

Tatsächlich haben die ganzen Zahlen auch im Hinblick auf den Computer etwas vollkommenes an sich. Ihre Verknüpfungen lassen sich nämlich auf wunderbare Weise durch einfache Schaltglieder wie UND- und ODER-Gatter durch Hardware fehlerfrei realisieren. Die Verknüpfungen für die reellen Zahlen können demgegenüber im Rechner nur approximativ ausgeführt werden.

Elektronische Rechenanlagen stellen daher heute in der Regel zwei unterschiedliche Zahlensysteme in Hardware bereit: die ganzen Zahlen und die Gleitkommazahlen. Mit der ganzzahligen Hardwarearithmetik wird ein beschränkter Bereich der ganzen Zahlen erfaßt. Durch Software läßt sich dieser Bereich im Prinzip beliebig erweitern. Software verlangsamt allerdings den Rechenprozeß. Sofern die Hardware (und auch die Software) intakt ist, arbeiten die ganzzahlige Arithmetik und ihre Softwareerweiterungen fehlerfrei.

Mittels der ganzzahligen Arithmetik läßt sich beispielsweise eine rationale Arithmetik oder eine Langzahlarithmetik aufbauen. Dabei führt der Rechner immer soviele Stellen mit, wie notwendig sind, um das Ergebnis einer Operation korrekt darzustellen. In der Zahlentheorie oder der Computeralgebra verwendet man vorzugsweise diese Art von Arithmetik. Auch die meisten Anwendungen in der Informatik lassen sich mit der ganzzahligen Arithmetik fehlerfrei ausführen. Man denke etwa an Übersetzungsvorgänge, an Such- oder Sortieralgorithmen.

In der Numerik oder beim sogenannten wissenschaftlichen Rechnen hat man es hingegen mit reellen Zahlen zu tun. Eine reelle Zahl wird im allgemeinen durch einen unendlich langen Dezimal- oder Binärbruch dargestellt. In einer Rechenanlage müssen diese durch endliche Brüche approximiert werden. Üblicherweise geschieht dies durch sogenannte Gleitkommazahlen. Eine Gleitkommazahl besteht aus einem Vorzeichen,

einer Mantisse von etwa 16 Ziffern, welche kleiner als eins ist, und einem Exponententeil, welcher die Größenordnung der Zahl angibt, z. B. $-0,4751234567895382 * 10^4$. Die erste Zahl rechts vom Komma muß von 0 verschieden sein.

Die Gleitkommaarithmetik wird ebenfalls durch Hardware bereitgestellt und ist damit sehr schnell. Als Ergebnis der Verknüpfung zweier Gleitkommazahlen erhält man wieder eine Gleitkommazahl. Die einzelne Gleitkommaoperation ist heute, wie man sagt, maximalgenau. D. h. das Ergebnis jeder Verknüpfung von zwei Gleitkommazahlen unterscheidet sich vom korrekten Verknüpfungsergebnis um weniger als eine oder eine halbe Einheit in der letzten Stelle der Mantisse je nach Wahl der Rundungsfunktion. Soweit klingt alles ganz gut.

Aber schon nach zwei oder nur wenigen Gleitkommaoperationen kann das Ergebnis völlig falsch sein. Wir wollen versuchen, dies zu erläutern. Im allgemeinen werden die Exponenten von zu addierenden Zahlen nicht gleich sein. Dann müssen die Mantissen erst einmal stellenrichtig untereinandergeschoben werden, d. h. die Mantisse der Zahl mit dem kleineren Exponenten wird um einige Stellen nach rechts geschoben. Dann wird addiert und (auf unsere 16 Stellen) gerundet. Was dabei über die 16. Stelle rechts hinausragt, wird abgeschnitten und weggeworfen (s. Bild 7). Wenn nun weitere Additionen folgen mit der Eigenschaft, daß das Ergebnis einer langen Summe von der Größenordnung des abgeschnittenen Teiles von b ist, kann die Summe nicht mehr richtig berechnet werden, da der abgeschnittene Teil von b nach der ersten Addition nicht mehr zur Verfügung steht.

Was sich hier abspielt, ist schlichtweg eine Tragödie, und man muß sich wirklich fragen, warum man dies nicht schon längst behoben hat. Die Antwort lautet: Weil man sich inzwischen daran gewöhnt hat, daß das so ist, vielleicht sogar glaubt, daß dies so sein muß, und weil die Anwender keine Abhilfe fordern (vgl. dazu den obigen Ausspruch von J. H. Wilkinson). Man nimmt die Situation gewissermaßen als höhere Gewalt oder vermeintlich unabänderliches Übel in Kauf. Natürlich kann man versuchen, für das vorliegende Problem eine Fehleranalyse durchzuführen. Dabei wird der Fehler jeder einzelnen Gleitkommaoperation abgeschätzt. Man wird dabei vielleicht feststellen, daß ein ernsthafter Fehler begangen wurde und das Ergebnis nicht stimmen kann. Mit einem anderen Algorithmus läßt sich dieser Fehler dann vielleicht vermeiden. Das Problem besteht dabei nur darin, daß ein schneller Rechner heute eine Milliarde Gleitkommaoperationen in der Sekunde ausführt. So viel Fehleranalyse wie dies bräuchte, kann aber selbst die ganze Menschheit nicht durchführen. Es bleibt daher eigentlich gar nichts anderes übrig, als dem Ergebnis einer langen Gleitkommarechnung zu vertrauen oder es vielleicht mit heuristischen Methoden zu bestätigen oder plausibel zu machen.

Dabei ist Abhilfe relativ einfach möglich. Man braucht sich nur daran zu orientieren, wie die alten Rechner vor hundert Jahren das Problem gelöst haben. Ein solcher Rechner (s. Bild 8) war etwa mit einem Eingaberegister von neun Dezimalstellen ausgestattet. Das Ergebnisregister war wesentlich breiter und hatte vielleicht 25 Dezimalziffern. Es war ein sogenanntes Festkommaregister, d. h. während der Rechnung saß das Komma immer an der gleichen Stelle im Ergebnisregister. Jede Ziffer (Stelle) des Ergebnisregisters trug eine eindeutige Exponentenkennung (Einer, Zehner, Hunderter, ..., Zehntel, Hundertstel, ...). Das wesentlich breitere Ergebnisregister ermöglichte es, eine große Anzahl von Summanden immer richtig positioniert in das Ergebnisregister

zu addieren, ohne nach jeder Addition das Ergebnis gleich runden zu müssen. Es konnten sogar Produkte aufaddiert werden. Dabei wurden die Vielfachen des einen Faktors entsprechend den Ziffern des zweiten Faktors addiert. Man konnte so ohne weiteres tausend Summanden oder Produkte aufaddieren. Solange kein Unter- oder Überlauf an einem Ende des Ergebnisregisters auftrat, waren alle Ziffern des Ergebnisregisters korrekt. Eine Fehleranalyse war nicht notwendig. Diese Operation wurde damals schlicht als „Auflaufenlassen" bezeichnet. Sie wurde so oft wie irgend möglich angewendet; denn es war die schnellste Art, Berechnungen durchzuführen. Zwischenergebnisse brauchten nicht ausgelesen und für eine nachträgliche Operation wieder eingelesen zu werden. Gerundet wurde erst am Schluß der Akkumulation und zwar nur ein einziges Mal.

Bild 9 zeigt zwei andere mechanische Rechner, welche auf anderen Schaltprinzipien beruhen, aber ebenfalls mit der Technik des Auflaufenlassens betrieben werden können. Man erkennt wieder die im Vergleich zum Eingaberegister wesentlich breiteren Ergebnisregister.

Diese extrem nützliche, schnelle und immer korrekte Operation des Auflaufenlassens ist beim Übergang zu den ersten elektronischen Rechenanlagen offenbar verlorengegangen. Bei den in der Anfangszeit verwendeten Technologien wäre sie auch viel zu teuer gewesen. Bei den heute verfügbaren Technologien trifft dies nicht mehr zu. Die Forderung, die sich an dieser Stelle ziemlich zwangsläufig stellt, besteht nun darin, diese alte und bewährte Technik des Auflaufenlassens (Akkumulierens) als fünfte Operation zu den vier Gleitkommaverknüpfungen wieder hinzuzunehmen.

Nun wird argumentiert, dies sei doch viel zu kompliziert. Im Vergleich zu den alten Maschinen deckt der Gleitkommabereich heute einen riesigen Zahlenbereich ab, und so breite Register könne man doch gar nicht zur Verfügung stellen. Dies stimmt einfach nicht! Bei der klassischen /360- und /370-Architektur, welche sich fast 30 Jahre lang im Markt bewährt und diesen vielleicht sogar dominiert hat, wird der gesamte Gleitkommabereich bei Verwendung des Datenformates double precision (long) durch 568 **Bits**, wohlgemerkt **Bits**, abgedeckt. Die Mantisse besteht aus 14 hexadezimalen Ziffern, und man hat einen Exponentenbereich von -64 bis +64. Der ganze Gleitkommabereich besteht also aus 64 + 14 + 64 = 142 hexadezimalen Ziffern, das sind 568 Bits. Bei dem heute vielfach verwendeten Datenformat des sogenannten IEEE-Arithmetik-Standards 754 benötigt man wegen des dort verwendeten wesentlich größeren Exponentenbereiches etwa viermal so viele Bits.

Bei der heutigen Technologie ist es überhaupt kein Problem, ein solches Register (langes Datenwort) auf der Arithmetikeinheit zur Verfügung zu stellen. Auf solchen Maschinen laufen dann Akkumulationen und Akkumulationen von Produkten, bei denen kein Unter- und kein Überlauf auftritt, immer absolut fehlerfrei ab. Man braucht keine Fehleranalyse mehr auszuführen auch dann nicht, wenn eine Million Produkte aufsummiert werden. Diese Festkommaakkumulation als fünfte Gleitkommaoperation ist in den letzten 20 Jahren auf Institutsebene wiederholt in den verschiedensten Technologien realisiert worden, in Software, in Assembler, in Mikrocode, in diskreter Bit-Slice-Technik und in VLSI-Technologie. Sie war dabei immer schneller als eine auf der gleichen Technologiestufe ausgeführte Gleitkommaakkumulation. Dies ist auch durchaus verständlich; denn der Festkommaakkumulationsprozeß ist einfacher. Der ankommende Summand wird nur an die richtige Stelle geschoben und addiert.

Im Vergleich zur Gleitkommaakkumulation entfällt bei jeder Operation die Normalisierung, das Runden, das Packen zu einer Gleitkommazahl und das Entpacken für die nächste Operation. Man braucht auch nicht abzufragen, welcher von zwei Summanden geschoben werden muß (den kleineren Exponenten trägt). Bei Summen von Produkten entfällt das ständige Abspeichern der Zwischensumme, usw.

Es zeigt sich, daß in moderner Technologie bei einer Festkommaakkumulation von Produkten, d. h. bei der Berechnung von Skalarprodukten für die Ausführung der Arithmetik überhaupt keine Rechenzeit mehr benötigt wird. Die Produktbildung und die Akkumulation lassen sich in einer Pipeline (Fließbandverarbeitung) versetzt, in der Zeit ausführen, welche für das Beschaffen der Daten benötigt wird. Dies besagt noch einmal, daß es keine andere Art, Skalarprodukte zu berechnen, geben kann, welche schneller ist. Alles was zusätzlich benötigt wird, ist ein relativ kleiner, lokaler Speicher (ein langes Datenwort) auf der Arithmetikeinheit. Die Vorteile, die sich damit für die Numerik ergeben, sind immens.

Nach dieser informellen Beschreibung der Festkommaakkumulation von Gleitkommazahlen und -produkten, wollen wir die Fragestellung noch etwas formaler betrachten. Tatsächlich ist dies auch der Weg, auf dem diese fünfte Gleitkommaoperation wiederentdeckt wurde. Ausgangspunkt ist das Problem der Fehleranalyse numerischer Algorithmen oder die Frage nach der Zuverlässigkeit numerisch berechneter Ergebnisse. Mit zunehmender Rechenleistung der einzelnen Prozessoren wurde eine Fehleranalyse von Hand mehr und mehr unmöglich. Trotzdem ist man natürlich bei vielen Anwendungen dringend an der Größe des Fehlers berechneter Ergebnisse interessiert. Wir erinnern noch einmal an die einleitend erwähnte Berechnung der Eigenwerte einer schnellen schweren Turbine. Hier kann man nicht experimentieren. Die Eigenwerte müssen genau und sicher berechnet werden, sonst geht das Aggregat explosionsartig zu Bruch.

Nun sind Rechenanlagen einmal dazu erfunden worden, komplizierte Aufgaben dem Menschen abzunehmen. Die offensichtliche Diskrepanz zwischen Rechenleistung und Beherrschung des Rechenfehlers legt es nahe, zu versuchen, den Prozeß der Fehlerabschätzung selbst wieder dem Rechner zu übertragen. Ziel muß es sein, den Rechner so weiterzuentwickeln, daß man nicht nur Näherungsrechnung, sondern echte Mathematik damit betreiben kann, in dem Sinne, daß genaue Schranken für die Lösung von Problemen berechnet oder sogar Existenz- und Eindeutigkeitsaussagen mit dem Rechner nachgewiesen werden. Die Numerik sollte letztlich einmal den gleichen Ansprüchen genügen wie der Rest der Mathematik.

Grundlage der Mathematik ist die Arithmetik. Um das genannte Ziel zu erreichen, wird man versuchen, die arithmetische Basis des Rechners zu erweitern und außer den Verknüpfungen für reelle Zahlen auch kompliziertere mathematische Verknüpfungen im Rechner maximal genau direkt zur Verfügung zu stellen oder sogar in Schranken einzuschließen, anstelle sie über die vier Gleitkommaverknüpfungen zu approximieren. Die in Frage kommenden Verknüpfungen sind schnell aufgezählt. Neben den reellen Zahlen bilden die komplexen Zahlen die Grundlage der Analysis. Will man berechnete Ergebnisse garantieren, so braucht man zudem auch die Intervalle über den reellen und komplexen Zahlen. Die Intervalle bringen das Kontinuum auf den Rechner. Ein Intervall mit Gleitkommazahlen als Schranken beschreibt das Kontinuum der reellen Zahlen zwischen diesen Schranken. Anstelle nur eines haben wir damit bereits vier Grundda-

tentypen. Über jedem dieser vier Datentypen werden jetzt noch Vektoren und Matrizen aufgebaut. Man erhält so insgesamt zwölf Räume (Mengen mathematischer Objekte). In der Theorie der Rechnerarithmetik [8], [9] wird nun gezeigt, daß es vernünftig ist, alle Verknüpfungen in diesen zwölf Räumen nach dem Prinzip des Semimorphismus bereitzustellen. Wir wollen dieses Prinzip hier kurz angeben. Der mathematisch weniger geübte Leser kann diese Formeln einfach überlesen.

Ist M einer unserer zwölf Räume und N die Teilmenge der im Rechner darstellbaren Elemente, so erklären wir für jede Verknüpfung $\circ$ in M eine Approximation $\boxdot$ in N nach der Vorschrift

$$(RG) \qquad a \boxdot b := \Box(a \circ b) \qquad \text{für alle } a, b \in N \text{ und alle Verknüpfungen } \circ \text{ in } M.$$

Dabei ist $\Box : M \to N$ eine Abbildung von M nach N, welche als Rundung bezeichnet wird, wenn sie die beiden folgenden Eigenschaften besitzt.

$$(R1) \qquad \Box a = a \qquad \text{für alle } a \in N \qquad \text{(Projektion)}$$

$$(R2) \qquad a \leq b \Rightarrow \Box a \leq \Box b \qquad \text{für alle } a, b \in M \qquad \text{(Monotonie)}$$

Beim Semimorphismus verlangt man ferner noch, daß die Rundung auch antisymmetrisch ist, d. h. daß gilt

$$(R3) \qquad \Box(-a) = -\Box a \qquad \text{für alle } a \in M \qquad \text{(Antisymmetrie)}$$

Weitere wichtige Rundungen sind die nach unten bzw. oben gerichtete Rundung mit den Eigenschaften

$$(R4) \qquad \nabla a \leq a \quad \text{für alle } a \in M \text{ bzw. } a \leq \triangle a \quad \text{für alle } a \in M \qquad \text{(gerichtet)}$$

Die gerichteten Rundungen sind durch $(R1), (R2)$ und $(R4)$ eindeutig bestimmt [8], [9].

Mit diesen fünf Formeln (RG) und $(R1, 2, 3, 4)$ werden in den auf dem Rechner darstellbaren Teilmengen unserer zwölf Räume eine Vielzahl von Verknüpfungen definiert. Im Unterschied zur herkömmlichen Approximation dieser Verknüpfungen durch Gleitkommaoperationen sind die neuen, mittels Semimorphismus definierten Verknüpfungen alle maximal genau, d. h. zwischen dem korrekten und dem berechneten Verknüpfungsergebnis liegt kein weiteres Element aus dem betreffenden Raster. Dies ist leicht einzusehen; denn ist für zwei Elemente $a, b \in N \quad \alpha \in N$ die größte untere und $\beta \in N$ die kleinste obere Schranke des korrekten Verknüpfungsergebnisses $a \circ b$ in M, d. h.

$$\alpha \leq a \circ b \leq \beta,$$

so folgt

$$\Box\alpha \overset{(R1)}{=} \alpha \overset{(R2)}{\leq} \Box(a \circ b) \overset{(RG)}{=} a \boxdot b \overset{(R2)}{\leq} \Box\beta \overset{(R1)}{=} \beta.$$

Es liegt also auch das Verknüpfungsergebnis $a \boxdot b$ in N zwischen α und β.

In der Rechnerarithmetik wird nun gezeigt, daß man alle Verknüpfungen in den oben genannten zwölf Räumen (reelle und komplexe Zahlen, reelle und komplexe Intervalle sowie Vektoren und Matrizen über diesen vier Grundtypen) auf der Ebene

von Programmiersprache und Compiler mittels einer modularen Technik bereitstellen kann, wenn auf niedriger Ebene, möglichst in Hardware, 15 Grundoperationen bereitstehen. Dabei handelt es sich um die fünf Verknüpfungen $+, -, *, /, \cdot$ jeweils mit den drei Rundungen $\Box, \nabla, \Delta$. Dabei bezeichnet $\cdot$ das Skalarprodukt zweier Vektoren (beliebiger aber endlicher Dimension), $\Box$ eine antisymmetrische Rundung und ∇ und Δ die beiden gerichteten Rundungen von den reellen Zahlen in die Gleitkommazahlen. Alle 15 Verknüpfungen $\boxed{\circ}$, $\overset{\nabla}{\circ}$, $\overset{\Delta}{\circ}$ mit $\circ \in \{+, -, *, /, \cdot\}$ sind nach (RG) definiert. Im Falle des Skalarproduktes sind a und b Vektoren.

Die Gesellschaft für Angewandte Mathematik und Mechanik (GAMM) und die International Association for Mathematics and Computers in Simulation (IMACS) haben im Jahre 1993 ein „Proposal for Accurate Floating-Point Vector Arithmetic“ [13] verabschiedet, welches von künftigen Rechnern genau diese Verknüpfungen fordert. Die herkömmliche Numerik verwendet davon nur die vier Operationen $\boxed{\circ}$, $\circ \in \{+, -, *, /\}$. Der oben bereits erwähnte IEEE-Arithmetik-Standard 754 fordert zwölf dieser 15 Operationen ($\boxed{\circ}$, $\overset{\nabla}{\circ}$, $\overset{\Delta}{\circ}$ für $\circ \in \{+, -, *, /\}$). Sie stehen heute praktisch auf allen Personal Computern und Workstations zur Verfügung. Die Verknüpfungen mit den Rundungen ∇ und Δ können über die verfügbaren Programmiersprachen jedoch in der Regel nicht angesprochen werden.

Zur Erzielung hoher Rechengenauigkeit sowie für das Rechnen mit Vektoren und Matrizen werden darüberhinaus insbesondere die drei Skalarprodukte $\boxed{\cdot}$, $\overset{\nabla}{\cdot}$ und $\overset{\Delta}{\cdot}$ benötigt. Dies sind Summen von Produkten von Gleitkommazahlen. Nach (RG) müssen diese Summen zunächst völlig korrekt im Raum der reellen Zahlen ausgeführt werden. Anschließend werden sie nur ein einziges Mal gerundet. Stellt man, wie bereits angesprochen, auf der Arithmetikeinheit ein langes Festkommaregister zur Verfügung, welches den ganzen Gleitkommabereich abdeckt, so lassen sich alle 15 Verknüpfungen nach (RG) korrekt ausführen. Damit lassen sich alle Verknüpfungen in den oben beschriebenen zwölf Räumen nach der Formel (RG) modular aufbauen. In numerischen Rechnungen werden dadurch sehr viele unnötige Rundungen ein für alle Male beseitigt.

Eigentlich sollte die hier skizzierte, allein auf mathematischen Prinzipien beruhende Definition der arithmetischen Verknüpfungen in Rechenanlagen Bestandteil jeder Programmiersprache sein. Der Benutzer einer Rechenanlage muß doch genau wissen, was passiert, wenn er in einem Programm eine arithmetische Operation aufruft. Nur so ist eine systematische Kontrolle des Rechenprozesses überhaupt möglich.

Mit den hier skizzierten Bausteinen für eine universelle Rechnerarithmetik läßt sich sehr einfach eine Arithmetik für mehrfache (doppelte, dreifache, vierfache, usw.) Genauigkeit auch für Intervalle aufbauen. Damit lassen sich instabile, kritische Stellen in Algorithmen häufig sehr einfach aufspüren, analysieren und kontrollieren. Eine Variable vierfacher Genauigkeit ist beispielsweise einfach ein Feld von vier Gleitkommazahlen einfacher Genauigkeit, deren Mantissen hintereinanderliegen (s. Bild 10). Variable eines solchen Typs lassen sich in dem langen Datenwort auf der Arithmetikeinheit addieren, subtrahieren und auch akkumulieren. Über das optimale Skalarprodukt läßt sich auch ihre Multiplikation einfach ausführen. Es ist nämlich

$$(a_1 + a_2 + a_3 + a_4)(b_1 + b_2 + b_3 + b_4) = a_1b_1 + a_1b_2 + a_1b_3 + a_1b_4 + a_2b_1 + \cdots + a_4b_3 + a_4b_4,$$

nichts anderes als ein Skalarprodukt. Die Division solcher Variablen erfolgt mittels eines

einfachen Iterationsprozesses.

Die bereits des öfteren zitierte allgemeine Theorie der Rechnerarithmetik gibt es seit 1972. In Form eines Vorlesungsmanuskriptes wurde sie damals an mehrere Fachkollegen weitergegeben. In Buchform ist sie im Jahre 1976 zunächst in deutscher Sprache beim Bibliographischen Institut erschienen [8]. Eine englischsprachige Ausgabe gibt es seit 1978 als IBM Research Report und als Buch bei Academic Press seit 1981 [9]. Die Realisierbarkeit in verschiedenen Technologien wurde mehrmals nachgewiesen, zuletzt 1994 durch Bau eines VLSI-Koprozessorchips, welcher über den PCI-Bus an Personal Computer angeschlossen werden kann [16].

Dieser Prozessor (s. Bild 11) erfüllt erstmals den von der GAMM und der IMACS vorgeschlagenen Standard für verifizierte, hochgenaue Vektorarithmetik in vollem Umfang. Im Computer wirkt er wie ein Katalysator in doppeltem Sinn. Er beschleunigt die Rechnung und eliminiert viele unnötige Rundungsfehler. Skalarprodukte berechnet er immer mit voller Genauigkeit oder im Bedarfsfall mit nur einer einzigen Rundung. Der Chip kann über Sprachen wie PASCAL-XSC [5] und C-XSC [6] direkt angesprochen werden. Er beschleunigt die Softwareimplementierungen der betreffenden Operationen in diesen Sprachen um den Faktor 100.

Der gegenwärtige Prozessor ist an die Geschwindigkeit des PCs angepaßt, bei dem die Daten in 32 Bit-Portionen übertragen werden. Wünschenswert wäre der Bau eines noch schnelleren Vektorarithmetik-Koprozessors für die Klasse der 100 MHz Workstations, bei denen die Daten in 64 Bit-Portionen übertragen werden. Ein solcher Chip könnte der Welt des wissenschaftlichen Rechnens wesentliche Impulse verleihen.

In den achtziger Jahren hat die Firma IBM viele dieser Ideen aufgegriffen und auf ihren Rechnern der /370-Architektur implementiert und zur Verfügung gestellt [14], [15]. Im Bereich des wissenschaftlichen Rechnens werden diese Rechner heute aber kaum noch verwendet. Sie sind inzwischen von moderneren, schnellen Workstations abgelöst worden, auf denen entsprechende Hardwareunterstützung der Arithmetik nicht vorhanden ist.

Allgemein muß man feststellen, daß sich diese neue, umfassende Rechnerarithmetik auf kommerziellen Rechenanlagen trotz gelegentlicher Ansätze noch nicht endgültig durchgesetzt hat. Die Gründe hierfür sind vielfältig. Einige davon sollen genannt werden.

Ein wesentlicher Grund besteht sicherlich darin, daß in Europa praktisch keine Rechner mehr gebaut werden. Ein Schwerpunkt der skizzierten Entwicklung lag hier, und es ist sicher nicht einfach, eine Rechnerentwicklung in den USA oder in Japan von Europa aus zu beeinflussen, zumal es für die Umsetzung der theoretischen Ideen in die Praxis eine Anzahl gut geschulter und ausgebildeter Mitarbeiter bedarf, welche in Übersee nicht von selbst entstehen. NIH (Not Invented Here) heißt ein Schlagwort, mit dem man sich solche Wünsche fernhält, zumal der erstmalige Entwurf einer wesentlich leistungsfähigeren Arithmetikeinheit natürlich auch etwas kostet. Die Entwicklung und der Bau eines VLSI-Vektorarithmetik Koprozessors für schnelle Workstations wäre sicherlich eine Möglichkeit, auf einem wichtigen Teilgebiet der elektronischen Datenverarbeitung Aktivität, Einfluß und Aufmerksamkeit wieder nach Europa zu ziehen. Vielleicht findet sich irgendwo die Bereitschaft, so etwas zu finanzieren.

Ein anderer Grund besteht sicher darin, daß Fachkollegen die neue Arithmetik noch

nicht vehement genug fordern. Ein netter Artikel in einer wissenschaftlichen Zeitschrift hilft da wenig. Wirklichen Einfluß könnte man nur über das Geld bei der Beschaffung teurer Anlagen ausüben. Der Rechner ist in Form einer leistungsfähigen Workstation, eines Vektorrechners oder Supercomputers oder einer massiv parallel arbeitenden Anlage auch mit herkömmlicher Gleitkommaarithmetik bereits ein extrem mächtiges und auch erfolgreiches Werkzeug, und der Umgang mit diesem Werkzeug prägt auch das Denken und sogar den Menschen. Dadurch entsteht eine Hemmschwelle, die man nicht gerne übertritt. „Der Schuster bleibt bei seinem Leisten". Ein auf der anderen Seite befindliches, noch mächtigeres Werkzeug, welches man nicht kennt, kann man natürlich zunächst auch nicht souverän oder gar virtuos nutzen. Die Hemmschwelle wird dadurch noch zusätzlich vergrößert. Wer den Umgang mit dem schwierigen Werkzeug Computer erst einmal gelernt und seinen Arbeitsstil gefunden hat, verhält sich naturgemäß konservativ und nimmt neuem gegenüber eine Abwehrhaltung ein. Bereits im Jahre 1789 beklagt unser Dichter Friedrich Schiller in seiner Antrittsvorlesung als Professor für Geschichte an der Universität Jena den Typ des Brotgelehrten: *„Jede wichtige Neuerung seiner Wissenschaft schreckt ihn auf, denn sie zerbricht die alte Schulform, die er sich so mühsam zu eigen machte, ...".*

In diesem Sinne ist der Übergang zu einem mathematisch-arithmetisch wesentlich leistungsfähigeren Rechner vielleicht sogar ein Generationenproblem. Es müssen erst genügend junge Wissenschaftler ausgebildet werden, welche das Neue genauso selbstverständlich und souverän beherrschen wie die alten das Alte. Auch der für die gesamte Mathematik so fundamentale Begriff des linearen Raumes hat seit seiner Entdeckung durch H. Grassmann (1844) über S. Banach (1922) bis zu dem berühmten Buch „Moderne Algebra" von B. L. van der Waerden (1931) fast hundert Jahre gebraucht, bis er sich endgültig durchgesetzt hat. Auch die für viele Anwendungen so überaus nützliche Matrizenrechnung hat ein ähnliches Schicksal erfahren. Seit den Anfängen bei Cayley (1858) ist ein Dreivierteljahrhundert vergangen, bis die enorme Leistungsfähigkeit dieses Kalküls allgemein anerkannt und akzeptiert war. Im Vorwort zur zweiten Auflage seiner „Ausdehnungslehre" (1. Auflage 1844) schreibt H. Grassmann: *„Das Werk ... hat in den ersten 23 Jahren nach seinem Erscheinen nur eine geringe und meist nur gelegentliche Beachtung gefunden. ... Ich konnte die Ursache davon nur in der streng wissenschaftlichen, auf die ursprünglichen Begriffe zurückgehenden Behandlungsweise finden. Eine solche Behandlungsweise erforderte aber ein nicht bloß gelegentliches Auffassen dieser oder jener Resultate, sondern ein sich Versenken in die zugrunde liegenden Ideen und eine zusammenhängende Auffassung des ganzen auf dies Fundament aufgeführten Baues, dessen einzelne Teile erst durch das Überschauen des Ganzen ihr volles Verständnis erhalten konnten. ...".*[7], S.11.

4 Intervallrechnung, ein einfacher, sehr mächtiger Kalkül zum Rechnen mit Ungleichungen

In der Analysis hat man es mit dreierlei Arten von Strukturen zu tun, der algebraischen Struktur, der Ordnungsstruktur und der topologischen Struktur. Beim Übergang auf den Rechner unterliegen die erste und die letzte starken Veränderungen, während für

das Rechnen mit Ungleichungen, die Ordnungsstruktur, bzgl. $\leq$ vermöge von (R2) die gleichen Gesetzmäßigkeiten gelten wie in den entsprechenden Grundräumen [8], [9]. Praktisch bedeutet dies, daß Eigenschaften eines Verfahrens oder Algorithmus, welche allein unter Verwendung der Ordnungsstruktur hergeleitet werden, auch auf dem Rechner streng erfüllt sind. In diesem Sinne kommt der Ordnungsstruktur und dem Rechnen mit Ungleichungen eine ganz besondere Bedeutung zu. In den letzten Jahrzehnten ist mit der Intervallrechnung ein sehr mächtiger Kalkül entwickelt worden, welcher das Rechnen mit Ungleichungen systematisiert.

Betrachten wir die beiden Ungleichungen $a_1 \leq a \leq a_2$ und $b_1 \leq b \leq b_2$. Für die Summe und die Differenz gilt dann $a_1 + b_1 \leq a + b \leq a_2 + b_2$ und $a_1 - b_2 \leq a - b \leq a_2 - b_1$. Im Falle der Multiplikation ist die Regel nicht so einfach. Man muß neun Fälle unterscheiden, je nachdem, ob a_1, a_2, b_1, b_2 kleiner oder größer als Null sind. Im Falle der Division gilt entsprechendes [8], [9], [1]. Aufgrund dieser vielen Fallunterscheidungen gestaltet sich ein explizites Rechnen mit Ungleichungen von Hand sehr schwierig. Bei komplizierten Ausdrücken ist es praktisch nicht mehr ausführbar.

Die Intervallrechnung faßt diese komplizierten Regeln für das Rechnen mit Ungleichungen zusammen. Mit A = $[a_1,\ a_2]$ und B = $[b_1,\ b_2]$ gilt für alle Verknüpfungen $\circ \in \{+, -, *, /\}$, ($0 \notin B$ im Falle der Division):

$$A \circ B = \{a \circ b \mid a \in A \wedge b \in B\} = [\min_{i,j=1,2}(a_i \circ b_j),\ \max_{i,j=1,2}(a_i \circ b_j)].$$

Bei Ausführung auf dem Rechner wird die untere Schranke des Ergebnisses nach unten und die obere nach oben gerundet. Die vielen Fallunterscheidungen werden ein für allemal ausprogrammiert und verschwinden bei entsprechend mächtigen Programmiersprachen fortan im Laufzeitsystem des Compilers. Der Benutzer braucht sich darum nicht mehr zu kümmern.

Die Leistungsfähigkeit dieses Kalküls soll an einem einfachen Beispiel illustriert werden. Wir betrachten ein lineares Gleichungssystem in Fixpunktform $x = A \cdot x + b$ mit einer kontrahierenden Matrix A. Der Intervallvektor X sei eine grobe Anfangseinschließung der Lösung $x^* \in X$. Man setzt nun X auf der rechten Seite des Gleichungssystems ein und denkt sich damit formal das Gesamtschrittverfahren, das Einzelschrittverfahren und das Relaxationsverfahren intervallmäßig aufgeschrieben. Man kann dabei auch noch den Durchschnitt zweier aufeinander folgender Näherungen bilden. Zerlegt man nun die so entstehenden Rechenvorschriften in Formeln für die Schranken, so erhält man u. a. eine Anzahl von Verfahren zur Berechnung von Schranken für die Lösung linearer Gleichungssysteme, welche vor etwa 50 Jahren von bekannten Mathematikern in mühsamer Kleinarbeit explizit hergeleitet wurden [1]. Der Kalkül der Intervallrechnung reproduziert diese und andere Verfahren auf einfachste Weise. Um die vielen bei der Matrixvektormultiplikation notwendigen Fallunterscheidungen braucht sich der Benutzer nicht mehr zu kümmern. Sie werden vom Rechner anhand des vorprogrammierten Kalküls vollautomatisch durchgeführt und es werden sogar die Rundungsfehler mit erfaßt. Der Kalkül entwickelt eine eigene Dynamik!

Worin bestehen andere Vorteile des Kalküls der Intervallrechnung? Aufgrund der Herleitung über das Rechnen mit Ungleichungen ist sofort klar, daß man damit Einschließungen des Wertebereiches von arithmetischen Ausdrücken und Funktionen in

Abhängigkeit von darin auftretenden Intervalldaten berechnen kann. Bei naiver Betrachtungsweise fallen diese Schranken für den Wertebereich allerdings häufig sehr groß und ungenau aus. Man spricht von einer Überschätzung des Wertebereiches. Dies hat der Intervallrechnung anfangs ein negatives Image eingebracht. Häufig hört man das Argument „da kann doch als Ergebnis nur ein Intervall von minus unendlich bis plus unendlich herauskommen".

In der Intervallrechnung wird nun aber gezeigt, daß diese Überschätzung des Wertebereiches bei stetigen Funktionen mit dem Durchmesser der Argumentintervalle gegen null konvergiert, d. h. die intervallmäßige Auswertung eines Ausdruckes geht gegen den Wertebereich, in der Grenze gegen den Wert der Funktion. Bei spezieller Darstellung des Ausdruckes oder der Funktion durch die sogenannte zentrierte Form [1] erfolgt diese Konvergenz sogar quadratisch. Die zentrierte Form läßt sich (bei differenzierbaren Funktionen) mittels der automatischen Differentiation wiederum vollautomatisch mit dem Rechner herstellen. Das heißt mit anderen Worten: Bei kleinen Intervallargumenten erhält man durch die intervallmäßige Auswertung ein quadratisch konvergentes Verfahren zur Berechnung des Wertebereiches von Funktionen und Ausdrücken und, wenn man die automatische Differentiation hier gleich mit einbezieht, auch zur Berechnung von Einschließungen des Wertebereiches von Ableitungen, Taylorkoeffizienten, Gradienten, Jacobimatrizen, Hessematrizen usw. Diese Größen werden vom Rechner wiederum vollautomatisch ohne weiteres Zutun des Benutzers aus dem arithmetischen Ausdruck heraus ermittelt. Die eigentliche Intelligenz wird dabei in das Laufzeitsystem des Compilers verlagert. Insbesondere durch das Zusammengehen mit der automatischen Differentiation erschließt sich der Intervallrechnung ein riesiges Feld von Anwendungen (Einschließung linearer und nichtlinearer Gleichungssysteme, verifizierte adaptive numerische Quadratur durch Einschließung des Restgliedes der Quadraturformel, verifizierte Integration gewöhnlicher Differentialgleichungen und von Integralgleichungen usw.).

Nach dem soeben Gesagten bieten sich Unterteilungsverfahren in der Intervallrechnung direkt an. Dies ist keine besondere Einschränkung, da in der Numerik viele Verfahren ohnehin in kleinen Schritten ablaufen. Wir illustrieren das Gesagte durch einige Beispiele:

Wir nehmen an, es soll die Frage untersucht werden, ob eine etwa durch einen arithmetischen Ausdruck gegebene reelle Funktion keine Nullstelle in einem vorgegebenen Intervall X besitzt. Hat man nur Gleitkommaarithmetik zur Verfügung, so läßt sich diese Frage mit mathematischer Sicherheit nicht beantworten. Man kann die Funktion in dem Intervall X etwa 1000 mal auswerten. Falls alle Funktionswerte positiv ausfallen, wird die Funktion mit hoher Wahrscheinlichkeit keine Nullstelle in X besitzen. Sicher aber ist diese Antwort keineswegs; denn es könnte ja aufgrund von Rundungsfehlern für einen negativen Funktionswert ein positives Ergebnis berechnet worden sein. Die Funktion könnte aber auch einen Durchhänger ins Negative besitzen, welcher durch die Wahl der Auswertestellen nicht erfaßt wurde (s. dazu Bild 12). Eine einmalige intervallmäßige Auswertung der Funktion kann hingegen ausreichen, um die gestellte Aufgabe mathematisch völlig einwandfrei, streng und sicher zu beantworten. Enthält nämlich das berechnete Ergebnisintervall nicht den Wert Null, so kann auch der Wertebereich die Null nicht enthalten; denn das berechnete Ergebnisintervall ist ja eine

Obermenge des Wertebereiches. Die Funktion besitzt folglich keine Nullstelle in dem Intervall X. Dieses mit wesentlich weniger Rechenoperationen berechnete Ergebnis ist so sicher wie ein (mit Hilfe des Rechners) bewiesener mathematischer Satz. Wenn man die Null nicht beim ersten Versuch ausschließen kann, unterteilt man einfach das Intervall in einige Teilintervalle.

Als zweites Beispiel betrachten wir die Aufgabe, das globale Minimum einer Funktion in einem vorgegebenen Bereich zu bestimmen. Wir beschränken uns wieder auf den eindimensionalen Fall. Das zu untersuchende Intervall wird in Teilintervalle zerlegt. In jedem dieser Teilintervalle wird nun die Funktion intervallmäßig ausgewertet. Dies liefert eine Obermenge des Wertebereiches in jedem Teilintervall. Man greift nun etwa dasjenige Teilintervall mit der kleinsten unteren Schranke für den Wertebereich heraus und berechnet in einem inneren Punkt dieses Intervalles den Funktionswert intervallmäßig. Dies liefert eine garantierte obere Schranke für einen Wert der Funktion. Diejenigen Teilintervalle, deren untere Schranke für den Wertebereich größer ist als dieser Wert, können mit Sicherheit das absolute Minimum nicht enthalten (s. dazu Bild 13). Sie brauchen nicht weiter betrachtet zu werden. Die verbleibenden Intervalle werden nach dem gleichen Prinzip weiter unterteilt. Das Verfahren funktioniert auch bei höheren Dimensionen sehr gut, da ganze Bereiche sehr rasch ausgeschlossen werden können. In [3], [4] findet man Verfeinerungen dieser Techniken. Häufig findet man sichere Schranken für das absolute Minimum damit schneller als mit herkömmlichen Methoden eine Näherung, deren Qualität noch unbekannt ist.

Als nächstes Beispiel betrachten wir die adaptive numerische Quadratur. Wir nehmen an, es soll das bestimmte Integral über eine bestimmte Funktion zwischen zwei Grenzen a und b berechnet werden. Die übliche Vorgehensweise besteht darin, daß man das Intervall $[a, b]$ in n Teile unterteilt. Das Integral wird dann dargestellt als Summe aus einer Näherungsformel plus einem Restglied. Die Näherungsformel ist ein Skalarprodukt aus gewissen Gewichten und den Funktionswerten an den Teilpunkten x_i. Bei hinreichend glattem Integranden läßt sich in der Regel das Restglied in einer Form darstellen, welche eine höhere Ableitung der Funktion an einer oder mehreren unbekannten Zwischenstellen im Intervall $[a, b]$ enthält. Die im Restglied auftretenden unbekannten Zwischenstellen werden nun einfach durch das ganze Intervall ersetzt in dem sie liegen können. Die höhere Ableitung wird durch automatische Differentiation über diesem ganzen Intervall ausgewertet. Dadurch werden sowohl die unbekannten Zwischenstellen als auch der Wert der höheren Ableitung an diesen Stellen mit Sicherheit eingeschlossen. Damit erhält man Schranken für das Restglied, welche vermöge der sonst darin auftretenden Größen wie $h^n/n!$ in der Regel sehr klein sind. Insgesamt hat man dadurch stets sichere Schranken für den durch die Näherunsformel begangenen Fehler [10].

Diese Information kann ausgenutzt werden, um die Schrittweite adaptiv an die Eigenschaften des Integranden anzupassen. Bei flachem Funktionsverlauf möchte man natürlich mit relativ großer Schrittweite integrieren, während bei steilem Verlauf der Funktion die Schrittweite kleiner gewählt werden muß.

Da man immer den Fehlerterm selbst auswertet bzw. zur Schrittweitensteuerung heranzieht, eignet sich die Vorgehensweise auch für die Gauß-Quadratur. Die sonst übliche Verwendung einer weiteren Näherungsformel, um den Fehler zu schätzen, wird

dadurch überflüssig. Eine verifizierende adaptive numerische Quadratur ist daher unter Umständen schneller als eine nur auf Näherungsrechnung basierende Vorgehensweise [10]. Entsprechendes gilt ja auch für die beiden anderen oben aufgeführten Beispiele.

Auch die verifizierte numerische Integration von Anfangswertproblemen bei gewöhnlichen Differentialgleichungen macht von der Restgliedeinschließung Gebrauch. Hierzu sind jedoch noch etwas genauere Betrachtungen notwendig, da das Restglied in diesem Fall auch die unbekannte Funktion enthält. Man muß daher zusätzlich noch Fixpunktsätze mit heranziehen.

Wir haben die Intervallrechnung als einen sehr mächtigen Kalkül für das Rechnen mit Ungleichungen erkannt. Die damit gewonnenen Aussagen bleiben bei der Übertragung auf den Rechner in der Regel gültig. Wichtig ist dabei, daß man von Anfang an mit Ungleichungen arbeitet und nicht etwa erst eine Norm zieht und diese dann mit Ungleichungen abschätzt. In einem solchen Fall hat man die strenge Gültigkeit auf dem Rechner bereits verlassen.

Jeder Mathematiker weiß, daß es von den Rechenregeln für komplexe Zahlen bis zu den wunderschönen Ergebnissen der Funktionentheorie ein weiter Weg ist, den man erst einmal zurücklegen muß, bevor man Funktionentheorie erfolgreich anwenden kann. Mit der Intervallrechnung ist es nicht anders. Auch ihre Eigenheiten muß man zunächst mühsam studieren, um sie erfolgreich anwenden zu können. Dann kann man allerdings viel damit erreichen. Naives Anwenden der Grundverknüpfungen hingegen führt in der Regel zu Mißerfolg.

Die Intervallrechnung war viele Jahre lang dazu verurteilt, auf der Stelle zu treten, da eine angemessene Programmierbarkeit nicht vorhanden war. Wer sie verwenden wollte, mußte erst einmal eine größere Manpower investieren, um die nötigen Grundroutinen und Programmierumgebungen zu schaffen. Aus Gründen der Rechengeschwindigkeit mußte vieles in Assembler codiert werden. Mit dem nächsten Rechner war dann alles wieder verloren und die Arbeit ging wieder von vorne los. Nur an ganz wenigen Stellen konnten leistungsfähige Programmierumgebungen geschaffen werden, mit denen dann auch die wesentlichen Fortschritte erzielt wurden. Beispielsweise ist die automatische Differentiation aus dem Bedürfnis heraus entstanden, Wertebereiche höherer Ableitungen einfach und schnell einzuschließen, da diese für die verifizierte Berechnung von Lösungen gewöhnlicher Differentialgleichungen oder auch bestimmter Integrale benötigt werden. Ohne geeignete Programmierumgebungen wären derartige Techniken nicht entstanden, und sie sind auch ohne solche nicht beherrschbar.

Diese geschilderten Schwierigkeiten haben aber auch dazu beigetragen, den Blick dafür zu schärfen, daß Numerik nicht nur aus Algorithmen und FORTRAN-Programmen besteht, sondern daß auch der ganze Unterbau, die Arithmetik, die Programmiersprache, der Compiler, das Laufzeitsystem, das Betriebssystem, die Architektur, die Hardware, die Technologie und unter Umständen sogar die Peripherie mit dazugehören. Genau dies ist auch mit der obigen Aussage von J. Wilkinson gemeint.

5 Einfluß der Programmierung auf Fortschritte in der Numerik

Um das Jahr 1955 wurden die ersten Computer im Interncode programmiert. Sehr bald kamen aber die Assembler-Sprachen auf. Jedem Maschinenbefehl entspricht dabei ein symbolischer Befehl, dessen Name möglichst bereits seine Bedeutung zum Ausdruck bringt, den man sich dadurch leichter merken kann, und es werden relative Adressen verwendet. Ein Compiler besorgt die Übersetzung des mit symbolischen Befehlen geschriebenen Programmes in Maschinenbefehle und rechnet die symbolischen Adressen in absolute Adressen um. Die Übersetzung ist eine 1:1 Umsetzung. Man spricht von maschinennaher Programmierung. Ein so geschriebenes Programm ist in der Regel nicht auf eine Maschine eines anderen Typs übertragbar. Der Wunsch nach maschinenunabhängiger Programmierung lag daher auf der Hand.

So kamen Ende der fünfziger Jahre die ersten „problemorientierten" Programmiersprachen auf. Aus Sicht der Numerik waren es FORTRAN und ALGOL. Der Schritt und der Fortschritt von den Assembler-Sprachen zu diesen Sprachen war gewaltig und genial zugleich. Er ist eigentlich nur aus der seinerzeitigen Euphorie heraus begreifbar. Problemorientiert bedeutet im Hinblick auf die Numerik im wesentlichen, daß man arithmetische Ausdrücke und Funktionen, welche einen ganzzahligen, reellen oder logischen Wert liefern, in einer weitestgehend der üblichen mathematischen Notation angepaßten Schreibweise niederschreiben kann. Er braucht nicht weiter in Einzeloperationen zerlegt zu werden. Ein Compiler erledigt die Übersetzung in Maschinenbefehle. Für kompliziertere Funktionen, welche einen Wert der genannten Typen liefern, gibt es in der Unterprogrammtechnik den Begriff der Funktion. Er erlaubt den Gebrauch von Funktionen in Programmen in einer Form, welche ebenfalls weitestgehend der in der Mathematik üblichen Notation entspricht. Für Dinge, welche nicht in dieses Schema passen, wurde der Begriff der Prozedur bereitgestellt.

Da man in der Mathematik jede Rechnung letztlich auf das Rechnen mit reellen Zahlen zurückführt, waren die meisten Numeriker mit diesem Zustand über mehrere Jahrzehnte hinweg völlig zufrieden. Es ist allerdings bemerkenswert, daß der Schritt von den Assembler-Sprachen zu den höheren Programmiersprachen praktisch widerstandslos akzeptiert wurde, obwohl ein in einer höheren Programmiersprache formuliertes Programm in der Regel eine längere Laufzeit benötigt. Man war damals offenbar bereit, ein höheres Abstraktionsniveau gegen Rechenzeit einzutauschen!

Mitte der sechziger Jahre kam die Intervallrechnung auf. Eine vernünftige Verwendung dieses äußerst nützlichen Kalküls war jedoch bis Anfang der neunziger Jahre praktisch nicht möglich, da die kommerziell verbreiteten Programmiersprachen der ersten Generation wie FORTRAN, ALGOL, BASIC, PASCAL, MODULA oder C einen einfachen Umgang damit nicht zuließen. Will man etwa einen einfachen arithmetischen Ausdruck, etwa ein Polynom vierten Grades, für ein reelles Argument auswerten, so kann man ihn in diesen Sprachen wie in der Mathematik üblich niederschreiben. Möchte man ihn hingegen für ein kleines Intervall auswerten, so erfordert jede arithmetische Operation einen Prozeduraufruf. Die Programmierung wird dadurch wieder auf das Assembler-Niveau zurückgeworfen. Ein derartiges Programm ist nicht mehr lesbar. Ein

Programm von wenigen Seiten versteht der Programmierer einige Tage später selbst nicht mehr. Die meisten potentiellen Anwender der Intervallrechnung sind vermutlich aus diesem Grund bei den Anfangsgründen steckengeblieben.

Noch in den sechziger Jahren hat H. W. Wippermann gezeigt, wie man aus dieser Schlinge herauskommt. Man muß den Funktionsbegriff in den Programmiersprachen verallgemeinern und wie in der Mathematik üblich auch Funktionen zulassen, welche mehr als ein Ergebnisdatum liefern. Im Gegensatz zu einer Prozedur kann man eine Funktion in einem arithmetischen Ausdruck mehrmals aufrufen, z. B. $f(f(f(a,b),c),d)$. Von hier ist es zu der in der Mathematik üblichen Operatornotation nur noch ein kleiner, notationeller Schritt. Man schreibt den Funktionsnamen nicht mehr vor die beiden Argumente, sondern dazwischen und ersetzt ihn dann noch durch ein Operatorsymbol. Man erhält dann z. B. $a + b + c + d$. Dabei können jetzt a, b, c, d reelle oder komplexe Zahlen, Intervalle, Vektoren, Matrizen, usw. sein. Der Compiler prüft den Typ der Operanden und führt dann den zugehörigen +-Operator aus.

Dieses Konzept hat dann bei der Schaffung von ALGOL-68 Pate gestanden und dazu beigetragen, daß dort ein Operatorkonzept vorgesehen war. ALGOL-68 war im Hinblick auf numerische Anwendungen eine sehr fortgeschrittene Sprache. Sie hat sich leider nicht durchgesetzt, vielleicht weil die Rechner damals noch nicht leistungsfähig genug waren. Sicherlich aber auch, weil die Anwender diesen Programmierkomfort nicht vehement genug gefordert haben. Es gibt heute Compiler, welche die Verwendung von Funktionen mit allgemeinem Ergebnistyp zulassen. Man sollte dann auch noch den zweiten Schritt gehen und ein Operatorkonzept zur Verfügung stellen.

Ohne ein Operatorkonzept in der Programmiersprache ist auch ein größerer Fortschritt im Hinblick auf die Rechengenauigkeit nicht möglich. Semimorphe Verknüpfungen in den üblichen Produkträumen der Numerik, wie wir sie im dritten Abschnitt definiert haben, lassen sich in den Sprachen der ersten Generation (FORTRAN, ALGOL, BASIC, PASCAL, MODULA, C) nicht vernünftig codieren. Ein maximal genaues Skalarprodukt oder eine maximal genaue Matrixmultiplikation läßt sich über diese Sprachen praktisch nicht ansprechen. Will man so etwas darin zur Verfügung stellen, so muß man auf das Niveau der Codeprozeduren hinuntersteigen. Programmiert man hingegen etwa eine Matrixmultiplikation in der üblichen Weise mittels dreier geschachtelter Laufanweisungen, so werden die darin auftretenden Produkte und Summen wieder mit der gewöhnlichen Gleitkommaarithmetik ausgewertet, d. h. es wird nach jeder Multiplikation und jeder Addition gerundet, was viele unnötige Fehler verursacht.

Die Lösung dieses Problems besteht in der Bereitstellung eines Operatorkonzeptes in der Programmiersprache. Alle Verknüpfungen in den üblichen Räumen der Numerik wie reelle und komplexe Zahlen, Vektoren und Matrizen und den zugehörigen Intervallräumen können dann mit den in der Mathematik üblichen Operatorsymbolen für entsprechend vordefinierte Datentypen angesprochen werden. Sind etwa A, B, C reelle oder komplexe Matrizen, so wird eine Matrixmultiplikation einfach durch die Anweisung $C := A * B$ angesprochen. Das Operatorzeichen $*$ ruft dabei auf Maschinenniveau eine möglichst in Hardware realisierte Elementaroperation auf, welche jede Komponente der Produktmatrix maximal genau berechnet. Diese Operatornotation für alle üblichen arithmetischen Verknüpfungen vereinfacht das Programmieren erheblich. Viele Probleme werden überhaupt erst dadurch zuverlässig programmierbar und

beherrschbar. Die Programme lassen sich wesentlich leichter lesen. Sie lassen sich leichter austesten und werden dadurch wesentlich zuverlässiger. Vor allem aber sind die Verknüpfungen maximal genau.

Am Institut des Autors gab es seit 1967 immer einen entsprechenden Programmierkomfort. Zunächst war es der von H. W. Wippermann implementierte Compiler für die bereits erwähnte ALGOL-60 Erweiterung. In den siebziger Jahren wurde dann in Zusammenarbeit mit der Firma Nixdorf eine sehr mächtige PASCAL Erweiterung (PASCAL-SC) geschaffen und auf einem Mikroprozessor implementiert. In den achtziger Jahren entstand in Zusammenarbeit mit der Firma IBM die Programmiersprache ACRITH-XSC [15] als FORTRAN-77 Erweiterung. Parallel dazu wurde auf Institutsebene die bereits vorhandene PASCAL Erweiterung zu PASCAL-XSC [2] [5] weiterentwickelt. In all diesen Sprachen sind alle arithmetischen Verknüpfungen nach dem Prinzip des Semimorphismus definiert und implementiert. Sie können mit der in der Mathematik üblichen Operatornotation angesprochen werden.

Für die Definition und die Implementierung derartiger, in sich konsistenter Programmiersprachen müssen jeweils etwa 20 Mannjahre aufgewendet werden. Dies erklärt vielleicht, warum Entsprechendes nicht auch an anderen Orten entstanden ist. Der dadurch gewonnene Programmierkomfort und das Ausdrucksniveau sind allerdings beträchtlich. Sie vereinfachen die Programmierung ganz erheblich. Dadurch werden Problemlösungen ermöglicht, welche weit außerhalb der Reichweite der Sprachen der ersten Generation liegen. Problemlösungen wie die hochgenaue, verifizierte Berechnung von Lösungen gewöhnlicher Differentialgleichungen oder von Integralgleichungen oder auch nur die Einschließung des Wertebereiches von Ableitungen und Taylorkoeffizienten über gewissen Bereichen werden eigentlich erst auf diesem Niveau möglich und auch erst verständlich.

Neuerdings gibt es mit C++ und FORTRAN-90 auch zwei standardisierte, verbreitete Programmiersprachen, welche über ein Operatorkonzept verfügen. Zumindest die Entwicklung der letzteren ist durch die Existenz der XSC-Sprachen wesentlich beeinflußt worden. In C++ und FORTRAN-90 hat man es aber leider wieder unterlassen, die Eigenschaften der Arithmetik präzise durch das Konzept des Semimorphismus zu definieren. Mit Hilfe des Operatorkonzeptes kann man jedoch in diesen Sprachen selbst Bibliotheken aufbauen, welche alle Verknüpfungen in den interessierenden zwölf Räumen der Numerik (reelle und komplexe Zahlen, reelle und komplexe Intervalle sowie Vektoren und Matrizen über diesen vier Grundtypen) mittels des Prinzips des Semimorphismus bereitstellen, ohne einen neuen Compiler entwickeln zu müssen. Dies ist inzwischen auch ausgeführt worden. Die so entstandenen Bibliotheken stehen unter den Namen C-XSC [6] und FORTRAN-XSC zur Verfügung.

Sprachen auf diesem Niveau könnte man als Sprachen der 2. Generation bezeichnen. Typische und unverzichtbare Merkmale dieser Sprachen sind: Ein Modulkonzept, ein Operatorkonzept, Funktionen und Operatoren mit allgemeinem Ergebnistyp, Überladen von Funktionen, Prozeduren und Operatoren, Überladen des Zuweisungsoperators sowie von read und write, dynamische Felder, Zugriff auf Unterbereichsfelder, Rundungskontrolle durch den Benutzer, semimorphe Arithmetik mit zugehörigen Operato-

[2] PASCAL-**XSC**: PASCAL for e**X**tended **S**cientific **C**omputation, Springer Verlag, 1991, englische Ausgabe 1992

ren für die Grundtypen der Numerik (reelle und komplexe Zahlen, reelle und komplexe Intervalle sowie Vektoren und Matrizen über diesen Räumen), semimorphe mehrfachgenaue Arithmetik auch für Intervalle, genaue Auswertung von Ausdrücken, automatische Differentiation.

Bild 14 zeigt eine PASCAL-XSC Prozedur zur Berechnung einer verifizierten Einschließung der Lösung eines linearen Gleichungssystems $A * x = b$. Es werden dynamische Felder, vordefinierte Arithmetikmodule, exakte Auswertung von Skalarproduktausdrücken und Intervallarithmetik verwendet. Die erste Anweisung berechnet eine Näherungsinverse der Matrix A. Die beiden folgenden Anweisungen enthalten zwei „Scharf"-Symbole. Das jeweils rechts stehende Symbol bewirkt, daß der dahinter in Klammern stehende Ausdruck exakt, ohne Informationsverlust ausgewertet wird. Das links davon stehende Symbol bewirkt eine Rundung in das kleinste einschließende Intervall. Danach ist D eine Einschließung des Defektes der Näherungsinversen. In der Schleife wird dann ein Iterationsverfahren (Gesamtschrittverfahren) mit Intervallen mit der im allgemeinen rasch konvergenten Matrix D durchgeführt. Um zu erreichen, daß die Lösung irgendwann einmal im Innern von x liegt, wird die Näherung in jedem Iterationsschritt ein klein wenig expandiert. Sobald die neue Näherung in der vorigen enthalten ist, wird das Verfahren abgebrochen. Nach dem Brouwerschen Fixpunktsatz hat man dann die Lösung eingefangen. Ergibt sich während zehn Iterationen keine Inklusion, so wird das Verfahren mit einer Fehlermeldung abgebrochen. Als Alternative kann auch vom Rechner selbsttätig ein leistungsfähigerer Algorithmus aufgerufen werden.

Bild 15 zeigt ein PASCAL-XSC Programm zur Berechnung von Einschließungen der Wertebereiche der Taylor-Koeffizienten der Funktion $e^{\frac{5000}{sin(11+(x/100)^2)+30}}$ im Intervall [1.001,1.005]. Die Variable x und das Ergebnis f werden als Feld der Dimension 40 vereinbart. In den Komponenten dieser Felder werden Einschließungen der Wertebereiche der Taylor-Koeffizienten von x bzw. f im Intervall [1.001,1.005] geführt. Mit „use itaylor" wird die im Rechner vordefinierte Interval-Taylor-Arithmetik aktiviert. Die im Ausdurck für f auftretenden arithmetischen Operationen und Standardfunktionen werden damit ausgeführt. Nach Beendigung der Rechnung enthält die i-te Komponente von f, $i = 0(1)40$, eine Einschließung des Wertebereiches des i-ten Taylorkoeffizienten von f im Intervall [1.001,1.005].

Mit einer solch einfachen Routine lassen sich beispielsweise Restglieder von Quadraturformeln sehr einfach auswerten. Die Integration kann damit adaptiv gesteuert werden. Die Verwendung von Fehlerschätzern mit all ihren Unsicherheiten kann damit entfallen.

An dieser Stelle muß im Grunde vieles von dem, was am Ende des Abschnittes über die Arithmetik gesagt wurde, noch einmal wiederholt werden. Der Rechner ist auch bei Verwendung von Programmiersprachen der ersten Generation (FORTRAN, ALGOL, BASIC, PASCAL, MODULA, C) bereits ein sehr mächtiges und auch erfolgreiches Werkzeug. Vierzig Jahre Programmierung auf dem Niveau dieser Sprachen prägt auch das Denken und sogar den Menschen. Auf diesem Niveau ist sehr viel Software entstanden, welche auch erfolgreich vermarktet wird. Dies erzeugt naturgemäß Widerstand Neuem gegenüber. Es liegt hier sehr viel Trägheit und Konservativismus im System, was wahrscheinlich nur durch Ausbildung neuer Anwendergenerationen überwunden

werden kann. Es wird sicherlich noch einige Zeit vergehen, bis die Möglichkeiten von Werkzeugen, welche wir als Sprachen der 2. Generation bezeichnet haben, voll erkannt und auch ausgenutzt werden. Den Anwendern und Numerikern kann man nur empfehlen, sich mit diesen neuen Möglichkeiten auseinanderzusetzen. Sie sind um ein Vielfaches mächtiger und leistungsfähiger als die Sprachen der ersten Generation und man kann praktisch bei jeder Anwendung Nutzen daraus ziehen. Viele Probleme sind überhaupt erst auf diesem Niveau lösbar.

Auf dem Gebiet der Rechnerarithmetik und der Programmiersprachen läßt sich Neues heute nur über die einschlägigen Standardisierungsgremien letztlich durchsetzen. In diesen Gremien sind auch die Hersteller und damit wirtschaftliche und kommerzielle Interessen und gegenseitige Abhängigkeiten vertreten, so daß gelegentlich auch mit für Außenstehende vielleicht unverständlicher Härte gefochten wird. Auch diese Situation wird in der Antrittsvorlesung unseres Dichters Friedrich Schiller im Jahr 1789 bereits vortrefflich beschrieben: *„Wer hat über Reformen mehr geschrieen als der Haufe der Brotgelehrten? Wer hält den Fortgang nützlicher Revolutionen im Reich des Wissens mehr auf als eben diese? ... Sie fechten mit Erbitterung, mit Heimtücke, mit Verzweiflung, weil sie bei dem Schulsystem, das sie verteidigen, zugleich für ihr ganzes Dasein fechten.“* Es bleibt die Hoffnung, daß sich immer wieder genügend unabhängige Wissenschaftler auch in den Standardisierungsgremien für den Fortschritt unserer Wissenschaft einsetzen. Bei den bereits erreichten und prognostizierten Fortschritten in der Rechnerhardware braucht man dabei nach Ansicht des Autors nicht mehr in erster Linie auf Erzielung hoher Ausführungsgeschwindigkeiten zu achten. Sicherheit in der Programmierung und hohes Abstraktionsniveau sowie Sicherheit in den Ergebnissen sollten allerhöchste Priorität erhalten, was hohe Ausführungsgeschwindigkeit nicht ausschließt.

Aus einem Land heraus, in dem man sich an der Weiterentwicklung des Rechners praktisch nicht mehr beteiligt, ist die Beeinflussung solcher Entwicklungen schwierig. Dennoch müssen wir uns daran beteiligen, wenn wir uns nicht selbst aufgeben wollen; denn fortschrittliche und erfolgreiche Software muß sich an den Geräten von morgen und nicht an denen, welche vor zehn Jahren in den Entwicklungslabors entworfen wurden, orientieren.

6 Numerik mit automatischer Ergebnisverifikation

In diesem Abschnitt sollen exemplarisch einige Problemlösungen angesprochen werden, welche außerhalb der Reichweite von einfacher Gleitkommaarithmetik und Programmiersprachen der ersten Generation liegen. Eine universelle Rechnerarithmetik und eine Programmiersprache vom Typ der XSC-Sprachen sind erforderlich. Für ausführlichere Darstellungen muß auf die Literatur verwiesen werden [3], [4], [10].

6.1 Verifizierte Berechnung der Lösung linearer Gleichungssysteme und anderer algebraischer Probleme

Als erstes Beispiel wollen wir kurz skizzieren, wie man etwa im Falle eines linearen Gleichungssystems die Korrektheit des berechneten Ergebnisses verifizieren kann. Sehr viele numerische Probleme führen auf lineare Gleichungssysteme. Auch ein korrekt programmiertes Lösungsverfahren muß numerisch nicht auf die richtige Lösung führen. Man berechnet zunächst mit einem üblichen, favorisierten Verfahren, beispielsweise mit dem Gauß-Algorithmus, eine Näherung für die Lösung. Diese Näherung dehnt man nun aus, indem man in jeder Komponentenrichtung eine kleine Zahl $\pm\epsilon$ draufschlägt. So entsteht ein n-dimensionaler Quader X mit der berechneten Näherung als Mittelpunkt. Dieser Quader ist nichts anderes als ein Intervallvektor, ein Intervall in jeder Koordinatenrichtung. Nun bringt man das Gleichungssystem in geeigneter Weise in Fixpunktform $x = Dx + z$, wobei man natürlich die Tatsache, daß man ja schon gerechnet hat und vermutlich bereits eine Näherungslösung besitzt, ausnutzt. Man setzt nun den Intervallvektor X auf der rechten Seite ein und berechnet das Bild $Y := DX + z$, Bild 16. Ist Y in X enthalten, was man durch einfachen Schrankenvergleich leicht feststellen kann, so enthält Y nach dem Brouwerschen Fixpunktsatz [3] einen Fixpunkt der Gleichung $x = Dx + z$. Ist Y strikt in X enthalten, so daß sich die Ränder nicht berühren, so ist der Fixpunkt sogar eindeutig, das heißt, es wurde rechnerisch nachgewiesen, daß die ursprüngliche Matrix des Gleichungssystems nicht singulär ist, das Gleichungssystem also genau eine Lösung besitzt. Durch Iteration läßt sich der Fixpunkt mittels des optimalen Skalarproduktes praktisch beliebig genau bestimmen. Der Benutzer erhält auf diese Weise ein mathematisch einwandfreies und sicheres Ergebnis, welches in seltenen Fällen allerdings auch darin bestehen kann, daß der Verifikationsschritt nicht zum Erfolg geführt hat. In diesem Fall kann ein anderes Lösungsverfahren oder eine verfeinerte Technik herangezogen und vom Rechner selbst aufgerufen werden.

Ähnliche Techniken der numerischen Ergebnisverifikation lassen sich bei vielen anderen algebraischen Problemstellungen anwenden, wie z. B. der Lösung nichtlinearer Gleichungssysteme, bei der Berechnung von Nullstellen, bei der Berechnung von Eigenwerten und Eigenvektoren von Matrizen, bei Optimierungsproblemen usw. Hierzu gehört insbesondere auch die hochgenaue und sichere Auswertung von arithmetischen Ausdrücken und Funktionen im Rechner. Die entsprechenden, entwickelten und verfügbaren Routinen arbeiten auch für Probleme mit (kleinen) Intervallen als Daten. In diesem Fall wird die Menge aller Lösungen eingeschlossen [3], [4], [10]. Ihr Durchmesser gibt Aufschluß über die Sensitivität des Problems.

Die Funktionsauswertung wird dabei auf die Lösung eines einfachen linearen Gleichungssystems zurückgeführt, dessen Näherungslösung gegebenenfalls mittels Defektkorrektur und optimalem Skalarprodukt bis zu garantierter, hoher Genauigkeit nachkorrigiert wird. Diese Technik, Funktionswerte hochgenau in Schranken einzuschließen, stellt ein ganz wesentliches Hilfsmittel für die Numerik dar. Ein Newton-Verfahren zur Berechnung der Nullstelle eines Gleichungssystems ist in der Umgebung der Nullstelle

[3] Der Brouwersche Fixpunktsatz besagt folgendes: Es sei $\phi : I\!R^n \to I\!R^n$ eine stetige Abbildung und $X \epsilon I\!R^n$ nicht leer, konvex, abgeschlossen und beschränkt, sowie $Y := \phi(X)$. Ist $Y \subseteq X$, so besitzt ϕ mindestens einen Fixpunkt in Y.

häufig instabil (Bild 5). Eine Null kommt nämlich nur dadurch zustande, daß sich positive und negative Terme gegeneinander wegheben. In der unmittelbaren Umgebung der Nullstelle sind diese Terme nur näherungsweise gleich und es tritt bei gewöhnlicher Gleitkommarechnung Auslöschung auf. Bei nicht genügend genauer Funktionsauswertung aber schießt das Newton-Verfahren dann unter Umständen weit über das Ziel hinaus, eventuell sogar in den Einzugsbereich einer anderen Nullstelle. Ein im Reellen gegen eine Lösung x konvergentes Iterationsverfahren muß so in gewöhnlicher Gleitkommarechnung keineswegs gegen den Wert x konvergieren. Es kann sich unter Umständen auf einen völlig verschiedenen Wert einpendeln. Ein abschließender Verifikationsschritt ist daher auch bei Iterationsverfahren angebracht.

Eine bewährte Methode, hohe Genauigkeit zu erzielen besteht darin, nicht die Lösung selbst, sondern den Fehler (Defekt) einer Näherung einzuschließen.

6.2 Automatische Differentiation

Im Bereich der Numerik gehört es sicherlich zu den wesentlichsten und fruchtbarsten Entdeckungen der letzten Jahrzehnte, daß man Werte und Einschließungen von Werten und Wertebereichen von Ableitungen und Taylor-Koeffizienten, Jacobi-Matrizen, Gradienten, Hesse-Matrizen usw. aus dem arithmetischen Ausdruck heraus rein zahlenmäßig berechnen kann, ohne den Ausdruck zunächst formal zu differenzieren. Die automatische Differentiation unterscheidet sich wesentlich von der numerischen Differentiation durch Differenzenquotienten.

Als automatische Differentiation bezeichnet man im einfachsten Fall eine Technik, welche es gestattet, sowohl den Wert einer durch einen arithmetischen Ausdruck gegebenen Funktion als auch denjenigen der Ableitung an einer bestimmten Stelle zu berechnen. Die Stelle kann dabei auch ein Intervall sein. In diesem Fall werden Schranken für den Wertebereich der Funktion und denjenigen der Ableitung über diesem Intervall berechnet. Anstelle mit Zahlen bzw. Intervallen wertet man den Ausdruck für Zahlenpaare bzw. Paare von Intervallen aus. Eine Weiterentwicklung der automatischen Differentiation ist die automatische Generierung von Taylor-Koeffizienten und Einschließung von Wertebereichen von Taylor-Koeffizienten über bestimmten Intervallen. Derartige Techniken sind bei alleiniger Verfügbarkeit von einfacher Gleitkommaarithmetik und Programmiersprachen der ersten Generation praktisch nicht kodierbar.

Bei der automatischen Differentiation eines arithmetischen Ausdruckes für eine Funktion wird für jeden Operanden des Ausdruckes (das sind Zahlen, Variable und Standardfunktionen) ein Zahlenpaar eingesetzt, wobei die erste Komponente den Wert dieses Operanden an der auszuwertenden Stelle und die zweite den Wert der Ableitung angibt. Diese Zahlenpaare werden nun nach den im Ausdruck vorgegebenen Operationen bzw. den bekannten Ableitungsregeln für Summe, Differenz, Produkt, Quotient und die Kettenregel sowie die Ableitungsregeln für elementare Funktionen miteinander verknüpft. Als Ergebnis erhält man ein Zahlenpaar, wobei die erste Komponente den Wert des Ausdruckes und die zweite den Wert der Ableitung an der betreffenden Stelle angibt. Bei der automatischen Generierung von Taylor-Koeffizienten rechnet man entsprechend mit n-Tupeln, wobei die erste Komponente den Wert der Funktion und die weiteren Komponenten die Werte des ersten, zweiten usw. Taylor-Koeffizienten der

Funktion an der betreffenden Stelle angibt. Wichtig ist dabei, daß in beiden Fällen nur mit Zahlen und nicht mit formalen Ausdrücken gerechnet wird. Einschließungen von Ableitungen und Taylorkoeffizienten über gewissen Bereichen erhält man mit den gleichen Algorithmen, indem man die Rechnung anstelle von Zahlen mit Intervallen ausführt.

Die automatische Differentiation und die Generierung von Taylor-Koeffizienten sind bei etwas komplizierteren Ausdrücken von Hand kaum durchführbar, weil etwa die Kettenregel oder die Quotientenregel relativ komplizierte Vorschriften darstellen, so daß man sich regelmäßig sehr schnell verrechnet. Eine Maschine kann aber diese Vorschriften sehr stabil, schnell und korrekt ausführen. Die Programmierung der automatischen Differentiation oder die Generierung von Taylor-Koeffizienten gestaltet sich mit den in den vorigen Abschnitten beschriebenen arithmetischen und programmiersprachlichen Erweiterungen sehr einfach. Mittels des Operatorkonzeptes und der verfügbaren Arithmetik für Intervalle läßt sich die mathematische Intelligenz praktisch in das Laufzeitsystem des Compilers verlagern. Die im arithmetischen Ausdruck auftretenden arithmetischen Operatoren werden vom Compiler einfach entsprechend den Regeln der automatischen Differentiation für Paare bzw. n-Tupel anders interpretiert. Man spricht auch von Operatorüberladung. Die Maschine liefert den Wert der Ableitung bzw. der höheren Taylor-Koeffizienten. Bei Intervalldaten liefert sie Einschließungen der Wertebereiche der Ableitungen und Taylor-Koeffizienten.

Die automatische Differentiation ist auf dem Rechner sehr schnell und effizient ausführbar. Es wird nur mit Zahlen gerechnet und nicht erst der Ausdruck für die (höhere) Ableitung formal hergeleitet. Man unterscheidet zwischen der Vorwärts- und der Rückwärtsmethode. Überraschend ist, daß man mit der Rückwärtsmethode beispielsweise den Gradienten mit einem arithmetischen Aufwand berechnen kann, welcher von der gleichen Größenordnung ist, wie derjenige für die Berechnung des Funktionswertes selbst. Allerdings wächst dabei der Speicherplatzbedarf stark an. Die Kunst besteht darin, ein Optimum zwischen Rechenaufwand und Speicherplatzbedarf zu finden.

6.3 Verifizierte Berechnung der Lösung von gewöhnlichen Differentialgleichungen

Bei gewöhnlichen Differentialgleichungen gibt es Verfahren sowohl für Anfangs- als auch für Rand- und Eigenwertaufgaben. Bei den Anfangswertaufgaben wird etwa im Fall eines Systems erster Ordnung die rechte Seite der Differentialgleichung mit automatischer Differentiation in ein Taylor-Polynom mit Restglied entwickelt. Das Restglied wird nun mit einer geeigneten Schrittweite ausgewertet. Da man eine Einschließung der Lösung erhalten möchte, erfolgt die Rechnung intervallmäßig. Im Unterschied zur numerischen Quadratur enthält das Restglied bei Differentialgleichungen außer der unabhängigen Variablen im allgemeinen auch noch die gesuchte Funktion selbst. Will man das Restglied mittels Intervallarithmetik sicher einschließen, so braucht man daher auch bereits eine sichere Einschließung der gesuchten Funktion im ganzen Integrationsintervall. Es scheint also so, als hätte man sich hier in einem Zirkel verfangen. Diesen kann man aber aufschneiden. Zunächst braucht man nämlich nur eine grobe Einschließung der gesuchten Funktion über dem Integrationsintervall. Diese läßt sich folgendermaßen be-

schaffen. Die Differentialgleichung wird zunächst in eine äquivalente Integralgleichung umgeformt. Der Integrand enthält nun außer der unabhängigen Variablen wiederum die gesuchte Funktion. Will man das Integral mittels Intervallrechnung sicher in Schranken einschließen, so braucht man wiederum bereits eine Einschließung der gesuchten Funktion im ganzen Integrationsintervall. Im einfachsten Fall beschafft man sich eine solche durch Schätzung. Da die Differentialgleichung einer Lipschitzbedingung genügt, gibt es immer eine solche Einschließung. Unter Umständen muß man dabei die Schrittweite verkleinern. Diese, zunächst noch rudimentäre, geschätzte Anfangseinschließung wird nun mittels des Banachschen Fixpunktsatzes abgesichert, indem man die rechte Seite der Integralgleichung damit intervallmäßig auswertet und prüft, ob sich eine erwartete Inklusion ergibt. Ist dies nicht der Fall, so kann man die Schrittweite und/oder die Anfangsschätzung verändern. Ist hingegen die Inklusion eingetreten, so hat man eine sichere Anfangseinschließung der gesuchten Funktion im Integrationsintervall. Damit wird nun das Restglied des Taylor-Polynoms ausgewertet. Das Restglied ist jetzt klein von höherer Ordnung. Man erhält so insgesamt eine wesentlich bessere Approximation der gesuchten Funktion im Integrationsintervall durch ein Polynom mit Intervallkoeffizienten. Der bei Fortsetzung dieser Vorgehensweise über mehrere Einzelschritte sich ergebende sogenannte Wrapping-Effekt läßt sich durch Mitführung lokaler Koordinaten beherrschen.

Rand- und Eigenwertprobleme bei gewöhnlichen Differentialgleichungen lassen sich im einfachsten Fall mittels Schießverfahren auf Anfangswertprobleme zurückführen. Dabei wird auch die Existenz und Eindeutigkeit der Lösung vom Algorithmus, das heißt mit dem Rechner, nachgewiesen.

Diese Methoden zur verifizierten Berechnung von Einschließungen von Lösungen gewöhnlicher Differentialgleichungen sind im allgemeinen aufwendiger als Näherungsverfahren mit einfacher Gleitkommaarithmetik, was im Gegensatz zu einer weitverbreiteten Meinung für andere Intervallverfahren keineswegs immer der Fall ist. Dieser Mehraufwand wird aber durch die gewonnene Ergebnissicherheit oft mehr als wett gemacht, da sich dadurch ein oft angewendetes vermeintliches Absichern der Lösung durch Experimentieren erübrigt.

Es gibt inzwischen viele Anwendungen dieser Methoden auf Probleme, bei denen man ohne diese Techniken ziemlich hilflos ist. Auf die Einsatzmöglichkeit in der Chaosforschung wurde in der Einleitung bereits hingewiesen. Sehr schöne Erfolge sind auch beim Aufspüren von periodischen Lösungen von Differentialgleichungen erzielt worden. Hier kann man gegebenenfalls mit dem Rechner mathematisch völlig einwandfrei beweisen, daß eine gegebene Differentialgleichung periodische Lösungen besitzt und diese sogar noch in sichere und enge Schranken einschließen. Wenn man bedenkt, wieviel Überlegungen und Abschätzungen mit Bleistift und Papier in solchen Fällen früher durchgeführt werden mußten, so belegt gerade dieses Beispiel drastisch die Mächtigkeit des Kalküls der Intervallrechnung. Die Tatsache, daß man solche Ergebnisse heute mit dem Rechner herleiten kann, ist im Grunde eine echte Revolution sowohl in der Numerik wie in der Mathematik. Allein so etwas reicht aus, um die Bereitstellung einer mächtigeren Arithmetik und Programmierumgebung zu rechtfertigen.

6.4 Verifizierte Berechnung des Schnittes von Kurven und Flächen

Als weiteres Beispiel soll noch die verifizierte Berechnung des Schnittes von Kurven und Flächen genannt werden. Ein herkömmliches Pfadverfolgungsverfahren zur Berechnung des Schnittes zweier Flächen kann, falls sich zwei Äste der Schnittkurve nahekommen, durchaus von einem Ast auf den anderen springen und dadurch eine völlig falsche Topologie der Schnittkurve vortäuschen. Bei der verifizierten Berechnung der Schnittkurve wird zunächst eine gewöhnliche Differentialgleichung dafür hergeleitet. Diese wird dann verifiziert integriert. Hierbei ist bereits der allererste, der Banach-Schritt, ausreichend; denn hat man erst einmal eine sichere Einschließung am rechten Ende des Integrationsintervalles, so kann man diese mittels der Parameterdarstellung der Fläche als Gleichungssystem mit dem Newton-Verfahren praktisch wieder auf einen Punkt zusammenziehen. Dadurch wird auch die Eindeutigkeit der berechneten Lösung im Integrationsintervall nachgewiesen. Ein Wrapping-Effekt kann nicht auftreten (Bild 17).

6.5 Behandlung von Integralgleichungen und Anwendungen bei partiellen Differentialgleichungen

Auch für Integralgleichungen sind Methoden zur verifizierten Berechnung von Lösungen entwickelt worden. Als Hilfsmittel werden wieder die automatische Differentiation und Fixpunktsätze im Zusammenwirken mit der Intervallrechnung verwendet. Bei Fredholmschen Integralgleichungen zweiter Art beispielsweise wird der Kern mittels automatischer Differentiation in eine zweidimensionale Taylorreihe mit Restglied entwickelt. Er besteht dann aus einer Summe aus einem entarteten Kern und einem kontrahierenden Kern. Beide Teile können dann durch Verwendung von Intervallarithmetik mit im Grunde bekannten Methoden behandelt und die Lösung eingeschlossen werden. Diese Methode ist auf große Systeme nichtlinearer Integralgleichungen erfolgreich angewandt worden. Auch Probleme bei partiellen Differentialgleichungen sind durch Zurückführung auf Integralgleichungen mit automatischer Ergebnisverifikation behandelt worden. Umgekehrt sind auch Probleme bei Integralgleichungen auf Differentialgleichungen zurückgeführt und dann mit automatischer Ergebnisverifikation gelöst worden.

Neuerdings werden verifizierende Methoden auch für Probleme bei partiellen Differentialgleichungen entwickelt. Traditionell kommen bei diesen insbesondere Differenzenverfahren und Finite-Elemente-Methoden zum Einsatz. Durch die dadurch entstehenden großen Gleichungssysteme wird auch bei heutigen Rechenanlagen bezüglich Rechenzeit und Speicherplatzverwaltung die Leistungsgrenze sehr schnell erreicht.

Von besonderem Interesse sind daher solche Verfahren, mit denen sich partielle Differentialgleichungen über gewöhnliche Differentialgleichungen lösen lassen. Dabei kann zwar nicht von einer Verringerung der Rechenzeit ausgegangen werden. Es wird aber die Datenverwaltung stärker auf das System verlagert und dadurch automatisiert. In diese Kategorie fallen alle Verfahren, welche sich unter dem Stichwort „Linienmethode“ zusammenfassen lassen, aber auch Charakteristikenverfahren unter anderem bei den partiellen Differentialgleichungen erster Ordnung. Bei den Letzteren erhält man

verifizierte Werte der Lösung der partiellen Differentialgleichung längs Linien. Bei der Linienmethode besteht eine Schwierigkeit darin, daß der Diskretisierungsfehler noch irgendwie abgeschätzt werden muß.

Inzwischen sind auch für nichtlineare elliptische Randwertprobleme Methoden entwickelt worden, mit denen sich ausgehend von einer Näherungslösung eine strenge Lösungseinschließung gewinnen läßt, wobei gleichzeitig die Existenz der Lösung vom Verfahren mitbewiesen wird. Auch hierbei werden Fixpunktsätze als wesentliches Hilfsmittel eingesetzt.

7 Schlußbemerkung

In Europa ist genügend geistige und auch finanzielle Kraft vorhanden, um sich sowohl an der Weiterentwicklung der Rechnerhardware als auch der Basissoftware erfolgreich zu beteiligen. Die Kulturlandschaft Europa braucht eine eigenständige Rechnerentwicklung, zumal es sich um eine völlig problemlose Technologie handelt. Der heute vielfach beklagte technologische Rückstand Europas hat seine Ursache weitgehend in der Aufgabe und Verdrängung einer eigenständigen Computerentwicklung mit ihren vielfältigen Basiskomponenten. Eine erfolgreiche Rechnerentwicklung muß auch die Technologie und die Prozessorstruktur einbeziehen. Ein Sprung aus dem Stand an die Spitze ist nicht zu erwarten. Bei der extrem langen Zurückhaltung auf diesem Gebiet ist Geduld erforderlich. Es muß langsam und systematisch damit begonnen werden, wieder Know-how aufzubauen. Dazu ist auch die Politik gefordert. Chancen, wieder Tritt zu fassen, tun sich immer wieder auf. Mangelnde Risikobereitschaft oder Angst vor der eigenen Courage helfen nicht weiter. Sie führen nur immer tiefer in die Sackgasse. Das Problem muß vielmehr strategisch und frontal angegangen werden. Erinnern und besinnen wir uns zurück auf unsere Pioniere wie Konrad Zuse und andere, die vor mehr als 50 Jahren durch klare und konsequente Lösungen den Weg aufgezeigt und gewiesen haben!

8 Literaturhinweise

[1] Alefeld, G., Herzberger, J.: *An Introduction to Interval Computations*, Academic Press, New York, 1983.

[2] de Beauclair, W.: *Rechnen mit Maschinen*, Vieweg, Braunschweig, 1968.

[3] Hammer, R. et al.: *Numerical Toolbox for Verified Computing I, Theory, Algorithms and PASCAL-XSC Programs*, Springer, Berlin, 1991.

[4] Hammer, R. et al.: *C++ Toolbox for Verified Computing I*, Springer, Berlin, 1995.

[5] Klatte, R. et al.: *PASCAL-XSC-Sprachbeschreibung mit Beispielen*, Springer, Berlin, 1991, engl. Übersetzung, Springer, Berlin, 1992.

[6] Klatte, R. et al.: *C-XSC, A C++ Class Library for Extended Scientific Computing*, Springer, Berlin, 1993.

[7] Koecher, M.: *Lineare Algebra und analytische Geometrie*, Springer, Berlin, 1983.

[8] Kulisch, U.: *Grundlagen des Numerischen Rechnens*, BI Wissenschaftsverlag, Mannheim, 1976.

[9] Kulisch, U., Miranker, W.L.: *Computer Arithmetic in Theory and Practice*, Academic Press, New York, 1981.

[10] Lohner, R. et al.: *Numerical Toolbox for Verified Computing II, Theory, Algorithms and PASCAL-XSC Programs*, Springer, Berlin, 1995.

[11] Ungerer, T.: *Mikroprozessoren - heute, morgen und übermorgen*, Jahrbuch Überblicke Mathematik 1996, Vieweg.

[12] Zuse, K.: *The Plankalkül*, Oldenbourg, München, 1989.

[13] GAMM-IMACS, *Proposal for Accurate Floating-Point Vector Arithmetic*, in Rundbrief der GAMM, 1993, Brief 2, und in *Mathematics and Computers in Simulation*, Vol. 35, No. 4, IMACS, 1993.

[14] IBM, *High Accuracy Arithmetic Subroutine Library (ACRITH)*, General Information Manual, 3rd ed., GC33-6163-02, IBM, 1986.

[15] IBM, *High Accuracy Arithmetic–Extended Scientific Computation (ACRITH-XSC)*, General Information, GC33-6461-01, IBM, 1990.

[16] Zeitschrift *Elektronik* 26, 1994, *Genauer und trotzdem schneller, Ein neuer Coprozessor für hochgenaue Matrix- und Vektoroperationen*, Titelgeschichte.

In dieser Reihe sind bisher die folgenden Arbeiten erschienen:

1/1996 Ulrich Kulisch: *Memorandum über Computer, Arithmetik und Numerik.*

2/1996 Andreas Wiethoff: *C–XSC — A C++ Class Library for Extended Scientific Computing.*

3/1996 Walter Krämer: *Sichere und genaue Abschätzung des Approximationsfehlers bei rationalen Approximationen.*

4/1996 Dietmar Ratz: *An Optimized Interval Slope Arithmetic and its Application.*

5/1996 Dietmar Ratz: *Inclusion Isotone Extended Interval Arithmetic.*

1/1997 Astrid Goos, Dietmar Ratz: *Praktische Realisierung und Test eines Verifikationsverfahrens zur Lösung globaler Optimierungsprobleme mit Ungleichungsnebenbedingungen.*

2/1997 Stefan Herbort, Dietmar Ratz: *Improving the Efficiency of a Nonlinear-System-Solver Using a Componentwise Newton Method.*

3/1997 Ulrich Kulisch: *Die fünfte Gleitkommaoperation für top-performance Computer — oder — Akkumulation von Gleitkommazahlen und -produkten in Festkommaarithmetik.*

4/1997 Ulrich Kulisch: *The Fifth Floating-Point Operation for Top-Performance Computers — or — Accumulation of Floating-Point Numbers and Products in Fixed-Point Arithmetic.*

5/1997 Walter Krämer: *Eine Fehlerfaktorarithmetik für zuverlässige a priori Fehlerabschätzungen.*