Kleine Fehlerschranken bei Matrixproblemen

Zur Erlangung des Akademischen Grades eines

DOKTORS DER NATURWISSENSCHAFTEN

von der Fakultät für Mathematik

der Universität Karlsruhe

genehmigte

DISSERTATION

von

Dipl.–Math. Siegfried M. Rump

aus Wuppertal

Tag der mündlichen Prüfung: 15. 2. 1980

Referent: ..... Professor Dr. U. Kulisch

Korreferent: ..... Professor Dr. W. Niethammer

ANERKENNUNG

Allen, die am Zustandekommen dieser Arbeit einen Anteil haben, möchte ich an dieser Stelle meinen herzlichen Dank ausdrücken. Dies gilt insbesondere für die Mitarbeiter des Instituts für Angewandte Mathematik der Universität Karlsruhe, die in vielen Gesprächen und steter Hilfsbereitschaft ihren Teil dazu beitrugen. Dies gilt insbesondere für Herrn Dr. E. Kaucher; ohne die unzähligen Diskussionen mit ihm und seinen Ideenreichtum läge diese Arbeit heute sicher nicht vor. Mein Dank geht an Herrn cand. math. H. Böhm, der mit viel Engagement alle vorkommenden Programme auf der Rechenanlage implementierte, und an Frau Lioba Schindele, die Teile des Manuskripts mit (der bereits gewohnten) großen Sorgfalt zu Papier brachte.

Mein besonderer Dank gilt den beiden Referenten Herrn Professor Dr. U. Kulisch und Herrn Professor Dr. W. Niethammer. Es ist ein großes Glück, in unserem Institut bei Herrn Kulisch arbeiten zu dürfen. Allein für das Schaffen dieser Voraussetzung bin ich ihm zu tiefstem Dank verpflichtet. Durch seine weitestgehende stete und verständnisvolle Förderung stehe ich tief in seiner Schuld.

Vorwort

In der heutigen Zeit, wo in Großrechenanlagen riesige Algorithmen mit Millionen von Rechenoperationen laufen, wird immer häufiger die Frage gestellt: "Kann ich den berechneten Ergebnissen überhaupt noch trauen?" Tatsächlich können - wie die Vergangenheit gezeigt hat - durch Akkumulierung von Rundungsfehlern große Fehler entstehen, die bis zur völligen Verfälschung des Ergebnisses führen. Viele Algorithmen arbeiten ohne beweisbare Fehlerschranken. Doch auch in den Fällen, in denen eine Fehlerabschätzung des Ergebnisses angegeben wird, das mittels eines Gleitkomma-Algorithmus erzielt wurde, ist Vorsicht geboten. Bei der Lösung eines linearen Gleichungssystems $Ax = b$ etwa kann man aus $(A + \Delta A)\tilde{x} = b$ für das Gleitkommaergebnis $\tilde{x}$ und einer Abschätzung für $\|\Delta A\|$ noch nicht auf den Fehler von $\tilde{x}$ gegenüber der wahren Lösung schließen. Um diesen Fehler abzuschätzen, wird die Kondition der Matrix benötigt, die i.a. nicht bekannt ist. Weiter ist nicht zu vergessen, daß die Richtigkeit der Abschätzung für $\|\Delta A\|$, sagen wir $\|\Delta A\| \leq \eta$, voraussetzt, daß alle Matrizen B mit $\|B - A\| \leq \eta$ nicht singulär sind.

Diese kurzen Überlegungen zeigen bereits, daß die Verwendung von Gleitkommaalgorithmen sehr wohl möglich ist, jedoch Geschick und feine Beobachtung der Vorgänge vom Anwender verlangt. Gleichwohl sei betont, daß auf diesem Wege keine mathematisch beweisbaren Fakten erzielt werden. In der Praxis kommt es leider oft vor, daß die Programme einerseits unsachgemäß angewandt werden und andererseits die Probleme derart komplex sein können, daß jegliche Übersicht verloren geht. Darüberhinaus ist vom Anwender in jeder Phase viel Energie zu investieren, um folgenschwere Fehler zu vermeiden.

In der vorliegenden Arbeit werden zu einigen Problemen Algorithmen angegeben, die garantierte, mathematisch beweisbare Fehlerabschätzungen für das Ergebnis errechnen, und zwar mit hoher Genauigkeit. Es gelingt für die Lösung linearer Gleichungssysteme bei 8-stelliger Rechnung mindestens 15 Dezimalstellen zu garantieren. Obwohl diese Algorithmen nur mit solchen direkt vergleichbar sind, die ebenfalls garantierte

Schranken liefern sei vermerkt, daß die Rechenzeit höchstens die 6-fache eines entsprechenden Gleitkommaalgorithmus ist. Außerdem werden sämtliche Vorteile der Gleitkommaalgorithmen verwendet. Mit diesen Algorithmen gelingt es, auf dem Rechner numerische Beweise zu führen.

Im ersten Kapitel wird zunächst eine naheliegende und bekannte Form solcher Algorithmen zur Lösung linearer Gleichungssysteme untersucht: Das Intervall-Gauß- und das Intervall-Gauß-Jordan-Verfahren. Ist eines der beiden Verfahren (intervallmäßig) durchführbar, so kann gezeigt werden, daß die Eingabe(punkt- oder intervall)-matrix nicht singulär ist bzw. keine singuläre Matrix enthält und die Lösung garantiert im Ergebnisintervall liegt (siehe Alefeld/Herzberger). Hervorzuheben ist die Beobachtung, daß durch die Intervallrechnung offenbar nicht so viel "verloren geht", wie oft angenommen. Dies wird an verschiedenen Beispielen gezeigt. Ja es wird sogar bewiesen, daß bei einer Erhöhung der Rechengenauigkeit um 1 Stellen sich auch die Genauigkeit der Ergebnisintervalle um mindestens 1 Stellen erhöhen muß. Diese Tatsache wird für eine allgemeine Klasse von Algorithmen bewiesen (Satz 1.3). Es wurde ein Programmpaket für eine Intervallarithmetik mit variabler Genauigkeit geschrieben und implementiert. Das Paket beinhaltet ein Integer- und Floatingpaket und ist für die UNIVAC 1108 der Universität Karlsruhe verfügbar. Mittels dieses Programmpakets wurde nachgewiesen, daß beide Intervallalgorithmen bei etwas höherer Rechengenauigkeit bereits sehr gute Ergebnisse liefern. Diese können mit geringem Mehraufwand sogar noch um ein Vielfaches verbessert werden. Es werden Beispiele hierfür angegeben. Daneben wird der Aufwand für diese und andere Intervallalgorithmen zur Einschließung der Lösung eines linearen Gleichungssystems untersucht. Es wird gezeigt, daß die Variante von Crout und Doolittle trotz besserer Fehleranalyse i.a. etwa gleich gute Lösungsintervalle wie der intervallmäßige Gauß- und Gauß-Jordan-Algorithmus liefert. Es ergab sich weiter, daß im Sinne des Absolutbetrages stark differierende

Lösungskomponenten offenbar keinen Einfluß auf die Genauigkeit der Ergebnisintervalle haben. Sodann wurde die Empfindlichkeit der Algorithmen untersucht und gezeigt, wo die Durchführung der Rechnung mit höherer Genauigkeit angebracht ist. Bei Betrachtung der Defektiteration stellte sich heraus, daß hier wie für andere Algorithmen die Genauigkeit der Lösung erheblich verbessert werden kann, und zwar mit geringem Aufwand (siehe auch Kapitel 2). Schließlich wird zum Ende des ersten Kapitels noch kurz auf die exakte Lösung von linearen Gleichungssystemen eingegangen.

In Kapitel 2 wird ein neues Verfahren zur Einschließung der Lösung eines linearen Gleichungssystems angegeben. Im Gegensatz zu bekannten Verfahren wird ausgehend von einer Gleitkommanäherung der Lösung eine garantierte Einschließung berechnet. Die Richtigkeit des Verfahrens wird mittels des allgemeinen Einschließungssatzes 2.2 bewiesen. Basierend auf diesem Satz und den Sätzen 2.3 bis 2.5 werden im dritten Kapitel noch weitere Einschließungsverfahren behandelt. Die Sätze gestatten es, für einen gegebenen Intervallvektor zu beweisen, daß er die Lösung des gegebenen Gleichungssystems enthält. Darüberhinaus wird algorithmisch bewiesen, daß die Eingabematrix nicht singulär ist. Weiter werden zahlreiche Verbesserungen gezeigt, die den Algorithmus beschleunigen und die Genauigkeit der Lösungsintervalle erheblich verbessern. Dies sind z.B. die Einschließung des Defekts, ein Prinzip, daß auch in vielen anderen Algorithmen angewendet werden kann. Weiter etwa die ε-Erweiterung, die eine Konvergenzbeschleunigung des Intervallverfahrens bedeutet. Durch die Verwendung eines langen Akkumulators zur Berechnung der auftretenden Skalarprodukte konnte der Algorithmus noch wesentlich verbessert werden. Es werden Abschätzungen angegeben, für welche Konditionszahlen der Eingabematrix der Algorithmus durchführbar ist. Für den Spezialfall diagonaldominanter Matrizen wird eine erheblich beschleunigte Version des Algorithmus beschrieben. Es wird gezeigt, daß der Aufwand des Intervallalgorithmus höchstens der sechsfache des des gleitkommamäßigen Gaußalgorithmus ist. Sodann wird kurz darauf eingegangen, wie bei einem eventuellen Versagen des Algorithmus die

Genauigkeit in optimaler Weise erhöht werden kann. Im letzten Abschnitt des zweiten Kapitels werden zahlreiche numerische Beispiele gegeben. Es zeigt sich insbesondere, daß die im Algorithmus auftretenden Intervalle kaum die Tendenz haben, sich aufzublähen sondern im Gegenteil auf die Lösung mit annähernd gleichbleibendem Durchmesser "hinkonvergieren". Insbesondere werden Beispiele mit extrem hohen Konditionszahlen betrachtet. Dabei wurden Gleichungssysteme bis zum Grad 200 untersucht, wobei die Zahl 200 durch den beschränkten Speicherplatz der UNIVAC 1108 der Universität Karlsruhe bedingt ist. Der Grad 200 stellt keine Schranke für den Algorithmus dar. Weiter werden verschiedene Varianten des Algorithmus verglichen, wobei die mit Abstand beste als Paket in die Rechenzentrumsbibliothek der UNIVAC 1108 der Universität Karlsruhe aufgenommen wurde.

Im dritten Kapitel schließlich werden weitere Anwendungsbeispiele der Einschließungssätze angegeben. So wird ein einfacher Algorithmus zum numerischen Beweis der Nicht-Singularität einer Matrix aufgezeigt. Dieser wie die anderen Algorithmen sind sowohl für (dichte) Punktmatrizen als auch für Intervallmatrizen brauchbar. Mittels des allgemeineren Satzes 3.1 lassen sich Aussagen über die Eigenwerte einer Matrix treffen. So wird ein Algorithmus zum numerischen Beweis, daß die Eigenwerte einer Matrix positiven Realteil haben sowie einer, daß eine symmetrische Matrix positiv definit ist angegeben. Anhand von Beispielen wird die Wirkungsweise der Algorithmen demonstriert. Darüberhinaus werden Algorithmen zur Einschließung der Inversen einer Matrix mit dem dazugehörigen Beweis der Nicht-Singularität gegeben. Im letzten Abschnitt von Kapitel 3 schließlich werden - wieder aus Gleitkommanäherungen - Verfahren zur Einschließung der Eigenwerte und Eigenvektoren einer beliebigen reellen Matrix angegeben. Ein herausragendes Ergebnis ist, daß die Eindeutigkeit von Eigenwert und Eigenvektor im Lösungsintervall getrennt nachgewiesen werden kann, was den Algorithmus auch zur Bestimmung von Ausschließungsintervallen für Eigenwerte und solchen für Eigenvektoren tauglich macht.

Die Einschließungssätze aus Kapitel 2 haben über die aufgeführten Algorithmen hinaus noch ein weites Anwendungsfeld. Insbesondere können z.B. die hier aufgezeigten Algorithmen leicht auf den komplexen Fall umgeschrieben werden. Weiter gibt es Anwendungen bei Randwertproblemen, Differentialgleichungen Nullstellen von Polynomen etc. die in Zukunft noch untersucht werden.

Beim Entwurf eines Algorithmus geht man zunächst von einem reellen Verfahren aus. Es wird eine direkte Formel (wie das Gauß-Verfahren) oder eine Rekursionsformel (wie das Newton-Verfahren) angegeben. Im ersten Fall möchte man vielleicht ein Näherungsergebnis verbessern, so daß wir uns in unserer Betrachtung auf eine Rekursionsformel beschränken. Hierfür wird unter gewissen Voraussetzungen die Konvergenz (im Körper der reellen Zahlen) mathematisch bewiesen. Um aus der Rekursionsformel zu einem Algorithmus zu gelangen, ist letztendlich ein Computer-Programm zu erstellen. Hier wird bei bloßer Ersetzung der reellen Operationen durch (genäherte) Gleitkommaoperationen ein Fehler begangen. Um eine beweisbare Aussage über das erhaltene Gleitkommaergebnis zu bekommen, sind sämtliche möglicherweise aufgetretenen Rundungsfehler des Algorithmus zu erfassen. Kann ein Konvergenzkriterium für das Gleitkommaverfahren angegeben werden, muß dessen Gültigkeit auf dem Rechner wieder unter Berücksichtigung aller Rundungsfehler nachgeprüft werden. In allen hier entwickelten Algorithmen ist das Erfülltsein einer Kontraktionsbedingung notwendig und hinreichend für die Konvergenz des reellen Verfahrens. Die Kontraktionsbedingung wird in den vorliegenden Algorithmen nicht explizit sondern implizit während der Durchführung geprüft. Die erfolgreiche Beendigung der hier angegebenen Algorithmen ist mit dem impliziten Beweis der Kontraktionsbedingung gleichzusetzen. Gleichzeitig werden in diesem Fall Schranken ausgegeben, die bewiesenermaßen die gesuchte Lösung enthalten. Bei bekannten Verfahren ist ein Starten des Algorithmus überhaupt nur möglich, wenn die Konvergenz durch Normabschätzungen a priori gesichert ist und damit bereits eine Einschließung der Lösung bekannt ist. Das Verfahren beschränkt sich dann darauf, die Lösungs-

schranken zu verbessern. Die hier beschriebenen Algorithmen beweisen für eine Gleitkommanäherung Fehlerabschätzungen. Der Beweis kann je nach Kondition des Problems im Rahmen der jeweiligen Genauigkeit geführt werden oder nicht. In diesem Sinn sind die hier behandelten Algorithmen als ein hinreichendes Kriterium zu betrachten. Die schmale Kluft zwischen notwendig und hinreichend wird hierbei allein durch die Rundungsfehler bestimmt. Hat das Kriterium einmal gegriffen werden überdies Methoden angegeben, die es bei gleichbleibender Grundgenauigkeit der Rechnung (!) allein unter Verwendung eines langen Akkumulators gestatten, die Lösungsschranken beliebig zu verfeinern.

In einem Appendix wird noch kurz auf die Darstellung der reellen Zahlen auf dem Rechner eingegangen. Es wird gezeigt, daß ein Homomorphismus der reellen Zahlen auf die "Rechnerzahlen" nicht möglich ist. Darüberhinaus wird deutlich gemacht, daß das Assoziativgesetz auf dem Rechner praktisch keine Gültigkeit haben kann. Hieraus folgt insbesondere, daß man von der algebraischen Struktur der "Rechnerzahlen" über die in Kulisch 1 entwickelten Eigenschaften nicht viel mehr erwarten zu können scheint.

Im Anhang wird der Quelltext der FORTRAN- und Assemblerunterprogramme des in der Programmbibliothek des Rechenzentrums der Universität Karlsruhe installierten Programmpakets angegeben.

0. Einleitung

Seit den ersten grundlegenden Arbeiten der Intervallrechnung besteht die weit verbreitete Ansicht, bei Anwendung der Intervallrechnung blähten sich die Intervalle sofort auf und bereits nach wenigen Schritten sei das Ergebnis unbrauchbar. Dieses Gerücht bekommt insbesondere Nahrung durch Ratschläge "Ein reelles Verfahren kann sofort in ein Intervallverfahren umgeschrieben werden, wenn statt reeller Zahlen jeweils Intervalle und statt der reellen Operationen Intervalloperationen eingesetzt werden". Man erkannte schnell, daß dieses Versprechen nicht eingehalten werden kann. Betrachten wir als Beispiel das Newton-Verfahren

$$x^{n+1} := x^n - f(x^n) \,/\, f'(x^n) \,. \tag{1}$$

Nach bekannten Abschätzungen folgt sofort

$$d(x^{n+1}) = d(x^n) + d(f(x^n) \,/\, f'(x^n)) \geq d(x^n) \,,$$

das Verfahren kann also in der vorgeschlagenen Weise gar nicht gegen die Lösung konvergieren. Offenbar müssen beim Entwurf von Intervallalgorithmen feinere Methoden angewandt werden.

Um das Problem besser in den Griff zu bekommen, soll zunächst die Aufgabenstellung präzisiert werden und anschließend Möglichkeiten der Lösung diskutiert werden. Gegeben sei also ein mathematisches Problem, gesucht dessen Lösung, und zwar mit Hilfe eines Computers. Was in der numerischen Mathematik zur Lösung eines Problems angeboten wird, sind Algorithmen. Knuth verwendet gleich 9 Seiten in seinem ersten Band, um diesen Begriff näher zu fassen. Man könnte eine Definition etwa so formulieren:

> "Ein Algorithmus ist ein endliches Verfahren, in dem der erste Schritt und zu jedem Schritt auch der folgende Schritt eindeutig gestgelegt sind".

Ohne auf eine genauere Interpretation eingehen zu wollen sei nur betont, daß der Algorithmus determinstisch sein muß und nach endlich vielen Schritten enden muß. Ein solcher Algorithmus ist etwa das Newton-Verfahren mit Abbruchkriterium, das unter gewissen Voraussetzungen konvergiert. Die Konvergenz setzt voraus, daß die auftretenden Zahlen reelle Zahlen sind, d.h. insbesondere unendliche und zwar auch nicht periodische Dezimalbrüche sein können. Um aus dem theoretischen Verfahren ein praktisches zu machen, werden z.B. in (1) für die Variablen Zahlen eingesetzt. Doch bereits bei einfachsten Funktionen wie etwa $f(x) = x - \pi$ steht die Praxis zunächst vor schier unüberwindlichen Schwierigkeiten. Setzt man nämlich $x^0 := 1$ und will (1) anwenden, heißt das

$$x^1 := 1 - (1 - \pi) / 1 .$$

Der Ausdruck in der Klammer $1 - \pi$ müßte bei einer "wirklichen" Anwendung von (1) exakt berechnet werden, d.h. es müßten unendlich viele Stellen angegeben werden. Genau genommen müßte das bereits bei der Angabe von f geschehen. Denn die Definition "π ist das doppelte der kleinsten positiven Nullstelle von $\cos x = o$" gibt noch keine Ziffern und $2 \cdot \arccos(o)$ setzt wieder einen Algorithmus voraus. Kurzum sind die wenigsten Zahlen endlich darstellbar, will man nicht in irgendwelchen algebraischen oder transzendenten Erweiterungskörpern der rationalen Zahlen rechnen. Um also überhaupt erst anfangen zu können, müssen die reellen Zahlen durch geeignete rationale Zahlen "approximiert" werden. Dabei wird jedesmal ein Fehler begangen. Schreibt man etwa 3.14159 oder 355/113 für π, so ergibt sich ein

$$\text{absoluter Fehler } |\pi - 3.14159| < 2.7_{10}\text{-}6 \text{ und } |\pi - 355/113| < 2.7_{10}\text{-}7 \text{ und ein}$$

$$\text{relativer Fehler } \left|\frac{\pi - 3.14159}{\pi}\right| < 8.5_{10}\text{-}7 \text{ und } \left|\frac{\pi - 355/113}{\pi}\right| < 8.5_{10}\text{-}8 .$$

Wie man sieht, ist die Approximation von π mit 6 Ziffern unterschiedlich gut ausgefallen. Zurück zur Praxis. Das Endziel des Obergangs vom theoretischen Verfahren zum praktischen Algorithmus ist die Implementierung auf einer Rechenanlage. Nun

wird auf großen Rechenanlagen mit verschiedenen Zahlensystemen gerechnet. Nehmen wir an, unsere Rechenanlage besitzt ein Gleitkommasystem zur Basis 10, was ohnehin am sinnvollsten ist. Der Exponentenbereich sei im Moment so groß angenommen, daß kein Überlauf eintritt (ein etwaiger Abbruch wegen Exponentenüberlauf ist maschinenabhängig und daher für unsere Betrachtungen weniger interessant). Nehmen wir weiter an, der Mantissenbereich ist dreistellig, d.h. alle Zahlen der Form $\pm 0.abc_{10}e$ sind zugelassen, wo a,b,c Ziffern und der Exponent e eine ganze Zahl ist. Die beste Näherung in diesem System für die Zahl π wäre also $0.314_{10}1$ mit einem relativen Fehler von $< 5.1_{10}\text{-}4$. Wendet man das Newton-Verfahren (1) auf $f(x) = x - \pi$ an mit einem Startwert $x^0 := 2$ und der Näherung $0.314_{10}1$ für π, bekommt man bereits in einem Schritt $x^1 = 3.14$, also die beste mögliche Näherung in unserem Gleitkommasystem. Es können jedoch auch bei einem so einfachen Algorithmus und einfachsten Funktionen erhebliche Ungenauigkeiten auftreten. Nimmt man etwa

$$(2) \qquad f(x) = 1.24x^2 - 121x + 2950, \text{ also } f'(x) = 2.48x - 121$$

(man beachte, daß alle Koeffizienten exakt in dem genannten Gleitkommasystem mit 3-stelliger Mantisse darstellbar sind), so ist die gesuchte Nullstelle zwischen 49 und 51 auf 3 Stelle gerundet 50.0 (dies ist sogar die exakte Nullstelle); berechnet man mit unserem Rechner nach dem Newton-Verfahren (1) für $x^0 = 49$ die nächste Näherung x^1 mittels des Horner-Schemas, so erhält man (zur Übersichtlichkeit ohne Exponent geschrieben; ein Pfeil bedeutet Rundung auf dem Rechner zur nächstgelegenen Gleitkommazahl)

$$\begin{aligned} &f(49) = (1.24 * 49 - 121) * 49 + 2950 = (60.76 - 121) * 49 + 2950 \longrightarrow \\ &\quad \longrightarrow (60.8 - 121) * 49 + 2950 = -60.2 * 49 + 2950 = -2949.8 + 2950 \longrightarrow \\ &\quad \longrightarrow -2950 + 2950 = o \qquad (3) \\ &f'(49) = 2.48 * 49 - 121 = 121.52 - 121 \longrightarrow 122 - 121 = 1 , \end{aligned}$$

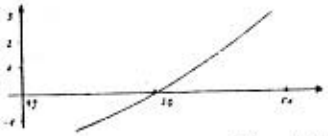
also

$$x^1 = x^0 - f(x^0)/f'(x^0) = 49 - 0/1 = 49 = x^0 .$$

Der Rechner wäre in diesem Fall wegen $x^0 = x^1$ (!) natürlicherweise davon ausgegangen, daß $x = 49$ die exakte Lösung von (2) für $49 \leq x \leq 51$ ist. Der Fehler ist offenkundig: Statt des wahren Wertes $f(49) = -1.76$ hat der Rechner $f(49) = 0$ berechnet, nur verursacht durch den internen Rundungsfehler. Das "wirkliche" Newton-Verfahren, nämlich das reelle nicht gerundete, hätte die Folge

$$x^0 = 49,\ x^1 = 52.38\ldots,\ x^2 = 50.79\ldots,\ x^3 = 50.156\ldots,\ x^4 = 50.0089\ldots,$$
$$x^5 = 50.000\ldots,\ x^6 = 50.000\ldots$$

geliefert. Ähnliche (auch noch viel schlimmere) Beispiele lassen sich ebenso für andere Gleitkommasysteme konstruieren. Nebenstehend ist ein Bild der Funktion f aus (2) zwischen 49 und 51 gezeichnet. Bedenkt man weiter, welchen Fehler hier



ein paar Operationen angerichtet haben, kann man sich ausmalen, welche verheerende Ausmaße der Fehler in einem Programm annehmen kann, in dem gleich Millionen von Operationen durchgeführt werden, so wie es heute gang und gäbe ist.

Offenbar wird deutlich, welchen Wert man Zahlen beizumessen hat, die Endwerte eines Algorithmus sind, der nur mit Gleitkommazahlen rechnet ohne irgendwelche Vorkehrungen den Fehler betreffend (die Endwerte können nicht als Ergebnisse bezeichnet werden). Selbstverständlich heißt das nicht, daß Gleitkommarechnungen von vornherein sinnlos sind; nur müß(t)en sie immer in Verbindung mit einer Fehlerabschätzung durchgeführt werden. Im obigen Beispiel hieße das zum Beispiel für die Berechnung der Ableitung (siehe (3)), daß man bei der Multiplikation $2.48 * 49$ nur aussagen kann, daß das Ergebnis zwischen 121 und 122 liegt, dementsprechend der Wert der Ableitung zwischen 0 und 1. Würde man die Berechnung der Ableitung mit 4 rechnenden Stellen saldieren, könnte bereits $0.5 \leq f'(49) \leq 0.6$ verbindlich ausgesagt werden und bei 5 rechnenden Ziffern bereits der exakte Wert $f'(49) = 0.52$ garantiert werden. Entsprechendes gilt für die Berechnung von $f(49)$.

Ein reelles Verfahren ohne Vorkehrungen auf Gleitkommazahlen zu übertragen ist nicht zulässig; die erhaltenen Werte als Lösung des reellen Problems zu bezeichnen ohne eine entsprechende Fehlerabschätzung ist nicht statthaft. Jeder Algorithmus auf dem Rechner ausgeführt muß eine Fehlerabschätzung in irgendeiner Form enthalten. Es sind so z.B. bereits viele Intervallverfahren bekannt, die wahre Lösungen liefern, d.h. die Garantie geben, daß die Lösung zwischen zwei angegebenen Gleitkommazahlen liegt. In der Zukunft gilt es daran zu gehen, diese Algorithmen weiter zu verbessern, um noch schärfere Schranken angeben zu können.

Nun zum Vorwurf, die Intervallalgorithmen würden die Intervalle sofort aufblähen. Zum einen ist dies eine recht globale Behauptung, die im speziellen sicher nicht zutrifft (siehe Kapitel 2 dieser Arbeit). Andererseits ist es falsch, dies der Intervallrechnung anzulasten. Betrachten wir die Lösung eines linearen Gleichungssystems nach der Gauß'schen Eliminationsmethode. Wilkinson hat zahlreiche Abschätzungen für die Genauigkeit des Gleitkommaergebnisses in Bezug auf die wahre Lösung angegeben. Diese Abschätzungen sind naturgemäß Überschätzungen, denn für eine genaue Abschätzung müßte man den ganzen Algorithmus wiederholen und als Formel mit einbringen. Der Sinn von Fehlerabschätzungen ist ja gerade, einfache Formeln anzugeben, welche den gemachten Fehler dennoch möglichst genau abschätzen. Rechnet man den Gaußalgorithmus mit Intervallen statt mit reellen Zahlen (d.h. Überall werden reelle Zahlen durch Intervalle und reelle Operationen durch Intervalloperationen ersetzt), bekommt man ein Ergebnisintervall. Sicher ist, daß die wahre Lösung in diesem Intervall liegt. Eine Intervalloperation $\circledast$ ist aber definiert als

$$C = A \circledast B = \{a * b \mid a \in A,\ b \in B\} \quad \text{für} \quad * \in \{+,-,\times,/\},$$

d.h. zu jedem $c \in C$ gibt es ein Paar $(a,b) \in (A,B)$ mit $a * b = c$. Da C in dem jeweiligen Gleitkommasystem dargestellt werden muß, wird für die untere bzw. obere Grenze von C jeweils die größte bzw. kleinste Gleitkommazahl eingesetzt, die gerade noch kleiner bzw. größer ist, resp. Berechnet man $a * b$, wo a,b reell sind,

weiß man a priori nicht, ob a * b auf dem Rechner nach oben oder unten gerundet wird. Daher muß man bei einem allgemeinen Gleichungssystem Ax = b davon ausgehen, daß bei der Ersetzung von a * b durch die Intervalloperation a * b sowohl die untere als auch die obere Grenze angenommen werden kann. Das wiederum heißt genau, daß die intervallmäßige Lösung eines linearen Gleichungssystems die beste bekannte Fehlerabschätzung für den Wert der Gleitkommarechnung darstellt (die Fehlerabschätzungen von Wilkinson sind außerdem in der Praxis dann nicht anwendbar, solange keine garantierte Abschätzung der Konditionszahl der Ausgangsmatrix vorliegt, was in den seltensten Fällen zutrifft). Ist also das Ergebnisintervall recht groß oder versagt gar die Rechnung, weil in einer Pivotspalte nur Nullintervalle auftreten, so heißt dies gerade, daß man auf den Ausgabewert der Gleitkommarechnung gerade soviel vertrauen kann, wie das Intervall groß ist oder überhaupt nichts mehr aussagen kann. Daß es trotzdem Wege und Mittel gibt, die wahre Lösung dennoch besser einzuschließen, hat einen anderen Grund. Man geht von der Erfahrung aus, daß die Grenzen der Fehlerabschätzung nicht voll ausgeschöpft werden (das ändert nichts an der Tatsache, daß es möglich ist!). Sodann gibt man mit geeigneten Methoden (siehe Kapitel 2 dieser Arbeit) eine Abschätzung für den Fehler der Gleitkommawerte an, die dann wesentlich besser ausfallen kann. Das heißt nichts anderes als: ein anderes Verfahren liefert andere Abschätzungen. Tritt allerdings der Fall ein, daß die Gleitkommawerte zu schlecht sind, wird wahrheitsgemäß deren Fehler angegeben, der dann entsprechend groß ausfällt.

In den folgenden Kapiteln werden zahlreiche Wege und Algorithmen angegeben, die ein vorzeitiges Aufblähen der Intervalle wirksam unterdrücken und im Regelfall eine sehr genaue Einschließung der Lösung angeben.

1. Intervallmäßige Lösung von linearen Gleichungssystemen

Eine naheliegende Verallgemeinerung des Gauß'schen Algorithmus für reelle Zahlen besteht in der bloßen Ersetzung der reellen Zahlen durch Intervalle und der reellen Operationen durch Intervalloperationen. Die erhaltenen Lösungsintervalle enthalten die exakte Lösung des reellen Gleichungssystems (für einen Beweis siehe z.B. Alefeld/Herzberger). Entsprechende Aussagen gelten für das Gauß-Jordan-Verfahren. In Alefeld/Herzberger wurde gezeigt, daß das Intervall-Gauß-Verfahren mindestens dann eine Einschließung liefert, wenn die Ausgangsmatrix streng diagonaldominant ist. Die Voraussetzung ist recht scharf (läßt sich allerdings manchmal durch die Methode von Hansen nachträglich herstellen). Doch selbst wenn diese Voraussetzung nicht erfüllt ist, kann das Verfahren gerechnet werden in der Hoffnung, daß in keiner Pivotspalte nur Nullintervalle auftreten. Dies ist oft auch noch der Fall, wenn die Ausgangsmatrix nicht streng diagonaldominant oder keine M-Matrix war.

Mancheiner wird das Verfahren bereits programmiert und festgestellt haben, daß sich die Lösungsintervalle rasch aufblähen und oft numerische Singularität auftritt (d.h., eine Pivotspalte besteht nur aus Nullintervallen). Der Hauptgrund hierfür liegt in der geringen Stellenzahl, mit der gerechnet wurde. Nicht zuletzt auch folgendes Beispiel gab Anlaß, das Intervall-Gauß-Verfahren mit höherer Genauigkeit durchzurechnen.

Betrachten wir die Funktion

$$(1.1) \qquad e^{-x} = \sum_{i=0}^{\infty} (-1)^i \cdot \frac{x^i}{x!} = 1 - x + \frac{x^2}{2!} - \frac{x^3}{3!} + - \ldots$$

Bekannterweise ist die Funktion für größere x von sehr kleinem absoluten Betrag gleichwohl in der Berechnung nach (1.1) sehr große Auslöschungen auftreten. Im folgenden wurde e^{-x} für x = 10(10)50 mit einer Intervallarithmetik mit maximaler Mantissenlänge 50 berechnet. In der nachstehenden Tabelle bezeichnet s_{max} den maximalen Summand dem Betrag nach in (1.1) für das jeweilige x, also

$$s_{max} = \max_i \frac{x^i}{i!} .$$

Wie man sieht ist etwa für x = 20 der Wert von e^{-x} bei 10^{-9}, während s_{max} bei 10^{+7} liegt. Bei der Aufsummierung gehen also durch Auslöschung mindestens h = 17 = 9 + 1 + 7 Stellen verloren, d.h. höchstens 33 Stellen können garantiert übrig bleiben; in der Tat bleiben 32 Stellen garantiert.

x	e^{-x}	s_{max}	garantierte Stellen	h	i_{max}
10	4.5_{10}-5	2.8_{10} 3	41	41	100
20	2.1_{10}-9	4.3_{10} 7	32	33	150
30	9.4_{10}-14	7.8_{10} 11	23	24	190
40	4.2_{10}-18	1.5_{10} 16	14	15	230
50	1.9_{10}-22	2.9_{10} 20	5	7	220

Die entsprechenden Werte sind in der letzten Spalte aufgetragen. Wie man sieht, gehen selbst bei x = 50 und 270 Additionen nur 2 Stellen verloren durch den Auffang von Rundungsfehlern in den Intervalloperationen. Diese Beobachtung gibt Anlaß zu der Vermutung, daß genaue Intervalle erhalten werden können, wenn nur genügend genau gerechnet wird. In den folgenden Abschnitten wird sich diese Vermutung bestätigen.

1.a) Bemerkungen zum Intervall-Gauß- und Gauß-Jordan-Algorithmus

In der Einleitung wurde bereits versucht zu verdeutlichen, daß ein "ungenaues" Intervall nur auf ein schlecht konditioniertes Problem deutet und nichts anderes heißt, als daß die Gleitkommanäherung ebenfalls sehr schlecht sein kann. Hat man nur die Gleitkommanäherung vor sich, weiß man zunächst nicht viel, denn selbst ein kleiner Defekt sagt zunächst nichts über den relativen Fehler der Gleitkommanäherung aus, wenn nicht Abschätzungen über die Kondition der Matrix vorliegen.

Z.B. ist in

$$100x + 99y = 199$$
$$99x + 98y = 197$$

der Defekt für $\tilde{x} = (0.1;2)$ verschwindend klein, nämlich $\Delta x = b - A\tilde{x} = (0;0.01)$. Man ist geneigt, $\tilde{x}$ für eine gute Näherung der Lösung zu halten, obwohl in Wirklichkeit zur exakten Lösung (1;1) kaum Ähnlichkeit besteht. Die direkte intervallmäßige Durchführung des Gauß-Algorithmus vergröbert die Ergebnisintervalle scheinbar dadurch, daß abhängige Intervalle wie nicht abhängi verwendet werden. Sind z.B. A,B,C positive Intervalle und D = A + B und E = -A + C, so wird zur Berechnung der oberen Grenzen von D und E einmal die obere und einmal die untere Grenze von A verwendet. Im Nachhinein sind solche Beobachtungen leicht zu machen, und zwar für spezielle Probleme. Im allgemeinen Fall ist a priori nicht bekannt, wann nach oben und wann nach unten gerundet wird. Tatsächlich gibt es Beispiele, wo immer "ungünstig" gerundet wird. Nehmen wir die nach unten gerichtete Rundung in einem 13-Bit Computer und das Gleichungssystem

$$6875x + 2922y - 3113z = -1391$$
$$3013x + 1941y - 6039z = -4788$$
$$3162x + 5988y - 4937z = 762 \quad ,$$

so lautet die Gleitkommanäherung $\tilde{x}$ und die Intervall-Gaußeinschließung X

$$\tilde{x} = \begin{pmatrix} -0.192291 \\ 1.09277 \\ 1.04785 \end{pmatrix} \qquad X = \begin{pmatrix} [-0.193024, -0.191467] \\ [\ 1.09229, \ 1.09424] \\ [\ 1.04785, \ 1.044907] \end{pmatrix} ,$$

während die auf 6 Stellen gerundete exakte Lösung

$$z = \begin{pmatrix} -0.192238 \\ 1.09303 \\ 1.04824 \end{pmatrix}$$

lautet. Die Gleitkommanäherung für z stimmt hier in der dritten Komponente mit

der linken Grenze der Intervall-Gaußeinschließung überein. Offenbar trat hier der Fall ein, daß an der entsprechenden Stelle der Gleitkommarechnung immer so gerundet wurde, daß die linke Intervallgrenze des entsprechenden Intervallschritts angenommen wurde. Die Weite des Intervalls für z ist in diesem Fall 10 Bit (von hinten), die Gleitkommanäherung weicht 3 Bit vom gerundeten exakten Wert ab.

Die naive Intervallrechnung erweitert nur die vier Grundrechenoperationen $+, -, \cdot, /$, sowie die von der Ordnungsrelation $\leq$ herrührende Ordnungsstruktur auf Intervalle und durchläuft damit Algorithmen. Auf dieser Stufe steht man offenbar, wenn man einen gegebenen Algorithmus für gerundetes Rechnen mit Intervallen durchrechnet.

In der nicht-naiven Intervallrechnung macht man sich die Erkenntnisse zu Nutze, daß Intervalle Mengen sind und mit der Inklusion eine zweite Ordnungsstruktur tragen. Die Relation "Element aus", die Inklusion, sowie das zugehörige Infimum und Supremum, das sind Durchschnitt und konvexe Hülle von Intervallen nimmt man als numerisch leicht ausführbare Operationen zu den Grundrechenoperationen hinzu. Dadurch erhält man insgesamt die folgenden Operationen:

$$+, -, \cdot, /, \in, \subseteq, \cap, \overline{\cup},$$

Praktisch gehören alle bisher veroffentlichten gut funktionierenden Algorithmen der Intervallrechnung der nicht-naiven Intervallrechnung an und machen in Herleitung und Anwendung von der erweiterten Menge der Grundrechenoperationen Gebrauch.

Mit diesen Begriffsbildungen könnte man die bisherigen Erfahrungen wie folgt zusammenfassen: Nicht-naive Intervallrechnung ist ein gut brauchbares und nützliches Hilfsmittel der Numerik. Naive Intervallrechnung aber führt i.a.

zu unbrauchbaren Fehlerschranken. Dieses Ergebnis ist aber nur beschränkt richtig. Richtig ist es, soweit es die nicht-naive Intervallrechnung betrifft. Die bezüglich der naiven Intervallrechnung gemachte Aussage soll hingegen noch genauer analysiert werden.

Über die naive Intervallrechnung gibt es einen ganz zentralen und wohl bekannten Satz, der mit einfachen Worten und ohne großen Formalismus etwa folgendes aussagt (bezüglich einer Präzisierung dieser Ergebnisse siehe etwa das Buch von Alefeld und Herzberger, S. 28 ff): Betrachten wir einen numerischen Algorithmus bzw. einen arithmetischen Ausdruck für reelle oder komplexe Argumente. Von dem zu Grunde liegenden Funktionsausdruck nehmen wir an, daß er Lipschitz-stetig ist. Wertet man diesen Funktionsausdruck intervallmäßig aus, d.h. setzt man an Stelle der Argumente Intervalle ein, so liefert die intervallmäßige Auswertung eine Überschätzung des Wertebereiches der Funktion im Bereich der Argumentintervalle.

Läßt man die Durchmesser aller Argumentintervalle gegen Null gehen, so konvergiert die Überschätzung des Wertebereiches gegen Null oder die intervallmäßige Auswertung der Funktion gegen den Wertebereich bzw. in der Grenze gegen den Wert der Funktion. Bei geeigneter Darstellung der Funktion geht die Überschätzung des Wertebereiches sogar quadratisch mit dem Durchmesser der Argumentintervalle gegen Null.

Bei jedem Algorithmus tritt nun aufgrund der unvermeidlichen Rundungsfehler ein Genauigkeitsverlust ein. Dieser Genauigkeitsverlust nimmt im allgemeinen bei Vergrößerung der mitgeführten Ziffernlänge nicht zu. D.h. führt man einen Algorithmus mit 10-stelligen Ausgangsdaten 10-stellig durch und verliert man während der Rechnung 5 Stellen durch Rundungsfehler, so verliert man bei 15-stelliger Rechnung durch Rundungsfehler nicht mehr als 5 Stellen. Führt man nun dieselbe Rechnung mit Intervallen durch und beginnt mit denselben 10-stelligen Ausgangsdaten, also mit Punktintervallen, so bewirken die Rundungsfehler eine gewisse

Aufblähung der Intervalle. Diese wird dann noch verstärkt durch die Überschätzung des Wertebereiches der Funktion durch die Intervallrechnung. Im Fall von linearen Gleichungssystemen tritt jedoch durch die Intervallrechnung praktisch kein Genauigkeitsverlust mehr ein. Man kann erwarten, daß wenn bei 10-stelliger Rechnung noch 7 Stellen übrig bleiben, daß dann bei 15-stelliger Rechnung wenigstens $12 = 7 + 5$ Stellen erhalten bleiben. Hier kann sogar durch eine sehr preiswerte Nachkorrektur das Ergebnis um Größenordnungen verbessert werden, wie die Beispiele später zeigen werden.

Man fragt sich, wie ein solcher Effekt zustande kommt. Beim Intervall-Gaußalgorithmus kann es (bei zu geringer Mantissenlänge) passieren, daß in einer Pivotspalte nur Nullintervalle auftreten. In diesem Fall kann der Algorithmus nicht fortgeführt werden. Andererseits tritt erfahrungsgemäß bei einer erfolgreich durchgeführten Gaußelimination (ohne Rücksubstitution) während der ganzen Rechnung praktisch kein Nullintervall auf. Gerade beim Rechnen mit Nullintervallen ist das Ergebnis (bei Multiplikation und Division) ja gar nicht mehr von allen Intervallgrenzen der Operanden abhängig, d.h. in diesem Fall träte eine Überschätzung des Wertebereichs ein. Sobald die Vorwärtselimination erfolgreich durchgeführt wurde, kann bei der Rückwärtselimination keine Division durch Nullintervalle mehr auftreten, d.h. der gesamte Algorithmus kann erfolgreich beendet werden. Gleichwohl können die Ergebnisintervalle große Durchmesser haben und sogar Nullintervalle sein. Trotzdem steckt in der intervallmäßigen LU-Zerlegung soviel Information, daß mit wenig Mehraufwand sehr gute Ergebnisse erzielt werden können (siehe unten).

Unter der Annahme, daß während der Durchführung des Intervall-Gauß-Algorithmus keine Nullintervalle auftreten soll jetzt gezeigt werden, daß sich bei Erhöhung der Genauigkeit der Rechnung um 1 Stellen die Genauigkeit der Ergebnisintervalle um wenigstens 1 Stellen erhöht. Wir zeigen dies unabhängig vom Intervall-Gauß-Algorithmus für jeden Intervall-Algorithmus, der ausgehend von Punktintervallen die vier Grundrechnungsarten verwendet.

Definition 1.1 Für $X \in \mathbb{IR}$ sei

$$\|X\| := \min_{x \in X} |x| .$$

Definition 1.2 Für $X \in \mathbb{IR}$ und $o \notin X$ bezeichnen wir

$$\Delta(X) := \frac{d(X)}{\|X\|}$$

als relativen Fehler der Intervalls X.

Geht man davon aus, daß ein Intervall X eine exakte Lösung $\hat{x}$ einschließen soll, ist offenbar

$$\Delta(X) = \max_{x,y \in X} \frac{|x-y|}{|x|} ,$$

also insbesondere

$$\max_{x \in X} \frac{|x - \hat{x}|}{|\hat{x}|} \leq \Delta(X) .$$

Demnach ist $\Delta(X)$ der "maximale relative Fehler" von X in Bezug auf $\hat{x}$.

Im Folgenden sei vorausgesetzt, daß für alle auftretenden Intervalle gilt:

$\alpha)$ $o \notin X$

$\beta)$ $\Delta(X) < 1$ ($\Leftrightarrow$ $d(X) < \|X\|$) .

Daraus folgt, daß $|X| = \|X\| + d(X) \approx \|X\|$.

Im Folgenden werde irgendein Algorithmus intervallmäßig ausgeführt. Alle Eingabedaten seien Punktintervalle. Die Basis des benutzten Gleitkommasystems

sei b. Es soll untersucht werden, wie sich die Genauigkeit verbessert, wenn die Rechnung mit l Stellen mehr in der Mantisse ausgeführt wird. Das entspricht einer Erhöhung der Genauigkeit um 1 Stellen. Die bei der ersten Rechnung auftretenden Intervalle werden mit A,B,..., die bei der um 1 Stellen genaueren Rechnung auftretenden Intervalle mit $A^*,B^*,\ldots$ bezeichnet, resp.

1) Rundung von reellen Zahlen.

Eine reelle Zahl a werde bei der ersten Rechnung zum Intervall A, bei der genaueren Rechnung zum Intervall A^* gerundet. Dann gilt

$$d(A^*) \leq b^{-1}\cdot d(A)$$

und wegen $\|A^*\| \geq \|A\|$

$$\Delta(A^*) = \frac{d(A^*)}{\|A^*\|} \leq b^{-1}\cdot\Delta(A) \ .$$

=> Bei der Rundung von reellen Zahlen wird bei Erhöhung der Genauigkeit um 1 Stellen der Durchmesser und der relative Fehler der Ergebnisintervalle um 1 Stellen verbessert.

2) Addition von Intervallen.

Es gelte

$$d(A^*) \leq b^{-1}\cdot d(A) \ ; \quad d(B^*) \leq b^{-1}\cdot d(B) \ ;$$
$$\Delta(A^*) \leq b^{-1}\cdot\Delta(A) \ ; \quad \Delta(B^*) \leq b^{-1}\cdot\Delta(B) \ .$$

Dann ist wegen der Monotonie

$$A^* \subseteq A \ , \quad B^* \subseteq B \ .$$

Also ist

$$d(A^* + B^*) = d(A^*) + d(B^*) \leq b^{-1}\cdot(d(A)+d(B)) =$$
$$= b^{-1}\cdot d(A+B)$$

und

$$\|A^* + B^*\| \geq \|A + B\| \ ,$$

da β) für alle aufgetretenen Intervalle, also insbesondere für A + B gilt.

Demnach ist

$$\Delta(A^* + B^*) = \frac{d(A^*+B^*)}{\|A^*+B^*\|} \leq b^{-1}\cdot\frac{d(A+B)}{\|A+B\|} = b^{-1}\cdot\Delta(A+B) \ .$$

=> Wird d und Δ von A und B um 1 Stellen verbessert, verbessert sich auch das Ergebnisintervall der Summe von A und B um 1 Stellen in d und Δ.

3) Subtraktion von Intervallen.

Der Fall wird durch Ersetzen von B durch -B auf 2) zurückgeführt.

4) Multiplikation von Intervallen.

Mit den Voraussetzungen von 2) ist für A,B > o

$$\begin{aligned} d(A^*\cdot B^*) &= |A^*|\cdot|B^*| - \|A^*\|\cdot\|B^*\| = \\ &= (\|A^*\| + d(A^*))\cdot(\|B^*\| + d(B^*)) - \|A^*\|\cdot\|B^*\| = \\ &= \|A^*\|\cdot d(B^*) + \|B^*\|\cdot d(A^*) + d(A^*)\cdot d(B^*) \leq \\ &\leq b^{-1}\cdot\{\|A^*\|\cdot d(B) + \|B^*\|\cdot d(A) + d(A)\cdot d(B)\} = \quad \text{nach } \beta) \\ &= b^{-1}\cdot\{\|A\|\cdot d(B) + \|B\|\cdot d(A) + d(A)\cdot d(B)\} = \\ &= b^{-1}\cdot(|A|\cdot|B| - \|A\|\cdot\|B\|) = \\ &= b^{-1}\cdot d(A\cdot B) \ . \end{aligned}$$

Wegen d(A) = d(-A) und $\|A\| = \|-A\|$ gilt die Schlußkette auch ohne die Voraussetzung A,B > o; es genügt $o \notin A,B$, also α) . Wegen $A^*\cdot B^* \subseteq A\cdot B$ und α) folgt $\|A^*\cdot B^*\| \geq \|A\cdot B\|$ und mit β) folgt weiter

$$\Delta(A^*\cdot B^*) = \frac{d(A^*\cdot B^*)}{\|A^*\cdot B^*\|} \leq b^{-1}\cdot\frac{d(A\cdot B)}{\|A^*\cdot B^*\|} \leq b^{-1}\cdot\frac{d(A\cdot B)}{\|A\cdot B\|} = b^{-1}\cdot\Delta(A\cdot B) \ .$$

=> Wird d und Δ von A und B um 1 Stellen verbessert, erhöht sich auch die Genauigkeit des Ergebnisintervalls des Produkts von A und B um 1 Stellen in d und Δ.

5) Division von Intervallen.

Mit den Voraussetzungen von 2) ist für A,B > o

$$d(A^*/B^*) = |A^*|/\|B^*\| - \|A^*\|/|B^*| =$$
$$= \frac{\|A^*\| + d(A^*)}{\|B^*\|} - \frac{\|A^*\|}{\|B^*\| + d(B^*)} =$$

$$= \frac{\|A^*\| \cdot d(B^*) + \|B^*\| \cdot d(A^*) + d(A^*) \cdot d(B^*)}{\|B^*\| \cdot (\|B^*\| + d(B^*))} \leq$$

$$\leq b^{-1} \cdot \frac{\|A^*\| \cdot d(B) + \|B^*\| \cdot d(A) + d(A) \cdot d(B)}{\|B^*\| \cdot |B^*|} \approx \quad \text{nach } \beta)$$

$$\approx b^{-1} \cdot \frac{\|A\| \cdot d(B) + \|B\| \cdot d(A) + d(A) \cdot d(B)}{\|B\| \cdot |B|} =$$

$$= b^{-1} \cdot (|A| / \|B\| - \|A\| / |B|) =$$

$$= b^{-1} \cdot d(A / B) \quad .$$

Wie unter 4) kann die Voraussetzung $A,B > o$ wegfallen. Weiter ist wie unter 4)

$$\Delta(A^* / B^*) = \frac{d(A^* / B^*)}{\|A^* / B^*\|} \leq b^{-1} \cdot \frac{d(A / B)}{\|A^* / B^*\|} \leq b^{-1} \cdot \frac{d(A / B)}{\|A / B\|} = b^{-1} \cdot \Delta(A / B)$$

wieder wegen $\beta)$ und $A^* / B^* \subseteq A / B$.

=> Wird d und Δ von A und B um 1 Stellen verbessert, erhöht sich auch die Genauigkeit des Ergebnisintervalls des Quotienten von A und B um 1 Stellen in d und Δ .

Nunmehr können wir folgenden Satz formulieren:

Satz 1.3 Gegeben sei ein Algorithmus zur Berechnung der Funktion $f : V_n\mathbb{R} \to IV_m\mathbb{R}$, in dem nur die Intervalloperationen $+,-,\times,/ : IT \to IT$ für ein Raster T vorkommen. Weiter seien für alle während des Algorithmus auftretenden Intervalle die Bedingungen $\alpha)$ und $\beta)$ erfüllt.
Wird dann der Algorithmus für die gleichen Eingabedaten in IS, also mit $+,-,\times,/ : IS \to IS$ für ein Raster S durchgeführt, wobei für alle Eingabedaten x (d.s. reelle Zahlen)

$$(*) \qquad |\Delta_S(x) - \nabla_S(x)| \leq b^{-1} \cdot |\Delta_T(x) - \nabla_T(x)|$$

erfüllt sei, so verbessert sich für alle Ergebnisintervalle der Durchmesser d und der relative Fehler Δ wenigstens um den Faktor b^{-1} (für Δ_S, ∇_S, Δ_T, ∇_T siehe Kulisch 1).

Beweis. Mit den Bezeichnungen von vorhin folgt aus $(*)$ sofort

$$d(A^*) \leq b^{-1} \cdot d(A) \quad \text{und} \quad \Delta(A^*) \leq b^{-1} \cdot \Delta(A) \ .$$

Alle während des Algorithmus auftretenden Intervalle sind entweder durch Rundung der Anfangsdaten entstanden oder sind Ergebnis der Verknüpfung zweier während der Rechnung bereits aufgetretenen Intervalle durch $+,-,\times$ oder $/$. In allen Fällen wurde nachgewiesen, daß sich jeweils der Durchmesser d und der relative Fehler Δ mindestens um den Faktor b^{-1} verbessern. Das gilt für alle während der der Rechnung aufgetretenen Intervalle, also insbesondere auch für die Ergebnisintervalle.

Korollar. Wird bei einer zweiten Durchführung des Intervall-Gauß- oder Intervall-Gauß-Jordan-Algorithmus die Rechengenauigkeit gegenüber vorher um 1 Stellen erhöht (1 Stellen mehr in der Mantisse zur Basis b) und sind $\alpha)$ und $\beta)$ erfüllt, so verbessert sich der Durchmesser und der relative Fehler der Ergebnisintervalle mindestens um den Faktor b^{-1}.

Wesentlich bei der Beweisführung ist, daß die Eingabedaten reelle Zahlen und exakt bekannt sind. Sind die Eingabedaten ihrerseits Intervalle, kann eine entsprechende Aussage allgemein nicht mehr gemacht werden.

Die Erfahrung hat gezeigt, daß die Voraussetzungen $\alpha)$ und $\beta)$ bei der intervallmäßigen Durchführung des Gauß- oder Gauß-Jordan-Algorithmus fast immer erfüllt sind. Im Wesentlichen tritt also einer der folgenden beiden Fälle ein:

a) Die Durchführung versagt, da in einer Pivotspalte nur Nullintervalle auftraten

b) Der Algorithmus ist durchführbar und es treten während der Rechnung keine Nullintervalle auf.

Natürlich könnte man leicht ein Gegenbeispiel konstruieren; in der Praxis kam

unter vielen hundert gerechneten Beispielen keines vor. Mit den obigen Überlegungen erklärt sich die beobachtete Linearität der Genauigkeitsübertragung.

Als stellvertretendes Beispiel betrachten wir das Gleichungssystem mit der 7 × 7-Hilbert-Matrix und rechter Seite (1,...,1). Es wurde nach dem Intervall-Gauß- und Intervall-Gauß-Jordan-Algorithmus mit Genauigkeiten 15,20,25,30 und 35 Dezimalstellen gerechnet. Das Ergebnis ist in der folgenden Tabelle wiedergegeben, und zwar wurden für alle 7 Komponenten die relativen Fehler angegeben.

IV-Gauß	15-stellig	20-stellig	25-stellig	30-stellig	35-stellig
1	> 1	$.11_{10}-3$	$.11_{10}-8$	$.11_{10}-13$	$.11_{10}-18$
2	$.34_{10}0$	$.29_{10}-5$	$.29_{10}-10$	$.29_{10}-15$	$.29_{10}-20$
3	$.18_{10}-1$	$.17_{10}-6$	$.17_{10}-11$	$.17_{10}-16$	$.17_{10}-21$
4	$.16_{10}-2$	$.16_{10}-7$	$.16_{10}-12$	$.16_{10}-17$	$.16_{10}-22$
5	$.26_{10}-3$	$.25_{10}-8$	$.25_{10}-13$	$.25_{10}-18$	$.25_{10}-23$
6	$.64_{10}-4$	$.64_{10}-9$	$.64_{10}-14$	$.64_{10}-19$	$.64_{10}-24$
7	$.58_{10}-4$	$.58_{10}-9$	$.58_{10}-14$	$.58_{10}-19$	$.58_{10}-24$
IV-Gauß-Jordan					
1	$.13_{10}-3$	$.13_{10}-8$	$.13_{10}-13$	$.13_{10}-18$	$.13_{10}-23$
2	$.11_{10}-3$	$.11_{10}-8$	$.11_{10}-13$	$.11_{10}-18$	$.11_{10}-23$
3	$.92_{10}-4$	$.92_{10}-9$	$.92_{10}-14$	$.92_{10}-19$	$.92_{10}-24$
4	$.80_{10}-4$	$.80_{10}-9$	$.80_{10}-14$	$.80_{10}-19$	$.80_{10}-24$
5	$.71_{10}-4$	$.71_{10}-9$	$.71_{10}-14$	$.71_{10}-19$	$.71_{10}-24$
6	$.64_{10}-4$	$.64_{10}-9$	$.64_{10}-14$	$.64_{10}-19$	$.64_{10}-24$
7	$.58_{10}-4$	$.58_{10}-9$	$.58_{10}-14$	$.58_{10}-19$	$.58_{10}-24$

Tabelle 1.1 Intervall-Gauß- und Gauß-Jordan-Algorithmus für verschiedene Genauigkeiten

Der relative Fehler > 1 beim 15-stelligen IV-Gauß-Algorithmus heißt, daß dort ein Nullintervall als Lösung ausgegeben wurde. Rechnet man das Gleichungssystem mit 10 Dezimalstellen Genauigkeit, so produzieren beide Algorithmen als Lösungsintervalle nur Nullintervalle. Bei 5 Dezimalstellen Genauigkeit versagen beide Algorithmen. Wie erwähnt steht das angegebene Beispiel für viele gerechnete Beispiele, wobei immer die gleich Beobachtung gemacht wurde: Bei Erhöhung der Rechengenauigkeit um k Stellen erhöht sich auch die Genauigkeit der Lösung um k Stellen. An obiger Tabelle beobachtet man auch den typischen Effekt, daß die Genauigkeit der Lösung von der letzten Komponente zur ersten beim Gauß-Algorithmus stark abnimmt; beim Gauß-Jordan-Algorithmus hingegen bleibt die Genauigkeit annähernd konstant. Führt man die in Abschnitt e) beschriebene Defektiteration durch, führen jedoch beide Algorithmen auf weit bessere (untereinander gleich gute) Lösungsintervalle.

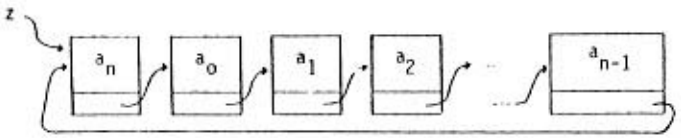
1.b) Zur Implementierung einer Intervallarithmetik mit flexibler Genauigkeit

Ein Hauptanliegen der genaueren Arithmetik war die Länge der verwendeten Zahlen dynamisch zu halten. Daher wurde die Programmiersprache PASCAL und das Element pointer verwandt. Die Basis β sollte eine 10er-Potenz sein um Rundungsfehler zu vermeiden (siehe Kulisch), andererseits sollte $2 \cdot \beta^2 - 1$ noch ein darstellbarer integer sein. Daher wurde $\beta = 10^5$ gesetzt für die UNIVAC 1108 der Universität Karlsruhe.

Als erstes wurde ein Paket für lange Integer geschrieben. Der Typ eines Longintegers LINT ist definiert als

```
TYPE  LIST = RECORD  ELEMENT : INTEGER;
                     SUCC : ↑LIST
             END;
      LINT = ↑LIST;
```

Vereinbarungsgemäß sieht die interne Darstellung der Zahl $z = \sum_{i=o}^{n} a_i \cdot \beta^i$, a_i sind alle nichtnegativ oder nichtpositiv und $a_k \neq o$, wie folgt aus (symbolisch)



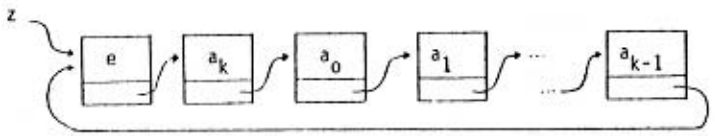
Die Darstellung ist in der Reihenfolge $a_o, a_1, \ldots$ vorteilhaft bei Addition, Subtraktion und Multiplikation; bei der Division allerdings von Nachteil. Das Vorzeichen kann jedoch, da z auf a_n zeigt, im Gegensatz zum SAC1-System sofort bestimmt werden, d.h. die maximale Rechenzeit zur Vorzeichenbestimmung geht von O(n) auf O(1) zurück. Neben zahlreichen Algorithmen für Input/Output, Konvertierung etc. wurden geschrieben und implementiert:


```
IADD(a,b)    := a + b
INEG(a)      := -a
ISUB(a,b)    := a - b
IMULT(a,b)   := a + b
IDIV(a,b)    :=  a/b
IPOWER(a,n)  := a^n
IGCD(a,b)    := gcd(a,b)
```

und andere. Die Operationen werden jeweils in **Z** ausgeführt.

An dieser Stelle noch eine Bemerkung zur Schnelligkeit der Algorithmen. Grundsätzlich wurde versucht, die Algorithmen nach dem neuesten Stand der Forschung zu schreiben (z.B. bei IDIV Multiplikation mit der Inversen, IGCD nach Lehmer usw. ; für eine genauere Beschreibung siehe Knuth). Leider ist die Implementierung trotzdem recht langsam, jedoch offenbar ohne Verschulden der Algorithmen. Zur Begründung sei der Karatsuba-Algorithmus (siehe Knuth) zur Multiplikation langer Zahlen angeführt. Durch die Methode gelingt es, die Rechenzeit für die Multiplikation zweier longinteger der Länge n von $O(n^2)$ auf $O(n^{1.58})$ zu reduzieren. Der Algorithmus wurde implementiert. Obwohl bereits gute Erfahrungen damit in anderen Systemen gemacht wurden (Verbesserung bis zum Faktor 10), konnte hier nur für sehr große Zahlen (1000-stellig und mehr) überhaupt nur annähernd die gleiche Rechenzeit erzielt werden! Der Grund hierfür liegt an dem enormen Overhead, den die Listenverarbeitung mittels Pointer erzeugt. An dieser Stelle sei noch vermerkt, daß jede Operation (die vier Grundrechnungsarten, Exponentiation, ect.) als Ergebnis eine Liste erzeugt, d.h. daß während der Abarbeitung des Algorithmus immer mehr dynamische Speicherplätze gefordert werden. Es muß also Vorsorge getroffen werden, daß von Zeit zu Zeit nicht mehr benötigter Speicherplatz wieder verfügbar gemacht wird. Dies geschieht in dem vorgestellten System mit Hilfe eines Garbage Collectors (siehe Rump 2).

Weiter wurde ein Floating-Paket geschrieben. Der Typ ist der gleiche wie LINT, jedoch die Interpretation eine andere. Die Zahl $z = 0.(\sum_{i=0}^{k} a_i \cdot \beta^i) \cdot 10^e$ (hier wird die Zahl mit den Ziffern von $a_k, a_{k-1}, \ldots, a_1, a_0$ hintereinander geschrieben nach o. dargestellt; außerdem wird $\beta/2 \leq |a_k| < \beta$ vorausgesetzt) wird im Paket dargestellt als (symbolisch)



Die Prozeduren sind die gleichen wie für longinteger (außer GCD), wobei der Buchstabe I durch FLOAT zu ersetzen ist. Hinzu kommen wie üblich Input/Output-Algorithmen und diverse Typenumwandlungsalgorithmen.

Als dritte Gruppe sind Algorithmen für Intervalle geschrieben worden, und zwar mit angebbarer Genauigkeit. Der Typ eines Intervalls (linke und rechte Grenzen sind Floating-Zahlen) ist definiert als

```
TYPE  TYPES = (FLOATING,INTERVAL);
      REC = RECORD  LEFT : FLOAT;
                    CASE TYP : TYPES of FLOATING : ( );
                                        INTERVAL : (RIGHT : FLOAT)
            END;
      FLINTV = ↑REC;
```

Liegt ein Punktintervall P vor, hat TYP den Wert FLOATING und Pf.LEFT ist der Wert der Zahl; liegt ein Intervall vor, hat TYP den Wert INTERVAL und die Komponenten LEFT und RIGHT sind die Grenzen des Intervalls. Es wurden u.a. folgende Algorithmen geschrieben:

FADD(a,b,k)	:= $a \oplus b$	auf Länge k
FSUB(a,b,k)	:= $a \ominus b$	auf Länge k
FMULT(a,b,k)	:= $a \otimes b$	auf Länge k
FDIV(a,b,k)	:= $a \oslash b$	auf Länge k

Für $* \in \{+,-,\times,/\}$ bezeichnet $\circledast$ die entsprechende Intervalloperation. Die Algorithmen berechnen jeweils a * b mittels der FLOAT-Algorithmen. Waren a und b Punktintervalle und ist a * b in k Stellen darstellbar, ist das Ergebnis das Punktintervall mit dem Wert a * b, andernfalls wird a * b auf k Stellen intervallmäßig nach oben und unten gerundet und dem Prozedurnamen zugewiesen.

Das Paket ist an der UNIVAC 1108 der Unversität Karlsruhe implementiert und verfügbar.

1.c) Numerische Ergebnisse und Vergleich des Intervall-Gauß- und Gauß-Jordan-Algorithmus mit Pivotsuche

Zunächst sei etwas zur Angabe der Stellenzahl bemerkt. Gegeben sei ein Problem und

$\hat{x}$	die genaue Lösung
$\tilde{x}$	die durch Gleitkommarechnung bestimmte Näherung
x_l, x_r	die untere und obere Grenze eines Näherungsintervalls.

Der relative Fehler von $\tilde{x}$ berechnet sich nach

$$\Delta_{rel}(x) = \frac{|\tilde{x} - \hat{x}|}{|\hat{x}|} ,$$

der maximale relative Fehler von $[x_l, x_r]$ mit $0 \notin [x_l, x_r]$ nach

$$\Delta_{rel}([x_l, x_r]) \leq \frac{|x_r - x_l|}{\min\{|x_l|, |x_r|\}} ,$$

wobei das Gleichheitszeichen nur für $\hat{x} = x_l$ bzw. $\hat{x} = x_r$ gilt, je nachdem das Intervall positiv oder negativ ist.

Mit beiden Algorithmen (Gauß-Verfahren und Gauß-Jordan-Verfahren) werden mit verschiedenen Genauigkeiten folgende Beispiele gerechnet ($a_{i(n+1)}$ ist die rechte Seite b):

a) Exp ± 5 — Die Koeffizienten a_{ij}, i = 1(1)n j = 1(1)n+1, sind definiert als $m \cdot 10^e$, wobei $0 \leq m \leq 1$ eine Gleitkommazufallszahl mit 5-stelliger Mantisse und $-5 \leq e \leq +5$ eine zufällige ganze Zahl ist.

b) Exp ± 15 — wie a), jedoch $-15 \leq e \leq 15$

c) Exp ± 30 — wie a), jedoch $-30 \leq e \leq 30$

d) Int 10 — Die Koeffizienten a_{ij}, i = 1(1)n, j = 1(1)n+1 sind ganze Zufallszahlen mit $|a_{ij}| \leq 10$

e) Q - 1 — $S = (s_{ij})$ und $R = (r_{ij})$ mit $s_{ij} = 1$ und $0 \leq r_{ij} \leq 1$ eine zufällige Gleitkommazahl mit 5-stelliger Mantisse und Exponent 0 für i,j = 1(1)n. Dann ist $a_{ij} = S - q \cdot R$ für $q = 10^{-1}$ und b so gewählt, daß (1,...,1) die exakte Lösung des Gleichungssystems ist.

f) Q - 2 — wie e) mit $q = 10^{-2}$

g) Q - 3 — wie e) mit $q = 10^{-3}$

h) HIV — $a_{ij} = [1/(i+j-1)]$ für i,j = 1(1)n das engste Intervall, das 1/(i+j-1) enthält: Und zwar mit k-stelliger Mantisse, wenn k-stellig gerechnet wird; b = (1,...,1).

i) HIV1 — wie h), jedoch $a_{ij} = [1/(i+j)]$ für i = 2(1)n, j = 1(1)n und $a_{1j} = 1$.

j) HR — $a_{ij} = |1/(i+j-1)|$ auf 5-stellige Mantisse abgeschnitten; b = (1,...,1) .

k) HE — $a_{ij} = f/(i+j-1)$, wobei f = kgV(1,2,...,2n-1); b = (1,...,1) .

Für die Intervallalgorithmen wurde als Pivotelement jeweils dasjenige gewählt, das vom Nullpunkt den größten Abstand hat. Existieren mehrere solche, wird das mit dem geringsten Durchmesser genommen. Im Folgenden wurden für die Testmatrizen a) bis g) jeweils mindestens 3 Beispiele gerechnet, Die Anzahl der mindestens garantierten Stellen heißt dabei immer über alle Beispiele und alle Komponenten genommen, im Durchschnitt entsprechend über alle Beispiele und alle Komponenten. 15-stellige Rechnung z.B. heißt mit 15-stelliger Mantisse und dem System aus 1.b) gerechnet. G bzw. GJ bedeuten Gauß- bzw. Gauß-Jordan-Verfahren, intervallmäßig mit Pivotsuche durchgeführt.

Die Spalte NS gibt an, wieviele der gerechneten Beispiele numerisch singulär waren, d.h. in einer Pivotspalte nur Nullintervalle aufgetreten sind. Aus Abbildung 1.1 ersieht man, daß nur für Exp ± 30 und 5-stellige Rechnung für Grad 10 ein numerisch singuläres Beispiel auftrat. Daß (wie z.B. für Exp ± 5, Grad 7) für höheren Grad teilweise mehr Stellen im Minimum garantiert sind als für kleineren Grad, ist auf die statistische Streuung zurückzuführen. Eine 0 bei den mindestens garantierten Stellen heißt, daß in wenigsten einem Beispiel in einer Komponente einmal keine Stelle garantiert war. Z.B. für Exp ± 30, Grad 7 war das für G und GJ jeweils tatsächlich nur in einem Beispiel in einer Komponente der Fall; ansonsten waren hier immer wenigstens 10 Stellen garantiert!

Unter sämtlichen (vielen hundert) gerechneten Beispielen kam es nur dreimal vor, daß das Gauß-Jordan-Verfahren numerisch singulär war und der Gauß-Algorithmus nicht. Der umgekehrte Fall trat nie ein. Gleichwohl liefert GJ insbesondere für schlecht konditionierte Probleme bessere Ergebnisintervalle. Ein typisches Beispiel hierfür ist die Hilbertmatrix vom Grad 10, wobei die Nenner durch einen geeigneten Faktor wegmultipliziert werden (siehe Definition von HE unter k)); die Darstellung ist also exakt. In Tabelle 1.5 ist links der relative Fehler der G- und rechts der der GJ-Lösungsintervalle angegeben.

Tabelle 1.3

Anzahl der garantierten Stellen

		5 - stellig					15 - stellig				
		mindestens		im Durchschnitt			mindestens		im Durchschnitt		
Testmatrix	Grad	G	GJ	G	GJ	NS	G	GJ	G	GJ	NS
Exp ± 5	15						9	10	12	12	-
	20						9	11	11	12	-
	25						8	10	9	10	-
Int 10	15						7	10	10	11	-
Q - 1	3	2	2	3	3	-	12	12	13	13	-
	5	1	1	2	2	-	11	11	12	12	-
	7	o	o	1	1	-	9	10	11	11	
Q - 2	3	1	1	1	1	-	11	11	11	11	-
	5	o	o	1	1	-	10	10	11	11	-
	7	o	o	1	1	1	9	9	10	10	-
Q - 3	3	o	o	1	1	-	10	10	10	11	-
	5	o	o	o	o	1	9	9	10	10	-
	7	o	o	o	o	2	8	8	9	9	-

Tabelle 1.2

Anzahl der garantierten Stellen

		5-stellig					15-stellig			
		mindestens		im Durchschnitt			mindestens		im Durchschnitt	
Testmatrix	Grad	G	GJ	G	GJ	NS	G	GJ	G	GJ
Exp ± 5	3	2	2	4	4	-	12	12	14	14
	5	2	2	4	4	-	12	13	14	14
	7	3	3	4	4	-	13	13	14	14
	10	o	o	3	3	-	11	11	13	13
	12						10	10	11	12
Exp ± 15	3	o	o	4	4	-	6	6	14	14
	5	3	3	4	4	-	13	13	14	14
	7	3	3	4	4	-	13	13	14	14
	10	1	1	3	3	-	11	11	13	13
Exp ± 30	3	1	1	4	4	-	11	11	14	14
	5	1	1	4	4	-	11	11	14	14
	7	o	o	4	4	-	o	o	13	13
	10	o	o	3	3	1	o	o	13	13

Tabelle 1.4

Testmatrix	Grad	Anzahl der garantierten Stellen: 15-stellig mindestens G	15-stellig mindestens GJ	15-stellig im Durchschnitt G	15-stellig im Durchschnitt GJ	25-stellig mindestens G	25-stellig mindestens GJ	25-stellig im Durchschnitt G	25-stellig im Durchschnitt GJ
HIV	5	5	8	8	8	15	18	17	18
	7	o	4	3	4	9	14	12	14
	10	numerisch singulär				3	7	5	7
HIV1	5	6	8	8	8	16	18	17	18
	7	o	4	3	4	9	14	12	14
	10	numerisch singulär				o	6	5	6
HR	5	7	10	10	10	17	20	20	20
	7	3	7	8	8	14	17	16	18
	10	o	4	4	4	9	14	13	14
HE	3					22	23	23	23
	5					16	19	19	19
	7					10	14	15	15
	10					o	7	8	8

> 1	2.5_{10}-8
4.2_{10}-1	2.1_{10}-8
9.1_{10}-3	1.9_{10}-8
3.4_{10}-4	1.7_{10}-8
2.1_{10}-5	1.5_{10}-8
1.8_{10}-6	1.4_{10}-8
2.1_{10}-7	1.3_{10}-8
3.9_{10}-8	1.2_{10}-8
1.1_{10}-8	1.1_{10}-8
1.0_{10}-8	1.0_{10}-8

Tabelle 1.5 relativer Fehler der Lösungsintervalle bei G und GJ

Das drastische Absinken der Genauigkeit bei G ist typisch, während die Genauigkeit von GJ annähernd konstant bleibt (siehe dazu auch Abschnitt e) dieses Kapitels).

Kommen wir zum Aufwand der beiden Verfahren. Zunächst sei die Anzahl der Operationen für ein paar gängige Algorithmen aufgeschrieben.

	# Additionen	# Multiplikationen	# Divisionen
LU-Zerlegung	$\frac{n^3}{3} - \frac{n^2}{2} + \frac{n}{6}$	$\frac{n^3}{3} - \frac{n^2}{2} + \frac{n}{6}$	$\frac{n^2}{2} - \frac{n}{2}$
$\tilde{x}$ aus LU-Zerlegung	$n^2 - n$	$n^2 - n$	n
Näherung R für A^{-1} aus LU-Zerlegung	$\frac{2}{3}n^3 - \frac{n^2}{2} - \frac{n}{6}$	$\frac{2}{3}n^3 - \frac{n^2}{2} - \frac{n}{6}$	$n^2 - n$
$\tilde{x}$ aus R ($x = R \cdot b$)	$n^2 - n$	n^2	-
Defektiterationsschritt mit LU	$2n^2$	$2n^2 - n$	-
Defektiterationsschritt mit R	$2n^2$	$2n^2$	-

Tabelle 1.6 Aufwand einiger Verfahren

Ein Defektiterationsschritt heißt $x' = \tilde{x} - \Delta x$, wobei Δx als Lösung von $A \cdot \Delta x = A\tilde{x} - b$ einmal mittels LU-Zerlegung und einmal mittels R berechnet wird. Die Anzahl der Additionen bei Übergang zur Intervallrechnung verdoppelt sich gegenüber dem reellen Verfahren; die Anzahl der Multiplikationen schwankt zwischen 2 und 4 (siehe Kulisch). Je nach der Art des Problems und des Algorithmus treten praktisch keine Nullintervalle auf oder sehr viele, und demnach wird man die mittlere Anzahl der Multiplikationen bestimmen. Bei der intervallmäßigen Durchführung des G bzw. GJ-Algorithmus gilt die Faustregel, daß praktisch keine Nullintervalle auftreten, wenn der Algorithmus erfolgreich beendet wird. In Tabelle 1.7 wurde jeweils $\leq$ für die maximale und $\sim$ für die mittlere Anzahl der Operationen geschrieben.

# Additionen	# Multiplikationen	# Divisionen
	$\sim \frac{2}{3}n^3 + n^2 - \frac{5}{3}n$	
$\frac{2}{3}n^3 + n^2 - \frac{5}{3}n$		$n^2 - n$
	$\leq \frac{4}{3}n^3 + 2n^2 - \frac{10}{3}n$	

Tabelle 1.7 Aufwand für Intervall-Gauß-Algorithmus

Geht man von einfacher zu k-facher Genauigkeit über, erhöht sich der Aufwand für

1 lange Addition auf k Additionen

1 lange Multiplikation auf k^2 Multiplikationen + k^2 Addtionen

1 lange Division auf $\frac{k(k+1)}{2}$ Divisionen + $\frac{k(k+1)}{2}$ Multiplikationen + $\frac{k(k+1)}{2}$ Additionen

Damit verhält sich der Gesamtaufwand des Intervall-Gauß-Algorithmus für verschiedene Genauigkeiten wie folgt

k	2	3	4	5
Aufwand	$\sim 2.7n^3$	$\sim 6n^3$	$\sim 10.7n^3$	$\sim 16.7n^3$

Tabelle 1.8 Rechenaufwand des Intervall-Gauß-Algorithmus

Dies gilt nur, wenn die Intervalloperationen (wie heute leider noch erforderlich) umständlich mit eigens geschriebenen Prozeduren simuliert werden müssen. Wieder einmal erhebt sich die Forderung, solche Algorithmen ein für allemal maschinenseitig zur Verfügung zu stellen, damit nicht wieder und wieder dieselben Algorithmen

programmiert werden müssen, und zwar meist in umständlicher Weise, da der Zugriff zu maschineninternen Größen von höheren Programmiersprachen meist recht schwierig oder gar nicht möglich ist. Der Aufwand für den Intervall-Gauß-Algorithmus mit einfacher Länge beträgt im Mittel $\sim \frac{2}{3} n^3$. Der Aufwand für den reellen Gauß-Jordan-Algorithmus ist bekannterweise bereits $\frac{1}{2} n^3$, so daß bei Übergang zu Intervallen sich der Aufwand auf n^3 erhöht.

Der Speicherbedarf erhöht sich bei beiden Verfahren bei Übergang zu k-facher Genauigkeit auf $(n^2 + n) \cdot 2k$.

1.d) Die Variante von Crout und Doolittle

In der Variante von Crout/Doolittle (kurz C/D) wird der Gaußsche Algorithmus derart umgeschrieben, daß die Verwendung von genauen Skalarprodukten ermöglicht wird. Wie in [Grüner] gezeigt wird, verbessert sich die Fehlerabschätzung gegenüber dem ursprünglichen Gauß-Algorithmus wesentlich. Um die beiden Varianten zu vergleichen, wurden folgende Matrizen verwendet:

a) Exp 5 vom Grad 25

b) Int 10 vom Grad 25

c) Hilbert 12 Hilbertmatrix vom Grad 12, wie HE unter k) beschrieben.

In den Beispielen a), b) und c) wurde jeweils 10-stellig, 15-stellig und 35-stellig gerechnet. In der nachfolgenden Tabelle 1.9 ist jeweils der minimale relative Fehler Δ_{min} und der maximale relative Fehler Δ_{max} jeweils für das Intervall-Gauß-Verfahren und die Variante von Crout und Doolittle angegeben und der durchschnittliche Faktor f zwischen den relativen Fehlern der Komponenten der Lösungsvektoren der beiden Verfahren (ein Faktor > 1 heißt, die relativen Fehler der Gauß-Lösungsintevalle sind im Durchschnitt um diesen Faktor schlechter).

	Gauß		Crout/Doolittle		f
	Δ_{min}	Δ_{max}	Δ_{min}	Δ_{max}	
a)	$7.9_{10}-8$	$2.0_{10}-5$	$1.1_{10}-8$	$2.8_{10}-6$	~7
b)	$1.1_{10}-7$	$2.1_{10}-2$	$4.4_{10}-8$	$1.6_{10}-2$	~2.2
c)	$2.8_{10}-9$	> 1	$5.0_{10}-9$	> 1	~0.6

Tabelle 1.9 relativer Fehler der Ergebnisintervalle bei G und C/D

Wie man sieht, ergibt die Variante eine Verbesserung des relativen Fehlers, bei Hilbert-Matrizen jedoch eine Verschlechterung.

Bei dieser Gelegenheit wurde nochmals geprüft, wie sich die Erhöhung der Genauigkeit der Rechnung auf den relativen Fehler des Ergebnisses auswirkt. Es ergab sich, daß diese Beziehung wieder annähernd linear ist, d.h. eine Erhöhung der Genauigkeit um 10 Stellen bewirkt eine Verbesserung des relativen Fehlers der entsprechenden Ergebnisintervalle um 10 Zehnerpotenzen. Zum Beispiel wurde die Testmatrix a) mit 10 und 20 Stellen Genauigkeit mit C/D durchgerechnet. Für die relativen Fehler einiger Ergebnisintervalle ergab sich:

10-stellig	20-stellig
$2.71_{10}-7$	$2.76_{10}-17$
$2.95_{10}-7$	$3.00_{10}-17$
$6.39_{10}-7$	$6.49_{10}-17$
$3.12_{10}-7$	$3.18_{10}-17$
$2.59_{10}-7$	$2.64_{10}-17$

Tabelle 1.10 relativer Fehler der Ergebnisintervalle bei verschiedenen Genauigkeiten

Wie hier erhärtete sich auch bei zahlreichen anderen Beispielen die lineare Abhängigkeit (siehe auch Tabelle 1.1).

1.e) Weitere Ergebnisse und Verbesserungen

Man könnte vermuten, daß im Sinne des Absolutbetrages stark differierende Lösungen auch Intervalle großen Durchmessers implizieren. In dieser Allgemeinheit ist die Vermutung sicher zu verneinen. Wie die Erfahrung vieler Testbeispiele zeigte, hängt der Durchmesser der erhaltenen Lösungsintervalle im wesentlichen von der Kondition der Eingabematrix ab. Im folgenden Beispiel etwa wurde ein Gleichungssystem vom Grad 10 mit 5 Stellen (!) Genauigkeit gerechnet. In der untenstehenden Tabelle sind alle 10 Lösungsintervalle angegeben nebst einer Einschließung für die Determinante. Das Gleichungssystem ist vom Typ Exp ± 30.

$$
\begin{aligned}
-0.93632_{10}\text{-}19 &\leq x_1 \leq -0.93553_{10}\text{-}19\\
0.91294_{10}\text{-}11 &\leq x_2 \leq 0.91330_{10}\text{-}11\\
-0.45503_{10}\text{-}41 &\leq x_3 \leq -0.45454_{10}\text{-}41\\
0.33509_{10}\text{-}23 &\leq x_4 \leq 0.33524_{10}\text{-}23\\
0.67809_{10}\text{-}23 &\leq x_5 \leq 0.68394_{10}\text{-}23\\
0.40303_{10}\text{-}23 &\leq x_6 \leq 0.40323_{10}\text{-}23\\
0.33592_{10}\text{-}7 &\leq x_7 \leq 0.33620_{10}\text{-}7\\
-0.13319_{10}\text{-}7 &\leq x_8 \leq -0.13814_{10}\text{-}7\\
0.25342_{10}\text{-}17 &\leq x_9 \leq 0.25367_{10}\text{-}17\\
-0.19591_{10}\text{-}14 &\leq x_{10} \leq -0.19566_{10}\text{-}14\\
-0.80528_{10}240 &\leq \text{Determinante} \leq -0.80365_{10}240
\end{aligned}
$$

Tabelle 1.11 Lösungsintervalle eines 10×10 Gleichungssystems auf 5 Stellen gerechnet

Wie man sieht, sind die Lösungsintervalle durchweg sehr genau (bei 6-stelliger Rechnung), obwohl zwischen der dem Betrag nach kleinsten und größten Lösung eine Differenz von 34 im Exponenten liegt. Es bestätigt sich die an mehreren Stellen in [Forsythe/Mohler] unterstrichene Tatsache, daß die Determinante allein noch nichts über die Kondition einer Matrix aussagt.

Eine Möglichkeit, die Durchmesser der Lösungsintervalle zu verkleinern ist, die Zwischenrechnung

$$a_{ij}^{(k+1)} := a_{ij}^{(k)} - (a_{ik}^{(k)} / a_{kk}^{(k)}) * a_{kj}^{(k)}$$

mit höherer Genauigkeit zu rechnen. Im folgenden wurden z.B. Matrizen des Typs Q - 3 mit 15 Stellen Genauigkeit gerechnet, die Zwischenrechnungen jedoch auf 20 Stellen Genauigkeit durchgeführt und dann auf Intervalle der Mantissenlänge 15 intervallmäßig gerundet. Es wurden jeweils wenigstens 3 Beispiele gerechnet, wobei Δ_{min} und Δ_{max} den jeweils kleinsten und größten relativen Fehler der Ergebnisintervalle aller Beispiele angibt. f ist der durchschnittliche Verbesserungsfaktor, wobei f > 1 heißt, daß der Algorithmus mit genaueren Zwischenrechnungen im Durchschnitt einen um den Faktor f kleineren relativen Fehler besitzt. Weiter wurden exakte Hilbertmatrizen (siehe Definition unter k)) verschiedener Grade 25-stellig gerechnet mit 30-stelliger Zwischenrechnung. Schließlich seien noch einige Ergebnisse für Q - 3 Matrizen angegeben bei 5-stelliger Rechnung mit 10-stelliger Zwischenrechnung. Man beachte jeweils, daß der Speicherplatz praktisch gleich bleibt und die "Gesamtgenauigkeit" der Rechnung; es werden nur Zwischenrechnungen mit erhöhter Genauigkeit berechnet um bessere, genauere Zwischenergebnisse abspeichern zu können.

Die Verbesserung für Q - 3 für 5/10-stellige Rechnung ist erwartungsgemäß groß. Sie kann jedoch kaum als Maß dienen, da hier in den Zwischenrechnungen bereits mit doppelter Genauigkeit gerechnet wurde. Wie man an den beiden anderen Beispielen

Testmatrizen	Grad	Intervall-Gauß-Algorithmus Δ_{min}	Intervall-Gauß-Algorithmus Δ_{max}	Intervall-Gauß-Algorithmus und genauere Zwischenrechnungen Δ_{min}	Intervall-Gauß-Algorithmus und genauere Zwischenrechnungen Δ_{max}	f
Q - 3	3	$6.5_{10}-11$	$1.2_{10}-10$	$3.2_{10}-14$	$1.2_{10}-13$	$\sim 8_{10}-4$
15/20-stellig	5	$8.2_{10}-11$	$1.1_{10}-9$	$6.7_{10}-14$	$6.2_{10}-12$	$\sim 8_{10}-4$
	7	$2.9_{10}-10$	$1.1_{10}-8$	$3.3_{10}-13$	$1.2_{10}-11$	$\sim 8_{10}-4$
HE	3	$2.7_{10}-22$	$2.4_{10}-21$	$1.7_{10}-22$	$1.5_{10}-21$	$\sim 6_{10}-1$
25/30-stellig	5	$2.1_{10}-18$	$1.7_{10}-15$	$1.4_{10}-18$	$1.1_{10}-15$	$\sim 6_{10}-1$
	7	$2.9_{10}-14$	$6.1_{10}-9$	$2.0_{10}-14$	$4.2_{10}-9$	$\sim 6_{10}-1$
	10	$2.3_{10}-7$	> 1	$1.6_{10}-7$	> 1	$\sim 6_{10}-1$
Q - 3	3	8.0 -1	ns	$3.1_{10}-4$	$1.4_{10}-3$	
5/10-stellig	5	2.4 -0	ns	$7.3_{10}-4$	$6.2_{10}-2$	
	7	> 1	ns	$3.3_{10}-3$	$9.4_{10}-1$	

Tabelle 1.12 Relative Fehler bei genaueren Zwischenrechnungen

sieht, ist der Gewinn unterschiedlich. Auf jeden Fall scheint sich eine etwas genauere Berechnung der Zwischenergebnisse zu lohnen. Eine noch genauere Berechnung der Zwischenresultate bringt dagegen keinen sichtbaren Gewinn. Es wurde etwa versucht, bei 15-stelliger Rechnung durch das Rechnen der Zwischenergebnisse auf 30 Stellen eine weitere Genauigkeitssteigerung zu erzielen. Dies trat jedoch nicht ein.

Der wesentliche Teil des Gesamtaufwands bei der Durchführung des Gauß-Algorithmus geht in die Berechnung der LU-Zerlegung (siehe Tabelle 1.6 . Daneben wird die Rückwärtssubstitution ($x = U^{-1} \cdot L^{-1} \cdot b$) durchgeführt. Weiter kann man das Ergebnis nach der Defektiteration weiter verbessern. Ist $\tilde{x}$ ein Näherungswert für die Lösung eines linearen Gleichungssystems $Ax = b$ und bezeichnet $\Delta = A\tilde{x} - b$ den zugehörigen Defekt, so wäre

$$x^* = \tilde{x} - \tilde{y}, \text{ wobei } y \text{ die Lösung von } Ay = \Delta \text{ ist,}$$

die exakte Lösung des Gleichungssystems ($Ax^* = A\tilde{x} - A\tilde{y} = A\tilde{x} - \Delta = b$). Mit einer Einschließung Y von y ist dann $X = \tilde{x} - Y$ offenbar auch eine (hoffentlich bessere) Einschließung der exakten Lösung. Im Folgenden wurde genauer untersucht, welchen Einfluß die verschieden genaue Rechnung der Einzelschritte auf die Genauigkeit der Ergebnisintervalle hat. Es wurde die Genauigkeit der LU-Zerlegung, der Rückwärtssubstitution, der Berechnung von $x^{k+1} := m(x^k) - y^k$ und der Berechnung des Defekts $\delta^k := A \cdot m(x^k) - b$ variiert, wobei δ^k intervallmäßig ausgerechnet wird und y^k die mittels der bereits vorliegenden LU-Zerlegung berechnete intervallmäßige Lösung von $A \cdot y^k = \delta^k$ ist. Als Testmatrix diente eine 7×7-Matrix $A = (a_{ij})$, wobei a_{ij} zufällige Integer und $|a_{ij}| \leq 10$ waren. Die rechte Seite b wurde so berechnet, daß die exakte Lösung $(1,\ldots,1)$ lautete. Es sind wieder der minimale und der maximale relative Fehler Δ_{min} und Δ_{max} der 7 Ergebnisintervalle für die verschiedenen Genauigkeitskombinationen angegeben. Ein relativer Fehler von $1.1_{10}-9$ bedeutet übrigens ein Lösungsintervall

[0.9999999999 , 1.0000000001] .

Anzahl der Stellen bei der Berechnung von				IV Gaußzerlegung		1. Iteration		2. Iteration	
LU-Zerlegung	Rückwärts-substitution	$x^{k+1}:=m(x^k)-y^k$	$\delta^k:=A\cdot m(x^k)-b$	Δ_{min}	Δ_{max}	Δ_{min}	Δ_{max}	Δ_{min}	Δ_{max}
10	10	10	15	5.0_{10}-8	7.2_{10}-7	1.1_{10}-9	1.1_{10}-9	1.1_{10}-9	1.1_{10}-9
10	10	10	20	5.0_{10}-8	7.2_{10}-7	1.1_{10}-9	1.1_{10}-9	1.1_{10}-9	1.1_{10}-9
10	10	10	25	5.0_{10}-8	7.2_{10}-7	1.1_{10}-9	1.1_{10}-9	1.1_{10}-9	1.1_{10}-9
10	10	[illegible]	20	5.0_{10}-8	7.2_{10}-7	9.6_{10}-18	1.1_{10}-16	$1-2_{10}$-18	$2-2_{10}$-17
10	10	20	25	5.0_{10}-8	7.2_{10}-7	9.6_{10}-18	1.4_{10}-16	1.1_{10}-19	1.1_{10}-19
10	10	20	30	5.0_{10}-8	7.2_{10}-7	9.6_{10}-18	1.4_{10}-16	1.1_{10}-19	1.1_{10}-19
10	10	20	35	5.0_{10}-8	7.2_{10}-7	9.6_{10}-18	1.4_{10}-16	1.1_{10}-19	1.1_{10}-19
10	15	20	20	4.2_{10}-8	5.8_{10}-7	1.4_{10}-17	2.1_{10}-16	1.7_{10}-18	3.0_{10}-17
10	15	20	25	4.2_{10}-8	5.8_{10}-7	1.4_{10}-17	2.1_{10}-16	1.1_{10}-19	1.1_{10}-19
10	15	20	30	4.2_{10}-8	5.8_{10}-7	1.4_{10}-17	2.1_{10}-16	1.1_{10}-19	1.1_{10}-19
10	15	20	35	4.2_{10}-8	5.8_{10}-7	1.4_{10}-17	2.1_{10}-16	1.1_{10}-19	1.1_{10}-19

Tabelle 1.13 Genauigkeit bei verschieden genauen Zwischenrechnungen

Es fällt zunächst auf, daß beim Übergang von (10,10,20,20) zu (10,15,20,20) die Genauigkeit des Ergebnisses abnimmt, obwohl die Rechnung genauer durchgeführt wurde! In diesem Fall ist zufälligerweise der Mittelpunkt des ungenaueren Ergebnisintervalls näher bei der wahren Lösung als der des engeren Intervalls bei der zweiten Rechnung. Weiter sieht man, daß bereits eine etwas genauere Defektberechnung zu sehr genauen Ergebnisintervallen führt ($\Delta_{max} = 1.1_{16}$-9), und das beim ersten Iterationsschritt. Gleichzeitig kann die relativ ungenau berechnete LU-Zerlegung zu viel genaueren Ergebnisintervallen führen, wenn nur die Defektiteration mit höherer Genauigkeit durchgeführt wird. Dabei sollte der Defekt selbst etwas genauer als die Addition zur vorigen Iterierten ausgeführt werden. Man kann sogar dazu übergehen, die Rückwärtssubstitution und $x^{k+1} = m(x^k) + \delta^k$ etwas ungenauer auszuführen, den Defekt dafür aber etwas genauer. Hierzu ein extremes Beispiel. Testmatrix ist die Hilbert 15×15-Matrix mit exakten Koeffizienten (siehe HE unter k)) mit rechter Seite (1,...,1). Bei der Rechnung wurde die Kombination (40,20,20,35) verwandt, d.h. LU-Zerlegung 40-stellig, Rückwärtssubstitution und $x^{k+1} = m(x^k) - y^k$ je 20-stellig und Defekt 35-stellig. Dabei ergab sich:

0) direkte Gaußzerlegung. 6 der 15 Ergebnisintervalle haben einen relativen Fehler > 1, der kleinste relative Fehler ist 2.9_{10}-8 (das erste Ergebnisintervall z.B. lautet $[-3.15_{10}10, +3.15_{10}10]$ wobei der genaue Wert 15 ist).

1) erster Iterationsschritt. $\Delta_{max} = 2.1_{10}$-1; $\Delta_{min} = 1.4_{10}$-18

2) zweiter Iterationsschritt. $\Delta_{max} = 6.0_{10}$-2; $\Delta_{min} = 5.2_{10}$-19

3) dritter Iterationsschritt. Alle Ergebnisintervalle sind Punktintervalle, die Lösung ist exakt angegeben!

Der Grund für die exakte Gewinnung der Lösung liegt einfach darin, daß die bei 2) gewonnenen Intervalle die Lösung (das sind nur ganze Zahlen) gerade als Mittelpunkt haben, der Defekt exakt o wird in allen Komponenten.

Wie die Beispiele aus Tabelle 1.13 deutlich zeigen, steckt in der durchgeführten LU-Zerlegung vielmehr Information, als man bei der bloßen Rückwärtssubstitution herausziehen kann. Wir möchten dies noch an zwei Beispielen verdeutlichen. Die Testmatrizen sind 7×7 und 8×8-Hilbert-Matrizen (wie oben exakt) mit rechter Seite $(1,\ldots,1)$. Die Rechnung erfolgt mit:

LU-Zerlegung	15-stellig
Rückwärtssubstitution	15-stellig
$x^{k+1} = m(x^k) - y^k$	30-stellig
Defekt	35-stellig .

Für den kleinsten und größten relativen Fehler ergibt sich für die beiden Beispiele bei 4 Iterationen:

	Hilbert 7×7		Hilbert 8×8	
	Δ_{min}	Δ_{max}	Δ_{min}	Δ_{max}
Intervall-Gauß-Algorithmus	1.0_{10}-4	> 1	1.3_{10}-2	> 1
1. Iteration	1.3_{10}-12	2.6_{10}-7	5.4_{10}-7	> 1
2. Iteration	2.0_{10}-19	4.0_{10}-15	2.2_{10}-11	8.7_{10}-5
3. Iteration	3.2_{10}-28	6.0_{10}-23	9.2_{10}-16	3.6_{10}-9
4. Iteration	6.0_{10}-30	3.7_{10}-29	3.8_{10}-20	1.5_{10}-13

Tabelle 1.14 Genauigkeitsgewinn durch Defektiteration

Im 5. Iterationsschritt wäre das Ergebnis bei der 7×7-Hilbert-Matrix übrigens sogar exakt angegeben worden (Punktintervalle). Wie man sieht, ist die Lösung trotzdem die LU-Zerlegung mitsamt Rückwärtssubstitution nur 15-stellig durchgeführt wurde im ersten Fall auf wenigstens 29, im zweiten Fall auf wenigstens 13 Stellen oder genauer berechnet worden. Hinzu kommt das wichtige Argument, daß die Kosten der Rechenzeit der Defektiteration und Rückwärtssubstitution bei $\sim 2n^2$,

währenddessen für die LU-Zerlegung bei $\frac{1}{3} n^3$ liegen. Das heißt, durch insgesamt geringen Mehraufwand kann die Genauigkeit der Lösung um Größenordnungen verbessert werden. Es mag vielleicht verwundern, wie so etwas überhaupt möglich ist. Grob gesprochen ist hier der Wunschtraum verwirklicht, nach und nach die Stellen zu berechnen, d.h. zum Beispiel zunächst 15 Stellen, von denen vielleicht nur ein paar genau sind, dann wieder 15, die aber auf die ersten 15 quasi überlappt werden,

erste Näherung — zweite Näherung

so daß im Endeffekt eine viel größere Genauigkeit entsteht. Leider ist der Intervall-Gauß-Algorithmus so geartet, daß bei Durchführung mit zu geringer Stellenzahl das Verfahren versagt: in einer Pivotspalte sind nur Nullintervalle aufgetreten. Hat die LU-Zerlegung allerdings einmal funktioniert, ist gleich viel mehr Information berechnet worden, als vielleicht benötigt wurde. Es bleibt die Frage nach einem Mittelweg, einem Verfahren, daß mit weniger Aufwand vielleicht weniger Information liefert, in jedem Fall jedoch sagen wir mindestens k garantierte Stellen bei k-stelliger Rechnung. Ein Verfahren, das diesem Ziel recht nahe kommt, soll im nächsten Kapitel vorgestellt werden.

1.f) Exakte Lösung von linearen Gleichungssystemen

Es soll noch kurz auf die exakte Lösung von linearen Gleichungssystem eingegangen werden. Exakt ist hier in dem Sinne zu verstehen, daß sämtliche auftretenden Zahlen während der Gauß-Elimination exakt, das heißt auf volle Länge gespeichert werden. Beim normalen Gauß-Verfahren tritt zunächst die Schwierigkeit der Division auf:

$$a_{ij}^{(k+1)} := a_{ij}^{(k)} - (a_{ik}^{(k)} / a_{kk}^{(k)}) \cdot a_{kj}^{(k)}$$

Hier kann man sich jedoch derart behelfen, daß (beim k-ten Eliminiationsschritt) die k-te Zeile mit $a_{ik}^{(k)}$ multipliziert von der i-ten Zeile ($i > k$) mit $a_{kk}^{(k)}$ multipliziert abgezogen wird:

$$(1.3) \qquad a_{ij}^{(k)} = a_{kk}^{(k)} \cdot a_{ij}^{(k)} - a_{ik}^{(k)} \cdot a_{kj}^{(k)} \quad .$$

Auf diese Art und Weise kommen nur Multiplikationen vor, die leicht exakt ausgeführt werden können; der Rechenaufwand steigt jedoch auf das doppelte. Eine weitere Beobachtung ist die, daß $a_{kk}^{(k)}$ in der (k+2)-ten,...,n-ten Zeile ohne Rest aufgeht (k = 1(1)n-2). Auf diese Art und Weise können die Längen der Zahlen reduziert werden. Wir geben ein kleines Beispiel an, in dem mit E ein Eliminationsschritt und mit K die Kürzung durch einen Faktor bezeichnet wird:

8	3	4	8		8	3	4	8		8	3	4	8
9	4	3	2	E →	0	+5	-12	-56	E →	0	+5	-12	-56	K :8 →
7	9	2	4		0	+51	-12	-24		0	0	552	2736	
8	5	1	2		0	-16	-24	-48		0	0	72	656	

	8	3	4	8		8	3	4	8		8	3	4	8
K :8 →	0	5	-12	-56	E →	0	5	-12	-56	K :5 →	0	5	-12	-56
	0	0	69	342		0	0	69	342		0	0	69	342
	0	0	9	82		0	0	0	2580		0	0	0	516

Man kann übrigens darüberhinaus leicht zeigen, daß das verbleibende Element $a_{nn}^{(n)}$, in diesem Fall 516, gleich der Determinante der ursprünglichen Matrix ist. Bei exakter Rechnung, d.h. Speicherung aller Stellen jedes Ergebnisses, bekommt man allerdings mit der Summe (1.3) sehr rasch Schwierigkeiten, wenn die Exponenten der Summanden weit auseinanderliegen. Betrachtet man etwa das Beispiel

$$A = \begin{pmatrix} 1 & 1 \\ 1 & 10^{-99} \end{pmatrix} \quad ,$$

so wäre $a_{22}^{(2)} = -1\text{-}10^{-99}$, also bereits 100 Dezimalen lang. Diese Schwierigkeit könnte man durch einen modularen Algorithmus umgehen, wie ihn McClellan beschrieben hat. Grob gesprochen wird das Gleichungssystem für eine Reihe von Primzahlen p_i durchgerechnet (nachdem vorher das Gleichungssystem durch einen geeigneten Faktor, etwa zehn hoch minus den kleinsten vorkommenden Exponenten, so multipliziert wurde, daß alle a_{ij} ganze Zahlen sind). Ist dann das Produkt Πp_i größer als jede während der genauen Rechnung vorkommende Zahl (die vorher abgeschätzt wird, z.B. durch die Formel von Cauchy-Hadamard), so kann die Dreieckszerlegung nach dem Chinesischen Restsatz (oder nach der Variante von Garner, siehe Knuth) exakt rekonstruiert werden. McClellan gibt Rechenzeiten für die Lösung von linearen Gleichungssystemen für verschiedene Grade an, wobei die a_{ij} ganze Zufallszahlen mit $-2^{32} + 1 \leq a_{ij} \leq 2^{32} - 1$ sind:

n	5	10	15
Rechenzeiten [sec]	1.1	6.6	18.5

Die Rechenzeiten wurden auf einer UNIVAC 1108 ermittelt. Wie man sieht, sind die Zahlen recht hoch; für weit auseinanderliegende Eingabekoeffizienten, d.h. Matrizen mit großem

$$d := \max|a_{ij}| - \min|a_{ij}| \quad ,$$

wird der Algorithmus unbrauchbar.

2. Ein neues Verfahren zur Lösung linearer Gleichungssysteme

Wie wir bereits gesehen haben, ist das Intervall-Gauß-Verfahren brauchbar, wenn mit größerer Mantissenlänge gerechnet wird. Nun wird man jedoch diese höhere Genauigkeit i.a. nicht zur Verfügung haben oder muß entsprechende Algorithmen entwerfen wie in Kapitel 1 (siehe auch Rump 1). Geht man von der Gegenwart und heute existierenden Rechnern aus, wäre ein Algorithmus wünschenswert, der nur mit einfachlangen Zahlen auskommt und ein möglichst genaues Ergebnis liefert. Dabei sind leider immer noch für die nach oben und nach unten gerichteten Rundungen kleine Assemblerprogramme zu schreiben, da in den meisten Rechnern die Möglichkeit gerichtet zu runden nicht besteht.

In Kapitel 1 haben wir bereits gesehen, daß auch zunächst schlecht aussehende Ergebnisse (oder weite Intervalle) mit wenig Mehraufwand zu sehr guten Resultaten führen können. In Alefeld/Apostolatos und Kulisch 2 ist ein ähnlicher Weg beschritten. Aus zunächst sehr groben Abschätzungen (die die Lösung garantiert einschließen) ist durch sukzesive Verbesserung die Lösung immer schärfer eingegrenzt, bis zuletzt recht enge Lösungsintervalle erzielt werden. In Wongwises ist ein ähnlicher Weg beschritten. Den bekannten Verfahren ist gemeinsam, daß ausgehend von einer Ersteinschließung der Lösung die Ergebnisintervalle verfeinert werden. Im folgenden Kapitel ist ein völlig neuer Weg beschritten. Es wird umgekehrt ausgehend von einer Gleitkommanäherung ein Lösungsintervall durch Vergröberung erzielt. Diese Vergröberung findet iterativ solange statt, bis der Algorithmus in der Lage ist, für das gegebene Intervall zu beweisen, daß es die Lösung enthält. Tatsächlich wird durch den Algorithmus ein hinreichendes Kriterium an die Hand gegeben, um von einem gegebenen Intervall festzustellen, ob es die Lösung enthält oder nicht.

2.a) Einschließungssätze und Folgerungen

Zunächst wiederholen wir den Brouwer'schen Fixpunktsatz (siehe Collatz):

Satz 2.1: Eine stetige Abbildung einer abgeschlossenen, beschränkten, konvexen Teilmenge M des Euklidschen Raumes $\mathbb{R}^n$ in sich hat mindestens einen Fixpunkt.

Im folgenden wird ein Spezialfall eines allgemeinen Einschließungssatzes angegeben. Die vollständige Theorie findet sich in Kaucher/Rump . Die verblüffende Einfachheit läßt den Satz beinahe trivial erscheinen; die mächtige Tragweite scheint jedoch bis dato noch gar nicht überblickbar.

Satz 2.2: Gegeben seien stetige Funktionen $f : V_n\mathbb{R} \to V_n\mathbb{R}$ und $F : IV_n\mathbb{R} \to IV_n\mathbb{R}$ mit der Eigenschaft

$$(2.1) \qquad \bigwedge_{I \in IV_n\mathbb{R}} x \in I \Rightarrow f(x) \in F(I) .$$

Gilt dann für ein $\Omega \in IV_n\mathbb{R}$

$$(2.2) \qquad F(\Omega) \subseteq \Omega ,$$

so besitzt f in Ω mindestens einen Fixpunkt $\hat{x}$ und es ist

$$(2.3) \qquad \hat{x} \in \bigcap_{i \geq 0} F^i(\Omega) .$$

Beweis: Aus (2.2) und (2.1) folgt aus $x \in \Omega$ sofort $f(x) \in F(\Omega) \subseteq \Omega$, also $f : \Omega \to \Omega$. Nach Satz 2.1 besitzt f also wenigstens einen Fixpunkt $\hat{x}$ in $\Omega = F^0(\Omega)$. Für jedes $k \geq 0$ folgt dann

$$\hat{x} \in F^k(\Omega) \Rightarrow \hat{x} = f(\hat{x}) \in F(F^k)(\Omega)) = F^{k+1}(\Omega) ,$$

mittels vollständiger Induktion also die Behauptung (2.3).

Es sei betont, daß an die Intervallfunktion F keinerlei Voraussetzungen (außer (2.1)) geknüpft sind. Insbesondere wird F nicht als isoton vorausgesetzt.

Übertragen wir den Satz auf den Fall linearer Gleichungssysteme $Ax = b$. Ist f eine beliebige Funktion mit Fixpunkt $\hat{x} = f(\hat{x})$, wobei $\hat{x}$ die Lösung von $Ax = b$ ist, so benötigen wir nur noch eine Einschließungsfunktion F, so daß (2.1) gilt. Gleichzeitig müssen wir dafür Sorge tragen, daß (2.2) überhaupt eintreten kann.

Gehen wir von den Defektiterationen

$$x^{k+1} = x^k + R(b - Ax^k) \tag{2.4}$$

für eine beliebige Matrix R und Anfangswert x^0 aus. Definiert man $f : \mathbb{R}^n \to \mathbb{R}^n$ als

$$f(y) = y + R(b - Ay) , \tag{2.5}$$

so gilt

$$f(\hat{x}) = \hat{x} + Rb - RA\hat{x} = \hat{x} ,$$

$\hat{x}$ ist also Fixpunkt von f. Weiter ist bzgl. der euklidschen Norm $\|\cdot\|$

$$\begin{aligned} \|f(x) - f(y)\| &= \|x + Rb - RAx - y - Rb + RAy\| = \\ &= \|(E - RA)(x - y)\| \leq \|E - RA\| \cdot \|x - y\| . \end{aligned} \tag{2.6}$$

Ist der Spektralradius von E - RA kleiner als 1, so ist die Abbildung f im ganzen $\mathbb{R}^n$ kontrahierend. Würde man $F : IV_n\mathbb{R} \to IV_n\mathbb{R}$ so definieren, daß in (2.5) für den Punktvektor y Intervallvektoren eingesetzt werden, so würde die Bedingung (2.2) nie erfüllt werden (außer für den trivialen Fall $\Omega = o, y = o$). Schreiben wir die Defektiteration etwas um zu

$$\begin{aligned} f(y) &= \tilde{x} + Rb - RA\tilde{x} + y - \tilde{x} - RAy + RA\tilde{x} = \\ &= \tilde{x} + R(b - A\tilde{x}) + (E - RA)(y - \tilde{x}), \end{aligned} \tag{2.7}$$

wobei E die Einheitsmatrix ist. Nun kann die Einschließungsfunktion $\overline{F}$ direkt angegeben werden zu

$$\overline{F}(Y) := \{\tilde{x} + R(b - A\tilde{x})\} \boxplus (E - RA) \boxdot (Y \boxminus \tilde{x}), \text{ für } Y \in IV_n\mathbb{R} , \tag{2.8}$$

wobei hier und im Folgenden $X \circledast Y := \{x \circledast y \mid x \in X, y \in Y\}$ für $X, Y \in I\mathbb{R}$ und $\circledast \in \{+,-,\cdot,/\}$ den Komplex und $X \circledcirc Y$ evtl. eine Obermenge des Komplexes bezeichnet. Für Intervallvektoren, Intervallmatrizen etc. gilt die Definition entsprechend. Für das Intervallmatrixprodukt etwa gilt i.a. $A \boxdot B \subset A \circledcirc B$.

Trivialerweise gilt dann

$$\bigwedge_{Y \in IV_n\mathbb{R}} \bigwedge_{y \in Y} f(y) = \tilde{x} + R(b - A\tilde{x}) + (E - RA)(y - \tilde{x}) \in$$
$$\in \{\tilde{x} + R(b - A\tilde{x})\} \boxplus (E - RA) \boxdot (Y \boxminus \tilde{x}) = \overline{F}(Y),$$

da $\boxplus$ ja gerade so definiert war. Gilt also für irgendein $\Omega \in IV_n\mathbb{R}$ jetzt $F(\Omega) \subseteq \Omega$, so ist bereits $\hat{x} \in F(\Omega)$ bewiesen. In der Formel (2.7) wurde der Vektor $\tilde{x}$ einfach zur ursprünglichen Definition (2.5) hinzugenommen; er scheint zunächst ohne Einfluß. Um eine gute Konvergenz des Verfahrens zu sichern, wird man für $\tilde{x}$ eine Gleitkommanäherung für $\hat{x}$ einsetzen. Der Vektor $y - \tilde{x}$ wird dann von kleiner Norm, die Korrektur $(E - RA)\cdot(y - \tilde{x})$ also ebenfalls klein. Genauso wird man für R eine Gleitkommanäherung für A^{-1} einsetzen. Konvergenz tritt für $\rho(E - RA) < 1$ ein.

Nun zum praktischen Verfahren. In der Definition (2.8) kommen noch der reelle Vektor $(\tilde{x} + R\cdot(b - A\tilde{x}))$ und die reelle Matrix $(E - RA)$ vor. In der Praxis können diese i.a. nicht exakt ausgerechnet werden; daher muß eine Einschließung gefunden werden. Im allgemeinen Fall wird dies durch Ersetzen der reellen Operationen durch die entsprechenden Intervall-Operationen gesichert. Wir definieren also erneut für die Verwendung im Rechner

$$F(Y) := \tilde{x} \oplus R \odot (b \ominus A \odot \tilde{x}) \oplus (E \ominus R \odot A) \odot (Y \ominus \tilde{x}) \supseteq \overline{F}(Y) . \tag{2.9}$$

(Für die Intervallmultiplikation $\odot : IM_n\mathbb{R} \times IM_n\mathbb{R} \to IM_n\mathbb{R}$ und $\odot : IM_n\mathbb{R} \times IV_n\mathbb{R} \to IV_n\mathbb{R}$ wird hier die gleichen Symbole eingesetzt). Es gilt offenbar

$$\bigwedge_{X \in IV_n\mathbb{R}} \bigwedge_{x \in X} f(\tilde{x}) = \tilde{x} + R(b - A\tilde{x}) + (E - RA)(x - \tilde{x}) \in F(X) ,$$

Voraussetzung (2.1) ist also erfüllt und der Satz anwendbar. Es bleibt die Frage, mit welchem $\Omega \in IV_n\mathbb{R}$ man den Test $F(\Omega) \subseteq \Omega$ versuchen soll. Einmal möchte man mit wenigen, möglichst mit einem Test auskommen, andererseits sollte Ω die wahre Lösung möglichst genau einschließen. Da man die Gleitkommanäherung $\tilde{x}$ bereits hat, liegt es nahe, mit dem Punktintervall $\tilde{\tilde{x}}$ zu beginnen: $Y^0 := \tilde{\tilde{x}}$. Ist nicht $\tilde{x}$ gerade die exakte Lösung, ist natürlich $F(Y^0) \subsetneq Y^0$. Nun iteriert man $F^k(Y^0) =: Y^k$ einfach solange, bis einmal $Y^{l+1} \subseteq Y^l$ für ein $l \in \mathbb{N}$ erfüllt ist. Damit ist bereits $\hat{x} \in Y^{l+1}$ bewiesen. Das auffallende an dem Verfahren ist, daß nicht wie üblich mit einer Einschließung für $\hat{x}$ begonnen wird. Im Gegenteil wird eine Einschließung konstruktiv berechnet. Man geht aus von einer Gleitkommanäherung $\tilde{x}$ für $\hat{x}$ und legt größer werdende Intervalle Ω um $\tilde{x}$ bis schließlich $\hat{x} \in \Omega$ bewiesen werden kann. Je genauer die Anfangsnäherung $\tilde{x}$ ist, desto weniger Iterationen werden benötigt und desto enger werden die Lösungsintervalle. In der Praxis berechnet man

$$z := \tilde{x} \oplus R \odot (b \ominus A \odot \tilde{x}) \in IV_n\mathbb{R} \quad \text{und} \quad B := E \ominus R \odot A \in IM_n\mathbb{R}$$

natürlich nur einmal, so daß (2.9) sich reduziert zu

$$F(Y) = z \oplus B \odot (Y \ominus \tilde{x}) . \tag{2.10}$$

Wie man sieht, ist der Aufwand gering; wir gehen darauf später noch ein.

Wegen $d(Y^k) \geq d(z)$ ist es sinnvoll, als Anfangswert $Y^0 := z$ zu wählen. Je näher der Näherungsvektor $\tilde{x}$ bei der wahren Lösung $\hat{x}$ liegt, desto kleineren Durchmesser hat z; je besser die Näherungsmatrix R die wahre Inverse A^{-1} von A approximiert, desto kleiner wird der Spektralradius von B und desto schneller konvergiert das Verfahren (2.10)!

Durch Satz 2.2 wird zunächst nur sichergestellt, daß f einen Fixpunkt $\hat{x} \in \Omega$ besitzt. Aus der Definition (2.5) folgt nur

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

Das bedeutet, daß $b - A\hat{x}$ im Kern von R liegt. Damit ist jedoch noch nicht bewiesen, daß $\hat{x}$ tatsächlich (eine) Lösung des Gleichungssystems $Ax = b$ ist. Diese Frage wird im nächsten Satz geklärt.

Satz 2.3: Gilt in Satz 2.2 mit $\overline{F}$ aus (2.8) statt F die verschärfte Voraussetzung

$$\overline{F}(\Omega) \subsetneq \Omega , \tag{2.11}$$

was bedeuten soll, daß ein reelles $\varepsilon > 0$ existiert mit $U_\varepsilon(F(\Omega)) \subseteq \Omega$ (also eine ganze Umgebung von $\overline{F}(\Omega)$ in Ω liegt), so folgt bereits, daß eine Lösung $\hat{x}$ von $Ax = b$ in Ω und sogar in $\overline{F}^i(\Omega)$ für $i \geq 0$ liegt.

Beweis: Aus (2.11) folgt die Existenz eines Vektors $\delta \in IV_n\mathbb{R}$ mit

$$\overline{F}(\Omega) + \delta = \Omega \quad \text{und} \tag{2.12}$$

$$\lambda(\delta_i) \neq 0 \neq \rho(\delta_i) \quad \text{für} \quad i = 1(1)n . \tag{2.13}$$

(λ und ρ bezeichnen wie üblich die linke und rechte Intervallgrenze).

Es sei $\varepsilon \in IM_n\mathbb{R}$ beliebig. Dann gilt mit $F^*(Y) := \{\tilde{x} \boxplus (R \boxplus \varepsilon) \boxdot (b - A\tilde{x})\} \boxplus$
$\boxplus \{E \boxminus (R \boxplus \varepsilon) \boxdot A\} \boxdot (Y \boxminus \tilde{x})$ für $Y \in IV_n\mathbb{R}$

$$\begin{aligned} &F^*(Y) \subseteq \overline{F}(Y) \boxplus \varepsilon \boxdot (b - A\tilde{x}) \boxplus \varepsilon \boxdot (A \boxdot (Y \boxminus \tilde{x})), \text{ also} \\ &d(F^*(Y)) \leq d(\overline{F}(Y)) + |\varepsilon| \cdot d(A \boxdot (Y \boxminus \tilde{x})) + d(\varepsilon) \cdot (|b - A\tilde{x}| + |A \boxdot (Y \boxminus \tilde{x})|) =: \\ &\qquad =: d(\overline{F}(Y)) + |\varepsilon| \cdot k_1 + d(\varepsilon) \cdot k_2 . \end{aligned} \tag{2.14}$$

(Für die entsprechenden Definitionen siehe Alefeld/Herzberger, Seite 152ff).

Wegen (2.13) kann ε so gewählt werden, daß

$$\lambda(\varepsilon_{ij}) \neq 0 \neq \rho(\varepsilon_{ij}) \quad \text{für} \quad i,j = 1(1)n \quad \text{und} \tag{2.15}$$

$$|\varepsilon| \cdot k_1 + d(\varepsilon) \cdot k_2 < d(\delta) \tag{2.16}$$

gilt. Dann ist nach (2.12) und (2.16) aber

(2.17) $F^*(\Omega) \subsetneq \Omega$.

Wegen (2.15) existiert bekanntermeise ein nicht-singuläres $\tilde{R} \in R + \varepsilon$. Mit $\tilde{F}(Y) := (\tilde{x} + \bar{R}(b - A\tilde{x})) \boxplus (E - \bar{R}A) \boxdot (Y \boxminus \tilde{x})$ folgt dann aus (2.17):

(2.18) $\tilde{F}(\Omega) \subsetneq \Omega$.

Für die Funktion $\tilde{f} : V_n\mathbb{R} \longrightarrow V_n\mathbb{R}$ mit

(2.19) $\tilde{f}(x) := x + \tilde{R}(b - Ax)$

gelten dann mit $\bar{F}$ und (2.18) die Voraussetzungen von Satz 2.2, es existiert demnach ein Fixpunkt $\hat{x}$ von $\tilde{f}$: $\tilde{f}(\hat{x}) = \hat{x}$. Somit folgt aus (2.19)

(2.20) $\tilde{R} \cdot (b - A\hat{x}) = o$.

Aus der Nicht-Singularität von $\tilde{R}$ und (2.3) ergibt sich die Behauptung.

Die entscheidende Voraussetzung von Satz 2.3 ist, daß nicht nur $\bar{F}(\Omega)$, sondern auch eine offene Umgebung noch ganz in Ω enthalten ist. So kann man "in der Nähe" von R eine nicht-singuläre Matrix $\tilde{R}$ finden, eine neue Funktion $\tilde{f}$ definieren, und so die Behauptung beweisen. Die Eindeutigkeit der Lösung, d.h. die Nicht-Singularität von A, ist damit allerdings noch nicht bewiesen. Im nachfolgenden Satz wird sogar die Nicht-Singularität von A und von R bewiesen.

Satz 2.4: Gilt in Satz 2.2 mit $\bar{F}$ aus (2.8) statt F die verschärfte Voraussetzung

(2.21) $\bar{F}(\Omega) \subsetneq \Omega$,

so folgt bereits die Nicht-Singularität von R und von A .

Beweis. Wie wir bereits oben gesehen haben, folgt aus $\bar{F}(\Omega) \subsetneq \Omega$, daß $f(y) = y + R(b - Ay)$ einen Fixpunkt $\hat{x} \in \bar{F}(\Omega)$ besitzt mit

(2.22) $b - A\hat{x} \in \mathrm{Ker}(R)$.

Für $y \in \mathrm{Ker}(A)$ gilt für $\lambda \in \mathbb{R}$

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

d.h. $\hat{x} + \lambda y$ ist ebenfalls Fixpunkt von f. Wegen $\hat{x} \in \Omega$ und $\Omega \in IV_n\mathbb{R}$ gibt es ein $\mu \in \mathbb{R}$ mit $\hat{x} + \lambda y \in \partial\Omega$ für $y \neq o$. Da $\hat{x} + \mu y$ jedoch Fixpunkt von f ist, folgt mit (2.1) für $\bar{F}$

(2.24) $$\hat{x} + \mu y = f(\hat{x} + \mu y) \in \bar{F}(\hat{x} + \mu y) \subseteq \bar{F}(\Omega) \subseteq \Omega \setminus \partial\Omega$$

wegen (2.21). Also kann nicht $y \neq o$ sein und es folgt $\mathrm{Ker}(A) = o$: A ist nicht singulär. Angenommen $y \in \mathrm{Ker}(R)$ und $y \neq o$. Da A nicht singulär ist, folgt $A^{-1} \cdot y \neq o$. Wie oben folgt, daß ein $\mu \in \mathbb{R}$ existiert mit $\hat{x} + \mu(A^{-1}y) \in \partial\Omega$. Andererseits ist aber

(2.25) $$f(\hat{x} + \mu(A^{-1}y)) = \hat{x} + \mu(A^{-1}y) + R(b - A\hat{x} - \mu y) = \hat{x} + \mu(A^{-1}y) + R(b - A\hat{x}) - \mu\cdot Ry = \hat{x} + \mu(A^{-1}y) .$$

$\hat{x} + \mu(A^{-1}y)$ ist also Fixpunkt von f und es folgt wie oben $\hat{x} + \mu(A^{-1}y) \in \Omega \setminus \partial\Omega$ im Widerspruch zur Voraussetzung. Also ist $\mathrm{Ker}(R) = o$ und R nicht singulär.

Wird die Funktion $\bar{F}$ nach (2.9) zu F vergröbert, so gelten die Sätze 2.3 und 2.4 mit F statt $\bar{F}$ entsprechend wegen $\bar{F}(Y) \subseteq F(Y)$.

Neben Punktgleichungssystemen können auch Intervallgleichungssysteme behandelt werden, wie der nachfolgende Satz zeigt.

Satz 2.5: Gegeben sei eine Intervallmatrix $A \in \mathbb{IM}_n\mathbb{R}$ und ein Intervallvektor $b \in \mathbb{IV}_n\mathbb{R}$. Für die Funktion $F : \mathbb{IV}_n\mathbb{R} \to \mathbb{IV}_n\mathbb{R}$ mit

$$F(Y) := \tilde{x} \oplus R \odot (b \ominus A \odot x) \oplus B \odot (Y \ominus \tilde{x})$$

mit beliebigen $\tilde{x} \in \mathbb{V}_n\mathbb{R}$ und $R \in \mathbb{M}_n\mathbb{R}$, wobei $B := E \ominus R \odot A$ gelte für ein $\Omega \in \mathbb{IV}_n\mathbb{R}$

$$F(\Omega) \subsetneq \Omega .$$

Dann folgt bereits, daß R und jedes $A \in A$ nicht singulär sind, jedes Gleichungssystem $Ax = b$ mit $A \in A$ und $b \in b$ eindeutig lösbar ist und

$$\bigwedge_{A \in A} \bigwedge_{b \in b} A \cdot x = b \Longrightarrow x \in \bigcap_{i \geq 0} F^i(\Omega) .$$

Beweis: Die Sätze 2.2, 2.3 und 2.4 sind für jedes $A \in A$ und $b \in b$ anwendbar. Hieraus folgt unmittelbar die Behauptung.

Es sei besonders darauf hingewiesen, daß die Sätze 2.4 und 2.5 ein Verfahren an die Hand geben, für eine Matrix A zu beweisen, daß sie nicht singulär ist bzw. für eine Intervallmatrix A zu beweisen, daß sie keine singuläre Matrix enthält.

Wir gehen hierauf in Kapitel 3, Abschnitt a) näher ein.

An einem einfachen Beispiel sei gezeigt, daß selbst die Konvergenz der Defektiteration beim Gleitkomma-Gauß-Algorithmus nicht den Schluß zuläßt, daß die Ausgangsmatrix nicht singulär ist. (1,1,1) löst das Gleichungssystem

$$\begin{aligned} -8392848 \cdot x \quad -3566221 \cdot y \quad -3799934 \cdot z &= -15759003 \\ 1699109 \cdot x \quad +3679519 \cdot y \quad +2370515 \cdot z &= 7749143 \\ -6693739 \cdot x \quad +113298 \cdot y \quad -1429419 \cdot z &= -8009860 . \end{aligned}$$

Die erste Iterierte der (doppeltlang berechneten) Defektiteration stimmt auf der UNIVAC 1108 identisch mit der zweiten Iterierten überein. Die Werte sind

$$x = 1.22; \quad y = 1.54; \quad z = 7.45_{10}-9$$

Gleichwohl ist die Ausgangsmatrix singulär, da (wie man sich leicht überzeugt) die Summe der ersten und der zweiten Zeile gleich der dritten Zeile ist.

Es sei betont, daß an keiner Stelle $\tilde{x} \in Y$ oder $\tilde{x} \in \Omega$ vorausgesetzt wurde; die Näherungslösung $\tilde{x}$ muß also keineswegs im Ergebnisintervall Ω liegen. Hierfür wird auf Seite 78mit der Hilbert 7×7-Matrix ein interessantes Beispiel angegeben.

2.b) Einschließung des Defekts

Eine Möglichkeit, die Lösungsintervalle eines Gleichungssystems zu verkleinern ist, statt eine Einschließung der Lösung eine Einschließung des Defekts zu bestimmen. Ist im Fall des linearen Gleichungssystems $Ax = b$

$$Ay = b - A\tilde{x} ,$$

so folgt

$$A(\tilde{x} + y) = A\tilde{x} + Ay = A\tilde{x} + b - A\tilde{x} = b ,$$

also ist $\tilde{x} + y$ die Lösung des ursprünglichen Gleichungssystems. Bestimmt man eine Gleitkommanäherung $\tilde{x}$ für die exakte Lösung von $\tilde{x}$ von $Ax = b$ und eine Einschließung Y für y, so folgt trivialerweise

(2.26) $\quad y \in Y \Rightarrow \hat{x} = \tilde{x} + y \in \tilde{x} + Y$.

Da $b - A\tilde{x}$ i.a. nicht exakt auf dem Rechner darstellbar ist, setzt man $b = b \ominus A \odot \tilde{x}$ und schließt die Lösung von $Ay = b$ ein. Man könnte auch zunächst $\hat{x}$ einschließen, etwa $\hat{x} \in X$ und die Defektiteration mit X und Durchschnittsbildung betreiben. Bei der Berechnung von $A \cdot X - b$ würden die Intervalle jedoch sofort relativ groß, so daß weit weniger Gewinn zu erwarten ist. Die rechte Seite $b = A \odot \tilde{x} \ominus b$ wird hingegen sehr klein, sobald der Defekt genau genug (z.B. doppeltgenau) berechnet wird. Der entscheidende Vorteil ist, daß $\tilde{x}$ ein Punktintervall ist. Bei der Berechnung von $A\tilde{x}$ treten zwangsläufig Auslöschungen auf. Diese können für das Punktintervall $\tilde{x}$ tatsächlich auftreten, für das Intervall X hingegen nicht ($X \ominus X \neq o$ sondern $d(X \ominus X) = 2 \cdot d(X)$).

Die Beobachtung, den Defekt statt der Lösung einzuschließen, kann in vielen Algorithmen angewendet werden. Zum Beispiel kann das Schulz-Verfahren in naheliegender Weise abgewandelt werden (siehe Alefeld/Herzberger, Seite 252). Gesucht ist die Lösung von $AX = E$. Ist Y^* die Lösung von $AY^* = E - A\tilde{X}$ für ein $\tilde{X}$, so ist

$$A(\tilde{X} + Y^*) = A\tilde{X} + AY^* = A\tilde{X} + E - A\tilde{X} = E .$$

Also ist $\tilde{X} + Y^*$ die exakte Inverse von A. Man berechnet also zunächst eine Gleitkommanäherung $\tilde{X}$ für A^{-1}, schließt Y^* durch $Y \in \mathbb{IM}_n\mathbb{R}$ ein und erhält eine Einschliessung $\tilde{X} \oplus Y$ für A^{-1}.

Wesentlich bei der Prozedur ist immer, daß der Defekt selbst (hier z.B. $E - A\tilde{x}$) mit doppelter Genauigkeit berechnet wird. Erheblich besser wäre natürlich der Einsatz des Bohlender-Algorithmus zur genauen Berechnung von Skalarprodukten. Bei Verwendung eines solchen Algorithmus könnte sogar der Defekt des Defekts etc. eingeschlossen werden. Gegeben seien $\tilde{x}$ und $\tilde{y}$, wobei näherungsweise

(2.27) $\quad A\tilde{x} = b \quad$ und $\quad A\tilde{y} = b - A\tilde{x}$

gelte ($\tilde{x}$ und $\tilde{y}$ sind Gleitkommanäherungen). Ist dann z die Lösung von

(2.28) $\quad Az = b - A\tilde{x} - A\tilde{y}$,

so gilt

$$A(\tilde{x} + \tilde{y} + z) = A\tilde{x} + A\tilde{y} + Az = A\tilde{x} + A\tilde{y} + b - A\tilde{x} - A\tilde{y} = b ,$$

$\tilde{x} + \tilde{y} + z$ ist also die exakte Lösung von $Ax = b$. Ist weiter $Z \in \mathbb{IV}_n\mathbb{R}$ eine Einschließung von z, so folgt

$$\hat{x} = \tilde{x} + \tilde{y} + z \in \tilde{x} + \tilde{y} + Z .$$

Wichtig ist immer, daß der Defekt (z.B. die rechte Seite von (2.28)) genau ausgerechnet wird, d.h. etwa auf Gleitkommagenauigkeit. Dann ist nämlich wirklich $b - A\tilde{x} - A\tilde{y} = o$ und z liefert eine wesentliche Verbesserung. Auf diese Weise ist es übrigens möglich, die Lösung $\hat{x}$ beliebig genau zu approximieren (solange kein Unterlauf eintritt), und zwar bei Rechnung ausschließlich mit einfacher Genauigkeit! Für die genaue Berechnung des Defekts kann übrigens auch der lange Akkumulator eingesetzt werden (siehe Kaucher/Klatte und Rump [3] und Diskussion und Beispiele in Abschnitt f) dieses Kapitels). Eine doppeltlange Auswertung von (2.28) lohnt kaum, da die Größenordnung des Fehlers bereits ungefähr die Größenordnung des Defekts selbst erreicht.

Die Benutzung von (2.26) schafft übrigens (selbstverständlich bei doppeltlanger Defektberechnung) mit insgesamt minimalem Aufwand eine Genauigkeitsverbesserung um fast 1 Stellen, wenn 1-stellige Mantisse einfache Genauigkeit bedeutet. Genauere Vergleiche folgen im Abschnitt f) dieses Kapitels.

2.c) Weitere Verbesserungen

Bisher wurde davon ausgegangen, daß man sich die Gleitkommanäherungen $\tilde{x}$ für $\hat{x}$ und R für A^{-1} "irgendwie" beschafft. Betrachten wir zunächst die Berechnung von R. Es gibt verschiedene Wege, R mittels Gleitkommarechnung anzunähern. Von einer LU-Zerlegung von A ausgehend, könnte man n lineare Gleichungssysteme mit den rechten Seiten e_i (also dem i-ten Einheitsvektor) lösen und sich so die Spalten von R

sukzessiv berechnen. Andererseits kann, wieder ausgehend von der LU-Zerlegung, auch nacheinander L und U invertiert werden. Man kann auch den Algorithmus von Gauß-Jordan direkt auf A anwenden und so auf dem Speicherplatz von A die Inversion durchführen. Alle drei Verfahren benötigen bei geeigneter Programmierung eine Rechenzeit von $\sim n^3$ Operationen. Dem letzten Verfahren wurde der Vorzug gegeben. Einmal läßt es sich recht einfach programmieren, zum andern scheint die erhaltene Inverse "etwas besser" zu sein als bei den ersten beiden Verfahren. Zum Beispiel bei der Hilbert 7×7 - Matrix war nur beim letzten Verfahren die Inverse gut genug, die Defektiteration konvergent zu machen (das heißt übrigens nicht, daß die entstandene Näherung R absolut gesehen näher an A^{-1} liegt, also etwa $\|A^{-1} - R\|$ am kleinsten ist für irgendeine Norm). Der Algorithmus kann dem Anhang entnommen werden.

Um eine höhere Genauigkeit zu erzielen, könnte man R iterativ verbessern, z.B. nach dem Schulz-Verfahren. Die Kosten hierfür liegen jedoch bei $2n^3$ für jeden Iterationsschritt, so daß es sinnvoller erscheint, R von vonherein mit höherer Genauigkeit neu zu berechnen. Die Kosten hierfür liegen etwa bei n^3 multipliziert mit den Kosten für eine längere Multiplikation. Da eine doppeltlange Multiplikation z.B. auf der UNIVAC 1108 nicht einmal zweimal soviel Zeit benötigt wie eine einfachlange, ist die neue Berechnung von R mit doppelter Genauigkeit hier sogar schneller als eine Iteration. Ein Problem stellt die Verdoppelung des Speicherplatzes dar. Hier wurde sich jedoch so beholfen, daß R von vornherein auf eine Datei ausgelagert wurde. Dabei entstehen vergleichsweise geringe Mehrkosten an Rechenzeit beim Lesen und Beschreiben der Datei.

Kommen wir zur Gleitkommanäherung $\tilde{x}$ für $\hat{x}$. Zunächst liegt es nahe, als erste Näherung $x^0 = R \cdot b$ zu verwenden, da R sowieso bereits berechnet wurde. Weiter ist es sinnvoll, die Näherung iterativ zu verbessern. Als Iterationsverfahren liegt die Defektiteration (wie in (2.4)) nahe. Eine Iteration nach

$$x^{k+1} := x^0 + R(b - Ax^0) + (E - RA)(x^k - x^0)$$

liefert übrigens numerisch etwas bessere Ergebnisse, oder zumindest mit weniger Iterationen. Der Unterschied ist jedoch minimal und aus technischen Gründen wurde (2.4) vorgezogen. Es bleibt die Frage nach dem Abbruchkriterium. Die Frage ist bisher in der Literatur nicht in befriedigender Weise beantwortet worden. Daher wurde das Abbruchkriterium nach der (sehr schlecht konditionierten) Hilbert 7×7-Matrix ausgesucht. Es wurde festgesetzt, daß solange iteriert wird, bis zum erstenmal

$$(2.29) \quad \left|\frac{x^{k+1} - x^k}{x^k}\right| < 10^{-7} \quad \underline{\text{oder}} \quad \left|\frac{x^{k+1} - x^k}{x^k}\right| \geq 10^{-(k-2)/2}$$

gilt. Man sieht sofort, daß maximal 17 Iterationen möglich sind, da sich (2.29) dann schreibt

$$\Delta_{rel} < 10^{-7} \qquad \underline{\text{oder}} \quad \Delta_{rel} \geq 10^{-(16-2)/2} = 10^{-7} .$$

Das Kriterium besagt im wesentlichen, daß in je zwei Iterationen mindestens eine neue Stelle gesichert werden muß. Das ist bei der Hilbert 7×7-Matrix gerade der Fall. Die hohe Zahl 17 sollte jedoch nicht erschrecken; im allgemeinen (bis auf ganz wenige, äußerst schlecht konditionierte) Ausnahmen kommt man mit zwei Iterationen aus. Wenn (2.29) zum erstenmal erfüllt wurde, wird $\tilde{x} := x^{k+1}$ gesetzt. Im vorigen Abschnitt haben wir gesehen wie vorteilhaft es ist, statt $\hat{x}$ den Defekt einzuschließen. Dafür braucht man aber auch eine Näherung $\tilde{y}$ für die Lösung y von $Ay = b - A\tilde{x}$. Man kann y auf dieselbe Art und Weise wie $\tilde{x}$ erhalten. Dann wäre etwa $y^0 = R(b - A\tilde{x})$ und

$$y^{k+1} := y^k - R(b - A\tilde{x} - Ay^k) .$$

Man könnte versuchen, beide Verfahren zu kombinieren und zu iterieren mit

$$(2.30) \quad \begin{aligned} &x^0 := Rb; \quad y^0 := R(b - Ax^0); \\ &x^{k+1} := x^k + y^k; \quad y^{k+1} := y^k + R(b - Ax^{k+1} - Ay^k) . \end{aligned}$$

Wäre der Zuschlag $R(b - Ax^{k+1} - Ay^k) =: \Delta$ in der zweiten Zeile die tatsächliche Lösung von $Az = b - Ax^{k+1} - Ay^k$, so folgte

$$A \cdot x^{k+2} = A(x^{k+1} + y^{k+1}) = Ax^{k+1} + Ay^k + b - Ax^{k+1} - Ay^k = b ,$$

also wäre x^{k+2} die exakte Lösung $\hat{x}$. Als $\bar{x}$ bzw. $\bar{y}$ wäre in diesem Fall x^{k+1} bzw. y^{k+1} zu verwenden. Der Aufwand für eine Iteration beträgt $3n^2$, der Aufwand für eine Iteration von $\bar{x}$ und $\bar{y}$ mit dem vorher beschriebenen Verfahren $2n^2$ und $3n^2$, resp. Würde man in (2.30) 5/3-mal so viele Iterationen benötigen wie vorher für $\bar{x}$ und $\bar{y}$ zusammen, hätte man erst die gleiche Rechenzeit verbraucht. Verschiedene Beispiele haben jedoch gezeigt, daß (2.30) der Einzeliteration von $\bar{x}$ und $\bar{y}$ in der Praxis klar unterlegen ist.

Eine Konvergenzbeschleunigung kann erzielt werden, wenn in (2.10) die Intervalle - grob gesprochen - etwas"erweitert werden.

Definition 2.6: Ein Intervall $J \in \mathbb{IR}$ heißt gegenüber $I = [a,b] \in \mathbb{IR}$ um ε erweitert (geschrieben $J = I \circ \varepsilon$), wenn gilt

$$J := \begin{cases} [a - (b - a) \cdot \varepsilon \ , \ b + (b - a) \cdot \varepsilon] & \text{für } a \neq b \\ [\nabla(a - \eta), \Delta(b + \eta)] & \text{für } a = b \end{cases} ,$$

wobei η die absolut-kleinste darstellbare Maschinenzahl ist. Ist J kein Nullintervall, so gilt

$$d(J) = (1 + 2\varepsilon) \cdot d(I) .$$

Es ist wichtig, zu bemerken, daß nur der Durchmesser von I um einen Faktor $1 + 2\varepsilon$ erweitert wird. Die Idee ist nun, statt (2.10) die Funktion $G(Y) = F(Y) \circ \varepsilon$ zu verwenden. Da durch die iterative Anwendung von F die Test-Einschließungs-Intervalle ohnehin immer mehr aufgebläht werden, nimmt man durch einen "freiwilligen Zuschlag" der Iteration "etwas Arbeit ab". Wir definieren demnach als Teilstück

des Algorithmus

$Y^0 := z$; $k := -1$;
repeat $k := k + 1$; $Y^k := Y^k \circ \varepsilon$; $Y^{k+1} := z \oplus B \odot (Y^k \ominus \bar{x})$
until $Y^{k+1} \subsetneq Y^k$;

(Das Programmstück ist für die Einschließung von $\bar{x}$, nicht für den Defekt $\bar{y}$ geschrieben).

Nach Verlassen der repeat-Schleife ist $\hat{x} \in Y^{k+1}$ bewiesen. Wie man sieht, wird $Y^{k+1} \subsetneq Y^k$ getestet, Y^{k+1} (freiwillig) um ε erweitert und Y^{k+2} errechnet, etc. Wie sich in der Praxis jedoch zeigt, wird nur bei äußerst schlecht konditionierten Problemen die Schleife überhaupt mehrmals durchlaufen. Die ε-Erweiterung (bereits für Y^0) hat die Vorteile, daß

a) in fast allen gerechneten Beispielen bereits $Y^1 \subsetneq Y^0$ ist,
b) das Ergebnisintervall praktisch nicht weiter wird,
c) im Fall, daß keine Einschließung möglich ist, die Schleife schneller abbricht.

Zum Abbruchkriterium dieser Iteration wird im nächsten Abschnitt dieses Kapitels noch etwas gesagt. In praxi hat sich der Wert $\varepsilon = 0.1$ als optimal erwiesen. Der Wert 10% war in den Beispielen der Größtmögliche, um a) und b) zu erhalten. Es sei nochmals betont, daß hier eine Konvergenzbeschleunigung eines Intervallverfahrens vorliegt.

Eine weitere Verbesserung des Verfahrens ist der Übergang vom Gesamt- zum Einzelschrittverfahren in (2.10). Bezeichnet Y_i bzw. z_i die i-te Komponente von Y bzw. z und B_i die i-te Zeile von B, so schreiben wir statt (2.10)

(2.31) for i := 1 to n do $Y_i := z_i \oplus B_i \odot (Y \ominus \bar{x})$;

wie wir noch sehen werden, ist es wichtig, wie die ε-Erweiterung eingebaut wird. Wir schreiben

$$
\begin{array}{l}
Y^0 := z;\ \ k := -1; \\
\underline{\text{repeat}}\ k := k+1;\ Y^{k+1} := Y^k := Y^k \circ \varepsilon\ ; \\
\qquad \underline{\text{for}}\ i := 1\ \underline{\text{to}}\ n\ \underline{\text{do}}\ Y_i^{k+1} := z_i \oplus B_i \odot (Y^{k+1} \ominus \tilde{x}) \\
\underline{\text{until}}\ \ Y^{k+1} \subsetneq Y^k\ ;
\end{array}
\tag{2.32}
$$

In dieser Form wird jede berechnete Komponente gleich weiterverwendet, es handelt sich also um ein Einzelschrittverfahren. Man könnte auf die Idee kommen, die ε-Erweiterung bei jedem Einzelschritt anzuwenden. U.U. wäre man gar versucht, die ε-Erweiterung nicht zu vollziehen, wenn für eine Komponente $Y_i^{k+1} \subsetneq Y_i^k$ gilt. In beiden Fällen könnte das Verfahren dann abgebrochen werden, wenn für n "aufeinanderfolgende" Komponenten $Y_i^{k+1} \subsetneq Y_i^k$ gilt (aufeinanderfolgend heißt im Sinne der Zählweise $1,2,\ldots,n-1,n,1,2,\ldots$). Es hat sich jedoch gezeigt, daß für beide Formen Beispiele existieren, wo (2.32) eine Einschließung der exakten Lösung liefert, die beiden anderen Verfahren jedoch nicht zum Ziel führen.

Wir wollen noch auf einen anderen Grund eingehen, warum die ε-Erweiterung in gewissem Sinne sogar notwendig sein kann. Um für F aus (2.10) ein $\Omega \in IV_n\mathbb{R}$ zu finden mit $F(\Omega) \subsetneq \Omega$ wird man ausgehend von einem $X^0 \in IV_n\mathbb{R}$ solange $X^i := F^i(X^0)$ bilden, bis zum ersten Mal $X^{i+1} \subsetneq X^i$ gilt. Da im vorliegenden Fall F sogar isoton ist, würde aus $X^i \subseteq X^{i+1} = F(X^i)$ sofort $X^k \subseteq X^{k+1}$ für $k \geq i$ folgen, die Iteration würde nie zum Ende kommen. Das gilt insbesondere für das "bestmögliche" (Punkt-)Anfangsintervall $X^0 := [x]$. Prüft man jedoch zunächst $F(X^0) \subsetneq X^0$, und setzt $X^1 := F(X^0) \circ \varepsilon$, wird durch diese Vergröberung u.U. erst die Konvergenz ermöglicht. Es ist zu empfehlen, den Wert für ε nach fünf Iterationen zu erhöhen (z.B. bei jeder Iteration zu verdoppeln), um divergente Fälle schneller auszuschließen.

Man könnte versuchen, die jeweiligen Ergebnisintervalle Y^{k+1} mit dem vorhergehenden Y^k zu vereinigen, um eine schnellere Konvergenz zu erzielen. Es zeigt sich jedoch, daß damit möglicherweise viel an Genauigkeit verschenkt wird, andererseits die Konvergenz jedoch nicht beschleunigt wird. Beim Abbruchkriterium (2.29) für die Gleitkommiteration von x kann es passieren, daß das Kriterium vorzeitig anspricht, d.h., daß die Iteration sehr langsam konvergiert. In diesem Fall konvergiert das Verfahren (2.32) weiter auf die Lösung zu. Während der Iteration blähen sich die Intervalle wenig auf und nähern sich immer mehr der Lösung, bis $Y^{k+1} \subsetneq Y^k$ erzielt wird. Entsprechende Beipspiele folgen in Abschnitt f) dieses Kapitels. Hätte man nach obigem Vorschlag zwei aufeinanderfolgende Iterationsintervalle jeweils vereinigt, währen die Lösungsintervalle um Größenordnungen schlechter ausgefallen. Ist einmal $Y^{l+1} \subsetneq Y^l$ erreicht, könnten die Ergebnisintervalle durch Fortfahren der Iteration mit Durchschnittsbildung noch verbessert werden. Der Algorithmus schreibt sich dann

$$
\begin{array}{l}
Y^0 := Y^{l+1};\ k := -1; \\
\underline{\text{repeat}}\ k := k+1;\ Y^{k+1} := Y^k; \\
\qquad \underline{\text{for}}\ i := 1\ \underline{\text{to}}\ n\ \underline{\text{do}}\ Y_i^{k+1} := (z_i \oplus B_i \odot (Y^k \ominus \tilde{x})) \cap Y_i^k \\
\underline{\text{until}}\ \ Y^{k+1} = Y^k;
\end{array}
\tag{2.33}
$$

Die Praxis hat jedoch gezeigt, daß selbst bei doppeltgenauer Rechnung von (2.33) der maximale relative Fehler der Ergebnisintervalle nur um Promille verbessert werden kann. Daraus folgt, daß der Algorithmus in dieser Hinsicht nahezu optimale Ergebnisintervalle liefert.

Ist σ eine Schranke über dem Spektralradius von $E - R \cdot A$, so kann die Formel

$$
\|x^0 - \hat{x}\| \leq \frac{1}{1-\sigma} \cdot \|x^0 - x^1\| \qquad \text{für } \sigma < 1 \tag{2.34}
$$

zur Abschätzung von Schranken für die Lösung $\hat{x}$ verwendet werden, wie etwa in Alefeld/Apostolatos und Kulisch 2 vorgeschlagen (x^1 wird aus x^0 nach (2.4) intervallmäßig berechnet). Eine ähnliche Abschätzung wurde von Wongwises verwendet, wobei das erhaltene Intervall nach

$$Y^{k+1} := (R \odot b \oplus (E \ominus R \odot A) \odot Y^k) \cap Y^k \tag{2.35}$$

verbessert wurde. Die Formeln (2.10) und (2.35) sind theoretisch äquivalent, in der Praxis ist (2.10) der Formel (2.35) jedoch deutlich überlegen. Das liegt daran, daß statt mit dem Intervall Y, das die Lösung $\hat{x}$ einschließt, mit dem Intervall $Y - \tilde{x}$ i.a. ein Nullintervall, multipliziert wird. Oben wurde bereits bemerkt, daß die von (2.10) gelieferten Intervalle selbst durch doppeltgenaue Ausführung von (2.33) kaum noch verbessert werden. Eine Abschätzung für σ kann man mittels

$$\sigma \leq \|E - R \cdot A\| \tag{2.36}$$

erhalten, wobei irgendeine Norm verwendet werden kann. Da Intervalloperationen verwendet werden müssen, stellt (2.36) u.U. eine Oberschätzung von σ dar. Wie die Sätze 3 bis 5 zeigen, arbeitet der im nächsten Abschnitt angegebene Algorithmus ohne daß eine Abschätzung für σ bekannt zu sein braucht. Es sollte noch bemerkt werden, daß (2.34) eine Kugel um x^0 angibt, die die Lösung $\hat{x}$ garantiert enthält. Der Radius der Kugel ist unabhängig von der Größe der Komponenten von $\hat{x}$. Sind also die Komponenten von $\hat{x}$ sehr unterschiedlich im Betrag, werden Komponenten von relativ kleinem Betrag proportional zur maximalen Differenz der Exponenten der Komponenten von $\hat{x}$ überschätzt. In unserem Algorithmus, der mit (2.10) arbeitet, wird dieser Nachteil dadurch vermieden, daß statt der einen Zahl σ "alle verfügbare Information" verwendet wird. Die Ausführung von (2.10) verursacht geringe Kosten im Vergleich zu den Gesamtkosten des Algorithmus

und der dafür gewonnenen Qualität. Abgesehen davon, daß (2.34) nur anwendbar ist, wenn die rechte Seite von (2.36) < 1 ist, lieferte der im nächsten Abschnitt angegebene Algorithmus mittels (2.10) in allen gerechneten Beispielen bessere Ergebnisse. Entsprechende Beispiele werden im Abschnitt f) dieses Kapitels angegeben.

Wie eben erwähnt, werden für Gleichungssysteme, bei denen die Komponenten der Lösung $\hat{x}$ stark differieren, bei Verwendung von (2.34) auch große Unterschiede in der Genauigkeit auftreten. Man kann versuchen, dies mittels einer geeigneten Equilibrierung der Matrix zu umgehen. Man könnte etwa zunächst Näherungslösungen berechnen, und die Matrix derart equilibrieren, daß alle Komponenten der Lösung des entstehenden Gleichungssystems in der Nähe von 1 liegen. Wir wollen hierzu ein Beispiel angeben. Gegeben sei das Gleichungssystem

$$\begin{aligned} x + 2y &= 201 \\ 2x + 3y &= 302 \end{aligned} \quad ; \tag{2.37}$$

die Lösung des Gleichungssystems ist offenbar $\hat{x} = (1,100)'$. In (2.38) ist die exakte Inverse A^{-1} und eine Näherungsinverse R angegeben

$$A^{-1} = \begin{pmatrix} -3 & 2 \\ 2 & -1 \end{pmatrix}; \qquad R = \begin{pmatrix} -2.9 & 2.1 \\ 1.9 & -0.9 \end{pmatrix} \tag{2.38}$$

Daraus errechnet sich

$$E - R \cdot A = \begin{pmatrix} -0.3 & -0.5 \\ -0.1 & -0.1 \end{pmatrix}, \tag{2.39}$$

nach der Zeilensummennorm also die Abschätzung $\sigma \leq 0.8$ für den wahren Spektralradius $\rho(E - R \cdot A) = 0.44$. Skaliert man das Gleichungssystem von rechts mit

$$D = \begin{pmatrix} 1 & 0 \\ 0 & 100 \end{pmatrix},$$

d.h. A geht über in $A \cdot D$, wird die Lösung des neuen Gleichungssystems $AD \cdot x = b$ zu $\hat{x} = (1,1)'$. Der Spektralradius von $E - R \cdot A$ bleibt durch die Skalierung unverändert. Als neue Abschätzung erhalten wir jedoch

$$\rho(E - D^{-1}R \cdot AD) \leq 51.3 \quad . \tag{2.40}$$

Dabei hat sich die Kondition von $\mathrm{cond}(A) = 18$ auf $\mathrm{cond}(AD) = 908$ verändert. Wie man sieht, kann (2.34) durch die Skalierung wegen (2.40) gar nicht mehr angewendet werden und die Kondition hat sich wesentlich verschlechtert.

2.d) Der Algorithmus

Im folgenden wird ein Algorithmus zur Einschließung der Lösung eines linearen Gleichungssystems, basierend auf den Sätzen 2 bis 5, angegeben. Dabei wurden die Beobachtungen des letzten Abschnitts miteingebaut. Der Algorithmus wird teilweise in PASCAL, teilweise in Klartext formuliert, so daß er leicht in jede Programmiersprache übertragbar ist. Der in FORTRAN formulierte Algorithmus ist als Paket LGL auf der UNIVAC 1108 des Rechenzentrums der Universität Karlsruhe verfügbar. Der Quelltext unter Verwendung des langen Akkumulators (siehe Seite 60) ist im Anhang angegeben.

Algorithmus 2.1 Garantierte Einschließung der Lösung des Punkt-Gls $Ax = b$.

1) Berechne $R \approx A^{-1}$.

2) Berechne $B := E \ominus R \odot A$ intervallmäßig.

3) $x^0 := R \cdot b$; $k := -1$;

   repeat $k := k + 1$; $x^{k+1} = x^k + R \cdot (b - Ax^k)$

   until $|x^{k+1} - x^k| \,/\, |x^k| \geq 10^{-k/2}$ oder

   $|x^{k+1} - x^k| \,/\, |x^k| < 10^{1-t}$;

   Setze $\tilde{x} := x^{k+1}$ und $r := k + 1$;

   $y^0 := R(b - A\tilde{x})$; $k := -1$;

   repeat $k := k + 1$; $y^{k+1} = y^k + R(b - A\tilde{x} - Ay^k)$

   until $|y^{k+1} - y^k| \,/\, |y^k| \geq 10^{-k/2}$ oder

   $|y^{k+1} - y^k| \,/\, |y^k| < 10^{2-t}$;

   Setze $\tilde{y} := y^{k+1}$ und $r := r + k + 1$;

4) Berechne $Y^0 := Z := \tilde{y} \oplus R \odot (b \ominus A \odot \tilde{x} \odot A \odot \tilde{y})$ intervallmäßig; $k := -1$;

5) repeat $k := k + 1$; $Y^{k+1} := Y^k := Y^k \circ \varepsilon$,

   for $i := 1$ to n do $Y_i^{k+1} := Z_i \oplus B_i \odot (Y^k \ominus \tilde{y})$

   until $Y^{k+1} \subseteq Y^k$ oder $\{k > 2.25 \cdot r - 1 =: \mathrm{bool}\}$;

   if bool then stop;

6) Setze $Y := Y^{k+1}$; A ist nicht singulär und es gilt $\hat{x} \in \tilde{x} \oplus Y$.

In der praktischen Implementierung des Algorithmus (siehe Anhang) ist der Speicherplatz durch Auslagerung von R auf n^2 reduziert.

Es ist ratsam, die Defekte doppeltlang zu berechnen. Das sind die Ausdrücke in den Klammern in den Schritten 3 und 4, in Schritt 3 mit Gleitkomma-, in Schritt 4 mit Intervallrechnung. Auch die Berechnung B in Schritt 2 kann zur Erzielung höherer Genauigkeit doppeltlang berechnet werden. Unter "doppeltlang berechnen" ist

jeweils zu verstehen, das Ergebnis doppeltlang zu berechnen, jedoch einfachlang zu speichern. Noch besser wäre natürlich, den Bohlender-Algorithmus zur genauen Berechnung von Skalarprodukten oder einen langen Akkumulator einzusetzen. Entsprechende Beispiele sind in Abschnitt f) dieses Kapitels angegeben. In Schritt 6 sollte das Ergebnisintervall doppeltlang berechnet werden, da i.a. mehr als einfache Genauigkeit erzielt wird. In Schritt 3 werden jeweils höchstens 17 bzw. 15 Iterationsschritte durchgeführt. Die Genauigkeit des Defekts ist mit 10^{-6} etwas geringer veranschlagt, da der Defekt gegenüber der tatsächlichen Lösung ohnehin klein ist. In Schritt 5, der Aufblähung der Intervalle, ist eine Schranke von höchstens $2.25 \cdot r - 1$ Iterationen eingebaut. Eine entsprechende Erklärung findet sich im nächsten Abschnitt dieses Kapitels. Aller Erfahrung nach ist eine Einschließung in der übergroßen Mehrzahl der Fälle bereits in einem Schritt gefunden. Selbst in äußerst schlecht konditionierten Beispielen ist die Anzahl der "Aufblähungen" nie über 7 gestiegen. Mit der Beschränkung werden divergente Fälle vorzeitig gestoppt, um Rechenzeit zu sparen.

Wie man leicht nachprüft, sind die Voraussetzungen von Satz 2, insbesondere (2.1) für F nach (2.10), erfüllt. Aus den Sätzen 3 und 4 folgt die Richtigkeit der Behauptung aus Schritt 6 von Algorithmus 2.1. Der Algorithmus kann in leicht abgeänderter Weise auch auf Intervallgleichungssysteme angewendet werden.
Sei $A \cdot x = b$ ein Intervallgleichungssystem, also

$$(2.41) \qquad A \cdot x = b \text{ mit } A \in \mathbb{IM}_n\mathbb{R} \text{ und } b \in \mathbb{IV}_n\mathbb{R} \qquad \text{und sei}$$

$$A \in \mathbf{A}, \text{ z.B. } A = m(\mathbf{A}) = \left(\frac{\underline{a}_{ij} + \overline{a}_{ij}}{2}\right) .$$

Der Algorithmus zur Lösung von (2.41) schreibt sich dann:

Algorithmus 2.2: Garantierte Einschließung der Lösung des Intervall-Gls $Ax = b$

1) Berechne $R \approx A^{-1}$

2) Berechne $B := E \ominus R \odot A$ intervallmäßig

3) $x^0 := R \cdot b$; $k := -1$;
repeat $k := k + 1$; $x^{k+1} = x^k + R \cdot (b - Ax^k)$
until $|x^{k+1} - x^k| \,/\, |x^k| \geq 10^{-k/2}$ oder
$|x^{k+1} - x^k| \,/\, |x^k| < 10^{1-t}$;
Setze $\tilde{x} := x^{k+1}$ und $r := k + 1$;
$y^0 := R(b - A\tilde{x})$; $k := -1$;
repeat $k := k + 1$; $y^{k+1} = y^k + R(b - A\tilde{x} - Ay^k)$
until $|y^{k+1} - y^k| \,/\, |y^k| \geq 10^{-k/2}$ oder
$|y^{k+1} - y^k| \,/\, |y^k| < 10^{2-t}$;
Setze $\tilde{y} := y^{k+1}$ und $r := r + k + 1$;

4) Berechne $Y^0 := Z := \tilde{y} \oplus R \odot (b \ominus A \odot \tilde{x} \ominus A \odot \tilde{y})$ intervallmäßig; $k := -1$;

5) repeat $k := k + 1$; $Y^{k+1} := Y^k := Y^k \circ r$,
for $i := 1$ to n do $Y_i^{k+1} := Z_i \oplus B_i \odot (Y^k \ominus \tilde{y})$
until $Y^{k+1} \subsetneq Y^k$ oder ($k > 2.25 \cdot r - 1$ =: bool);
if bool then stop;

6) Setze $Y := Y^{k+1}$. Für jedes $A \in \mathbf{A}$ und $b \in \mathbf{b}$ gilt:
- A ist nicht singulär ,
- für $\hat{x}$ mit $A\hat{x} = b$ ist $\hat{x} \in \tilde{y} + Y$.

Die wichtigsten Ergebnisse seien noch einmal zusammengefaßt:

- Algorithmus 2.2 ist für beliebige Intervallmatrizen $A \in \mathbb{IM}_n\mathbb{R}$ und Intervallvektoren $b \in \mathbb{IV}_n\mathbb{R}$ anwendbar, insbesondere auch für Punktmatrizen, Punktvektoren oder eine Kombination davon,
- wird Algorithmus 2.2 für eine Eingabe A, b erfolgreich durchgeführt, so ist damit bewiesen,
  a) daß alle $A \in A$ nicht singulär sind,
  b) daß jedes $Ax = b$ für alle $A \in A$, $b \in b$ eine eindeutig bestimmte Lösung $\tilde{x}$ besitzt und
  c) daß jede solche Lösung $\tilde{x}$ im Ergebnisintervall $\tilde{x} \oplus Y$ liegt.
- Die Rechenzeit beträgt $\sim 3n^3$.

Wir geben noch eine Abschätzung an, welcher Fehler etwa bei der Berechnung von R bzw. $E - F = A \cdot A$ zu erwarten ist. Berechnet man für A zunächst eine LU-Zerlegung und löst dann die Gleichungssysteme $Ax = e_i$ für die i-ten Einheitsvektoren e_i, so erhält man nacheinander die Spalten von R, einer Gleitkommanäherung für A^{-1}. Mit dieser Näherung R für die Inverse erhält man für die Fehlermatrix $F = A \cdot R - E$ nach Wilkinson S. 139

$$\|F\|_\infty \leq \|R\|_\infty \cdot g \cdot 2^{-t_1} \cdot (2.005n^2 + n^3 + \tfrac{1}{4} n^4 \cdot 2^{-t}) \quad (2.42)$$

Hierin ist $2^{-t_1} = 1.06 \cdot 2^{-t}$ (bei t-stelliger binärer Gleitkommaarithmetik) und g das betragsmäßig größte Pivotelement, das bei der LU-Zerlegung auftritt. Es gilt $g \leq 2^{n-1} \cdot \|A\|_\infty$, doch Wilkinson bemerkt, daß bereits $g > 8 \cdot \|A\|_\infty$ als äußerst ungewöhnlich zu bezeichnen ist. Legt man den Wert 8 zugrunde und bedenkt weiter, daß die Klammer in (2.42) im wesentlichen gleich n^3 ist, so folgt angenähert

$$\|F\|_\infty \geq 8 \cdot n^3 \cdot 2^{-t} \cdot \mathrm{cond}(A) . \quad (2.43)$$

F ist gleich $A \cdot R - E$, also (2.43) eine Abschätzung bezüglich der Rechts-Inversen von A. Benutzt man R als Linksinverse, müßte man zunächst $\|R \cdot A - E\| \leq \mathrm{cond}(A) \cdot \|F\|$ abschätzen. Es gibt wohl Beispiele, in denen die Rechts- und Linksinversen recht verschiedene Fehler für $\|E - RA\|$ und $\|E - AR\|$ aufweisen, doch haben Tests gezeigt, daß keine Formel allgemein bevorzugt wird und $\|E - RA\|$ und $\|E - AR\|$ fast immer annähernd gleich sind. Auch Wilkinson bemerkt, daß $\|E - RA\|$ und $\|E - AR\|$ i.a. gleiche Größenordnungen haben. Geht man weiter (bei doppeltlanger Rechnung) von 54-Bit Genauigkeit bei der Gleitkommarechnung aus und sagen wir einem 100 × 100 Gleichungssystem, so erhält man $8n^3 \cdot 2^{-54} \cdot \mathrm{cond}(A) = 4.44 \cdot 10^{-10} \cdot \mathrm{cond}(A)$ als rechte Seite von (2.43). Da die Funktion F aus (2.10) für $\rho(E - RA) < 1$ kontrahierend ist und, wie in Wilkinson bemerkt, die Abschätzung (2.43) angenähert auch für die Summennorm $\|\cdot\|_1$ gilt, können Matrizen bis zu einer Kondition

$$\mathrm{cond}(A) < 2.25_{10}9$$

bearbeitet werden. In der Praxis wird dieser Wert noch bei weitem überschritten.

2.e) Erweiterung des Algorithmus, Abschätzungen

Wie in [Alefeld/Herzberger] gezeigt wurde, kann der Gauß-Algorithmus intervallmäßig für streng diagonaldominante Matrizen mit Sicherheit erfolgreich beendet werden. Wir wollen zeigen, daß diese Art von Matrizen eine sehr einfache Durchführung des obigen Algorithmus gestattet. Wir betrachten das Problem zunächst für Punktmatrizen $A = (a_{ij})$. Dabei gilt für die Diagonale von A

$$\sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{ij}| < |a_{ii}| \quad \text{für } i=1(1)n . \quad (2.44)$$

Weiter sei in D die Diagonale von A und $D \in \mathbb{IM}_n\mathbb{R}$ eine Intervallmatrix, die D^{-1} enthält. Wir definieren

$$(2.45) \qquad D = (d_{ij}) \text{ mit } d_{ij} := \begin{cases} 1 \oslash a_{ij} & \text{für } i = j \\ 0 & \text{sonst.} \end{cases}$$

Die Elemente der Diagonalmatrix D sind also einfach die einschließenden Intervalle für a_{ii}^{-1}. Es gilt jetzt

$$E - D^{-1} \cdot A = C = (c_{ij}) \text{ mit } c_{ij} = \begin{cases} 0 & \text{für } i = j \\ -a_{ii}^{-1} \cdot a_{ij} & \text{sonst.} \end{cases}$$

Die Diagonale von C ist also identisch Null. Schreiben wir weiter

$$(2.46) \qquad C^* = (c_{ij}^*) \text{ mit } c_{ij}^* = \begin{cases} 0 & \text{für } i=j \\ -d_{ii} \odot a_{ij} & \text{sonst,} \end{cases}$$

so gilt offenbar

$$(2.47) \qquad C \subseteq C^* .$$

Bei der Berechnung von C^* wurde also ausgenutzt, daß die Diagonalelemente von $E - D^{-1} \cdot A$ exakt gleich Null sind. Mit den Bezeichnungen wie oben können wir also statt (2.10) die Funktion

$$(2.48) \qquad F(Y) = Z \oplus C^* \odot (Y \ominus X)$$

verwenden. Wir können voraussetzen, daß der Durchmesser der Komponenteninter-valle von C^* nach Definition (2.46) ε ist, wobei $|\varepsilon| \leq 2^{1-t}$ bei t-stelliger binärer Gleitkommaarithmetik ist. Für die i-te Zeile von C^* ist also

$$\sum_{j=1}^{n} |c_{ij}^*| = \sum_{j=1}^{n} |d_{ii} \odot a_{ij}| \leq \sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{ii}^{-1}| \cdot (1 + 2^{1-t}) \cdot |a_{ij}| =$$

$$= |a_{ii}^{-1}| \cdot (1 + 2^{1-t}) \cdot \sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{ij}| .$$

Nach (2.44) ist die letzte Summe kleiner als $|a_{ii}|$. Gilt die schärfere Voraussetzung $\sum_{\substack{j=1 \\ i \neq j}}^{n} |a_{ij}| < |a_{ii}| \cdot (1 + 2^{1-t})^{-1}$ für alle i,

so gilt

$$(2.48) \qquad \sum_{j=1}^{n} |c_{ij}^*| < 1 .$$

Für jede Zeile C^* ist also die Betragssumme und damit auch der Spektralradius kleiner als 1. Demnach ist F aus (2.48) also kontrahierend. Wenn die Rundungsfehler der Gleitkommaintervallarithmetik die Iterierten Y^k nicht zu sehr vergröbern, ist mit einer erfolgreichen Beendigung des Algorithmus zu rechnen. Wie man sieht, ist der Aufwand des gesamten Algorithmus in diesem Spezialfall $O(n^2)$.

Wie bereits erwähnt, kann der Algorithmus durch genauere Berechnung der Skalarprodukte verbessert werden. Zur Realisierung wurde auf der UNIVAC 1108 ein langer Akkumulator simuliert. Zu diesem Zweck wurde ein Feld AKKU vom Typ Integer eingeführt, dessen Komponenten man sich hintereinander geschrieben als eine große Zahl denkt, ein langer Akkumulator. Die Dimension des Feldes ist so ausgelegt, daß jede doppeltlange Zahl "hineinpaßt", d.h. ohne Fehler abgespeichert werden kann. Eine Prozedur AKADD(AKKU,D) gestattet es, die doppeltlange Zahl D zum AKKU zu addieren. Die Prozedur AKINT(AKKU,AL,AR) hat zum Ergebnis ein Intervall [AL,AR] mit Gleitkommagrenzen AL und AR, so daß die Zahl, die im AKKU steht im Intervall [AL,AR] liegt. Bei der Berechnung eines Skalarprodukts $p = \sum_{i=1}^{n} a_i \cdot b_i$ mit einfachlangen Gleitkommazahlen a_i und b_i können die einzelnen Produkte in einer doppeltlangen Variablen exakt abgespeichert werden. Bei Verwendung von AKADD und anschließend AKINT ist das Ergebnis also das engstmögliche Gleitkommaintervall, das p enthält. Die Methode wurde in Algorithmus 2.1 angewandt, und zwar in den Schritten 2 und 4. Es ergab sich dabei eine erhebliche Verbesserung der Lösungsintervalle, insbesondere wenn die Komponenten des Lösungsvektors von stark unterschiedlichem Betrag waren. Entsprechende Beispiele finden sich in Ab-

schnitt f) dieses Kapitels. Eine weitere Anwendung des langen Akkumulators ist die Folgende. Schritt 3 aus Algorithmus 2.1 wird erweitert, indem die Lösung von

$$A \cdot z = b - A\tilde{x} - A\tilde{y} \tag{2.50}$$

berechnet und iteriert wird, die Lösung von $A \cdot t = b - A\tilde{x} - A\tilde{y} - A\tilde{z}$ berechnet und iteriert wird usw. (z ist eine Näherung für z aus (2.50)). Die Schwierigkeit, die Defekte (rechte Seite von (2.50)) genau genug zu berechnen, ist durch Einsatz des langen Akkumulators gelöst. Hat man eine Näherung z für (2.50) gefunden, ersetzt man die rechte Seite der ersten Zeile von Schritt 4 durch

$$z \oplus R \odot (b \ominus A \odot \tilde{x} \ominus A \odot \tilde{y} \ominus A \odot \tilde{z}) \quad ,$$

in Schritt 5 $\tilde{y}$ durch $\tilde{z}$ und hat nach erfolgreicher Durchführung des Algorithmus bewiesen, daß für die Lösung $\hat{x}$ gilt

$$\hat{x} \in \tilde{x} \oplus \tilde{y} \oplus Y \quad .$$

Durch diese Methode kann das Ergebnisintervall ohne nennenswerten Mehraufwand in den Grenzen der Rechengenauigkeit beliebig verfeinert werden!

Eine andere Variante von Algorithmus 2.1 gestattet es, den "Defekt des Defekts" ohne Berechnung von $\tilde{z}$ einzuschließen. Hierzu wird die erste Zeile von 4 geschrieben zu

$$Y^0 := z := R \cdot (b - A \cdot \tilde{x} - A \cdot \tilde{y});$$

die Iteration in der zweiten Zeile von Schritt 5 schreibt sich zu

$$Y_i^{k+1} := Z_i \oplus B_i \odot Y^k;$$

und nach erfolgreicher Abarbeitung des Algorithmus ist bewiesen, daß

$$\hat{x} \in \tilde{x} \oplus \tilde{y} \oplus Y \quad .$$

Diese Variante von Algorithmus 2.1 bringt zunächst kaum einen Gewinn. Kombiniert man sie jedoch mit der vorhin beschriebenen Verwendung des langen Akkumulators zur genauen Berechnung der Skalarprodukte, kann der relative Fehler der Ergebnisintervalle oft um einige Zehnerpotenzen verbessert werden. Häufig sind die Ergebnisintervalle (deren Grenzen ja doppeltlange Zahlen sind) nur im letzten Bit verschieden, d.h. die auf der Maschine engstmöglichen!

Wir wollen noch die Rechenzeit des Algorithmus abschätzen. Seien α_1 die Kosten zur Berechnung einer Gleitkommaapproximation für $\hat{x}$ und α_2 die Kosten, um beweisbare Schranken für $\hat{x}$ mit Algorithmus 2.1 zu berechnen, beides mit vergleichbarer Genauigkeit. Ohne Beschränkung der Allgemeinheit verwenden wir den Gaußschen Algorithmus mit r Defekt-Iterationen, um eine Gleitkommanäherung zu berechnen. Als ein Maß für den Aufwand von Algorithmus 2.1 nehmen wir das Verhältnis $\mu = \alpha_2 / \alpha_1$. Ohne lineare Terme gilt $\alpha_1 \geq n^3/3 + 2rn^2$. Wird Schritt 5 von Algorithmus 2.1 s mal ausgeführt gilt

$$\alpha_2 \leq 2n^3 + n^2(3r + 4s + 4) \leq 6 \cdot \alpha_1 + n^2(-9r + 4s + 4) \quad .$$

Demnach kann μ mit

$$\mu \leq 6 + \frac{-9r + 4s + 4}{n/3 + 2r}$$

wie folgt abgeschätzt werden:

$$\mu \leq \begin{cases} 6 & \text{für } s \leq (9r - 4)/4 \\ 6 \cdot \left(1 + \frac{2s + 3.5r}{n}\right) & \text{sonst} \end{cases} \tag{2.51}$$

In Algorithmus 2.1 ist $\mu \leq 6$ durch die Abbruchbedingung in Schritt 5 garantiert. Es bleibt die Frage, was zu tun ist, wenn der Algorithmus versagt (bei sehr schlecht konditionierten Problemen). Die folgenden Überlegungen gelten allgemein, unabhängig von dem hier betrachteten Algorithmus. Es soll untersucht werden, was bei einer Erhöhung der Genauigkeit geschieht bzw. in welcher Art und Weise die Genauigkeit gesteigert werden soll. Wir nehmen an, daß bei einer Verdoppelung der Genauigkeit der Rechenaufwand sich vervierfacht. Dabei ist davon ausgegangen,

daß die Rechenzeit wesentlich von der Anzahl der Multiplikationen und Divisionen bestimmt wird. Allgemein sagen wir, der Aufwand erhöht sich um den Faktor l^2 bei einer ver-l-fachung der Genauigkeit. Die Genauigkeit wird beginnend mit 1 immer ver-l-facht, solange bis der Algorithmus das gewünschte Ergebnis liefert. Der schlimmste Fall, der eintreten kann ist der, daß wenn L die kleinste Genauigkeit ist, für die der Algorithmus arbeitet, man bei der Ver-l-fachung auf L - 1 stößt. Dann ist der Gesamtaufwand A gemessen an dem für die Genauigkeit L benötigten Aufwand a

$$A < a \cdot (l^2 + 1 + l^{-2} + l^4 + \ldots + l^{-2k}) = a \cdot f$$

bei (k + 2) Rechnungen insgesamt. Für den Faktor f ergibt sich

$$f < l^2 + \sum_{i=0}^{\infty} (l^{-2})^i = l^2 + (1 - l^{-2})^{-1} = l^2 + \frac{l^2}{l^2 - 1} .$$

Mit

$$(2.52) \qquad f(x) = x + \frac{x}{x - 1} \quad \text{und} \quad f'(x) = 1 - (x - 1)^{-2}$$

folgt

$$f'(x) = 0 \iff x = 0 \quad \text{oder} \quad x = 2 .$$

Der Faktor f wird also minimal für

$$l^2 = 2, \quad \text{also} \quad l = \sqrt{2} .$$

Nach dieser Überlegung ist es am optimalsten, die Genauigkeit immer zu ver-$\sqrt{2}$-fachen.

Im allgemeinen wird es so sein, daß nur eine ganzzahlige Vervielfachung der Genauigkeit zulässig ist. Für die hypothetische Vervielfachung mit dem Faktor $\sqrt{2}$ ergäbe sich ein Faktor f kleiner als 4, für eine jeweilige Verdoppelung der Genauigkeit ein Faktor kleiner als 16/3. Da im Normalfall die Wahrscheinlichkeit, daß der Algorithmus mit kleiner Genauigkeit bereits erfolgreich beendet werden

kann, recht hoch ist, scheint es am besten zu sein, zunächst mit 1,2,3,...,k-facher Genauigkeit zu rechnen und dann erst die Genauigkeit zu ver-$\sqrt{2}$-fachen (gerundet zur nächsten ganzen Zahl) oder zu verdoppeln. Ein günstiger Wert für k dürfte bei 6 liegen. Selbst bei Verdoppelung der Genauigkeit von Anfang an, ist der Gesamtaufwand im schlimmsten Fall höchstens $5\frac{1}{3}$ - mal so groß wie für die eine Berechnung mit kleinstmöglicher Genauigkeit!

2.f) Numerische Ergebnisse

Gleich an den letzten Abschnitt anschließend soll demonstriert werden, was geschieht, wenn in Schritt 3 von Algorithmus 2.1 "nicht oft genug" iteriert wird. Für eine diagonaldominante 5 × 5-Matrix, für die (2.44) "nicht gut" erfüllt war, ergab sich (gerundet)

Iterationen für die erste Komponente von x^k	Iterationen für die erste Komponente von y^k
1.500	5.859_{10}-3
0.750	2.930_{10}-3
1.125	4.395_{10}-3
0.938	3.662_{10}-3
1.031	4.028_{10}-3
0.984	3.845_{10}-3
1.008	3.937_{10}-3
0.996	3.891_{10}-3

Tabelle 2.1 Gleitkommaiterationen

In der letzten Zeile stehen jeweils die Werte für die erste Komponente von $\tilde{x}$ und $\tilde{y}$. Die Summe der letzten beiden Iterierten müßte die exakte Lösung approximieren. Die genaue Lösung ist $1 + 1.4_{10}$-8 , die Summe jedoch 9.999846_{10}-1 mit einem relativen Fehler von 1.54_{10}-5. Jetzt setzt die Iteration, die Aufblähung der mutmaßlichen Einschließungsintervalle in Schritt 5 ein. In der folgenden Tabelle sind

die errechneten Intervalle für die erste Komponente angegeben. Beim letzten Intervall trat (wie bei den anderen Intervallkomponenten auch) zum ersten mal Enthaltensein im vorangehenden Intervall ein und der Algorithmus konnte erfolgreich beendet werden.

$[3.91386356_{10}\ -3\ ,$	$3.91386362_{10}\ -3]$	$6_{10}\ -11$
$[3.90241947_{10}\ -3\ ,$	$3.90241973_{10}\ -3]$	$26_{10}\ -11$
$[3.90681508_{10}\ -3\ ,$	$3.90681555_{10}\ -3]$	$47_{10}\ -11$
$[3.90630600_{10}\ -3\ ,$	$3.90630652_{10}\ -3]$	$52_{10}\ -11$
$[3.90623015_{10}\ -3\ ,$	$3.90623059_{10}\ -3]$	$44_{10}\ -11$
$[3.90623283_{10}\ -3\ ,$	$3.90623321_{10}\ -3]$	$38_{10}\ -11$
$[3.90623405_{10}\ -3\ ,$	$3.90623443_{10}\ -3]$	$38_{10}\ -11$
$[3.90623415_{10}\ -3\ ,$	$3.90623443_{10}\ -3]$	$28_{10}\ -11$

Tabelle 2.2 Intervalliteration

In der dritten Spalte ist jeweils die Differenz der Intervallgrenzen angegeben. Wie man sieht wächst diese zunächst, bleibt dann jedoch trotz ε-Aufblähung von gleicher Größenordnung. Das Ergebnisintervall für die erste Komponente des Lösungsvektors lautet

$$[1.00000001391,\ 1.00000001429]$$

mit einem relativen Fehler von $3.78_{10}\ -10$. Wie man sieht, sind die Intervalle $F^i(\Omega)$ praktisch ohne Genauigkeitsverlust auf die exakte Lösung hin-konvergiert, und der erzielte relative Fehler ist

a) fast der beste erzielbare Fehler ausgehend von der schlechten Gleitkommanäherung 0.9998 und

b) um Größenordnungen besser als die Gleitkommanäherung (trotz Defektiteration).

Im vorliegenden Fall ist die Matrix so konstruiert worden, daß

$$\sum_{\substack{j=i \\ j\neq i}}^{5} |a_{ij}| = \frac{1}{2} \cdot |a_{ii}| \quad , \tag{2.53}$$

die Matrix ist also diagonaldominant und vom Grad 5. Die schlechte Konvergenz der Gleitkommaiteration ist auf die schlechte Näherung D^{-1} von A^{-1} zurückzuführen. Auch bei größeren Matrizen wurden ähnliche Beobachtungen gemacht. Weiterhin ist natürlich die Konditionszahl der Matrix entscheidend. Bei einer besser konditionierten 25×25-Matrix, mit (2.53), wurde nach 9 Gleitkomma- und 3 Intervalliterationen ein relativer Fehler von höchstens $3_{10}\ -16$ in allen Komponenten erreicht.

Wie man sieht, ist es von großer Wichtigkeit, die reellen Iterationen in Schritt 3 möglichst solange auszuführen, bis keine Verbesserung der Näherung mehr möglich scheint. Andererseits sollte natürlich auch nicht länger als notwendig iteriert werden, um Rechenzeit zu sparen. Es ist kaum möglich, ein greifendes Kriterium zu finden (das ist ja gerade die Crux bei der Gleitkommarechnung). Daher wurde in Anlehnung an die sehr schlecht konditionierten Hilbert-Matrizen das Kriterium des Algorithmus 2.1 gewählt.

Um die gemachten Aussagen zu unterstreichen, sei das Beispiel der Hilbert 7×7-Matrix angeführt. Die Koeffizienten sind durch Multiplizieren mit einem geeigneten Faktor ganzzahlig gemacht. Die folgende Tabelle zeigt wieder in der linken Spalte die Iterierten der ersten Komponente von $\tilde{x}$, in der rechten Spalte entsprechend für $\tilde{y}$.

11.658	0.2884
5.336	0.1853
7.594	0.2221
6.788	0.2090

Tabelle 2.3 Gleitkommaiterierte für Hilbert 7×7

Wieder sind nur die Iterierten der ersten Komponente angegeben, die anderen Komponenten verhalten sich ähnlich. Die exakte Lösung ist 7, die Gleitkommanäherung aus je 3 Iterationen für Lösung und Defekt ist 6.9965. Manches Abbruchkriterium hätte bei Vorlage der obigen Werte bereits mit der Meldung "das Verfahren konvergiert nicht" abgebrochen. Der relative Fehler der Gleitkommarechnung liegt bei 4.95_{10}-4 trotz Iteration. Es sollte betont werden, daß die erste "Näherung" 11.658 ist; für die exakte Lösung 7 ein katastrophaler Fehler. Nachfolgend sind wieder die "Aufblähungs"-Intervalle aus Schritt 5 von Algorithmus 2.1 angegeben, und in der dritten Spalte jeweils deren Durchmesser:

[0.213675300 , 0.213675311]	11_{10}-9
[0.211995313 , 0.211995425]	112_{10}-9
[0.212595554 , 0.212595772]	218_{10}-9
[0.212380990 , 0.212381326]	336_{10}-9
[0.212457556 , 0.212458059]	503_{10}-9
[0.212430058 , 0.212430779]	721_{10}-9
[0.212439708 , 0.212440703]	995_{10}-9
[0.212436024 , 0.212437393]	1369_{10}-9
[0.212437034 , 0.212438883]	1849_{10}-9
[0.212437201 , 0.212437824]	623_{10}-9

Im vorliegenden Beispiel blähten sich die Intervalle stärker auf, da $\varepsilon = 1.0$ gesetzt wurde. Die erzielte Einschließung für die Lösung lautet

[6.999999471 , 7.000000294]

mit einem relativen Fehler von 1.1_{10}-7. Der relative Fehler kann bei Übergang von 1.0 zu 0.1 für ε noch um etwa eine 10er-Potenz verbessert werden; läßt man die ε-Erweiterung ganz weg, ergibt sich praktisch keine Änderung zu $\varepsilon = 0.1$. Wieder ist die Verbesserung gegenüber der Gleitkommanäherung eklatant und es

bestätigt sich die Aussage, daß "genügend" viele Gleitkommaiterationen durchgeführt werden müssen, um ein gutes Ergebnis zu erzielen. Führt man etwa für die Näherung $\tilde{x}$ für die Lösung und $\tilde{y}$ für den Defekt je 10 Iterationsschritte durch, gewinnt man nocheinmal drei Stellen hinzu. Die Konditionszahl im vorliegenden Beispiel ist übrigens $4.75_{10}8$ (!); trotzdem ist eine Einschließung der Lösung auf 10 gültige Dezimalen in einfachlanger Rechnung gelungen. Die Abschätzung $\|E - R \cdot A\|$ für den Spektralradius lag bei 1.74; daher konnte eine Konvergenz der Gleitkommaiteration oder die Nicht-Singularität von A zunächst nicht garantiert werden. Es gibt krassere Beispiele, die belegen, daß die Abschätzung $\|E - R \cdot A\|$ nur sehr grob ist. So kam es bei einem 200×200 Gleichungssystem vor, daß bereits für $\|E - R \cdot A\| = 3$ die Aufblähung in Schritt 5 von Algorithmus 2.1 nicht zum Stillstand kam. Ein Beispiel für das andere Extrem sind die "PASCAL-Matrizen". Wir definieren eine n× n-Pascalmatrix $P = (p_{ij})$ durch

$$(2.54) \qquad p_{ij} := \binom{i+j}{j} = \frac{i! \cdot (i+j)!}{j!} \quad .$$

Die Kondition einer solchen Matrix ist sehr schlecht und steigt exponentiell mit dem Quadrat von n. Sie können exakt bis n = 16 auf der UNIVAC 1108 der Universität Karlsruhe in einfacher Genauigkeit gespeichert werden. Für größere n wurden einfach die Gleitkommanäherungen für die Koeffizienten eingesetzt. So ergab sich für n = 26 bei doppeltlanger Rechnung (siehe auch Tabelle 2.7)

a) eine Abschätzung der Konditionszahl von $8.23_{10}22$
b) eine Abschätzung des Spektralradius von E - RA zu 280 000
c) nach einer Iteration in Schritt 5 von Algorithmus 2.1 eine Einschließung der Lösung auf mindestens 6 Dezimalen in allen Komponenten.

An einem Beispiel soll gezeigt werden, daß bei den Operationen, die Intervalle verknüpfen, mit der nötigen Sorgfalt gearbeitet werden muß. Es wurde ein Unter-

programm DADD geschrieben zur Addition von Intervallen, wobei dei Grenzen jeweils doppeltlange Zahlen sind. In der ersten Version wurde bei der Addition von [a,b] und [c,d] das Intervall $[\underline{a+c}, \overline{b+d}]$ gebildet, wobei der Querstrich unter bzw. über der Zahl heißt, daß im letzten signifikanten Bit der Zahl 1 abgezogen bzw.hinzugezählt wurde. In diesem Fall wurde das Intervall grundsätzlich um ± 1 in jeder Richtung erweitert. Dann wurde der Algorithmus dahingehend verbessert, daß $\underline{a+c}$ nur dann gebildet wurde, wenn a + c nicht exakt darstellbar war (entsprechend für b+d); d.h., es wurde tatsächlich das engste Intervall I mit doppeltlangen Grenzen gewählt, so daß $[a+c, b+d] \subseteq I$ gilt. Es kommen ein paar Abfragen zum Algorithmus hinzu, die Gesamtrechenzeit ändert sich jedoch nicht merklich. Im Folgenden wurden als Beispiele die bekannten Matrizen $S = q \cdot R$ gewählt ($S = (s_{ij})$ mit $s_{ij} = 1$ für i,j = 1(1)n und $R = (r_{ij})$ mit r_{ij} zufällig aus [0,1], i,j = 1(1)n), wobei für q die Werte 10^{-i} für i = 1(1)5 eingesetzt wurden. Für den Grad n wurde 25 gewählt. In der folgenden Tabelle ist von links nach rechts die Anzahl der Iterationen für $\tilde{x}$, für $\tilde{y}$ und die Anzahl der "Aufblähungen" (Schritt 5 in Algorithmus 2.1), die Abschätzungen für den Spektralradius von E - R · A und die Konditionszahl von A angegeben. Diese Angaben sind in allen Beispielen für die einfache und verbesserte Version von DADD identisch. Schließlich sind der minimale und maximale relative Fehler der Lösungsintervalle für das einfache und verbesserte DADD angegeben.

	Iterat. in Schritt					DADD einf.		DADD verb.	
q	3	u. 5		$\|E - R \cdot A\|$	cond(A)	Δ_{min}	Δ_{max}	Δ_{min}	Δ_{max}
10^{-1}	2	2	1	4.4_{10}^{-5}	8.5_{10}^{3}	8.9_{10}^{-15}	2.4_{10}^{-14}	5.5_{10}^{-16}	1.6_{10}^{-15}
10^{-2}	2	2	1	5.0_{10}^{-4}	7.9_{10}^{4}	8.1_{10}^{-14}	2.0_{10}^{-13}	3.3_{10}^{-15}	4.0_{10}^{-14}
10^{-3}	2	3	1	3.2_{10}^{-3}	4.3_{10}^{5}	6.6_{10}^{-13}	1.6_{10}^{-11}	3.8_{10}^{-14}	9.1_{10}^{-13}
10^{-4}	3	3	1	4.6_{10}^{-2}	6.1_{10}^{6}	2.5_{10}^{-12}	1.7_{10}^{-10}	1.5_{10}^{-13}	1.0_{10}^{-11}
10^{-5}	5	4	2	7.4_{10}^{-1}	3.1_{10}^{7}	1.5_{10}^{-10}	3.3_{10}^{-8}	1.4_{10}^{-12}	3.4_{10}^{-10}

Tabelle 2.4 25 ×25 Testmatrizen

Wie man sieht ist ein deutlicher Gewinn zu verzeichnen, insbesondere bei schlechter konditionierten Matrizen.

Als nächstes werden wir die vorgeschlagenen Varianten von Algorithmus 2.1 anhand von Testbeispielen vergleichen. In der folgenden Tabelle werden den Varianten des Algorithmus 2.1 Namen zugeordnet.

LGL	Algorithmus 2.1
LGL/1	Verbesserung von LGL nach Seite 72, unten
LGLG	LGL mit Verwendung des langen Akkumulator nach Seite 71
LGLG/1	LGLG mit Verbesserung von Seite 72, unten
LLL	Algorithmus mit (2.34)
LLLG	LLL mit langem Akkumulator
LGLU	Algorithmus 2.2

Tabelle 2.5 Varianten des Algorithmus 2.1

In der folgenden Tabelle 2.6 sind die relativen Fehler der Ergebnisintervalle von Varianten des Algorithmus 2.1 angegeben. Die Koeffizienten der Matrix A und der rechten Seite b sind Zufallszahlen mit Exponenten aus dem Intervall [-10,10] . Das Gleichungssystem hat den Grad 50. Die Angaben in den beiden linken Spalten gelten für das Punktgleichungssystem Ax = b, in den beiden rechten Spalten für das Intervall-Gleichungssystem $A \cdot x = b$, wobei $b = b \cdot [1-10^{-8}, 1+10^{-8}]$. Wie man sieht, ist im Fall des Punktgleichungssystems LGLG/1 den anderen Algorithmen klar überlegen. Die Ergebnisintervalle sind in diesem Fall tatsächlich die kleinstmöglichen Maschinenintervalle. Im Fall des Intervallgleichungssystems schneiden die Varianten in etwa gleich ab, bis auf LLL und LLG. Obwohl zwischen

Algorithmus	PGLS max.	PGLS durchschn.	IVGLS max.	IVGLS durchschn.
LGL	3.0_{10}-14	2_{10}-15	8.9_{10}-7	4_{10}-8
LGL/1	3.0_{10}-14	1.8_{10}-15	8.9_{10}-7	4_{10}-8
LGLG	4.0_{10}-16	1_{10}-16	8.9_{10}-7	4_{10}-8
LGLG/1	4.8_{10}-18	1_{10}-18	8.9_{10}-7	4_{10}-8
LLL	1.2_{10}-11	5_{10}-14	4.8_{10}-4	1_{10}-6
LLLG	1.8_{10}-11	3_{10}-14	4.8_{10}-4	1_{10}-6

Tabelle 2.6 Relativer Fehler der Ergebnisintervalle;
A, b zufällig, Exponenten $\in [-10,+10]$, Grad 50

Relativer Fehler der Ergebnisintervalle für	LGL	LGLG	LGLG/1	LLLG
maximal	1.7_{10}-9	3.5_{10}-14	1.6_{10}-16	1.0_{10}-3
durchschn.	1.0_{10}-12	8.0_{10}-16	8.0_{10}-18	1.0_{10}-7

Tabelle 2.7 Relativer Fehler der Ergebnisintervalle;
A, $\hat{x}$ zufällig, Exponenten $\in [-10,+10]$, Grad 50

Relativer Fehler der Ergebnisintervalle für	LGL max.	LGL durchschn.	LGLG/1 max.	LGLG/1 durchschn.
Hilbert 5×5	2.4_{10}-13	1.3_{10}-13	nur Maschinen-	
Hilbert 6×6	4.0_{10}-12	2.7_{10}-12	intervalle	
Pascal 6×6	1.1_{10}-13	3.2_{10}-14		

Tabelle 2.8 Relativer Fehler der Ergebnisintervalle;
rechte Seite (1,...,1)

der dem Betrag nach größten Komponente des Lösungsvektors 9_{10} 0 und der kleinsten 5_{10}-4 nur eine Exponentendifferenz von 5 liegt, schneiden LLL und LLG sowohl bei Punkt- als auch bei Intervallgleichungssystemen deutlich schlechter ab. Die Rechenzeit für sämtliche Varianten betrug ca. 14 Sekunden auf der UNIVAC 1108 des Rechenzentrums der Universität Karlsruhe.

Als nächstes wurden die Koeffizienten von A und x zufällig mit Exponenten aus [-10,+10] erzeugt und die rechte Seite b mittels Gleitkommaarithmetik berechnet. Das Gleichungssystem hat wieder den Grad 50. Die Lösung $\hat{x}$ muß allerdings mit dem vorher zufällig erzeugten x durch die bei der Berechnung von b möglicherweise aufgetretenen Rundungsfehler nicht mehr übereinstimmen. Aus Tabelle 2.7 geht wieder die klare Überlegenheit von LGLG/1 hervor. In diesem Fall ist die Differenz zwischen größtem und kleinstem Exponenten der Lösungsvektorkomponenten gleich 13, was sich in den vergleichsweisen schlechten Resultaten von LLLG wiederspiegelt. In Tabelle 2.8 schließlich wurden als Testmatrizen die Hilbert 5×5 und 6×6 und Pascal 6×6 - Matrix betrachtet mit rechter Seite (1,...,1), resp. In diesem Fall produziert der Algorithmus LGLG/1 in allen drei Fällen nur "Maschinenintervalle", d.h. die engsten auf der UNIVAC 1108 überhaupt möglichen Intervalle in doppeltlangen Grenzen. Die Varianten LGL und LGL/1 liefern für diese (schlecht konditionierten) Beispiele die gleichen Lösungsintervalle.

Aus Tabelle 2.6 ersieht man, daß obwohl die Matrix A Punktmatrix geblieben und die rechte Seite b enge Intervalkomponenten besitzt, die Genauigkeit der produzierten Lösungsintervalle rapide absinkt. Dieser Effekt ist jedoch nicht den Algorithmen, sondern dem Gleichungssystem selbst zuzuschreiben. Wir geben hierfür ein einfaches Beispiel an. Gegeben sei das Gleichungssystem

$$(2.55) \qquad \begin{pmatrix} 100\,000 & 99\,999 \\ 99\,999 & 99\,998 \end{pmatrix} \cdot x = b$$

In der nachfolgenden Tabelle 2.9 ist die Lösung dieses Gleichungssystems für verschiedene rechte Seiten angegeben. Wie man sieht, verhält sich das Gleichungssystem "recht gutmütig", wenn die rechte Seite in beiden Komponenten nach unten oder nach oben gerundet wird. Man könnte daher die in der letzten Zeile mittels LGL berechneten Lösungsintervalle für die rechte Seite b als äußerst pessimistisch bezeichnen. Aus der vorletzten Zeile geht jedoch hervor, daß bei Rundungen der Komponenten von "b" (und der hier verwendeten 5-stelligen dezimalen Arithmetik) die Intervallgrenzen tatsächlich erreicht werden, d.h. die Lösungsintervalle sind mit dem Lösungskomplex identisch. Die äußerst starke Abhängigkeit der Lösung gegenüber kleinsten Änderungen der rechten Seite ist durch die hohe Kondition von etwa 150 000 der Matrix aus (2.55) zu erklären.

Mitte	$b = \begin{pmatrix} 200\,000 \\ 200\,000 \end{pmatrix}$	$\Rightarrow$	$\hat{x} = \begin{pmatrix} 200\,000 \\ -200\,000 \end{pmatrix}$
	$b = \begin{pmatrix} 199\,990 \\ 199\,990 \end{pmatrix}$	$\Rightarrow$	$\hat{x} = \begin{pmatrix} 199\,990 \\ -199\,990 \end{pmatrix}$
	$b = \begin{pmatrix} 200\,010 \\ 200\,010 \end{pmatrix}$	$\Rightarrow$	$\hat{x} = \begin{pmatrix} 200\,010 \\ -200\,010 \end{pmatrix}$
	$b = \begin{pmatrix} 199\,990 \\ 200\,010 \end{pmatrix}$	$\Rightarrow$	$\hat{x} = \begin{pmatrix} 2\,199\,970 \\ -2\,199\,990 \end{pmatrix}$
	$b = \begin{pmatrix} 200\,010 \\ 199\,990 \end{pmatrix}$	$\Rightarrow$	$\hat{x} = \begin{pmatrix} -1\,799\,970 \\ 1\,799\,990 \end{pmatrix}$
	$\boldsymbol{b} = \begin{pmatrix} [199\,990\ ,\ 200\,010] \\ [199\,990\ ,\ 200\,010] \end{pmatrix}$	$\Rightarrow$	$\boldsymbol{Y} = \begin{pmatrix} [-1\,800\,000\ ,\ +2\,200\,000] \\ [-2\,200\,000\ ,\ +1\,800\,000] \end{pmatrix}$

Tabelle 2.9 Lösung von (2.55) für verschiedene rechte Seiten

In der nächsten Tabelle 2.10 wird das Verhalten von Algorithmus 2.2 für Intervallgleichungssysteme wiedergegeben. Die Koeffizienten der Matrix A sind Zufallszahlen aus dem Intervall $[9,11]$. Die rechte Seite b ist so berechnet (mittels Gleitkommarechnung), daß die Lösung ungefähr gleich $(1,\ldots,1)$ ist. Die Matrix A und der Vektor b entstehen aus A und b durch Multiplikation aller Komponenten mit dem Intervall $[1 - 10^{-8}, 1 + 10^{-8}]$. Das Gleichungssystem $Ax = b$ wurde in beiden Fällen mittels LGL, die restlichen Gleichungssysteme mittels LGLU gelöst.

		maximal	durchschnittlich
Grad 50	$Ax = b$	7.8_{10}-16	2_{10}-16
	$Ax = b$	6.7_{10}-6	2_{10}-6
	$Ax = b$	4.4_{10}-5	1_{10}-5
	$Ax = b$	5.1_{10}-5	2_{10}-5
Grad 100	$Ax = b$	3.2_{10}-15	8_{10}-16
	$Ax = b$	3.9_{10}-5	8_{10}-6
	$Ax = b$	1.1_{10}-4	3_{10}-5
	$Ax = b$	1.2_{10}-4	3.5_{10}-5

Tabelle 2.10 Relativer Fehler der Lösungsintervalle bei Intervall-Gleichungssystemen

Jetzt geben wir in Tabelle 2.11 einige Beispiele für schlecht konditionierte Matrizen an. Zunächst die bekannten Hilbert-Matrizen, wobei wieder durch einen geeigneten Faktor die Koeffizienten ganzzahlig gemacht wurden. Dann die vorhin in (2.54) angegebenen Pascalmatrizen; weiterhin Pascal*-Matrizen P^* mit

$$P^* = (p^*_{ij}) \text{ und } p^*_{ij} = \binom{i+j-1}{j} \text{ für } i,j = 1(1)n .$$

Wie man sich leicht überzeugt, entstehen die Pascal*-Matrizen aus den Pascal-Matrizen durch Ränderung mit 1 links und oben. Sodann werden noch die Matrizen

$$S = (s_{ij}) \text{ mit } s_{ij} = \frac{\binom{n+j-1}{i-1} \cdot n \cdot \binom{n-1}{n-j}}{i+j-1} \text{ für } i,j = 1(1)n$$

getestet, die Zielke entnommen wurden. Alle rechten Seiten wurden so bestimmt, daß die exakte Lösung (1,...,1) war.

Matrix	Grad	Iterat. in Schritt 3	Iterat. in Schritt 5	$\|E - R \cdot A\|$	cond(A)	LGL Δ_{min}	LGL Δ_{max}	LGLG/1 mind. garantierte Dezimalst. jeder Lösungskomponente
Hilbert	5	2	1	$1.5_{10}-3$	$9.4_{10}5$	$1.1_{10}-13$	$2.4_{10}-13$	18
	6	4	1	$8.2_{10}-2$	$2.9_{10}7$	$1.9_{10}-12$	$4.0_{10}-12$	18
	7	17	2	$1.7_{10}0$	$1.3_{10}9$	$1.1_{10}-10$	$2.4_{10}-10$	17
Pascal	6	3	1	$3.8_{10}-3$	$1.8_{10}6$	$6.0_{10}-15$	$1.1_{10}-13$	18
	7	3	1	$1.4_{10}-1$	$2.5_{10}7$	$4.1_{10}-14$	$1.4_{10}-12$	18
	8	7	2	$1.2_{10}0$	$3.7_{10}8$	$3.6_{10}-13$	$2.8_{10}-11$	17
	9	3	1	$3.8_{10}1$	$1.3_{10}10$	stop in Schritt 5		
Pascal*	6	2	1	$3.0_{10}-4$	$2.1_{10}-$	$4.5_{10}-16$	$4.4_{10}-15$	18
	7	3	1	$2.3_{10}-3$	$2.9_{10}6$	$1.8_{10}-15$	$3.6_{10}-14$	18
	8	4	1	$1.1_{10}-1$	$4.0_{10}7$	$7.2_{10}-15$	$2.6_{10}-13$	18
	9	7	3	$3.5_{10}0$	$6.0_{10}8$	$5.1_{10}-14$	$4.3_{10}-12$	18
S	5	2	2	$3.1_{10}-3$	$2.2_{10}6$	$3.5_{10}-18$	$2.2_{10}-16$	18
	6	4	1	$2.9_{10}-2$	$1.1_{10}8$	$8.7_{10}-14$	$1.8_{10}-11$	18
	7	8	2	$1.6_{10}0$	$6.0_{10}9$	$3.5_{10}-18$	$3.5_{10}-18$	15

Tabelle 2.11 LGL und LGLG/1 für schlecht konditionierte Matrizen

Wie man sieht, kann die Anzahl der Gleitkomma-Iterationen differieren; es sind natürlich um so mehr, je schlechter die Kondition der Matrix ist. Die Anzahl der Aufblähungen in Schritt 5 ist meist gleich 1, nur in äußerst schlecht konditionierten Fällen kann sie 2 oder 3 werden. Alle Beispiele wurden einfachlang mit Version LGL und LGLG/1 (siehe Tabelle 2.5) gerechnet. Die Angaben in den ersten 7 Spalten sind für beide Versionen identisch. Die letzten drei Spalten geben den minimalen und maximalen relativen Fehler der Lösungsintervalle für LGL an und in der letzten Spalte ist die Anzahl der in allen Lösungsintervallen minimal garantierten Dezimalstellen gezeigt. Offenkundig ist die klare Oberlegenheit von LGLG/1. Wie man aus den Beispielen ersieht und sich auch in allen anderen Tests bestätigte, können Matrizen bis zur Konditionszahl 10^8 bei 8-stelliger Rechnung bearbeitet werden. Weiter sind im allgemeinen höchstens 6 Gleitkommaiterationen notwendig und erfahrungsgemäß bleibt die Aufblähung in Schritt 5 meist nach einem Schritt stehen. Anhand der folgenden Beispiele soll demonstriert werden, was bei noch schlechter konditionierten Gleichungssystemen und Übergang zu höherer Genauigkeit geschieht.

Recht naheliegend sind die Hilbert-Matrizen, die jedoch nur bis Grad 11 (nach wegmultiplizieren der Nenner) exakt auf der UNIVAC 1108 darstellbar sind. Also wurde von vorherein $H = (h_{ij})$ mit $h_{ij} = (i + j - 1)^{-1}$ für $i,j = 1(1)n$ gleitkommamäßig berechnet. Die rechte Seite wurde gleich (1,...,1) gesetzt. Weiter wurden die Pascal-Matrizen betrachtet nach (2.54). Diesmal wurde die rechte Seite so berechnet, daß die exakte Lösung "hätte ungefähr (1,...,1) sein müssen", nämlich $b_i = \sum_{j=1}^{n} a_{ij}$, $i = 1(1)n$ gleitkommamäßig . Durch die starken Rundungsfehler und Auslöschungen waren die tatsächlichen Lösungen jedoch teilweise sehr stark von 1 verschieden, und zwar bis zur Größenordnung 10^8 bei Pascal 26 × 26 . Wie man sieht, wird Schritt 5 in allen Beispielen trotz schlechtester Konditionszahlen nur einmal ausgeführt. Obwohl die Abschätzungen für die Konditionszahl bis $8.2_{10}22$ und für den Spektralradius E - R · A bis 280 000 reichen, sind in allen Beispielen bei allen Lösungsintervallen mindestens 5 Stellen garantiert, durchschnittlich jedoch noch weit mehr. Die teilweise auftretenden Unregelmäßigkeiten

Matrix	Grad	Iterationen in Schritt 3	5	$\|E - R \cdot A\|$	cond(A)	Δ_{min}	Δ_{max}
Hilbert	5	1	2	1	8.0_{10}-13	$9.4_{10}5$	1.1_{10}-13	2.3_{10}-13
	6	1	2	1	1.8_{10}-11	$2.9_{10}7$	2.8_{10}-12	6.0_{10}-12
	7	1	2	1	5.2_{10}-10	$9.9_{10}8$	7.7_{10}-11	1.7_{10}-10
	8	1	2	1	1.2_{10}-8	$2.2_{10}10$	1.1_{10}-9	5.5_{10}-9
	9	1	2	1	1.1_{10}-8	$1.7_{10}10$	9.6_{10}-10	6.7_{10}-9
	10	1	2	1	1.6_{10}-8	$1.6_{10}10$	5.3_{10}-10	7.4_{10}-8
	11	1	2	1	8.1_{10}-8	$9.1_{10}10$	3.9_{10}-10	2.7_{10}-8
	12	1	2	1	1.5_{10}-8	$2.0_{10}10$	7.9_{10}-10	1.6_{10}-8
	13	3	6	1	1.5_{10}-7	$1.5_{10}11$	7.0_{10}-9	2.6_{10}-8
	14	1	2	1	2.4_{10}-8	$2.4_{10}10$	3.7_{10}-10	3.7_{10}-8
	15	1	2	1	4.0_{10}-8	$3.8_{10}10$	8.5_{10}-10	4.8_{10}-8
	16	1	2	1	5.5_{10}-7	$5.9_{10}11$	1.7_{10}-8	3.1_{10}-8
	17	1	2	1	1.3_{10}-7	$4.4_{10}10$	8.4_{10}-10	5.0_{10}-8
	18	8	2	1	8.0_{10}-8	$1.0_{10}11$	1.5_{10}-9	5.4_{10}-8
	19	4	2	1	2.2_{10}-7	$6.2_{10}10$	7.6_{10}-10	7.5_{10}-8
	20	8	8	1	5.8_{10}-7	$5.9_{10}11$	7.7_{10}-9	8.7_{10}-8
	21	1	12	1	3.5_{10}-7	$2.1_{10}11$	3.8_{10}-9	2.7_{10}-7
	22	4	3	1	9.8_{10}-7	$2.6_{10}11$	3.1_{10}-9	4.5_{10}-7
	23	16	1	1	5.4_{10}-7	$2.3_{10}11$	3.9_{10}-9	3.5_{10}-7
	24	6	4	1	9.3_{10}-7	$2.4_{10}11$	5.5_{10}-9	1.5_{10}-7
	25	1	2	1	6.0_{10}-7	$1.9_{10}11$	2.4_{10}-9	7.7_{10}-8

Tabelle 2.12 Hilbert-Matrizen doppeltlang gerechnet

Matrix	Grad	Iterationen in Schritt 3	5	$\|E - R \cdot A\|$	cond(A)	Δ_{min}	Δ_{max}
Pascal	5	1	1	1	8.6_{10}-14	$1.1_{10}5$	2.9_{10}-15	2.6_{10}-14
	6	1	1	1	1.3_{10}-12	$1.8_{10}6$	1.6_{10}-14	3.3_{10}-13
	7	1	1	1	7.5_{10}-12	$2.5_{10}7$	2.8_{10}-14	9.2_{10}-13
	8	1	1	1	6.7_{10}-10	$4.0_{10}8$	4.6_{10}-13	3.3_{10}-11
	9	1	1	1	1.8_{10}-9	$5.8_{10}9$	4.0_{10}-12	5.5_{10}-10
	10	1	1	1	2.3_{10}-8	$9.0_{10}10$	3.1_{10}-12	8.1_{10}-10
	11	2	1	1	3.0_{10}-7	$1.3_{10}12$	1.1_{10}-11	4.8_{10}-9
	12	2	14	1	6.1_{10}-6	$2.0_{10}13$	1.6_{10}-10	1.4_{10}-7
	13	2	12	1	1.4_{10}-4	$3.0_{10}14$	7.2_{10}-10	1.2_{10}-6
	14	3	11	1	6.7_{10}-3	$4.7_{10}15$	3.3_{10}-9	1.1_{10}-5
	15	2	5	1	2.0_{10}-2	$2.0_{10}16$	1.8_{10}-6	1.2_{10}-5
	16	15	2	1	3.8_{10}-2	$4.0_{10}16$	2.2_{10}-7	1.6_{10}-6
	17	13	13	1	1.2_{10}-1	$7.6_{10}16$	1.7_{10}-8	2.9_{10}-6
	18	7	14	1	2.0_{10}-1	$2.8_{10}17$	5.0_{10}-8	4.7_{10}-7
	19	13	4	1	5.5_{10}-1	$1.2_{10}18$	2.2_{10}-8	1.7_{10}-5
	20	13	9	1	1.1_{10}+1	$6.6_{10}18$	1.5_{10}-7	2.2_{10}-6
	21	12	1	1	$5.1_{10}0$	$5.0_{10}18$	2.3_{10}-7	6.6_{10}-5
	22	14	3	1	2.1_{10}+2	$1.4_{10}20$	9.7_{10}-7	1.0_{10}-6
	23	14	3	1	3.0_{10}+2	$1.4_{10}20$	5.0_{10}-8	1.3_{10}-6
	24	13	13	1	6.7_{10}+2	$6.1_{10}20$	6.1_{10}-9	1.1_{10}-6
	25	14	1	1	2.1_{10}+3	$2.3_{10}21$	1.1_{10}-7	1.7_{10}-6
	26	14	14	1	2.8_{10}+5	$8.2_{10}22$	1.8_{10}-6	1.8_{10}-5

Tabelle 2.13 Pascal-Matrizen doppeltlang gerechnet

sind durch die bei der Berechnung der Koeffizienten auftretenden Rundungsfehler zu erklären. Die Anzahl der Gleitkomma-Iterationen für $\tilde{x}$ und $\tilde{y}$ kann bisweilen etwas größer werden. Das hängt natürlich von der Kondition der Matrix ab. Es kann auch passieren, daß nur eine Komponente des Lösungsvektors empfindlich gegen kleinste Störungen reagiert und daher nur für diese eine Komponente weiter iteriert werden muß.

Die Beispiele aus den Tabellen 2.12 und 2.13 wurden mit einer doppeltlangen Version des Algorithmus 2.1 gerechnet (auf der UNIVAC 1108 der Universität Karlsruhe als LGL2 in der Programmbibliothek verfügbar). Die vergleichsweise geringere Genauigkeit der Lösungsintervalle ist nicht auf den höheren Grad und weniger auf die schlechte Kondition der Matrizen zurückzuführen als auf die Tatsache, daß der lange Akkumulator nicht mehr eingesetzt wurde. In der Tat wurden mit Hilfe des langen Akkumulators insbesondere Skalarprodukte auf Maschinengenauigkeit berechnet. Das Produkt von zwei doppeltlangen Zahlen ist jedoch (ohne gesonderte Programme) auf der UNIVAC 1108 nicht mehr exakt (auf alle Stellen) ausführbar. Daher blieb keine andere Wahl, als das Ergebnis nach oben und unten zu runden um ein Intervall zu erhalten. Das geschieht bei jedem Zwischenschritt für jedes einzelne Produkt (für die Aufsummierung wurde ein spezielles DADD geschrieben, siehe Seite *80*). Durch diesen Prozess ergeben sich erhebliche Ungenauigkeiten, die bei Verwendung des langen Akkumulators nicht auftreten. So ist es gar erklärbar, daß (bei nicht zu schlecht konditionierten Problemen) die einfachlange Version LGLG/1 mit langem Akkumulator sogar genauere Ergebnisintervalle liefert als die doppeltlange Version LGL2.

Wie das Beispiel aus (2.55) bereits deutlich machte, sind die relativ schlechten Lösungsintervalle nicht auf den Algorithmus, sondern auf die Kondition der Matrix zurückzuführen. Hat die Matrix noch schlechtere Kondition, kann es vorkommen, daß die Lösungsintervalle für einige Komponenten Null-Intervalle sind und damit

keine Stelle mehr liefern (vergleiche Tabelle 2.9). Die Rechenzeit für Algorithmus LGLU für ein Gleichungssystem vom Grad 50 beträgt ca. 17 Sekunden auf der UNIVAC 1108.

Zuletzt wollen wir noch zeigen, daß der Algorithmus auch für Gleichungssysteme von höherem Grad gute Lösungsintervalle liefert. Es wurden die Beispielmatrizen $S - q \cdot R$ betrachtet für verschiedenes q und verschiedenen Grad des Gleichungssystems. Die rechte Seite b wurde so berechnet, daß die Lösung $(1,\ldots,1)$ ist. Das letzte Beispiel Exp±4 sind wieder Gleichungssysteme, bei denen die Matrix A und die rechte Seite b zufällig erzeugt wurden mit Exponenten im Intervall $[-4\ ,\ +4]$. Die Ergebnisse sind in der folgenden Tabelle 2.14 zusammengefaßt. Die Abschätzung für $\mathrm{cond}(A)$ ist durch $\mathrm{cond}(A) \leq \|A\| \cdot \|R\|$ und der Zeilensummennorm gegeben. Es bedeutet weiter $\sigma = \|E - R \cdot A\|$ (wieder nach der Zeilensummennorm), r die Anzahl der Gleitkommaiterationen aus Schritt 3), und s die Anzahl der "Aufblähungen" aus Schritt 5) von Algorithmus 2.1. In der letzten Spalte schließlich ist die Anzahl der mindestens garantierten Dezimalstellen in jeder Lösungskomponente für die verschiedenen Gleichungssysteme angegeben. Aus Tabelle 2.14 erhellt, daß selbst für "schlechtes" σ (teilweise sogar > 1) sehr enge Lösungsintervalle geliefert werden. Die Anzahl der Intervalliterationen ist gering, höchstens jedoch vier. Die Begrenzung der Ordnung des Gleichungssystems auf 200 ist durch den beschränkten Speicherplatz der UNIVAC 1108 verursacht und stellt keine Schranke für den Algorithmus dar.

Abschließend können wir feststellen, daß weitgehend unabhängig von Grad und Kondition der Matrix in jeder Lösungskomponente mit mindestens 15 garantierten Dezimalstellen zu rechnen ist.

Testmatrix		cond(A)	σ	r	s	mindestens garantierte Dezimalstellen in jeder Lösungskomponente
Grad 50	$q = 10^{-1}$	$9.2_{10}3$	$2.4_{10}-4$	4	1	18
	$q = 10^{-2}$	$9.0_{10}4$	$1.3_{10}-3$	4	1	18
	$q = 10^{-3}$	$9.0_{10}5$	$2.4_{10}-2$	6	1	18
	$q = 10^{-4}$	$9.0_{10}6$	0.15	7	1	18
	$q = 10^{-5}$	$9.0_{10}7$	1.2	10	2	17
Grad 100	$q = 1$	$2.8_{10}3$	$3.1_{10}-5$	4	1	18
	$q = 10^{-1}$	$2.6_{10}4$	$3.3_{10}-4$	4	1	18
	$q = 10^{-2}$	$2.6_{10}5$	$3.4_{10}-3$	4	1	18
	$q = 10^{-3}$	$2.6_{10}6$	$2.4_{10}-2$	6	1	17
	$q = 10^{-4}$	$2.6_{10}7$	0.56	9	2	18
	$q = 10^{-5}$	$2.6_{10}8$	2.3	16	4	17
Grad 150	$q = 10^{-2}$	$2.8_{10}6$	$6.5_{10}-3$	4	1	18
	$q = 10^{-3}$	$2.8_{10}7$	$1.3_{10}-2$	5	1	18
Grad 200	$q = 10^{-2}$	$8.0_{10}6$	$1.3_{10}-2$	3	1	18
	$q = 5_{10}-3$	$1.1_{10}7$	0.19	4	2	18
	$q = 10^{-3}$	$5.7_{10}7$	0.49	4	2	17
Expt4 Grad 100		$1.9_{10}3$	$3.7_{10}-5$	4	1	18
Grad 150		$2.8_{10}3$	$5.1_{10}-5$	4	1	18
Grad 200		$3.2_{10}3$	$1.0_{10}-4$	4	1	18

Tabelle 2.14 LGLG/1 für lineare Gleichungssysteme höheren Grades

Eine schnelle Methode zur Lösung beliebiger linearer Gleichungssysteme ist der Gaußsche Algorithmus. Die Ergebnisse, die dieser bei Gleitkommarechnung ohne jede Vorkehrung liefert, können mit großen Fehlern behaftet sein (siehe Hilbert 7×7-Matrix). Selbst wenn die Defektiteration zu konvergieren "scheint", ist damit noch nicht bewiesen, daß das erzielte Ergebnis "in der Nähe" der wahren Lösung liegt. Um garantierte Schranken zu erhalten wird ein spezieller Algorithmus benötigt (mit einer genau spezifizierten Arithmetik und Rundung). Wir haben einen solchen Algorithmus angegeben, der die folgenden Vorteile aufweist:

- Jeder Gleitkommaalgorithmus zur Gewinnung einer Approximation der Lösung ist brauchbar
- Die Matrix und die rechte Seite unterliegen keiner Einschränkung
- Der Algorithmus arbeitet auch für Intervall-Gleichungssysteme
- Es wird keine Ersteinschließung der Lösung oder der inversen Matrix benötigt
- Intervallarithmetik setzt sehr spät ein
- Der Algorithmus ist sehr schnell

Nach der erfolgreichen Abarbeitung des Algorithmus ist bewiesen:

- A ist nicht singulär
- $Ax = b$ ist eindeutig lösbar
- Es werden garantierte Lösungen von hoher Genauigkeit angegeben

Die Rechenzeit auf der UNIVAC 1108 der Universität Karlsruhe beträgt durchschnittlich 13 Sekunden für Grad 50 und ca. 88 Sekunden für Grad 100; die für den Gaußschen Algorithmus benötigte Rechenzeit war jeweils 1/6 , unabhängig

vom Grad. Überdies lassen sich die Algorithmen 2.1 und 2.2 leicht auf den Fall komplexer linearer Gleichungssysteme übertragen.

Die Algorithmen sind in ALGOL und FORTRAN programmiert. Der FORTRAN-Quelltext von Algorithmus 2.1, und zwar der besten Version LGLG/1 mit Verwendung des langen Akkumulators, und der notwendigen Assembler-Routinen finden sich im Anhang.

3. Weitere Einschließungssätze und -Algorithmen und numerische Beweise

In diesem Kapitel sollen u.a. als Folgerung der Sätze aus Kapitel 2 verschiedene Einschließungssätze und numerische Beweistechniken gebracht werden. Hier ist nur eine Auswahl einer Fülle weiterer Folgerungen angegeben. Die Tatsache an sich ist interessant, daß mittels numerischer Methoden (und einer genau definierten Rechnerarithmetik und -Rundung) mathematische Beweise geführt werden können, und zwar mit relativ geringem Mehraufwand gegenüber den bisherigen Gleitkommaverfahren. Daraus ergibt sich wieder die dringende Notwendigkeit, endlich entsprechende (siehe Kulisch 1) Arithmetiken fest in Großrechnern zu verdrahten. So können die angegebenen Algorithmen direkt implementiert werden ohne den bisherigen Mehraufwand an Assemblerprogrammen. Außerdem hat man es endlich mit klar definierten und weitgehend geklärten (siehe Kulisch 1) Strukturen zu tun: Es wird möglich, auf der Rechenanlage Mathematik zu treiben.

Hier wie im letzten Kapitel sind die angegebenen Beweistechniken (z.B. zum Nachweis der Nicht-Singularität einer Matrix) als hinreichendes Kriterium zu betrachten, dessen Gültigkeit algorithmisch auf dem Rechner nachgeprüft wird. In diesem Sinn haben wir es nicht mit einem Algorithmus im herkömmlichen Sinn zu tun und dementsprechend erscheint die Angabe eines Abbruchkriteriums als wenig sinnvoll.

3.a) Numerischer Beweis der Nicht-Singularität einer Matrix

Aus den Sätzen 2.2 bis 2.5 geht hervor, daß nach erfolgreicher Durchführung von Algorithmus 2.1 bzw. 2.2 für Punkt- bzw. Intervallmatrizen mit beliebiger rechter Seite b die Nicht-Singularität der Eingabematrix bzw. jedes Elements der Intervalleingabematrix bewiesen ist. Will man nur die Regularität (zunächst einer Punktmatrix A) beweisen, liegt es nahe, als rechte Seite b = o zu wählen.

Eine erste Näherung x der Lösung lautet selbstverständlich x = o. Auf eine Iteration mit mehrfachem Defekt wie in Algorithmus 2.1 kann verzichtet werden, da nicht die "Genauigkeit der Lösung" von Ax = o sondern nur die Regularität von A interessiert. Demnach kann in Anlehnung an Algorithmus 2.1 sofort der folgende Algorithmus formuliert werden:

Algorithmus 3.1 Numerischer Beweis der Nicht-Singularität der Punktmatrix $A \in M_n\mathbb{R}$

1) Berechne $R \approx A^{-1}$ gleitkommamäßig
2) Berechne $B := E \ominus R \odot A$ intervallmäßig
3) for $i := 1$ to n do $Y_i^0 := [-1, +1]$; $k := -1$;
4) repeat $k := k+1$; $Y^{k+1} := Y^k := Y^k \circ c$;
   for $i := 1$ to n do $Y_i^{k+1} := B_i \odot Y^k$
   until $Y^{k+1} \subsetneq Y^k$ oder $(k > \lceil \log_{10} n \rceil + 8) =: bool$;
   if bool then stop;
5) Es ist numerisch bewiesen, daß A nicht singulär ist

Die Richtigkeit des Algorithmus folgt sofort aus den Sätzen 2.2 bis 2.4. Für c gelten hier wie für die folgenden Algorithmen die in Kapitel 2 gemachten Aussagen und Bemerkungen entsprechend. Das Abbruchkriterium in Schritt 4 ist einerseits so hoch angesetzt, daß erfahrungsgemäß wahrscheinlich keine Einschließung mehr für größeres k eintritt, andererseits aber im divergenten Fall auch nicht zu viel Rechenzeit verbraucht wird. Für die Rechenzeit α gilt offenbar

$$\alpha = 2n^3 + 4s \cdot n^2 , \tag{3.1}$$

wenn Schritt 4 s-mal durchgeführt wird. Ist $\alpha(n)$ die maximale Rechenzeit für Grad n, so gilt

$$\alpha(100) \leq 2.4n^3 \quad \text{und} \quad \alpha(200) \leq 2.2n^3 .$$

Asymptotisch ist die maximale und mittlere Rechenzeit von Algorithmus 3.1 $\sim 2n^3$.

Stoppt Algorithmus 3.1 im ersten Aufblähungsiterationsschritt, also ist $Y^1 \subsetneq Y^0$, so heißt das nichts anderes als

$$\|E \ominus R \odot A\|_1 < 1 , \tag{3.2}$$

wobei $\|\cdot\|_1$ die Zeilensummennorm für Intervallmatrizen bedeutet :

$$A = (a_{ij}) \in IM_n\mathbb{R} \Rightarrow \|A\|_1 := \max_i \{ \sum_{j=1}^{n} |a_{ij}| \} .$$

Hieraus folgt selbstverständlich, daß sowohl A als auch R nicht singulär sind. In gewisser Weise stellt also der vorgestellte Algorithmus eine Verallgemeinerung des Kriteriums (3.2) dar. Für Matrizen, die bisher rechnerisch mittels (3.2) nicht als regulär entscheidbar waren, die gleichsam "fast-singulär" sind, liefert der Algorithmus oft eine Entscheidung. Dies sind bei einfacher Genauigkeit etwa Hilbert 7, Pascal 8 und 9, Pascal* 9 oder S 7 aus Tabelle 2.11. Aus Tabelle 2.14 entnimmt man auch Matrizen höheren Grades, für die eine Entscheidung der Regularität nunmehr möglich wird. Bei doppeltlanger Rechnung bilden die Pascal 20 bis 26 - Matrizen aus Tabelle 2.13ein krasses Beispiel.
Der Algorithmus läßt sich leicht auf Intervallmatrizen übertragen. Für m(A) wie in (2.41) ist

Algorithmus 3.2 Numerischer Beweis der Nicht-Singularität der Intervallmatrix $A \in IM_n\mathbb{R}$

1) Berechne $R \approx m(A)^{-1}$ gleitkommamäßig
2) Berechne $B := E \ominus R \odot A$ intervallmäßig
3) for $i := 1$ to n do $Y_i^0 := [-1, +1]$; $k := -1$;

4) repeat k := k+1; $Y^{k+1} := Y^k := Y^k \bullet \varepsilon$;
for i := 1 to n do $Y_i^{k+1} := B_i \odot Y^k$
until $Y^{k+1} \subsetneq Y^k$ oder ((k > $\lceil \log_{10} n \rceil$ + 8) =: bool) ;
if bool then stop ;

5) Es ist numerisch bewiesen, daß jedes $A \in \mathbf{A}$ nicht singulär ist

Die Richtigkeit des Algorithmus folgt wieder sofort aus Satz 2.5. Für das Abbruchkriterium in Schritt 4 gelten die gleichen Aussagen und Bemerkungen wie für Algorithmus 3.1. Für die Rechenzeit α gilt

(3.3) $$\alpha = 3n^3 + 4s \cdot n^2 ,$$

wenn Schritt 4 s-mal ausgeführt wird. Ist $\alpha(n)$ die maximale Rechenzeit für Grad n, so gilt

$$\alpha(100) \lesssim 3.4n^3 \quad \text{und} \quad \alpha(200) \lesssim 3.2n^3 .$$

Asymptotisch ist die maximale und mittlere Rechenzeit von Algorithmus 3.2 $\sim 3n^3$. Entsprechend ist Algorithmus 3.2 eine Verallgemeinerung des Kriteriums

$$\| E \ominus R \circledast A \|_1 < 1 .$$

Die Wirkungsweise der Algorithmen soll kurz an einem Beispiel verdeutlicht werden. Es sei

$$A = \begin{pmatrix} 1 & 2 \\ 2 & 3 \end{pmatrix} \Rightarrow A^{-1} = \begin{pmatrix} -3 & 2 \\ 2 & -1 \end{pmatrix} \text{ und sei } R = \begin{pmatrix} -2.8 & 2.2 \\ 1.8 & -0.8 \end{pmatrix}$$

eine Näherung für A^{-1}. Es ergibt sich

$$E - R \cdot A = \begin{pmatrix} -0.6 & -1 \\ -0.2 & -0.2 \end{pmatrix} ,$$

also $\| E - R \cdot A \|_1 = 1.6$: Der Nachweis der Nicht-Singularität kann auf diesem Wege nicht erbracht werden. Nach Algorithmus 2.1 ergibt sich

$$Y^0 = \begin{pmatrix} [-1,1] \\ [-1,1] \end{pmatrix} ; \quad Y^1 = \begin{pmatrix} [-1.6,1.6] \\ [-0.4,0.4] \end{pmatrix} ; \quad Y^2 = \begin{pmatrix} [-1.36,1.36] \\ [-0.4,0.4] \end{pmatrix} ; \quad Y^3 = \begin{pmatrix} [-1.216,1.216] \\ [-0.352,0.352] \end{pmatrix}$$

ohne ε-Aufblähung in Schritt 4. Aus $Y^3 \subsetneq Y^2$ folgt, daß A und R nicht singulär sind (man beachte, daß $Y^2 \subsetneq Y^1$ nicht gilt).

Man überlegt sich schnell, daß die Intervalle Y^i symmetrisch zum Ursprung liegen müssen. Daher kann der Algorithmus noch insofern vereinfacht werden, daß statt $B_i \odot Y^k$ in Schritt 4 nur

(3.4) $$s_i := \bigoplus_{j=1}^{n} |b_{ij}| \cdot |w_j^k|$$

gebildet wird und $s_i < w_i^k$ abgefragt wird. Die Rechenzeiten für die Algorithmen 3.1 und 3.2 verringern sich somit jeweils geringfügig auf

$$\alpha = 2n^3 + 2s \cdot n^2 \quad \text{und} \quad \alpha = 3n^3 + 2s \cdot n^2 .$$

3.b) Zur Lage der Eigenwerte einer Matrix

Im Folgenden soll zunächst Satz 2.4 dahingehend verallgemeinert werden, daß basierend auf den vorangegangenen Algorithmen allgemeine Aussagen über den Spektralradius und die Lage der Eigenwerte einer Matrix gemacht werden können. Wie in Kapitel 2 bezeichne $\circledast$ eine Operation, die den Komplex von $a \bullet b$ und $\circledast$ eine Operation, die eine Einschließung des Komplexes berechnet für $\bullet \in \{+,-,\cdot,/\}$. Sind a,b Intervallmatrizen, Intervallvektoren etc. gilt die Definition entsprechend.

Satz 3.1 Gilt für die Punktmatrix $C \in M_n\mathbb{R}$, den Punktvektor $z \in V_n\mathbb{R}$ und den Intervallvektor $\Omega \in IV_n\mathbb{R}$

$$z \boxplus C \boxdot \Omega \subsetneq \Omega \,, \tag{3.4}$$

so folgt $\rho(C) < 1$.

Beweis. Offenbar erfüllt $f(x) = z + C \cdot x : \Omega \to \Omega$ die Voraussetzungen von Satz 2.1. Es existiert also ein Fixpunkt $\hat{x}$ von f mit $z + C\hat{x} = \hat{x}$. Nach (3.4) ist dann

$$C \boxdot (\Omega \boxminus \hat{x}) \subseteq C \boxdot \Omega \boxminus (C \cdot \hat{x}) = C \boxdot \Omega \boxplus z \boxminus \hat{x} \subsetneq \Omega \boxminus \hat{x} \,.$$

Für $Y := \Omega \boxminus \hat{x}$ ist also

$$C \boxdot Y \subsetneq Y \tag{3.6}$$

und außer $o \in Y$ gilt sogar $U_r(o) \in Y$ für ein $r > o$. Für $C = o$ ist $\rho(C) = o < 1$. Für $C \neq o$ sei v ein beliebiger Eigenvektor und $\lambda \in \mathbb{C}$ der dazugehörige Eigenwert. Wir definieren $\Gamma := \{\gamma \in \mathbb{C} \mid \gamma \cdot v \in Y\}$ und $|Y^*| := \max_{\gamma \in \Gamma} |\gamma|$ $\left(\exists \Gamma\right)$. Damit ist

$$C \cdot (Y^* v) = \lambda Y^* \cdot v \in Y \setminus \partial Y$$

nach (3.6). Da $Y^* v$ auf dem Rand von Y liegt folgt also

$$|Y^*| < |\lambda Y^*| \quad \text{und daher} \quad |\lambda| < 1 \,.$$

Diese Schlußkette gilt für jeden Eigenwert λ von C und daraus ergibt sich die Behauptung.

Möchte man von einer Matrix $B \in M_n\mathbb{R}$ beweisen, daß $\rho(B) < 1$ ist, kann man direkt einen Algorithmus entsprechend den Schritten 4, 5 und 6 von Algorithmus 3.1 verwenden. Soll z.B. nachgewiesen werden, daß eine Matrix B nur Eigenwerte mit positivem Realteil hat, ist wie folgt vorzugehen. Für eine beliebige Matrixnorm gilt $\rho(B) \leq \|B\|$, also ist $\rho(C) \leq 1$ für $C := B/\|B\|$. Ist für ein $\Omega \in IV_n\mathbb{R}$ einmal $(E \boxminus C) \boxdot \Omega \subsetneq \Omega$ erfüllt, folgt nach Satz 3.1 $\rho(E \boxminus C) < 1$ für $z = o$. Für einen Eigenwert λ von C muß demnach $|\lambda| \leq 1$ und $|1 - \lambda| < 1$ gelten, also insbesondere $\mathrm{Re}\,\lambda > o$. Genauer muß λ im schraffierten Gebiet von Abbildung 3.1 liegen,

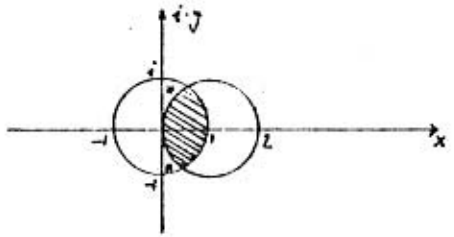


Abb. 3.1 Lage der Eigenwerte

wie man sich leicht überlegt. Wir können daher folgenden Algorithmus formulieren:

Algorithmus 3.3 Numerischer Beweis, daß die Eigenwerte einer Matrix B positiven Realteil haben

1) for i := 1 to n do $Y_i^o := [-1,+1]$; k := -1; $D := E \boxminus \|B\|^{-1} \boxdot B$;

2) repeat k := k+1; $Y^{k+1} := Y^k := Y^k \circ \varepsilon$;
   for i := 1 to n do $Y_i^{k+1} := D_i \boxdot Y^k$;
   until $Y^{k+1} \subsetneq Y^k$ oder $(k > \lceil \log_{10} n \rceil + 8) =: \mathrm{bool})$;
   if bool then stop ;

3) Für jeden Eigenwert λ von B ist numerisch bewiesen, daß $\mathrm{Re}\,\lambda > o$

Der Algorithmus läßt sich analog für Intervallmatrizen formulieren. Für das Abbruchkriterium in Schritt 2 gelten die Aussagen aus dem letzten Abschnitt entsprechend. In der Praxis wird man statt $\|B\|$ eine Zahl $\beta > \|B\|$ wählen, und zwar etwa eine Potenz der Basis der verwendeten Arithmetik, um die Division in Schritt 1 ohne Intervallbildung vollziehen zu können. Als Beispiel betrachten wir

$$B := \begin{pmatrix} 0.9 & -0.05 \\ 1 & 0.7 \end{pmatrix} .$$

Mit $\|B\| < 2$ bilden wir

$$D := E - \tfrac{1}{2} \cdot \|B\|\!\!\rightarrow \begin{pmatrix} 0.55 & 0.025 \\ -0.5 & 0.65 \end{pmatrix} .$$

Die Eigenwerte von D sind $\lambda_{1,2} = 0.6 \pm i \cdot 0.15$. Nach Algorithmus 3.3 (ohne ε-Aufblähung) ergibt sich

$$Y^0 = \begin{pmatrix} [-1,+1] \\ [-1,+1] \end{pmatrix}; \quad Y^1 = \begin{pmatrix} [-0.575,0.575] \\ [-1.15,1.15] \end{pmatrix}; \quad Y^2 = \begin{pmatrix} [-0.345,0.345] \\ [-1.035,1.035] \end{pmatrix};$$

und mit $Y^2 \subsetneqq Y^1$ also, daß die Realteile der Eigenwerte von B positiv sein müssen.

Liegen Eigenwerte im mit ✱ gekennzeichneten Bereich von Abbildung 3.1, kann der Algorithmus nicht erfolgreich beendet werden. Der Bereich kann durch Vergrößerung der oben erwähnten Konstante β verkleinert werden; die Eigenwerte von $\beta^{-1} \cdot B$ werden dadurch betragsmäßig kleiner. Andererseits rutschen die Eigenwerte von D näher an den Rand des Einheitskreises und die Einschließung wird schwieriger.

Ist die Ausgangsmatrix symmetrisch, sind alle Eigenwerte reell. In diesem Fall kann gezeigt werden, daß die Matrix positiv definit ist mit folgendem Algorithmus:

Algorithmus 3.4 Numerischer Beweis, daß eine symmetrische Matrix B positiv definit ist

1) for i := 1 to n do Y_i^0 := [-1,1] ; k := -1; D := E ⊖ ||B||$^{-1}$ ⊙ B ;

2) repeat k := k+1; Y^{k+1} := Y^k := Y^k ∘ ε ;
for i := 1 to n do Y_i^{k+1} := $D_i \odot Y^k$
until $Y^{k+1} \subsetneqq Y^k$ oder {(k > ⌈$\log_{10} n$⌉ + 8) =: bool} ;
if bool then stop ;

3) Für die symmetrische Matrix B ist bewiesen, daß sie positiv definit ist

Wir geben zur Veranschaulichung der Wirkungsweise ein kleines Beispiel. Es sei

$$A = \begin{pmatrix} 10 & 9 \\ 9 & 8 \end{pmatrix} \quad \text{und} \quad D := E - \frac{1}{20} \cdot A = \begin{pmatrix} 0.5 & -0.45 \\ -0.45 & 0.6 \end{pmatrix}.$$

Wir erhalten nach Algorithmus 3.4 (ohne ε-Aufblähung) für die rechte Intervallgrenze der

ersten Komponente	1	0.95	0.9475	0.949625	0.9523...
zweiten Komponente	1	1.05	1.0575	1.060875	1.0639...

von Y^k, $k \geq 0$. Die Intervalle sind wie vorhin symmetrisch zum Ursprung (daher ist oben auch nur eine Komponente angegeben); dementsprechend kann gemäß der gegebenen Anleitung (3.4) der Algorithmus (in diesem Fall sogar erheblich) verbessert werden. Algorithmus 3.4 wurde zwecks besserer Lesbarkeit in dieser Form belassen. Die Einschließung kann in unserem Beispiel nicht erfolgen, da die Eigenwerte von A gleich $\lambda_{1,2} = 9 \pm \sqrt{82}$ sind, also einer negativ ist. Andererseits ergibt sich für

$$A = \begin{pmatrix} 10 & 8.5 \\ 8.5 & 8 \end{pmatrix} \quad \text{und} \quad D := E - \frac{1}{20} \cdot A = \begin{pmatrix} 0.5 & -0.425 \\ -0.425 & 0.6 \end{pmatrix}$$

nach Algorithmus 3.4 (ohne ε-Aufblähung) für die rechte Intervallgrenze der

ersten Komponente	1	0.925	0.898125
zweiten Komponente	1	1.025	1.008125

von Y^k, $k \geq 0$. Demnach ist $Y^2 \subsetneqq Y^1$ und A als positiv definit nachgewiesen. Tatsächlich sind die Eigenwerte von A gleich $\lambda_{1,2} = 9 \pm \sqrt{73.25}$ positiv. Nach der Summennorm konnte übrigens $\rho(D) < 1$ nicht nachgewiesen werden. Die Rechenzeit α für die Algorithmen 3.3 und 3.4 beträgt

$$\alpha = s \cdot n^2,$$

wenn mit der angegebenen Vereinfachung (3.4) die Schleife in Schritt 2 je s-mal durchlaufen wird.

3.c) Einschließung der Inversen einer Matrix

Ganz analog zur Herleitung auf Seite 37 zur Gewinnung einer geeigneten Einschließungsfunktion F für die Lösung eines linearen Gleichungssystems $Ax = b$ kann eine Einschließungsfunktion G für die Inverse einer Matrix, also der Lösung von $AX = E$ konstruiert werden. Mit der Defektiteration (das ist das Schulz-Verfahren)

$$X^{k+1} := X^k + R(E - AX^k)$$

und $R \approx A^{-1}$ folgt für $g(X) := X + R(E - AX) : M_n\mathbb{R} \to M_n\mathbb{R}$ sofort

$$g(\hat{X}) = \hat{X} \quad \Rightarrow \quad R(E - A\hat{X}) = o,$$

d.h. $E - A\hat{X} \in \operatorname{Ker} R$ für $\hat{X} \in M_n\mathbb{R}$. Geben wir eine Einschließungsfunktion G^* an zu $G^*(Y) := (R + R(E - AR)) \boxplus (E - RA) \boxtimes (Y \boxminus R) : IM_n\mathbb{R} \to IM_n\mathbb{R}$, so folgt analog zu (2.7) sofort $X \in Y \Rightarrow g(X) \in G^*(Y)$. Wie bereits im letzten Kapitel angesprochen, kann die Funktion zu

$$G(Y) := R(E - AR) \boxplus (E - RA) \boxtimes Y \tag{3.7}$$

verbessert werden. Mit $Z := R(E - AR)$ und $C := E - RA$ folgt für $G(\Omega) \subsetneqq \Omega$ aus den Sätzen 2.1 und 3.1 die Existenz eines Fixpunktes $\hat{X} \in M_n\mathbb{R}$ mit $Z + C \cdot \hat{X} = \hat{X}$ und $\rho(C) = \rho(E - RA) < 1$. Daraus ergibt sich

Satz 3.2 Gilt für $A, R \in M_n\mathbb{R}$ und G wie in (3.7) und

$$G(\Omega) \subsetneqq \Omega$$

für ein $\Omega \in IM_n\mathbb{R}$, so ist A nicht singulär und es gilt

$$A^{-1} \in \bigcap_{i=0}^{\infty} (R \boxplus G^i(\Omega)) \quad .$$

Beweis. Aus den vorigen Betrachtungen folgt mit $\rho(E - RA) < 1$ sofort die Nicht-Singularität von A und R. Nach Satz 2.2 existiert ein Fixpunkt $\hat{X} \in M_n\mathbb{R}$ von $g(X) := R(E - AR) + (E - RA) \cdot X$ in $\bigcap_{i=0}^{\infty} G^i(\Omega)$. Es folgt

$g(\hat{X}) = \hat{X} \Rightarrow R - RAR - RA\hat{X} = o$, also $RAR + RA\hat{X} = R$. Durch Multiplikation auf beiden Seiten mit $A^{-1} \cdot R^{-1}$ von links ergibt sich $R + \hat{X} = A^{-1}$, und damit die Behauptung.

Damit kann folgender Algorithmus angegeben werden:

Algorithmus 3.5 Einschließung der Inversen einer Matrix

1) Berechne $R \approx A^{-1}$ gleitkommamäßig

2) $Y^0 := Q := R \circledcirc (E \circledcirc A \circledcirc R)$; $B := E \circledcirc R \circledcirc A$; $k := -1$;

3) repeat $k := k+1$; $Y^{k+1} := Y^k := Y^k \circledcirc \varepsilon$;

   for $l := 1$ to n do

   for $i := 1$ to n do $Y^{k+1}_{ij} := Q_{ij} \circledcirc \bigcirc_{l=1}^{n} B_{il} \circledcirc Y^k_{lj}$

   until $Y^{k+1} \subsetneqq Y^k$ oder $((k > \lceil 2 \cdot \log_{10} n \rceil + 2) =: \text{bool})$;

   if bool then stop ;

4) Es ist numerisch bewiesen, daß A nicht-singulär und

$$A^{-1} \in R \circledcirc Y^{k+1} \quad .$$

Mit der bei der Ersetzung von $\boxplus$ durch $\circledcirc$ aus G entstehenden Funktion $\overline{G}$ bleibt Satz 3.2 selbstverständlich erhalten. Für das Abbruchkriterium in Schritt 3 gelten im Wesentlichen die im vorletzten Abschnitt gemachten Aussagen. Die maximale Rechenzeit α von Algorithmus 3.5 beträgt

$$\alpha = 4n^3 + s \cdot 4n^3 \quad .$$

Ist nur eine Aufblähung notwendig, was in den allermeisten Beispielen der Fall war, beträgt die Rechenzeit $8n^3$.

Im obigen Algorithmus wurde bewußt aus eine weitere gleitkommamäßige Iteration von R (etwa nach dem Schulz-Verfahren oder $A^{-1} \approx (RA)^{-1} \cdot R$, da $RA \approx E$ evtl. leichter zu invertieren ist) verzichtet. Selbst mit doppeltlangen Skalarprodukten oder Verwendung des langen Akkumulators nach Seite 60 ergibt sich im Durchschnitt eine geringe Verbesserung der Einschließung von A^{-1}. In Anbetracht

der hohen Kosten von $2n^3$ pro Schritt wurde auf eine gleitkommamäßige Iteration verzichtet. Ein weiteres Argument gegen Algorithmus 3.5 ist der enorme Speicherbedarf von mindestens $4n^2$, was den maximalen Grad der Matrix stark einschränken kann.

Eine andere Möglichkeit, die Inverse einer Matrix einzuschließen, ist die direkte Verwendung von Algorithmus 2.1. Es werden einfach die Gleichungssysteme $Ax_i=e_i$, wobei e_i die i-ten Einheitsvektoren sind, gelöst und die Matrix mit den Spalten x_i stellt bereits eine Einschließung der Inversen dar. Entscheidend bei dem Verfahren ist, daß die Näherungsinverse R und $B = E \ominus R \odot A$ nur einmal berechnet zu werden braucht. Sind Schritt 1 und 2 in Algorithmus 2.1 also einmal ausgeführt, brauchen jeweils nur noch die Schritte 3 bis 6 durchgeführt zu werden mit einem Aufwand $\alpha \leq n^2(3r+4s+4)$ gemäß Seite 62. In dem in FORTRAN implementierten Algorithmus 2.1 (siehe Anhang) ist dies durch Setzen einer booleschen Variablen NRS möglich. Der entstandene Algorithmus hat neben dem geringen Speicherbedarf von n^2 auch den Vorteil, daß (mit wenig Mehraufwand) die Spalten x_i erheblich genauer berechnet werden. Bei der besten Variante LGLG/1 (die auch im Anhang wiedergegeben ist) kann in jeder Komponente mit 15 garantierten Stellen gerechnet werden. Das Verhältnis des Aufwands zu dem der Gleitkommversion (ohne Iteration!) ist

$$\mu \leq 2 + \frac{3r+4s+4}{n} \quad ,$$

asymptotisch ist also mit dem doppelten Aufwand zu rechnen unabhängig von n.

Algorithmus 3.6 Einschließung der Inversen einer Matrix

1) Berechne $R \approx A^{-1}$ gleitkommamäßig

2) Berechne $B := E \ominus R \odot A$ intervallmäßig

3) for i := 1 to n do

   begin $b := e_i$ (i-ter Einheitsvektor);

   Führe Schritt 3 bis 5 von Algorithmus 2.1 aus;

   Setze X_i (die i-te Spalte von X $\in IM_n R$) := $x \oplus Y$

   end ;

4) Es ist bewiesen, daß A nicht singulär ist und es gilt

$$A^{-1} \in X \ .$$

Wie Algorithmus 2.1 können beide Algorithmen zur Inverseneinschließung leicht auf Intervallmatrizen ausgedehnt werden. Für eine Intervallmatrix $A \in IM_n\mathbb{R}$ und eine die Inverse einschließende Intervallmatrix $B \in IM_n\mathbb{R}$ gilt dann:

Für alle $A \in \mathbf{A}$ ist A nicht singulär und es gilt $A^{-1} \in B$.

3.d) Einschließung der Eigenwerte und Eigenvektoren einer Matrix

Im Hinblick auf Satz 2.2 ist es naheliegend, zunächst die Potenzmethode (v. Mises - Verfahren) als Einschließungsalgorithmus zu formulieren. Es ist

$$x^{k+1} := \frac{(Ax^k)}{(Ax^k)_1} =: f(x^k) \qquad (3.8)$$

mit einem Startvektor x^0 ein Iterationsverfahren zur Approximation Eigenvektors mit dem betragsgrößten Eigenwert. Hierbei bedeutet v_1 die 1-te Komponente von $v \in V_n\mathbb{R}$. Man kann aus (3.8) direkt eine Intervallfunktion F gewinnen. Zu beachten ist hierbei nur, daß die 1-te Komponente "nicht wirklich" berechnet sondern gleich 1 gesetzt wird:

$$
\begin{aligned}
F(X)_i &:= (A \boxdot X) \oslash (A \odot X)_1 \quad \text{für } i = 1(1)n,\ i \neq 1 \\
F(X)_1 &:= 1 \ .
\end{aligned}
\tag{3.9}
$$

Offenbar gilt $x \in X \Rightarrow f(x) \in F(X)$ für $x \in V_n\mathbb{R}$ und $X \in IV_n\mathbb{R}$. Ist $F(\Omega) \subsetneq \Omega$, so besitzt f nach Satz 2.1 einen Fixpunkt $\hat{x}$ in $\Omega \in IV_n\mathbb{R}$ und es gilt

$$
f(\hat{x}) = \hat{x} \Rightarrow A\hat{x} = (A\hat{x})_1 \cdot \hat{x} \ .
$$

Demnach ist $F(X)$ eine Einschließung eines Eigenvektors und $(A \cdot \Omega)_1$ eine Einschließung des zugehörigen Eigenwerts von A. Man beachte, daß etwa die Normierung mittels der Euklidschen Norm, also

$$
x^{k+1} := |Ax^k|^{-1} \cdot (Ax^k) \quad \text{mit} \quad |x| := \Big(\sum_{i=1}^{n} x_i^2\Big)^{1/2}
$$

nicht zu einer Einschließung mit der daraus gebildeten Intervallfunktion führen kann. Das liegt daran (salopp gesagt), daß die Division durch $|Ax|$ i.a. nicht exakt ausgeführt werden kann und daher "in der Eigenrichtung" eine (unzulässige) Aufblähung stattfindet. Für (3.9) ist dies nicht der Fall, da hier nur in der Teilmenge $\{x \mid x_1 = 1\}$ des $\mathbb{R}^n$ gearbeitet wird.

Das Verfahren ist jedoch sehr empfindlich. Bereits bei einem Grad ≥ 5 und nicht gut konditionierter Matrix wird keine Einschließung mehr erreicht.
Es sei bemerkt, daß die Berechnung von F nach (3.9) im Raum der Intervallvektoren $IV_n\mathbb{R}$ praktisch die bestmögliche Einschließung darstellt, wenn die Skalarprodukte $A_i \odot X$ (A_i ist die i-te Zeile von A) "genau" berechnet werden. Darunter verstehen wir, daß das engstmögliche Intervall $I \in IV_n\mathbb{R}$ berechnet wird, daß $A_i \odot X$ einschließt:

$$
A_i \odot X \in J \in IV_n\mathbb{R} \quad \Rightarrow \quad I \subseteq J \ .
$$

Denn jede Komponente von $A \odot X$ ist ein Skalarprodukt, in dem jede Variable nur einmal vorkommt. Da die Komponenten unabhängig berechnet werden, ist die einzige Aufblähung bei Division durch $(A \odot X)_1$ möglich.

Doch selbst bei diesem Vorgehen gewinnt das Verfahren wenig und außerdem ist es nur für den betragsgrößten Eigenwert brauchbar. Daher wurde ein allgemein verwendbares Verfahren entworfen.

Gesucht sei eine Lösung des (nicht-linearen) Gleichungssystems

$$
\begin{aligned}
(A - \lambda E) \cdot x &= o \\
e_k' \cdot x - 1 &= o
\end{aligned}
\tag{3.10}
$$

in den Unbekannten $x \in \mathbb{R}^n$, $\lambda \in \mathbb{R}$ für e_k' der transponierte k-te Einheitsvektor.
Wir definieren die Funktion $f : V_{n+1}\mathbb{R} \to V_{n+1}\mathbb{R}$ zu

$$
f\begin{pmatrix} x \\ \lambda \end{pmatrix} := -R \cdot \begin{pmatrix} (A - \tilde{\lambda}E)\tilde{x} \\ e_k' \cdot \tilde{x} - 1 \end{pmatrix} + \left(E - R \cdot \begin{pmatrix} A - \tilde{\lambda}E & -\tilde{x} \\ e_k' & 0 \end{pmatrix}\right) \cdot \begin{pmatrix} x \\ \lambda \end{pmatrix}
\tag{3.11}
$$

für ein $R \in M_{n+1}\mathbb{R}$. Dann gilt der folgende Satz :

Satz 3.3 Die Funktion $F : IV_{n+1}\mathbb{R} \to IV_{n+1}\mathbb{R}$ sei für ein festes $A \in M_n\mathbb{R}$, $R \in M_{n+1}\mathbb{R}$, $\tilde{x} \in V_n\mathbb{R}$ und $\tilde{\lambda} \in \mathbb{R}$ wie folgt definiert :

$$
\begin{aligned}
F\begin{pmatrix} X \\ \Lambda \end{pmatrix} &:= Z \boxplus \left(E \boxminus R \boxtimes \begin{pmatrix} A \boxminus \tilde{\lambda}E & \boxminus X \boxplus \tilde{x} \\ e_k' & 0 \end{pmatrix}\right) \cdot \begin{pmatrix} X \\ \Lambda \end{pmatrix} \\
\text{mit } Z &:= \boxminus R \boxtimes \begin{pmatrix} (A \boxminus \tilde{\lambda}E) \boxtimes \tilde{x} \\ 0 \end{pmatrix} \in IV_{n+1}\mathbb{R} \ .
\end{aligned}
\tag{3.12}
$$

Die k-te Komponente des reellen Vektors $\tilde{x}$ sei gleich 1.
Gilt dann für ein $\Omega \in IV_{n+1}\mathbb{R}$ mit $\Omega = (\hat{X}, \hat{\Lambda})'$

$$
F(\Omega) \subsetneqq \Omega \qquad \text{außer für die k-te Komponente ,}
\tag{3.13}
$$

so besitzt A einen Eigenvektor $x \in V_n\mathbb{R}$ mit zugehörigem Eigenwert $\lambda \in \mathbb{R}$, so daß

$$
\lambda \in \tilde{\lambda} \oplus \hat{\Lambda} \quad \text{und} \quad x \in \tilde{x} \oplus \hat{X} \ .
\tag{3.14}
$$

Dieser Eigenwert hat die Vielfachheit 1. Für $(X^i, \Lambda^i) := F^i(\Omega)$ gilt überdies

$$
\lambda \in \tilde{\lambda} \oplus \Lambda^i \quad \text{und} \quad x \in \tilde{x} \oplus X^i \qquad \text{für } o \leq i \in \mathbb{N} \ .
$$

Beweis. Für f gilt offenbar

$$(3.15) \qquad y \in Y \in IV_n\mathbb{R} \text{ und } \mu \in M \in I\mathbb{R} \Rightarrow f\begin{pmatrix} y \\ \mu \end{pmatrix} \in f\begin{pmatrix} Y \\ M \end{pmatrix},$$

insbesondere wegen $e_k' \cdot \tilde{x} = 1$. Wegen Satz 2.1 folgt aus (3.13) die Existenz eines Fixpunktes $(\hat{x},\hat{\lambda})'$ von f. Für diesen gilt dann nach (3.11)

$$(3.16) \qquad \begin{pmatrix} \tilde{x} \\ \tilde{\lambda} \end{pmatrix} = \begin{pmatrix} \hat{x} \\ \hat{\lambda} \end{pmatrix} - R \cdot \left\{ \begin{pmatrix} (A - \tilde{\lambda}E)\tilde{x} \\ 0 \end{pmatrix} + \begin{pmatrix} A - \tilde{\lambda}E & -\hat{x}-\tilde{x} \\ e_k' & 0 \end{pmatrix} \cdot \begin{pmatrix} \hat{x} \\ \hat{\lambda} \end{pmatrix} \right\} \quad \text{und somit}$$

$$R \cdot \begin{pmatrix} (A - \tilde{\lambda}E)\cdot(\tilde{x}+\hat{x}) - \hat{\lambda}\cdot(\tilde{x}+\hat{x}) \\ e_k' \cdot \hat{x} \end{pmatrix} = R \cdot \begin{pmatrix} \{A - (\tilde{\lambda}+\hat{\lambda})E\} \cdot (\tilde{x}+\hat{x}) \\ e_k' \cdot \hat{x} \end{pmatrix} = 0 .$$

Aus Satz 3.1 folgt, daß für jedes $C \in E \boxminus R\boxplus\begin{pmatrix} A\boxminus\tilde{\lambda}E & \boxminus X\boxminus\tilde{x} \\ e_k' & 0 \end{pmatrix}$ gilt $\rho(C) < 1$. Daraus folgt insbesondere die Nicht-Singularität von R. Aus (3.15) ergibt sich daher

$$(3.17) \qquad \begin{matrix} A \cdot (\tilde{x}+\hat{x}) = (\tilde{\lambda}+\hat{\lambda})\cdot(\tilde{x}+\hat{x}) \\ e_k' \cdot \hat{x} = 0 \end{matrix} .$$

So folgt, daß $\tilde{x}+\hat{x}$ ein Eigenvektor der Matrix A ist zum Eigenwert $\tilde{\lambda}+\hat{\lambda}$. Die k-te Komponente von $\tilde{x}+\hat{x}$ ist gleich 1. Wegen $\hat{x} \in \hat{X}$ und $\hat{\lambda} \in \hat{\Lambda}$ folgt (3.14).

Für $x \in V_n\mathbb{R}$ und $\lambda \in \mathbb{R}$ ist

$$G\begin{pmatrix} x \\ \lambda \end{pmatrix} := -R \cdot \begin{pmatrix} (A - \tilde{\lambda}E)x \\ 0 \end{pmatrix} + \{E - R \cdot \begin{pmatrix} A - \tilde{\lambda}E - \tilde{\lambda}E & -\tilde{x} \\ e_k' & 0 \end{pmatrix}\}\begin{pmatrix} x \\ \lambda \end{pmatrix} = f\begin{pmatrix} x \\ \lambda \end{pmatrix} .$$

Für den Komplex

$$G\begin{pmatrix} X \\ \Lambda \end{pmatrix} := \{\psi \mid \exists\, x \in X,\ \lambda \in \Lambda :\ \psi = G\begin{pmatrix} x \\ \lambda \end{pmatrix}\} \text{ mit } X \in IV_n\mathbb{R},\ \Lambda \in I\mathbb{R}$$

gilt nach (3.13) sicher $G(\Omega) \subseteq \Omega$ wegen (3.15) und $G(\Omega) \subseteq F(\Omega)$. Nach Satz 3.1 muß insbesondere die Matrix

$$(3.18) \qquad \begin{pmatrix} A - (\tilde{\lambda}+\hat{\lambda})E & -\tilde{x} \\ e_k' & 0 \end{pmatrix}$$

nicht-singulär sein. Hatte $\lambda = \tilde{\lambda}+\hat{\lambda}$ eine Vielfachheit größer als 1, so hätte der linke obere n-n-Hauptminor aus (3.18) höchstens den Rang n-2. Durch 2-fache

Änderung kann sich somit der Rang auf höchstens n erhöhen. □

Man beachte, daß für die Anwendung von Satz 2.1 nur $F(\Omega) \subseteq \Omega$ benötigt wurde. Erst beim Beweis, daß λ von der Vielfachheit 1 ist, wurde von $F(\Omega) \subsetneq \Omega$ Gebrauch gemacht, und zwar nur von der eingeschränkten Version (3.13).

In der praktischen Durchführung wird man um schnelle Konvergenz zu erzielen

$$(3.19) \qquad R := \begin{pmatrix} A - \tilde{\lambda}E & -\tilde{x} \\ \zeta \cdot e_k' & 0 \end{pmatrix}^{-1}$$

wählen. Dabei kann davon Gebrauch gemacht werden, daß R die Gestalt

$$(3.20) \qquad \begin{pmatrix} * \\ 0 \ \ldots \ 0 \ 1 \\ * \end{pmatrix} \rightarrow \text{ k-te Zeile}$$

hat. Macht man sich diese Beobachtung zu Nutze erkennt man sofort, daß bei Anwendung von f aus (3.11) die k-te Komponente von $(x,\lambda)'$ gleich 0 ist. Bei der intervallmäßigen Auswertung nach (3.12) braucht Inklusion also für die k-te Komponente nicht überprüft zu werden. Die Beobachtung ist auch insofern wesentlich, da bei der Berechnung der k-ten Komponente nach (3.12) auch nur $F(\Omega) \subseteq \Omega$ kaum erfüllt werden könnte. Oblicherweise wird man, wenn die 1-te Komponente einer Gleitkommanäherung eines Eigenvektors den größten absoluten Betrag hat, k = 1 setzen.

Mit diesen Vorbereitungen könnten wir bereits einen effizienten Algorithmus zur Eigenvektor- und Eigenwert- Einschließung erstellen. In der Praxis hat sich jedoch gezeigt, daß hier und da teilweise entscheidende Verbesserungen angebracht werden können. So kann man bereits etwas Aufwand sparen durch die Beobachtung, daß in der letzten Zeile der Matrix

$$\begin{pmatrix} A - \lambda E & -x \\ e_k' & 0 \end{pmatrix}$$

nicht e_k' zu stehen braucht sondern auch ein beliebiges Vielfaches $\zeta \cdot e_k'$, $\zeta \neq o$ stehen kann. In diesem Fall ändert sich die k-te Zeile in (3.20) zu $(o,\ldots,o,\zeta^{-1})$. Bei der Anwendung von f nach (3.11) wird die k-te Komponente von $(x,\lambda)'$ jedoch wieder o. Für die Praxis heißt dies, daß für den Algorithmus nur ein k festgelegt werden muß und fortan die k-te Komponente überhaupt nicht mehr mehr berücksichtigt wird. Im Ergebnisintervallvektor $\hat{X}$ steht dann in der k-ten Komponente o und die k-te Komponente x_k des Eigenvektors x ist gleich $\tilde{x}_k$.

Weiterhin stellte sich heraus, daß die Gleitkommanäherungen $\tilde{x}$ und $\tilde{\lambda}$ für Eigenvektor und zugehörigen Eigenwert bisweilen sehr schlecht sein können, und zwar insbesondere für nicht-symmetrische Matrizen. Z.B. wurden in Bezug auf das Eigenproblem schlecht konditionierte Matrizen wie folgt konstruiert. Es wurden Gleitkommazahlen λ_i, $i = 1(1)n$ zufällig im Intervall $[-1,+1]$ gewählt und damit eine n×n-Diagonalmatrix D aufgebaut. Diese hat dann trivialerweise die Eigenwerte λ_i. Für nicht-singuläres A hat $A^{-1}DA$ die gleichen Eigenwerte λ_i; ist A schlecht konditioniert, ist auch das Eigenproblem von $A^{-1}DA$ schlecht konditioniert. Bei der gleitkommamäßigen Berechnung von $A^{-1}DA$ können sich die Eigenwerte gegenüber λ_i durch Rundungsfehler leicht verändern; das ist jedoch in unseren Beispielen ohne Belang. Für A wurden die Matrizen Q - 2 (siehe Abschnitt 1.c auf Seite 15, Definition f) gewählt. Zur Berechnung der Gleitkommanäherungen der Eigenvektoren und Eigenwerte wurden Algorithmen aus der Rechenzentrumsbibliothek der UNIVAC 1108 der Universität Karlsruhe verwendet. Sie arbeiten mittels Reduktion auf eine Hessenbergmatrix. Es waren in unserm Beispiel für Grad 5 i.a. nur 5 bis 6 von $8\frac{1}{2}$ Dezimalstellen der Eigenwerte und der Komponenten der Eigenvektoren richtig, für Grad 10 jedoch durchschnittlich nur noch 3. Es traten nicht selten erheblich krassere Fälle auf, so für Grad 5 ein relativer Fehler von 4.5_{10}-5 und für Grad 10 von 4.3_{10}-2. Im letzten Fall war also nur noch eine von $8\frac{1}{2}$ Dezimalen richtig für ein derart "kleines" Problem. Daher wurde eine Gleitkommaiteration für $\tilde{x}$ und $\tilde{\lambda}$ angestrebt. Wird $x^0 := \tilde{x}$ und $\lambda^0 := \tilde{\lambda}$ gesetzt, wird mittels

$$\begin{pmatrix} x^{k+1} \\ \lambda^{k+1} \end{pmatrix} := \begin{pmatrix} x^k \\ \lambda^k \end{pmatrix} - R \cdot \begin{pmatrix} (A - \lambda^k E)x^k \\ o \end{pmatrix} \tag{3.21}$$

iteriert mit R aus (3.19). Hierbei kann wieder sinnvoll der lange Akkumulator eingesetzt werden. Darüberhinaus kann wieder eine Art Defektiteration durchgeführt werden. Setzt man $\bar{x} := x^{k+1}$ und $\bar{\lambda} := \lambda^{k+1}$ (die letzten Gleitkommaiterierten aus (3.21)) und

$$\begin{pmatrix} \Delta x^o \\ \Delta\lambda^o \end{pmatrix} := - R \cdot \begin{pmatrix} (A - \bar{\lambda}E)\bar{x} \\ o \end{pmatrix},$$

so kann mittels

$$\begin{pmatrix} \Delta x^{k+1} \\ \Delta\lambda^{k+1} \end{pmatrix} := - R \cdot \begin{pmatrix} (A - \bar{\lambda}E + \Delta\lambda^k E) \cdot (\bar{x} - \Delta x^k) \\ o \end{pmatrix} \tag{3.22}$$

noch eine wesentliche Verbesserung der Näherung erreicht werden. Schließlich wird

$$\begin{pmatrix} x^* \\ \lambda^* \end{pmatrix} := \begin{pmatrix} \bar{x} \\ \bar{\lambda} \end{pmatrix} + \begin{pmatrix} \Delta x \\ \Delta\lambda \end{pmatrix}, \tag{3.23}$$

wobei Δx und $\Delta\lambda$ die letzten Näherungen aus (3.22) sind, als Gleitkommanäherungen im Algorithmus verwendet. x^* und λ^* werden jedoch nicht wirklich berechnet sondern bei ihrem Auftreten werden $\bar{x}$, $\bar{\lambda}$ und Δx, $\Delta\lambda$ mit dem Distributivgesetz verwendet. Dadurch wird (mit Hilfe des langen Akkumulators) quasi eine doppelte Genauigkeit simuliert, ohne jedoch doppeltlang rechnen zu müssen und insbesondere mit dem Vorteil, den langen Akkumulator einsetzen zu können (das Produkt zweier einfachlanger Gleitkommazahlen ist exakt in einer doppeltlangen Variablen berechenbar, das gilt jedoch nicht mehr für doppeltlange Zahlen und daher ist der lange Akkumulator dort nicht einsetzbar). Die Iterationen (3.21) und (3.22) werden vorteilhaft im Einzelschrittverfahren ausgeführt.

Statt der Funktion F aus (3.12) kann übrigens auch

$$\overline{F}\begin{pmatrix}X\\ \Lambda\end{pmatrix} := Z \oplus (E \ominus R \odot \begin{pmatrix} A \ominus \tilde{\lambda}E \ominus \Lambda E & -\tilde{x} \\ e_k' & 0 \end{pmatrix}) \odot \begin{pmatrix} X \\ \Lambda \end{pmatrix} \tag{3.24}$$

verwendet werden (Z wie in (3.12)). Bei der algorithmischen Durchführung beider Varianten ergeben sich gleich gute Lösungsintervalle. Gleichwohl könnte es bei beiden Verfahren passieren, daß der nach Satz 3.3 im Lösungsintervall X bzw. Λ existierende Eigenvektor bzw. Eigenwert nicht eindeutig ist, resp. Eine leichte Modifizierung der Definition sichert bereits die Eindeutigkeit, wie der folgende Satz zeigt.

Satz 3.4 Die Funktion $F : IV_{n+1}\mathbb{R} \to IV_{n+1}\mathbb{R}$ sei für ein festes $A \in M_n\mathbb{R}$, $R \in M_{n+1}\mathbb{R}$, $\tilde{x} \in V_n\mathbb{R}$ und $\tilde{\lambda}, \zeta \in \mathbb{R}$ wie folgt definiert:

$$F\begin{pmatrix}X\\ \Lambda\end{pmatrix} := Z \oplus (E \ominus R \odot \begin{pmatrix} A \ominus \tilde{\lambda}E \ominus \Lambda E & \ominus X \ominus \tilde{x} \\ \zeta \cdot e_k' & 0 \end{pmatrix}) \odot \begin{pmatrix} X \\ \Lambda \end{pmatrix} \tag{3.25}$$

mit Z wie in (3.12). Gilt dann für ein $\Omega \in IV_{n+1}\mathbb{R}$ mit $\Omega = (\hat{X}, \hat{\Lambda})'$, $\hat{X} \in IV_n\mathbb{R}$, $\hat{\Lambda} \in I\mathbb{R}$

$$F(\Omega) \subsetneq \Omega \qquad \text{außer für die k-te Komponente,} \tag{3.26}$$

und ist entweder $0 \in \hat{\Lambda}$ oder $0 \in \hat{X}$, so gelten alle Aussagen von Satz 3.3. Darüberhinaus ist der Eigenvektor $x \in \tilde{x} \oplus \hat{X}$ und der Eigenwert $\lambda \in \tilde{\lambda} \oplus \hat{\Lambda}$ eindeutig, d.h. gilt für einen Eigenwert μ bzw. einen Eigenvektor y von A $\mu \in \tilde{\lambda} \oplus \hat{\Lambda}$ bzw. $y \in \tilde{x} \oplus \hat{X}$, so folgt $\mu = \lambda$ bzw. $y = x$, resp.

Beweis: Unter Berücksichtigung der Bemerkung über ζ von vorhin kann der Beweis von Satz 3.3 direkt übernommen werden, wenn entweder $0 \in \hat{\Lambda}$ oder $0 \in \hat{X}$ gilt. Damit ist der erste Teil des Satzes bereits bewiesen. Zum Beweis des zweiten Teils definieren wir

$$H := \begin{pmatrix} A \ominus \tilde{\lambda}E \ominus \Lambda E & \ominus X \ominus \tilde{x} \\ \zeta \cdot e_k' & 0 \end{pmatrix} \in IM_{n+1}\mathbb{R}. \tag{3.27}$$

Mit (3.26) folgt aus Satz 3.1 für jedes $H \in \mathcal{H}$ die Nicht-Singularität von H. Angenommen, außer $0 \neq x \in \tilde{x} \oplus \hat{X}$, $\lambda \in \tilde{\lambda} \oplus \hat{\Lambda}$ mit $Ax = \lambda x$ gäbe es ein $\mu \in \tilde{\lambda} \oplus \hat{\Lambda}$, so daß ein $0 \neq y \in V_n\mathbb{R}$ existiert mit $Ay = \mu y$ (hierbei ist nicht $y \in \tilde{x} \oplus \hat{X}$ vorausgesetzt). Wir unterscheiden zwei Fälle.

a) Die k-te Komponente y_k von y ist gleich 0. Daraus folgt

$$H \cdot \begin{pmatrix} y \\ 0 \end{pmatrix} \supseteq \begin{pmatrix} A - \mu E & -x \\ \zeta \cdot e_k' & 0 \end{pmatrix} \cdot \begin{pmatrix} y \\ 0 \end{pmatrix} = \begin{pmatrix} (A - \mu E)y \\ 0 \end{pmatrix} = 0$$

wegen $e_k' \cdot y = 0$. Also folgt wegen der Nicht-Singularität jedes $H \in \mathcal{H}$ sofort $y = 0$ und damit die Eindeutigkeit von λ.

b) Die k-te Komponente y_k von y ist ungleich 0. Dann existiert ein $\xi \in \mathbb{R}$ mit $(x + \xi y)_k = 0$, also

$$H \cdot \begin{pmatrix} x + \xi y \\ \lambda - \mu \end{pmatrix} \supseteq \begin{pmatrix} A - \mu E & -x \\ \zeta \cdot e_k' & 0 \end{pmatrix} \cdot \begin{pmatrix} x + \xi y \\ \lambda - \mu \end{pmatrix} =$$

$$= \begin{pmatrix} (A - \mu E)(x + \xi y) - (\lambda - \mu)x \\ 0 \end{pmatrix} =$$

$$= \begin{pmatrix} (\lambda - \mu)x + 0 - (\lambda - \mu)x \\ 0 \end{pmatrix} = 0 .$$

Daraus folgt wie oben aus der Nicht-Singularität jedes $H \in \mathcal{H}$ $\lambda = \mu$ und daher insgesamt die Eindeutigkeit des Eigenwerts $\lambda \in \tilde{\lambda} \oplus \hat{\Lambda}$.

Bleibt die Eindeutigkeit des Eigenvektors x nachzuweisen. Angenommen, es gäbe ein $0 \neq y \in \tilde{x} \oplus \hat{X}$, so daß ein $\mu \in \mathbb{R}$ existiert mit $Ay = \mu y$ (es ist nicht $\mu \in \tilde{\lambda} \oplus \hat{\Lambda}$ vorausgesetzt). Dann folgt wegen $x_k \neq 0$ die Existenz eines $\xi \in \mathbb{R}$ mit $(y + \xi x)_k = 0$. Also ist

$$H \cdot \begin{pmatrix} y + \xi x \\ \mu - \lambda \end{pmatrix} \supseteq \begin{pmatrix} A - \lambda E & -y \\ \zeta \cdot e_k' & 0 \end{pmatrix} \cdot \begin{pmatrix} y + \xi x \\ \mu - \lambda \end{pmatrix} =$$

$$= \begin{pmatrix} (\mu - \lambda)y + 0 - (\mu - \lambda)y \\ 0 \end{pmatrix} = 0 .$$

Wegen der Nicht-Singularität jedes $H \in \mathcal{H}$ folgt also $\mu = \lambda$. Da λ aber die Vielfachheit 1 hat, ist sogar $x = y$ und damit die Eindeutigkeit des Eigenvektors x in $\tilde{x} \oplus \hat{X}$ gezeigt. Damit ist der Satz vollständig bewiesen.

Die Definition von F nach (3.25) ist eine Aufblähung gegenüber F nach (3.12). In der praktischen Anwendung macht sich dies sehr wohl bemerkbar, da etwa bei der Berechnung von $A \ominus \tilde{\lambda}E \ominus \Lambda E$ nicht mehr der lange Akkumulator eingesetzt werden kann. Dementsprechend ergeben sich zum Teil erheblich schlechtere Lösungsintervalle. Daher wurde nach Aussagen gesucht, die eine Eindeutigkeit von Eigenvektor und Eigenwert bereits mit schwächeren Voraussetzungen garantieren. Im obigen Beweis wurde ausschließlich von der Nicht-Singularität der Matrix H nach (3.27) Gebrauch gemacht. Da in diesem Fall der Kern nur aus der o besteht wurden im Fall der Mehrdeutigkeit entsprechende Vektoren aus dem Kern angegeben und daraus ein Widerspruch abgeleitet. Bei näherem Betrachten erhellt, daß zum Beweis stärkerer Aussagen mehr Information über die Funktion F nach (3.12) eingesetzt werden muß. Die beiden nachstehenden Sätze gelangen auf diesem Weg zu der starken Aussage, daß bereits die Funktion F nach (3.12) ausreicht, die Eindeutigkeit von Eigenwert und Eigenvektor bei Eintreten von $F(\Omega) \subsetneq \Omega$ zu beweisen.

Zuvor sei ein Lemma angegeben, das die Eindeutigkeit des Fixpunktes $(x, \lambda)'$ feststellt. Man beachte den Unterschied, daß jetzt nur nachgewiesen wird, daß nicht zwei Paare von Eigenvektor und zugehörigem Eigenwert im Lösungsintervall liegen.

Lemma 3.5 Unter den Voraussetzungen von Satz 3.1 besitzt die Funktion $f(x) := z + C \cdot x : \mathbb{R}^n \to \mathbb{R}^n$ genau einen Fixpunkt in $z \oplus C \odot \Omega$.

Beweis. Offenbar bildet f das Intervall Ω auf sich ab und besitzt daher nach Satz 2.1 mindestens einen Fixpunkt in Ω. Dieser Fixpunkt liegt auch in $z \oplus C \odot \Omega$.

Angenommen, f besäße zwei Fixpunkte x und y in Ω, also

$$x = f(x) = z + C \cdot x \qquad \text{und}$$
$$y = f(y) = z + C \cdot y .$$

Zieht man beide Gleichungen voneinander ab ergibt sich

$$x - y = C \cdot x - C \cdot y = C \cdot (x - y).$$

Wenn $x \neq y$ wäre, hätte C also einen Eigenwert gleich 1, im Widerspruch zu Satz 3.1.

Satz 3.6 Die Funktion $F : IV_{n+1}\mathbb{R} \to IV_{n+1}\mathbb{R}$ sei für ein festes $A \in \mathbb{M}_n\mathbb{R}$, $R \in M_{n+1}\mathbb{R}$, $\tilde{x} \in V_n\mathbb{R}$ und $\tilde{\lambda}, \zeta \in \mathbb{R}$ wie folgt definiert:

$$F\begin{pmatrix} X \\ \Lambda \end{pmatrix} := Z \oplus \left\{ E \ominus R \odot \begin{pmatrix} A \ominus \tilde{\lambda}E & \ominus X \ominus \tilde{x} \\ \zeta \cdot e_k' & o \end{pmatrix} \right\} \odot \begin{pmatrix} X \\ \Lambda \end{pmatrix} \tag{3.28}$$

mit Z wie in (3.12). Gilt dann für ein $\Omega \in IV_{n+1}\mathbb{R}$ mit $\Omega = (\hat{X}, \hat{\Lambda})'$, $\hat{X} \in IV_n\mathbb{R}$, $\hat{\Lambda} \in \mathbb{IR}$

$$F(\Omega) \subsetneq \Omega \qquad \text{außer für die k-te Komponente,} \tag{3.29}$$

so gelten alle Aussagen vnn Satz 3.3. Darüberhinaus ist der Eigenwert λ im Intervall $\tilde{\lambda} \oplus \hat{\Lambda}$ eindeutig, d.h. für einen beliebigen Eigenwert $\mu \in \tilde{\lambda} \oplus \hat{\Lambda}$ der Matrix A folgt $\mu = \lambda$.

Beweis. Für einen Moment gehen wir von der Defektschreibweise ab und schreiben für F

$$F\begin{pmatrix} X \\ \Lambda \end{pmatrix} := \begin{pmatrix} X \\ \Lambda \end{pmatrix} \oplus Z \oplus \left\{ E \ominus R \odot \begin{pmatrix} A \ominus \tilde{\lambda}E & \ominus X \\ \zeta \cdot e_k' & o \end{pmatrix} \right\} \odot \begin{pmatrix} X \ominus \tilde{x} \\ \Lambda \ominus \tilde{\lambda} \end{pmatrix} . \tag{3.30}$$

Nach Umschreiben von f aus (3.11) und leichter Umformung (siehe Beweis von Satz 3.3) ergibt sich

$$(3.31)\qquad f\begin{pmatrix} t \\ \nu \end{pmatrix} := \begin{pmatrix} t \\ \nu \end{pmatrix} - R \cdot \begin{pmatrix} (A - \nu E)\cdot t \\ \zeta \cdot e_k' t - 1 \end{pmatrix}$$

und $x \in I \subset IV_n\mathbb{R} \Rightarrow f(x) \subset F(I)$. Mit $X := \tilde{X} \oplus \hat{X}$ und $\Lambda := \tilde{\lambda} \oplus \hat{\Lambda}$ ergibt sich aus (3.29) mit dem temporären F aus (3.30) $F(X,\Lambda)' \subsetneq (X,\Lambda)'$. Es existiert also ein $x = \tilde{x} + \hat{x}$ und ein $\lambda = \tilde{\lambda} + \hat{\lambda}$ mit $Ax = \lambda x$. Angenommen, es gäbe ein $\mu \in \Lambda$, so daß ein $y \in \mathbb{R}^n$ existiert mit $Ay = \mu y$ (es ist nicht $y \in X$ vorausgesetzt). Dann definieren wir die Funktion $g_\mu(t) : \mathbb{R}^n \to \mathbb{R}^n$ als die Funktion die aus f entsteht, wenn in (3.31) $\nu = \mu$ gesetzt wird und nur die ersten n Komponenten betrachtet werden, also

$$f\begin{pmatrix} t \\ \mu \end{pmatrix} = \begin{pmatrix} g_\mu(t) \\ \cdot\cdot \end{pmatrix} .$$

Offenbar ist $g_\mu : X \to X$ und besitzt daher nach Satz 2.1 einen Fixpunkt z in X. Sei

$$(3.32)\qquad f\begin{pmatrix} z \\ \mu \end{pmatrix} = \begin{pmatrix} z \\ \mu^* \end{pmatrix} .$$

Für ein beliebiges aber festes $\nu \in \mathbb{R}$ ist

$$(3.33)\qquad f\begin{pmatrix} t \\ \mu \end{pmatrix} = \begin{pmatrix} t \\ \nu \end{pmatrix} \Leftrightarrow f\begin{pmatrix} t \\ \mu \end{pmatrix} = \begin{pmatrix} t \\ \mu \end{pmatrix} - R \cdot \begin{pmatrix} (A - \mu E)\cdot t \\ \zeta \cdot e_k' t - 1 \end{pmatrix} = \begin{pmatrix} t \\ \nu \end{pmatrix} \Leftrightarrow$$

$$\Leftrightarrow \begin{pmatrix} o \\ \mu - \nu \end{pmatrix} = R \cdot \begin{pmatrix} (A - \mu E)\cdot t \\ \zeta \cdot e_k' t - 1 \end{pmatrix} \Leftrightarrow R^{-1} \cdot \begin{pmatrix} o \\ \mu - \nu \end{pmatrix} = \begin{pmatrix} (A - \mu E)\cdot t \\ \zeta \cdot e_k' t - 1 \end{pmatrix}$$

wegen der Nicht-Singularität von R, die aus Satz 3.1 folgt. Es sei

$$(3.34)\qquad N(\nu) := \{ t \in \mathbb{R}^n \mid f\begin{pmatrix} t \\ \mu \end{pmatrix} = \begin{pmatrix} t \\ \nu \end{pmatrix} \} .$$

Ist s die letzte Spalte von R^{-1} ohne das letzte Element $(R^{-1})_{n+1,n+1} =: a$, so folgt nach (3.33)

$$(3.35)\qquad t \in N(\nu) \Leftrightarrow (A - \mu E)t = (\mu - \nu)s \quad \text{und} \quad \zeta \cdot e_k' t - 1 = (\mu - \nu)a.$$

Wäre die k-te Komponente y_k von y gleich o, so folgte wie unter Punkt a) des Beweises von Satz 3.4 ein Widerspruch. Also ist $y_k \neq o$. Wegen $z \in M(\mu^*)$ nach (3.32) und (3.34) ist für beliebiges aber festes $\nu \in \mathbb{R}$ mit $\zeta := (\mu - \nu)/(\mu - \mu^*)$ und $\eta := (1 - \zeta)/(\zeta y_k)$ unter der Voraussetzung $\mu \neq \mu^*$ wegen $\zeta \neq o$

$$(A - \mu E)(\zeta z + \eta y) = \zeta(A - \mu E)z = \zeta(\mu - \mu^*)s = (\mu - \nu)s \quad \text{und}$$

$$\zeta e_k'(\zeta z + \eta y) - 1 = \zeta(\mu - \mu^*)a - 1 + \zeta + \zeta\eta y_k = (\mu - \nu)a,$$

und daher $\zeta z + \eta y \in N(\nu)$ nach (3.35). Also ist $N(\nu)$ nicht leer für alle $\nu \in \mathbb{R}$. Wir definieren die Funktion $h(\nu) := \zeta z + \eta y : \mathbb{R} \to \mathbb{R}^n$. h ist stetig in ganz $\mathbb{R}$, denn

$$h(\nu + \varepsilon) - h(\nu) = \frac{-\varepsilon}{\mu - \mu^*} z + \frac{\varepsilon}{(\mu - \mu^*)\zeta y_k} y = \varepsilon \cdot \kappa ,$$

wobei κ konstant. Sind y und z linear unabhängig, wächst der Betrag von $h(\nu)$ für $\nu \to \infty$ über alle Grenzen. Sind y und z linear abhängig, also etwa $y = \rho z$, so ist

$$\zeta z + \eta y = z \cdot \{ \frac{\mu - \nu}{\mu - \mu^*} + \frac{\rho(1 - \frac{\mu - \nu}{\mu - \mu^*})}{\zeta y_k} \} .$$

Nur für $\rho = \zeta y_k$ wäre $h(\nu) = z = \text{const}$, für alle anderen ν wächst $h(\nu)$ für $\nu \to \infty$ über alle Grenzen. Für $\rho = \zeta y_k$ ist aber nach (3.32) und (3.35) und $z \in M(\mu^*)$ $(\mu - \mu^*)s = o$ und $\zeta y_k/\rho - 1 = o = (\mu - \mu^*)a$. Also ist entweder $\mu = \mu^*$ oder $s = o$ und $a = o$. Im letzten Fall wäre jedoch die letzte Spalte von R^{-1} gleich o im Widerspruch zur Nicht-Singularität von R. Nehmen wir immer noch $\mu \neq \mu^*$ an, kann wegen $h(\mu^*) = z \in X$, der Stetigkeit von h, der Beschränktheit von X und der Tatsache, daß $h(\nu) \to \infty$ für $\nu \to \infty$ ein γ gefunden werden mit $h(\gamma) \in \partial X$. Nach der Definition (3.34) von M ist jedoch $h(\nu)$ für jedes $\nu \in \mathbb{R}$ Fixpunkt von g_μ; insbesondere ist $h(\gamma)$ Fixpunkt von g_μ. Wegen $F(X,\Lambda)' \subsetneq (X,\Lambda)'$ kann jedoch kein Fixpunkt von g_μ auf dem Rand von X liegen. Aus diesem Widerspruch ergibt sich $\mu = \mu^*$.

Jetzt folgt aber aus (3.32) die Existenz eines Fixpunktes (y, μ) von f in X, woraus nach Lemma 3.5 sofort $y = x$ und $\mu = \lambda$ folgt. Damit ist der Satz vollständig bewiesen.

Satz 3.7 Unter den Voraussetzungen von Satz 3.6 ist der Eigenvektor x im Intervall $\breve{x} \oplus \hat{X}$ eindeutig, d.h. für einen beliebigen Eigenvektor $y \in \breve{x} \oplus \hat{X}$ der Matrix A folgt $y = x$.

Beweis. Wie im Beweis von Satz 3.6 machen zwecks übersichtlicherer Darstellung kurzfristig von F nach (3.30) und f nach (3.31) Gebrauch. Mit $X = \breve{x} \oplus \hat{X}$ und $\Lambda = \breve{\lambda} \oplus \hat{\Lambda}$ mit dem temporären F aus (3.30) ergibt sich wieder $F(X, \Lambda)' \subsetneq (X, \Lambda)'$ und die Existenz eines $x = \breve{x} + \hat{x}$ und $\lambda = \breve{\lambda} + \hat{\lambda}$ mit $Ax = \lambda x$. Angenommen, es gäbe ein $y \in X$, so daß ein $\mu \in \mathbb{R}$ existiert mit $Ay = \mu y$ (es ist nicht $\mu \in \Lambda$ vorausgesetzt). Wir definieren analog zum Beweis von Satz 3.6 die Funktion $g_y(v) : \mathbb{R} \to \mathbb{R}$ als die Funktion die aus f entsteht, wenn in (3.31) $t = y$ gesetzt wird und nur die letzte Komponente betrachtet wird, also $g_y(v) = f(y, v)_{n+1}$. Offenbar ist $g_y : \Lambda \to \Lambda$ und besitzt daher nach Satz 2.1 einen Fixpunkt σ in Λ. Weiter gilt für beliebiges aber festes $t \in \mathbb{R}^n$

$$f\begin{pmatrix} y \\ v \end{pmatrix} = \begin{pmatrix} t \\ v \end{pmatrix} \iff f\begin{pmatrix} y \\ v \end{pmatrix} - \begin{pmatrix} y \\ v \end{pmatrix} = R \cdot \begin{pmatrix} (A - vE)y \\ ce_k'y - 1 \end{pmatrix} = \begin{pmatrix} t \\ v \end{pmatrix} \iff$$

(3.36)

$$\iff \begin{pmatrix} y - t \\ 0 \end{pmatrix} = R \cdot \begin{pmatrix} (\mu - v)y \\ 0 \end{pmatrix} \iff R^{-1} \cdot \begin{pmatrix} y - t \\ 0 \end{pmatrix} = \begin{pmatrix} (\mu - v)y \\ 0 \end{pmatrix}.$$

Ist T die linke, obere n×n-Matrix in R^{-1} (also R^{-1} ohne die (n+1)-te Zeile und ohne die (n+1)-te Spalte) und ist C die letzte Zeile von R^{-1} ohne $(R^{-1})_{n+1,n+1}$ so folgt mit

(3.37) $$M(t) := \{v \in \mathbb{R} \mid f\begin{pmatrix} y \\ v \end{pmatrix} = \begin{pmatrix} t \\ v \end{pmatrix}\}$$

(3.38) $$v \in M(t) \iff T(y - t) = (\mu - v)y \quad \text{und} \quad C \cdot (y - t) = 0 .$$

Es sei

$$f\begin{pmatrix} y \\ \sigma \end{pmatrix} = \begin{pmatrix} z \\ \sigma \end{pmatrix}.$$

Für $v := z + \eta(y - z)$ mit $\eta := (v - \sigma)/(\mu - \sigma)$ folgt unter der Voraussetzung $\mu \neq \sigma$

$$T(y - v) = (1 - \eta) \cdot T(y - z) = (1 - \eta)(\mu - \sigma)y = (\mu - v)y \quad \text{und}$$
$$C(y - v) = (1 - \eta) \cdot C(y - z) = 0$$

Also kann für jedes $v \in \mathbb{R}$ ein Vektor v gefunden werden, so daß nach (3.38) gilt $v \in M(v)$. Jedes v, für das ein Vektor v gefunden werden kann mit $v \in M(v)$ ist nach der Definition (3.37) von M und der Definition von g_y zugleich Fixpunkt von g_y, insbesondere also auch ein Randpunkt von Λ. Wegen $F(X, \Lambda)' \subsetneq (X, \Lambda)$ kann jedoch kein Fixpunkt von g_y auf dem Rand von Λ liegen. Hieraus folgt $\mu = \sigma \in \Lambda$ und die Existenz eines Fixpunktes (y, μ) von f in (X, Λ). Mit Lemma 3.5 folgt demnach $y = x$ und der Satz ist bewiesen.

Natürlich ist es beim Beweis der beiden vorstehenden Sätze gleichgültig, ob mit den Funktionen f und F nach (3.11) und (3.28) oder mit denen nach (3.31) und (3.30) gearbeitet wird. Zur besseren Übersicht wurde der Beweis mit den letzteren geführt. Soll der Beweis auf die in Defektform gegebenen Funktionen f und F nach (3.11) und (3.28) umgeschrieben werden, müssen nur jeweils Eigenvektoren x und Eigenvektorintervalle X in $\breve{x} + x$ und $\breve{x} + X$ und Eigenwerte λ und Eigenwertintervalle Λ in $\breve{\lambda} + \lambda$ und $\breve{\lambda} + \Lambda$ geändert werden, resp. Ebenso gelten beide Sätze für $\bar{F}$ aus (3.24), da sowohl $\bar{F}$ als auch F nach (3.25) zum gleichen f nach (3.31) führen.

Mit diesen Vorbereitungen kann ein Algorithmus zur eindeutigen Einschließung eines Eigenvektors und Eigenwerts einer reellen Matrix angegeben werden.

Algorithmus 3.7 Einschließung eines Eigenvektors mit zugehörigem Eigenwert einer beliebigen reellen Matrix

1) $\tilde{x}$ und $\tilde{\lambda}$ seinen gleitkommamäßig berechnete Näherungen eines reellen Eigenvektors und zugehörigem Eigenwert einer Matrix $A \in \mathbb{M}_n\mathbb{R}$; sei die k-te Komponente $\tilde{x}_k$ von $\tilde{x}$ die mit dem größten Absoluthetrag;

2) Berechne

$$R \approx \begin{pmatrix} A - \tilde{\lambda}E & -\tilde{x} \\ e_k^T \cdot \|A\| & 0 \end{pmatrix}^{-1} \quad \text{gleitkommamäßig;}$$

3) for i := 1 to n do $y_i^0 := \tilde{x}_i$; $y_{n+1}^0 := \tilde{\lambda}$; l := -1;

repeat l := l+1; Δ := 0;

for i := 1 to n+1 do $y_i^{l+1} := y_i^l$;

for i := 1 to n+1 do

begin $y_i^{l+1} := y_i^{l+1} - R_i \cdot \{(A - y_{n+1}^{l+1}E) \cdot y^{l+1}\}$;

$\Delta := \max\{\Delta, |y_i^{l+1} - y_i^l| / |y_i^l|\}$

end

until $\Delta \geq 10^{-k/2}$ oder $\Delta < 10^{1-t}$;

for i := 1 to n+1 do $\bar{y}_i := y_i^{l+1}$;

for i := 1 to n+1 do $\Delta y_i^0 := -R_i \cdot \{(A - \bar{y}_{n+1}E) \cdot \bar{y}\}$; l := -1;

repeat l := l+1; Δ := 0;

for i := 1 to n+1 do $\Delta y_i^{l+1} := \Delta y_i^l$;

for i := 1 to n+1 do

begin $\Delta y_i^{l+1} := -R_i \cdot \{(A - \bar{y}_{n+1}E - \Delta y_{n+1}^{l+1}E) \cdot (\bar{y} + y^{l+1})\}$;

$\Delta := \max\{\Delta, |\Delta y_i^{l+1} - \Delta y_i^l| / |\Delta y_i^l|\}$

end

until $\Delta \geq 10^{-k/2}$ oder $\Delta < 10^{2-t}$;

seien $\bar{x}$, $\bar{\lambda}$ die (aufgespaltene) Näherung $\bar{y}$ und Δx, $\Delta\lambda$ die (aufgespaltene) Näherung Δy^{l+1}, wenn Δy^{l+1} die letzte Iterierte aus der letzten repeat-Schleife ist;

4) Es sei

$$Z := -R \odot \begin{pmatrix} (A \ominus \bar{\lambda}E \ominus \Delta\lambda E) \odot (\bar{x} \oplus \Delta x) \\ 0 \end{pmatrix};$$

l := -1; $Y^0 := Z$;

5) repeat l := l+1; $Y^{l+1} := Y^l := Y^l \circ \varepsilon$;

$$B := E \ominus R \odot \begin{pmatrix} A \ominus \bar{\lambda}E \ominus \Delta\lambda E & \ominus Y^l \ominus \bar{x} \ominus \Delta x \\ e_k^T \cdot \|A\| & 0 \end{pmatrix} \in \mathbb{IM}_{n+1}\mathbb{R};$$

for i := 1 to n+1 do

if $i \neq k$ then $Y_i^{l+1} := Z_i \oplus B_i \odot Y^l$

until $\{Y^{l+1} \subsetneq Y^l$ bis auf die k-te Komponente$\}$ oder

$\{(l > 2 \cdot \lceil \log_{10} \rceil + 4) =: \text{bool}\}$;

if bool then stop; $Y_k^{l+1} := \tilde{x}_k$;

6) Mit

$$\begin{pmatrix} X \\ \Lambda \end{pmatrix} := \begin{pmatrix} \bar{x} \\ \bar{\lambda} \end{pmatrix} \oplus \begin{pmatrix} \Delta x \\ \Delta\lambda \end{pmatrix} \oplus Y^{l+1}$$

gibt es genau einen Eigenvektor x und genau einen Eigenwert λ von A mit $x \in X$ und $\lambda \in \Lambda$ und es gilt $Ax = \lambda x$; λ hat die Vielfachheit 1;

Der Index i bedeutet bei einer Matrix wie immer die i-te Zeile der Matrix, bei einem Vektor die i-te Komponente des Vektors. In Schritt 3 des Algorithmus 3.7 wird das Abbruchkriterium wie in Schritt 3 des Algorithmus 2.1 verwendet.

"Bis auf die k-te Komponente" in Schritt 4 heißt wie vorhin, daß für die k-te Komponente die echte Inklusion nicht abgeprüft wird. Für das Abbruchkriterium

in Schritt 4 und ε gelten die Bemerkungen des letzten Abschnitts entsprechend. In Schritt 4 braucht die Intervallmatrix B natürlich nicht jedesmal neu berechnet zu werden. Es ändert sich jeweils nur die letzte Spalte. Bei der Bildung von $\ominus Y^1 \ominus x \ominus \Delta x$ in der Berechnung von B ist zu beachten, daß nur die ersten n Komponenten von Y^1 mit einbezogen werden. Aus Gründen der besseren Leserlichkeit wurde auf eine Umbenennung von Y^1 verzichtet. Bei der Berechnung der letzten Spalte S von B ist es naheliegend, zunächst $V := \ominus Y^1 \ominus \bar{x} \ominus \Delta x$ intervallmäßig zu berechnen und dann die Komponenten von S durch $S := e_{n+1}' \ominus R \oplus V$ zu gewinnen. Interessanterweise ist es auf der Rechenanlage günstiger, zunächst $T := R \oplus (\ominus \bar{x} \ominus \Delta x)$ mittels des langen Akkumulators zu berechnen, dann die letzte Spalte von B nach $S := e_{n+1}' \ominus T \oplus R \odot Y^1$ zu ermitteln. Obwohl nach der Subdistributivität

$$e_{n+1}' \ominus R \odot (\ominus Y^1 \ominus \bar{x} \ominus \Delta x) \subseteq e_{n+1}' \ominus R \odot (\ominus \bar{x} \ominus \Delta x) \oplus R \odot Y^1 \tag{3.39}$$

ist und i.a. das Gleichheitszeichen nicht gilt, wird die rechte Seite von (3.39) durch die Verwendung des langen Akkumulators auf dem Rechner genauer, da der zweite Summand der rechten Seite gleich dem auf dem Rechner engstmöglichen Intervall um den exakten Wert wird.

Bei der Formulierung des Algorithmus mit den hier beschriebenen Verbesserungen in FORTRAN ist es nicht gleichgültig, an welchen Stellen man nur einfachlang rechnet, wo es sinnvoll wird, doppeltlang zu rechnen oder wo der Einsatz des langen Akkumulators gerechtfertigt oder notwendig wird. Die in dieser Hinsicht mit Abstand optimalste Version des Algorithmus ist im Anhang als Quelltext angegeben.

Es sei erwähnt, daß Algorithmus 3.7 in begrenztem Maße auch zur Bestimmung von Ausschließungsintervallen für Eigenwerte oder Eigenvektoren verwendet werden kann. Hat einmal eine Einschließung für z.B. einen Eigenwert stattgefunden, ist also $\lambda \in \Lambda$ nachgewiesen, kann Schritt 5 des Algorithmus mit einem $\Lambda^* \supseteq \Lambda$ wiederholt werden. Tritt wieder Einschließung ein, ist bereits bewiesen, daß in $\Lambda^* \setminus \Lambda$

kein Eigenwert der Matrix liegt. Die Methode ist allerdings nur eingeschränkt brauchbar, da für zu große Λ^* keine Einschließung mehr erzielt werden kann. Das ist spätestens der Fall, wenn Λ^* zwei Eigenwerte enthält. Für symmetrische Matrizen wurden sehr gute Ergebnisse erzielt, d.h. zwischen den Ausschließungsintervallen wenig Spielraum war. Das Verfahren ist überdies insofern reizvoll, da der Mehraufwand pro Intervall nur

$$2n^2 + s \cdot 4n^2$$

bei s-maliger Ausführung von Schritt 5 beträgt.

Bei der Berechnung von R nach (3.19) stellte sich heraus, daß sehr große Fehler auftreten können, wenn die Komponenten von A sehr groß sind. Dann sind nämlich gar für eine gut konditionierte (!) Matrix A die Komponenten von A^{-1} relativ klein, insbesondere also gegen 1 aus e_k'. Wie vorhin festgestellt, kann e_k' durch $\zeta \cdot e_k'$ für beliebiges $0 \neq \zeta \in \mathbb{R}$ ersetzt werden. Daher erscheint es sinnvoll, $\zeta := \|A\|_1$ zu setzen. Tests haben gezeigt, daß nach dieser Änderung die beobachteten Ungenauigkeiten nicht mehr auftraten.

Die Funktion f in (3.11) ist bereits als "Defekt-Funktion" definiert. So wird mit Ω nicht der Eigenwert λ und der Eigenvektor x eingeschlossen, sondern deren Defekt, so daß sich (3.28) ergibt. Wie bereits in LGLG/1 wurde vorteilhaft von dieser Variante Gebrauch gemacht. Dies schlägt sich in den Ergebnisintervallen nieder, deren relativer Fehler sich um einige Zehnerpotenzen verbessert gegenüber der einfachen Form des Algorithmus.

Der Rechenaufwand α für Algorithmus 3.7 ergibt sich zu

$$\alpha = 2n^3 + 7n^2 + r \cdot 2n^2 + s \cdot 6n^2, \tag{3.40}$$

wenn in Schritt 3 r-mal (insgesamt) gleitkommamäßig und in Schritt 5 s-mal intervallmäßig iteriert wird.

Wie bereits oben bemerkt, ergaben sich teilweise sehr schlechte Gleitkommanäherungen. Dies kann in extremen Fällen dazu führen, daß der Algorithmus nicht erfolgreich beendet werden kann. In diesem Fall liegt es nahe, den Algorithmus erneut mit den in (3.23) berechneten, verbesserten Näherungen anzuwenden. In Schritt 3 des Algorithmus 3.7 wurde das vereinfachte Newton-Verfahren angewandt. Ein Wiederholen des Algorithmus in der gerade beschriebenen Weise läuft auf ein nicht-vereinfachtes Newton-Verfahren "mit Unterbrechungen" hinaus. Zunächst werden von einer Näherung aus ein paar vereinfachte Newton-Schritte durchgeführt. Kann mit der erzielten Gletikommanäherung Einschließung erzielt werden, ist man fertig (evtl. können die Lösungen mittels Durchschnittsbildung verfeinert werden). Andernfalls wird eine neue Steigung berechnet, wieder vereinfachte Newton-Schritte ausgeführt und so fort. Die Praxis lehrt, daß eine mehr als zweimalige Wiederholung des Prozesses nicht sinnvoll ist. Der Aufwand ist in diesem Fall $\sim 4n^3$. Schaut man sich die Defektiteration in (3.21) genauer an, läßt sich jeder Schritt auf die Lösung eines linearen Gleichungssystems mit variierenden rechten Seiten zurückführen. Soll nur die Defektiteration durchgeführt werden, kommt man also bereits mit einem Aufwand von $\sim \frac{1}{3}n^3$ aus. Daher liegt es nahe, zunächst eine reelle Defektiteration für Eigenwert und Eigenvektor und deren Defekte durchzuführen und mit den neuen Näherungen Algorithmus 3.7 zu starten. Der Gesamtaufwand beträgt in diesem Fall $\sim \frac{4}{3}n^3$. Man hat den Vorteil einer genaueren Gleitkommanäherung und damit letztlich auch schärferer Lösungsintervalle und der Mehraufwand gegenüber der einfachen Version beträgt höchstens 16 %. In praxi ist diese Zahl noch kleiner, so daß man insgesamt ca. mit einem Zeitfaktor von 1.07 rechnen kann. Dieser Wert ist insbesondere im Vergleich mit der Verdopplung des Aufwands nach obigem Verfahren interessant bei gleichem Gewinn.

3.e) Numerische Ergebnisse

Um eine ungefähre Anschauung der Wirkungsweise des Algorithmus 3.7 zu bekommen, betrachten wir zunächst folgendes einfache Beispiel:

$$A = \begin{pmatrix} 0 & 985 & 0 \\ 0 & 0 & 985 \\ -2786 & 1970 & 1393 \end{pmatrix} .$$

Es handelt sich um die Frobenius-Matrix des Polynoms

$$(x^2 - 2) \cdot (985x - 1393) ,$$

wobei die Koeffizienten durch einen geeigneten Faktor ganzzahlig gemacht wurden. 1393/985 ist ein Näherungsbruch für $\sqrt{2}$ mit einem relativen Fehler von nur $2.57_{10}-7$. Die Eigenwerte und Eigenvektoren der Matrix sind

$$\lambda_1 = -985\sqrt{2} \approx -1393.00036;\ \lambda_2 = 1393;\ \lambda_3 = 985\sqrt{2} \approx 1393.00036;$$

$$v_1 \approx (0.5\ ,\ -0.70710678\ ,\ 1);$$

$$v_2 \approx (0.50000026\ ,\ 0.70710696\ ,\ 1);$$

$$v_3 \approx (0.5\ ,\ 0.70710678\ ,\ 1);$$

Es ist $\lambda_3 - \lambda_2 \approx 36_{10}-8$ und $v_2 - v_3 = (26_{10}-8\ ,\ 16_{10}-8\ ,\ 0)$. Algorithmus 3.7 wurde mit verschiedenen Gleitkommanäherungen $\tilde{x}$, $\tilde{\lambda}$ gerechnet. Diese wurden nicht gemäß Schritt 3 des Algorithmus gleitkommamäßig iteriert. Es wurde also direkt $(\bar{x}, \bar{\lambda}) := (\tilde{x}, \tilde{\lambda})$ gesetzt mit $(\Delta x, \Delta\lambda) := (0, 0)$ und in Schritt 4 fortgefahren. Auf diese Weise wurde also direkt die Konvergenz oder Divergenz der Intervallaufblähung von Schritt 5 mit den in Tabelle 3.1 angegebenen Startwerten getestet. Aus den erzeilten Ergebnissen ersieht man, daß der Algorithmus empfindlich auf Störungen von (v_2, λ_2) und (v_3, λ_3) reagiert was auch nicht weiter verwundert, da hier äußerst eng benachbarte Fixpunkte vorliegen. Selbst völlige Verfälschung von (v_1, λ_1) macht jedoch nichts aus.

$\tilde{x} = v_2$	$\tilde{\lambda} = \lambda_2$	konvergent
$\tilde{x} = v_3$	$\tilde{\lambda} = \lambda_3$	konvergent
$\tilde{x} = v_2$	$\tilde{\lambda} = \lambda_3$	divergent
$\tilde{x} = v_3$	$\tilde{\lambda} = \lambda_2$	divergent
$\tilde{x} = v_3$	$\tilde{\lambda} = (\lambda_2 + \lambda_3)/2$	divergent
$\tilde{x} = v_3$	$\tilde{\lambda} = \lambda_3 - (\lambda_3 - \lambda_2)/3$	konvergent
$\tilde{x} = v_3$	$\tilde{\lambda} = \lambda_3 + (\lambda_3 - \lambda_2)$	konvergent
$\tilde{x} = v_3$	$\tilde{\lambda} = \lambda_3 + 3(\lambda_3 - \lambda_2)$	divergent
$\tilde{x} = v_1$	$\tilde{\lambda} = \lambda_1$	konvergent
$\tilde{x} = v_1$	$\tilde{\lambda} = 8 \cdot \lambda_1$	konvergent
$\tilde{x} = (1, -0.67, 0.33)$	$\tilde{\lambda} = -1393$	konvergent
$\tilde{x} = (1, -1, 1)$	$\tilde{\lambda} = -1000$	konvergent

Tabelle 3.1 Abhängigkeit vom Startwert

Das Verfahren (3.9) ist höchstens bis zum Grad 5 brauchbar. Es reagiert empfindlich auf nur wenig schlechte Kondition. Das liegt daran, daß es sich hier um ein "direktes" Verfahren handelt. Es zeigt sich die klare Überlegenheit der Defektverfahren. Mittels Algorithmus 3.7 können übrigens auch die Eigenwerte und Eigenvektoren einer Matrix vom Rang n-1 eingeschlossen werden. In diesem Fall erhält man für den Eigenwert 0 typisch eine Einschließung $[-2_{10}-26, +2_{10}-26]$. Weiter wurden in Bezug auf das Eigenproblem schlecht konditionierte Matrizen $A^{-1}DA$ betrachtet wie auf Seite 112 beschrieben. Es zeigt sich deutlich, daß die reelle Näherung sehr schlecht werden kann. Nimmt man für den Grad 6 etwa für A eine PASCAL*-Matrix (Definition siehe Seite 86 oben), sind die Gleitkommanäherungen durchschnittlich nur auf 2 Dezimalen genau. Die Ergebnisse, die für den Grad 10 mittels Matrizen Q-2 (Definition siehe Seite 24, f) für A erzielt wurden, sind in Tabelle 3.2 wiedergegeben. Die Eigenwerte wurden für die zwei Beispiele zufällig aus dem Intervall [-1,1] und [1,10] gewählt. Wie man sieht, waren die Gleitkommanäherungen teilweise nur auf eine Stelle genau und trotzdem wurde die Lösung in beiden Fällen auf mindestens 15 Dezimalen,

	max. relativer Fehler für $\tilde{x}$	max. relativer Fehler für $\tilde{\lambda}$	rel. Fehler der Lösungsintervalle maximal	rel. Fehler der Lösungsintervalle durchschn.
Eigenwerte $\in [-1,1]$	$5.4_{10}-2$	$5.7_{10}-3$	$1.0_{10}-15$	$1_{10}-18$
Eigenwerte $\in [1,10]$	$7.9_{10}-2$	$2.1_{10}-4$	$1.3_{10}-15$	$1_{10}-18$

Tabelle 3.2 Schlecht konditioniertes Eigenproblem

durchschnittlich sogar 18 Stellen eingeschlossen. Es sei vermerkt, daß für das erste Beispiel 5 und im zweiten 4 von 10 Eigenwert/Eigenvektor-Paaren nicht eingeschlossen werden konnten, wenn die Steigung nicht nach dem Vorschlag von Seite 126 korrigiert wurde. In der folgenden Tabelle 3.3 sind Beispiele gezeigt, in denen die Eigenwerte teilweise recht eng beieinander liegen. In der linken Spalte ist

Eigenwerte (u.a.)	Grad	maximaler relativer Fehler der Lösungsintervalle
$1, 1+3_{10}-7, 1+6_{10}-7$	10	$7_{10}-16$
$-5, -5+5_{10}-6, -5+_{10}-5,$ $-2-_{10}-3, -2, -2+_{10}-6$	10	$6_{10}-16$
$\lambda_i = 1 + (i-1) \cdot 0.01$	20	$2_{10}-17$

Tabelle 3.3 Eng benachbarte Eigenwerte

jeweils (ein Teil) der Eigenwerte angegeben, es folgt der Grad der Matrix und der maximale relative Fehler der Lösungsintervalle. Die zugrundeliegenden Matrizen sind vom Typ $A^{-1}DA$ mit einer Zufallsmatrix A, $|a_{ij}| \leq 1$. Wie man sieht, werden alle Eigenwerte und Eigenvektoren ohne Schwierigkeiten getrennt.

Bei der Berechnung der Eigenwerte der Hilbert- und Pascal-Matrizen zeigt sich wieder (siehe Tabelle 3.4), wie ungenau die Gleitkommanäherung sein kann. Die Abkürzung n.a. für den relativen Fehler eines Eigenwerts der Hilbert-7 × 7-Matrix

Matrix	Grad	max. relativer Fehler für $\tilde{x}$	max. relativer Fehler für $\tilde{\lambda}$	rel. Fehler aller Lösungsintervalle maximal	rel. Fehler aller Lösungsintervalle durchschn.
Hilbert	5	1.8_{10}-4	4.5_{10}-3	1.2_{10}-18	4_{10}-18
	6	2.6_{10}-3	1.7_{10}-2	1.2_{10}-18	4_{10}-18
	7	2.0_{10}-2	n.a.	1.1_{10}-18	4_{10}-18
	8	1.3	43	4.1_{10}-10	1_{10}-17
Pascal	6	1.0_{10}-5	1.5_{10}-5	1.2_{10}-18	4_{10}-18
	7	1.5_{10}-5	7.7_{10}-4	1.2_{10}-18	4_{10}-18
	8	1.5_{10}-3	8.2_{10}-4	1.1_{10}-18	4_{10}-18
	9	3.0_{10}-3	6.1_{10}-3	1.1_{10}-18	4_{10}-18

Tabelle 3.4 Relativer Fehler der Gleitkommanäherungen

bedeutet nicht angebbar. Der gerundete Wert für den Eigenwert war in diesem Fall 1.259_{10}-3, die Gleitkommanäherung aber -2.17_{10}-3, so daß nicht einmal das Vorzeichen richtig bestimmt wurde! Obwohl die Hilbert-Matrizen symmetrisch und positiv definit sind ergibt sich ein derart katastrophaler Fehler.

Die Anzahl der Gleitkommaiterationen betrug in allen Beispielen durchschnittlich 4, maximal jedoch 10 Iterationen. In fast allen Fällen wurden 2 Intervalliterationen durchgeführt und nur äußerst selten bis zu 5. Es zeigt sich wieder, daß je mehr gleitkommamäßig voriteriert wird desto weniger muß in Schritt 5 von Algorithmus 3.7 intervallmäßig iteriert werden. Insbesondere für die Hilbert-8×8-Matrix wäre der maximale relative Fehler der Lösungsintervalle um Größenordnungen besser ausgefallen (siehe Tabelle 3.4), wenn mehr Gleitkommaiterationen durchgeführt worden wären. Hier zeigt sich (in diesem speziellen Fall) eine Schwäche des Abbruchkriteriums in Schritt 3 von Algorithmus 3.7. Dies ist auf die allgemein bekannte Tatsache zurückzuführen, daß keine allgemein gültigen Gleitkommastrategien gefunden werden (siehe etwa Pivotisierung).

Abschließend noch ein Beipiel vom Grad 50 einer Zufallsmatrix A mit $|a_{ij}| \leq 1$. Die Matrix ist nicht symmetrisch und es ergab sich Tabelle 3.5. Die Angaben bezüglich des Fehlers beziehen sich (wie immer) auf alle Lösungsintervalle für sämtliche Eigenwerte und Eigenvektoren.

relativer Fehler der Lösungsintervalle für die Eigenwerte maximal	für die Eigenwerte durchschn.	für die Eigenvektoren maximal	für die Eigenvektoren durchschn.
7_{10}-17	2_{10}-18	3_{10}-16	2_{10}-18

Tabelle 3.5 Beispiel einer 50×50-Matrix

Der FORTRAN-Quelltext des auf der UNIVAC 1108 implementierten Algorithmus ist im Anhang wiedergegeben. Es sei noch darauf hingewiesen, daß sich Algorithmus 3.7 leicht sowohl auf Intervall-Matrizen als auch auf komplexe Matrizen und die Einschließung komplexer Eigenwerte und Eigenvektoren erweitern läßt. Die Beweise der Sätze 3.6 und 3.7 übertragen sich entsprechend.

Appendix

Als erste Idee für eine Realisierung einer Rechnerarithmetik könnte man den Versuch einer treuen Abbildung der reellen Zahlen auf die Gleitkommazahlen unternehmen. Sofort wird man an einen Homomorphismus denken. Das Ideale wäre natürlich ein ordnungstreuer algebraischer Homomorphismus. Weiter könnte jedoch noch von der Forderung der Ordnungstreue abgesehen und nur verlangt werden, daß das Bild "in der Nähe" des Urbildes liegen soll, ohne dies näher spezifizieren zu wollen. Wir werden jedoch sehen, daß nicht einmal dies realisierbar ist.

Satz. Gegeben sei eine endliche Menge T mit einer Operation $\boxplus$. Weiter sei $\sigma : \mathbb{R} \to T$ ein algebraischer Homomorphismus, d.h.

$$\bigwedge_{a,b \in \mathbb{R}} : \quad \sigma(a+b) = \sigma(a) \boxplus \sigma(b) ,$$

wobei $+$ die Addition in $\mathbb{R}$ bezeichnet. Dann gilt

$$\bigwedge_{a,b \in \mathbb{R}} \bigvee_{x \in [a,b]} : \sigma(x) = \sigma(o) .$$

Beweis. Gegeben seien $n+1$ verschiedene reelle Zahlen $c_1, \ldots, c_n, c_{n+1}$ aus dem Intervall $[a,b]$ mit $n = |T| < \infty$. Mit eventueller Umnumerierung gilt dann

$$\sigma(c_1) = \sigma(c_2) .$$

Demnach ist $\sigma(c_2 - c_1) = \sigma(c_2) \boxplus \sigma(-c_1) = \sigma(c_1) \boxplus \sigma(-c_1) = \sigma(c_1 - c_1) = \sigma(o)$. Mit $c = c_2 - c_1$ folgt

$$\sigma(2c) = \sigma(c+c) = \sigma(c) \boxplus \sigma(c) = \sigma(o) \boxplus \sigma(o) = \sigma(o+o) = \sigma(o) ,$$

und durch vollständige Induktion

$$\sigma(k \cdot c) = \sigma(o) \qquad \text{für alle } k \in \mathbb{N} .$$

Wegen $c = c_2 - c_1 \leq b - a$ gibt es daher ein $l \in \mathbb{N}$ mit $l \cdot c \in [a,b]$ und $\sigma(l \cdot c) = \sigma(o)$, q.e.d.

Der Beweis kann für wesentlich allgemeinere Strukturen geführt werden. Hier geht wesentlich die Archimedizität von $\mathbb{R}$ ein.

Aus dem Satz folgt, daß für einen Homomorphismus $\sigma : \mathbb{R} \to T$ in jedem (noch so kleinen) Intervall aus $\mathbb{R}$ ein Element existiert, daß zur Äquivalenzklasse der o gehört. D.h., die Menge aller $x \in \mathbb{R}$, die das gleiche Bild wie o haben, ist dicht in $\mathbb{R}$. Damit scheint jede sinnvolle Ordnung ausgeschlossen.

Man könnte versuchen, durch eine geeignete Definition von σ wenigstens die Assoziativität von $\oplus$ zu erhalten. Wir werden sehen, daß selbst dies praktisch ausgeschlossen ist.

Gegeben sei eine Menge von "Rechnerzahlen" $T \subseteq \mathbb{R}$, d.h.

$$T := \{t \mid t \in \mathbb{R} \text{ ist darstellbar auf dem Rechner}\} .$$

In T seien Operationen $\boxplus$ und $\boxminus$ definiert :

$$\boxplus , \boxminus : T \times T \to T .$$

Weiter sei $\phi : \mathbb{R} \to T$ eine Abbildung (oder Rundung), die mit $\boxplus$ und $\boxminus$ verträglich ist, also

$$\phi(a * b) = a \circledast b \quad \text{für alle } a,b \in T \text{ und } * \in \{+,-\} .$$

+,- sind dabei die gewöhnliche Addition und Subtraktion in $\mathbb{R}$. Sind $a,b \in T$ mit $a < b$ zwei aufeinanderfolgende Zahlen in T, also

$$(1) \qquad r \in \mathbb{R} : \quad a < r < b \quad \Rightarrow r \notin T ,$$

so folgt mit $\mu = (a+b)/2$ und $\gamma = (a-b)/2$, $\delta = (b-a)/2$

$$\phi(\mu) = a \Rightarrow (b \boxminus \delta) \boxplus \delta = \phi(b-\delta) \boxplus \delta = \phi(\mu) \boxplus \delta =$$
$$a \boxplus \delta = \phi(a+\delta) = \phi(\mu) = a \neq b$$
$$\text{und} \quad \phi(\mu) = b \Rightarrow (a \boxminus \gamma) \boxplus \gamma = \phi(a-\gamma) \boxplus \gamma = \phi(\mu) \boxplus \gamma =$$
$$b \boxplus \gamma = \phi(b+\gamma) = \phi(\mu) = b \neq a$$

Wird also nur <u>einmal</u> der Mittelpunkt zweier aufeinanderfolgender Zahlen $a,b \in T$ auf eine der Grenzen a oder b durch ϕ abgebildet (was recht naheliegend ist) und ist $\pm(a-b)/2 \in T$, gilt bereits

$$(2) \qquad \bigwedge a \in T,\ \delta \in \mathbb{R} : \quad (a \boxminus \delta) \boxplus \delta = a$$

<u>nicht</u> mehr. In üblichen Gleitkommasystemen ist $\gamma, \delta \in T$ praktisch immer erfüllt.

Selbst wenn der Mittelpunkt μ nicht auf eine der Grenzen a oder b durch ϕ abgebildet wird, soll gezeigt werden, daß die Gültigkeit von (2) auf in der Praxis kaum brauchbare Strukturen zu führen scheint.

Angenommen, es sei (2) erfüllt und

$$(3) \qquad \phi(\mu) = c \quad \text{mit } c < a \quad \text{o.B.d.A.}$$

Es ist zunächst für alle $f \in T$ und $\xi \in T$ mit $\phi(f+\xi) = f$:

$$\phi(f-\xi) = \phi(\phi(f+\xi) - \xi) = f$$

nach (2). Mit $f = c$, $\varepsilon = a - c$ und $\xi = \varepsilon + \delta$ folgt dann

$$(4) \qquad c = \phi(\mu) = \phi(c+\xi) = \phi(c-\xi) = \phi(c-\varepsilon-\delta) .$$

Für beliebiges $d \in T$ und $\Delta \in T$ mit $\phi(d-\Delta) = c$ folgt

$$(5) \qquad \phi(c+\Delta) = \phi(\phi(d-\Delta) + \Delta) = d$$

wieder nach (2). Unter der (naheliegenden) Annahme $\Delta \in T$ gilt dann für $d = \delta$ und $\Delta = -c-\varepsilon$ zunächst $\phi(d-\Delta) = \phi(\delta+c+\varepsilon) = \phi(c+\xi) = c$ nach (4), also folgt nach (5)

$$\delta = \phi(c+\Delta) = \phi(c-c-\varepsilon) = \phi(-\varepsilon) .$$

Wegen $c < a$ wird also die negative Zahl $-\varepsilon$ durch ϕ auf die positive Rechnerzahl δ abgebildet.

Die obigen Ausführungen zeigen, daß bereits mit geringsten Voraussetzungen zu erkennen ist, daß weder ein Homomorphismus der reellen Zahlen auf ein System von Rechnerzahlen noch eine assoziative Verknüpfung zwischen Rechnerzahlen

möglich ist. Dabei ist noch zu betonen, daß (2) nur eine Vorstufe zum Assoziativgesetz darstellt. Somit ist davon auszugehen, daß man von der algebraischen Struktur der "Rechnerzahlen" (siehe Kulisch 1) nicht viel mehr erwarten zu können scheint.

LITERATUR

Alefeld/Apostolatos	Auflösung nichtlinearer Gleichungssysteme mit zwei Unbekannten, Bericht des Rechenzentrums u. Inst. f. Angew. Math., Universität Karlsruhe (1967)
Alefeld/Herzberger	Einführung in die Intervallrechnung, Bibl. Inst., Mannheim, Wien, Zürich, 1974
Bohlender	Floating-point computation of functions with maximum accuracy, IEEE Comp. Soc., Symp. on Comp. Arithmetic, Dallas 1975
Collatz	Funktionalanalysis und numerische Mathematik, Springer-Verlag 1968
Grüner	Fehlerschranken für lineare Gleichungssysteme, Computing, Supplementband 1976
Kaucher/Klatte	Zum langen Akkumulator, Bericht des Inst. f. Angew. Math., Universität Karlsruhe (1977)
Kaucher/Rump	Generalized Iteration Methods for Bounds of the Solution of Fixed Point Operator-Equations, erscheint in Computing
Knuth	The Art of Computer Programming, Vol. 2, Reading, Mass., Addison-Wesley Publ. Comp. Inc. (1969)
Kulisch 1	Grundlagen des numerischen Rechnens, Bibl. Inst., Mannheim, Wien, Zürich, 1976
Kulisch 2	Grundzüge der Intervallrechnung, in 'Oberblicke Mathematik' 2, Bibl. Inst., Mannheim (1969)
McClellan	The exact solution of systems of linear equations with polynomial coefficients, J. Ass. Comp. Mach., v. 20, 1973
Rump 1	Untersuchungen verschiedener Rechnerarithmetiken, Bericht des Inst. f. Angew. Math., Universität Karlsruhe (1978)
Rump 2	Zur Rückführung nicht mehr benötigten Speicherplatzes in PASCAL, erscheint in elektronische Rechenanlagen
Rump/Kaucher	Small Bounds for the Solution of Systems of Linear Equations, erscheint im Supplementband Computing (1979)

Wongwises — Experimentelle Untersuchungen zur numerischen Auflösung von linearen Gleichungssystemen mit Fehlererfassung, Interner Bericht 75/1, Inst. f. Prakt. Math., Universität Karlsruhe

Zielke — ALGOL-Katalog Matrizenrechnung, Oldenbourg Verlag, München, Wien 1972

Tabellen

Sätze, Lemmata und Definitionen

Algorithmen

Anhang

FORTRAN - und Assembler - Unterprogramme

```
*********
*  LGL  *
*********
```

Die Subroutine LGL berechnet Schranken fuer die Loesung eines linearen Gleichungssystems A * x = b mit N Unbekannten. Die rechte Seite b kann mit Ungenauigkeiten behaftet sein, d.h. in einem Intervall bl <= b <= br liegen. Von der Matrix A wird angenommen, dass sie exakt gegeben ist. Man beachte, dass die Matrix A des Gleichungssystems diejenige Matrix ist, die tatsaechlich im Array A steht. Durch Rundungsfehler beim Einlesen koennen Fehler auftreten. Man vergewissere sich, dass dies nicht der Fall ist. Anderenfalls ist die Variante LGLU bzw. LGLU2 der Prozedur LGL zu verwenden (siehe dazu die entsprechende Beschreibung).

Das Programm LGL ist als FORTRAN-Unterprogramm aufrufbar.

Anwendung des Programms LGL:

1) Aufruf

```
CALL LGL (N, A, WK, GL, GR, NRS)
```

2) Parameter

Name	I	vereinbart als	I	Typ	I	E/A
N	I	INTEGER N	I	ganzzahlige Variable	I	E
A	I	REAL A(N,N)	I	N*N-Matrix	I	E
WK	I	REAL WK(N,6)	I	Feld der Laenge >= 6*N	I	
GL,GR	I	DOUBLE PRECISION	I	doppeltlange Felder	I	E/A
	I	GL(N),GR(N)	I	der Laenge N	I	
NRS	I	LOGICAL NRS	I	logische Variable	I	E

3) Beschreibung

Ist die rechte Seite b exakt gegeben, so setze GL = GR = b (GL und GR duerfen jedoch nicht dieselbe Variable sein). Sind fuer die rechte Seite nur Schranken bl und br mit bl <= b <= br bekannt, so setze GL = bl und GR = br. Die Loesung des Gleichungssystems A * x = b oder die Loesungsmenge S = { x | A * x = b und bl <= b <= br } liegt nach erfolgreicher Abarbeitung des Algorithmus zwischen GL und GR, es gilt also GL <= x <= GR fuer alle x aus S. Konnte der Algorithmus nicht erfolgreich beendet werden, gilt GL = (1, ..,1) und GR = (-1, ..,-1), erkennbar an GL > GR. In diesem Fall rufe man die Routine LGL2 auf (die Matrix A ist dann ein Feld doppelter Genauigkeit, siehe Beschreibung von LGL2). WK ist der Arbeitsspeicher des Algorithmus, ein Feld mindestens der Laenge 6*N. Ist nur eine rechte Seite b zu berarbeiten, ist NRS = FALSE. zu setzen. Fuer den Fall, dass mehrere rechte Seiten zu bearbeiten sind, rufe man LGL das erste Mal mit NRS = FALSE. auf, danach (ab der zweiten rechten Seite) mit NRS = TRUE. Dadurch wird erheblich Rechenzeit eingespart. Bei der ersten Abarbeitung werden Daten in die temporaeren Dateien mit den Nummern 25 und 26 gespeichert, die beim ersten Aufruf von LGL nicht assigniert sein duerfen. Waehrend der Loesung von A * x = b fuer weitere rechte Seiten mittels LGL(N, A, WK, GL, GR, TRUE.) darf kein Aufruf von LGL mit einer anderen Matrix A erfolgen. Die Matrix A ist nach Abarbeitung des Algorithmus in jedem Fall weiter verfuegbar.

Erfahrungsgemaess bricht der Algorithmus erst bei Konditionszahlen groesser als 10 ** 8 mit GL > GR ab.

Durch die erfolgreiche Abarbeitung (was aus GL <= GR folgt) von LGL wird gleichzeitig b e w i e s e n , dass

- die Matrix A nicht singulaer ist,
- das Gleichungssystem A * x = b fuer jedes b mit GL <= b <= GR eindeutig loesbar ist,
- fuer alle x mit A * x = b und GL <= b <= GR nach der Abarbeitung GL <= x <= GR gilt.

Der Algorithmus arbeitet im wesentlichen mit einfacher Genauigkeit. Die doppelte Laenge von GL und GR wird benoetigt, um dem Benutzer die hohe Genauigkeit der Ergebnisse zugaenglich zu machen. Ein Richtwert fuer die CPU-Zeit ist ca. 13 Sekunden fuer 50 Unbekannte und ca. 88 Sekunden fuer Gleichungssysteme mit 100 Unbekannten.

****
* LGLU *
****

Fuer die FORTRAN-Routine LGLU gilt genau die Beschreibung von LGL mit folgendem Unterschied

- der Aufruf erfolgt durch CALL LGLU (N,AL,AR,WK,GL,GR,NRS)
- fuer die Matrix A sind nur untere und obere Schranken bekannt, d.h. AL <= A <= AR (komponentenweise)
- Fuer die Loesungsmenge gilt S = < x : A * x = b und AL <= A <= AR und GL <= b <= GR >
- der Algorithmus arbeitet erfahrungsgemaess fuer
  norm(AR-AL) / max (norm(AR) , norm(AL)) <= 1/cond(A), (*)
  d.h. ist die Konditionszahl z.b. 10**5, darf der maximale relative Fehler der Eingabe(intervall)matrix nicht groesser als 10**-5 sein. ist (*) verletzt, so wird i.a. der Algorithmus nicht erfolgreich beendet, da wahrscheinlich ein singulaeres A mit AL <= A <= AR existiert
- Ein Richtwert fuer die CPU-Zeit ist ca. 15 Sekunden fuer 50 Unbekannte und ca. 110 Sekunden fuer 100 Unbekannte.

Durch die erfolgreiche Abarbeitung (was aus GL <= GR folgt) wird b e w i e s e n , dass

- fuer alle A mit AL <= A <= AR gilt: A ist nicht singulaer
- das Gleichungssystem A * x = b fuer jedes A mit AL <= A <= AR und GL <= b <= GR eindeutig loesbar ist
- fuer alle x mit A * x = b und AL <= A <= AR und GL <= b <= GR nach der Abarbeitung GL <= x <= GR gilt

LGL2

Fuer die FORTRAN-Routine LGL2 gilt genau die Beschreibung von LGL mit folgendem Unterschied:

- der Aufruf erfolgt durch CALL LGL2 (N,A,WK,GL,GR,NRS)
- die N*N-Matrix A hat doppelte Genauigkeit
- der Algorithmus arbeitet erfahrungsgemaess fuer Konditionszahlen bis 10**16
- Ein Richtwert fuer die CPU-Zeit ist ca. 17 Sekunden fuer 50 Unbekannte und ca 120 Sekunden fuer 100 Unbekannte.

Wieder wird durch die erfolgreiche Abarbeitung (was aus GL <= GR folgt) b e w i e s e n , dass

- die Matrix A nicht singulaer ist,
- das Gleichungssystem A * x = b fuer jedes b mit GL <= b <= GR eindeutig loesbar ist,
- fuer alle x mit A * x = b und GL <= b <= GR nach der Abarbeitung GL <= x <= GR gilt.

LGLU2

Fuer die FORTRAN-Routine LGLU2 gilt genau die Beschreibung von LGL mit folgendem Unterschied:

- der Aufruf erfolgt durch CALL LGLU2 (N,AL,AR,WK,GL,GR,NRS)
- Beide N*N-Matrizen AL und AR haben doppelte Genauigkeit
- fuer die Matrix A sind nur untere und obere Schranken bekannt, d.h. AL <= A <= AR (komponentenweise)
- Fuer die Loesungsmenge gilt S = < x : A * x = b und AL <= A <= AR und GL <= b <= GR >
- der Algorithmus arbeitet erfahrungsgemaess fuer
  norm(AR-AL) / max (norm(AR) , norm(AL)) <= 1/cond(A), (*)
  d.h. ist die Konditionszahl z.b. 10**10, darf der maximale relative Fehler der Eingabe(intervall)matrix nicht groesser als 10**-10 sein. ist (*) verletzt, so wird i.a. der Algorithmus nicht erfolgreich beendet, da wahrscheinlich ein singulaeres A mit AL <= A <= AR existiert
- Ein Richtwert fuer die CPU-Zeit ist ca. 19 Sekunden fuer 50 Unbekannte und ca. 130 Sekunden fuer 100 Unbekannte

Durch die erfolgreiche Abarbeitung (was aus GL <= GR folgt) wird b e w i e s e n , dass

- fuer alle A mit AL <= A <= AR gilt: A ist nicht singulaer
- das Gleichungssystem A * x = b fuer jedes A mit AL <= A <= AR und GL <= b <= GR eindeutig loesbar ist
- fuer alle x mit A * x = b und AL <= A <= AR und GL <= b <= GR nach der Abarbeitung GL <= x <= GR gilt.

```
      SUBROUTINE LGL (N,A,WK,BL,BR,NRS)
      DIMENSION A(N,N),WK(N,6),BL(N),BR(N)
      DOUBLE PRECISION BL,BR
      LOGICAL NRS
C
      LOGICAL OK
      IF ( NOT  NRS) GO TO 100
      CALL STEP1A (BL,BR,WK(1,5),WK(1,1),WK(1,2),N)
      GO TO 135
  100 CALL EXEC('@ASG,T 25    ',2,I,#110)
  110 CALL EXEC('@ASG,T 26    ',2,I,#120)
  120 REWIND 25
      REWIND 26
      WRITE (25) A
      CALL MATINV (A,N,WK)
      CALL STEP1 (A,BL,BR,WK(1,5),WK(1,1),N)
      DO 130 I=1,N
           WRITE (26) (A(I,J),J=1,N)
  130 CONTINUE
      REWIND 25
      READ (25) A
  135 CALL STEP2 (A,WK(1,5),WK(1,1),WK(1,2),WK(1,4),N,K1)
      CALL STEP3 (A,WK(1,5),WK(1,1),WK(1,4),WK(1,2),WK(1,5),N)
      CALL STEP4 (A,WK(1,4),WK(1,1),WK(1,2),WK(1,5),WK(1,3),N,K2)
      CALL OVERFL(I)
      CALL STEP5 (A,BL,BR,WK(1,1),WK(1,2),WK(1,3),WK(1,4),WK(1,5),
     *            BL,BR,N)
      IF ( NOT. NRS) CALL STEP6 (A,WK(1,5),BL,N)
      CALL STEP7 (WK(1,3),WK(1,4),WK(1,5),WK(1,6),WK(1,1),WK(1,2),
     *            BL,N,OK,9*(K1+K2)/4)
      IF ( NOT  OK) GO TO 200
      CALL STEP8 (BL,BR,WK(1,1),WK(1,2),WK(1,5),WK(1,6),N)
      CALL OVERFL(I)
      IF (I  EQ.  1) GO TO 200
      RETURN
C
  200 DO 210 I=1,N
           BL(I) = 1.0
           BR(I) = -1.0
  210 CONTINUE
      RETURN
C
C*****************************************************************
C          STEP1A BERECHNET BM := (B1+B2)/2 UND X := R * B       *
C          DIE MATRIX R WIRD ZEILENWEISE VON DATEI 26 EINGELESEN.*
C*****************************************************************
      SUBROUTINE STEP1A (B1,B2,BM,X,R,N)
      DIMENSION B1(N),B2(N),BM(N),X(N),R(N)
      DOUBLE PRECISION B1,B2,BM
C
      DOUBLE PRECISION H
      DO 1010 I=1,N
           BM(I) = 0.5 * (B1(I)+B2(I))
 1010 CONTINUE
C
      REWIND 26
      DO 1030 I=1,N
           READ (26) R
           H = 0
           DO 1020 J=1,N
                H = H + R(J) * BM(J)
 1020      CONTINUE
           X(I) = H
 1030 CONTINUE
C
C*****************************************************************
```

```
C          STEP1 BERECHNET BM := (B1+B2)/2 UND X := R * B        *
C          DIE MATRIX R IST IN A GESPEICHERT                     *
C*****************************************************************
      SUBROUTINE STEP1 (A,B1,B2,BM,X,N)
      DIMENSION A(N,N),B1(N),B2(N),BM(N),X(N)
      DOUBLE PRECISION B1,B2,BM
C
      DOUBLE PRECISION H
      DO 1010 I=1,N
           BM(I) = 0.5 * (B1(I)+B2(I))
 1010 CONTINUE
C
      DO 1030 I=1,N
           H = 0
           DO 1020 J=1,N
                H = H + A(I,J) * BM(J)
 1020      CONTINUE
           X(I) = H
 1030 CONTINUE
C
C*****************************************************************
C          STEP2 ITERIERT  X := X + R * (BM - A * X)            *
C          R WIRD BEI JEDER ITERATION NEU VON DATEI 26 GELESEN. *
C          K1 := ANZAHL DER ITERATIONEN.                        *
C*****************************************************************
      SUBROUTINE STEP2 (A,BM,X,R,Y,N,K1)
      DIMENSION A(N,N),BM(N),X(N),R(N),Y(N)
      DOUBLE PRECISION BM
C
      DOUBLE PRECISION H
      LOGICAL READY,CONV
      BOUND = 31.622777
      K1 = 0
 2010 CONTINUE
           K1 = K1+1
           REWIND 26
           READY = .TRUE.
           CONV = .TRUE.
           BOUND = BOUND * .31622777
           DO 2030 I=1,N
                H = BM(I)
                DO 2020 J=1,N
                     H = H - A(I,J) * X(J)
 2020           CONTINUE
                Y(I) = H
 2030      CONTINUE
           DO 2060 I=1,N
                READ (26) R
                S = 0
                DO 2040 J=1,N
                     S = S + R(J) * Y(J)
 2040           CONTINUE
                XNEU = X(I) + S
                IF (X(I)  EQ.  0.  OR.  XNEU .EQ.  0) GO TO 2050
                DELTA = MIN ( ABS(S/XNEU) , ABS(XNEU/X(I)) )
                IF (DELTA  GT.  1.0E-7) READY =  FALSE.
                IF (DELTA .GT.  BOUND) CONV = .FALSE.
 2050           X(I) = XNEU
 2060      CONTINUE
      IF (.NOT.  READY .AND.  CONV) GO TO 2010
C
C*****************************************************************
C          STEP3 BERECHNET Y := BM - A * X  UND  DX := R * Y     *
C          R WIRD VON DATEI 26 GELESEN                          *
C*****************************************************************
      SUBROUTINE STEP3 (A,BM,X,Y,DX,R,N)
```

```
      DIMENSION A(N,N),BM(N),X(N),Y(N),DX(N),R(N)
      DOUBLE PRECISION BM
C
      DOUBLE PRECISION H
      REWIND 26
      DO 3020 I=1,N
           H = BM(I)
           DO 3010 J=1,N
                H = H - A(I,J) * X(J)
 3010      CONTINUE
           Y(I) = H
 3020 CONTINUE
      DO 3040 I=1,N
           READ (26) R
           DX(I) = 0
           DO 3030 J=1,N
                DX(I) = DX(I) + R(J) * Y(J)
 3030      CONTINUE
 3040 CONTINUE
C
C*********************************************************************
C            STEP4 ITERIERT   DX := DX + R * (Y - A * DX)            *
C            R WIRD VON DATEI 26 GELESEN.                            *
C            K2 := ANZAHL DER ITERATIONEN.                           *
C*********************************************************************
      SUBROUTINE STEP4 (A,Y,X,DX,R,Z,N,K2)
      DIMENSION A(N,N),Y(N),X(N),DX(N),R(N),Z(N)
C
      LOGICAL READY,CONV
      DOUBLE PRECISION H
      BOUND = 31.622777
      K2 = 0
 4010 CONTINUE
           K2 = K2+1
           REWIND 26
           READY = .TRUE.
           CONV = .TRUE.
           BOUND = BOUND * .31622777
           DO 4030 I=1,N
                H = Y(I)
                DO 4020 J=1,N
                     H = H - A(I,J) * DX(J)
 4020           CONTINUE
                Z(I) = H
 4030      CONTINUE
           DO 4060 I=1,N
                READ (26) R
                S = 0
                DO 4040 J=1,N
                     S = S + R(J) * Z(J)
 4040           CONTINUE
                DXNEU = DX(I) + S
                IF (DX(I) .EQ. 0 .OR. DXNEU .EQ. 0) GO TO 4050
                DELTA = MIN ( ABS(S/DXNEU) , 1.0E9*ABS(DXNEU/X(I)) )
                IF (DELTA .GT. 1.0E-5) READY = .FALSE.
                IF (DELTA .GT. BOUND) CONV = .FALSE.
 4050           DX(I) = DXNEU
 4060      CONTINUE
      IF (.NOT. READY .AND. CONV) GO TO 4010
C
C*********************************************************************
C            STEP5 BERECHNET INTERVALLMAESSIG                        *
C            Z := R * (B - A * X - A * DX)                           *
C            R WIRD VON DATEI 26 GELESEN                             *
C*********************************************************************
      SUBROUTINE STEP5 (A,B1,B2,X,DX,Z1,Z2,R,W1,W2,N)
```

```
      DIMENSION A(N,N)
      DIMENSION B1(N),B2(N),X(N),DX(N),Z1(N),Z2(N),R(N),W1(N),W2(N)
      DOUBLE PRECISION B1,B2
C
      DOUBLE PRECISION H
      INTEGER AKKU(63)
      DO 5030 I=1,N
           DO 5005 J=1,63
                AKKU(J) = 0
 5005      CONTINUE
           DO 5010 J=1,N
                H = -A(I,J) * DX(J)
                CALL AKADD (AKKU,H)
 5010      CONTINUE
           DO 5020 J=1,N
                H = -A(I,J) * X(J)
                CALL AKADD (AKKU,H)
 5020      CONTINUE
           CALL AKADD (AKKU,B1(I))
           CALL AKINT (AKKU,G1,G0)
           CALL AKADD (AKKU,-B1(I))
           CALL AKADD (AKKU,B2(I))
           CALL AKINT (AKKU,G0,G2)
           W1(I) = G1
           W2(I) = G2
 5030 CONTINUE
C
      REWIND 26
      DO 5050 I=1,N
           READ (26) R
           F1 = 0
           F2 = 0
           DO 5040 J=1,N
                CALL MUL1 (R(J),W1(J),W2(J),G1,G2)
                CALL ADD (G1,G2,F1,F2,F1,F2)
 5040      CONTINUE
           Z1(I) = F1
           Z2(I) = F2
 5050 CONTINUE
C
C*********************************************************************
C            STEP6 BERECHNET INTERVALLMAESSIG   B := 1 - R * A       *
C            R WIRD ZEILENWEISE VON DATEI 26 GELESEN UND B WIRD      *
C            ZEILENWEISE AUF DATEI 25 GESCHRIEBEN                    *
C*********************************************************************
      SUBROUTINE STEP6 (A,R,B,N)
      DIMENSION A(N,N),R(N),B(N,2)
C
      DOUBLE PRECISION H
      INTEGER AKKU(63)
      REWIND 25
      REWIND 26
      DO 6030 I=1,N
           READ (26) R
           DO 6020 J=1,N
                DO 6005 K=1,63
                     AKKU (K) = 0
 6005           CONTINUE
                IF (I .EQ. J) CALL AKADD (AKKU,1.0D0)
                DO 6010 K=1,N
                     H = -R(K) * A(K,J)
                     CALL AKADD (AKKU,H)
 6010           CONTINUE
                CALL AKINT (AKKU,B(J,1),B(J,2))
 6020      CONTINUE
           WRITE (25) B
```

```
 6030 CONTINUE
C
C*****************************************************************
C           STEP7 BERECHNET Y := Z + [-EPS,EPS] * DIAM (Z)      *
C           UND ITERIERT INTERVALLMAESSIG  Y := Z + B * Y       *
C           WENN KEINE EINSCHLIESSUNG ERREICHT WERDEN KANN, WIRD*
C           OK = .FALSE. GESETZT.                               *
C           ES WIRD MAXIMAL K MAL ITERIERT.                     *
C*****************************************************************
      SUBROUTINE STEP7 (Z1,Z2,Y1,Y2,X,DX,B,N,OK,K)
      DIMENSION Z1(N),Z2(N),Y1(N),Y2(N),X(N),DX(N),B(N,2)
      LOGICAL OK
C
      REAL LOW
      LOGICAL READY
      DOUBLE PRECISION F1,F2,G1,G2
      OK = .TRUE.
      EPS = 0.1
      FACTOR = 1./FLOAT(2**26)
      DO 7005 I=1,N
           D = EPS * ( Z2(I) - Z1(I) )
           Y1(I) = Z1(I) - D
           Y1(I) = Y1(I) - FACTOR * ABS(Y1(I)) - 1.469368E-39
           Y2(I) = Z2(I) + D
           Y2(I) = Y2(I) + FACTOR * ABS(Y2(I)) + 1.469368E-39
 7005 CONTINUE
C
      KK = 0
 7010 CONTINUE
           KK = KK+1
           REWIND 25
           READY = .TRUE
           DO 7040 I=1,N
                READ (25) B
                G1 = 0
                G2 = 0
                DO 7030 J=1,N
                     CALL DMULR (B(J,1),B(J,2),Y1(J),Y2(J),F1,F2)
                     CALL DADD (F1,F2,G1,G2,G1,G2)
 7030           CONTINUE
                CALL DADD (DBLE(Z1(I)),DBLE(Z2(I)),G1,G2,G1,G2)
                IF (G1 .LE. DBLE(Y1(I)) .OR. G2 .GE. DBLE(Y2(I)))
     *               READY = .FALSE.
                IF (ABS(G1) .GT. 1.D+38 .OR. ABS(G2) .GT. 1.D+38)
     *               GO TO 7800
                Y1(I) = LOW(SNGL(G1))
                Y2(I) = -LOW(-SNGL(G2))
 7040      CONTINUE
           IF (READY) GO TO 7900
           IF (KK .EQ. K) GO TO 7800
           IF (KK .EQ. 5) EPS = 1.0
           IF (KK .GT. 5) EPS = EPS+EPS
           DO 7050 I=1,N
                D = EPS * ( Y2(I) - Y1(I) )
                Y1(I) = Y1(I) - D
                Y1(I) = Y1(I) - FACTOR * ABS(Y1(I))
                Y2(I) = Y2(I) + D
                Y2(I) = Y2(I) + FACTOR * ABS(Y2(I))
 7050      CONTINUE
      GO TO 7010
 7800 OK = .FALSE.
 7900 CONTINUE
C
C*****************************************************************
C          STEP8 BERECHNET INTERVALLMAESSIG B := X + DX + Y      *
C*****************************************************************
      SUBROUTINE STEP8 (B1,B2,X,DX,Y1,Y2,N)
      DIMENSION B1(N),B2(N),X(N),DX(N),Y1(N),Y2(N)
      DOUBLE PRECISION B1,B2
C
      DOUBLE PRECISION H,H1,H2
      DO 8010 I=1,N
           H = DX(I)
           H1 = Y1(I)
           H2 = Y2(I)
           CALL DADD (H,H,H1,H2,H1,H2)
           H = X(I)
           CALL DADD (H,H,H1,H2,B1(I),B2(I))
 8010 CONTINUE
      END
```

```
      SUBROUTINE LGLU (N,AL,AR,WK,BL,BR,NRS)
      DIMENSION AL(N,N),AR(N,N),WK(N,6),BL(N),BR(N)
      DOUBLE PRECISION BL,BR
      LOGICAL NRS
C
      LOGICAL OK
      IF ( NOT. NRS) GO TO 100
      CALL STEP1A (BL,BR,WK(1,5),WK(1,1),WK(1,2),N)
      GO TO 135
  100 CALL EXEC('@ASG,T 25    ',2,I,$110)
  110 CALL EXEC('@ASG,T 26    ',2,I,$120)
  120 REWIND 25
      REWIND 26
      WRITE (25) AL
      DO 123 I=1,N
           DO 122 J=1,N
                AL(I,J) = 0.5 * (AL(I,J)+AR(I,J))
  122      CONTINUE
  123 CONTINUE
      CALL MATINV (AL,N,WK)
      CALL STEP1 (AL,BL,BR,WK(1,5),WK(1,1),N)
      DO 130 I=1,N
           WRITE (26) (AL(I,J),J=1,N)
  130 CONTINUE
      REWIND 25
      READ (25) AL
  135 CALL STEP2 (AL,AR,WK(1,5),WK(1,1),WK(1,2),WK(1,4),N,K1)
      CALL STEP3 (AL,AR,WK(1,5),WK(1,1),WK(1,4),WK(1,2),WK(1,5),N)
      CALL STEP4 (AL,AR,WK(1,4),WK(1,1),WK(1,2),WK(1,5),WK(1,3),N,K2)
      CALL OVERFL(I)
      CALL STEP5 (AL,AR,BL,BR,WK(1,1),WK(1,2),WK(1,3),WK(1,4),WK(1,5),
     *            BL,BR,N)
      IF ( NOT. NRS) CALL STEP6 (AL,AR,WK(1,5),BL,N)
      CALL STEP7 (WK(1,3),WK(1,4),WK(1,5),WK(1,6),WK(1,1),WK(1,2),
     *            BR,BL,N,OK,9*(K1+K2)/4)
      IF ( NOT. OK) GO TO 200
      CALL STEP8 (BL,BR,WK(1,1),WK(1,5),WK(1,6),N)
      CALL OVERFL(I)
      IF (I .EQ. 1) GO TO 200
      RETURN
C
  200 DO 210 I=1,N
           BL(I) = 1.0
           BR(I) = -1.0
  210 CONTINUE
      RETURN
C
C*****************************************************************
C          STEP1A BERECHNET BM := (B1+B2) UND X := R * BM/2
C          DIE MATRIX R WIRD ZEILENWEISE VON DATEI 26 EINGELESEN      *
C*****************************************************************
      SUBROUTINE STEP1A (B1,B2,BM,X,R,N)
      DIMENSION B1(N),B2(N),BM(N),X(N),R(N)
      DOUBLE PRECISION B1,B2,BM
C
      DOUBLE PRECISION H
      DO 1010 I=1,N
           BM(I) = B1(I)+B2(I)
 1010 CONTINUE
C
      REWIND 26
      DO 1030 I=1,N
           READ (26) R
           H = 0
           DO 1020 J=1,N
                H = H + R(J) * BM(J)
 1020      CONTINUE
           X(I) = 0.5 * SNGL(H)
 1030 CONTINUE
C
C*****************************************************************
C          STEP1 BERECHNET BM := (B1+B2) UND X := R * BM/2
C          DIE MATRIX R IST IN A GESPEICHERT                          *
C*****************************************************************
      SUBROUTINE STEP1 (A,B1,B2,BM,X,N)
      DIMENSION A(N,N),B1(N),B2(N),BM(N),X(N)
      DOUBLE PRECISION B1,B2,BM
C
      DOUBLE PRECISION H
      DO 1010 I=1,N
           BM(I) = B1(I)+B2(I)
 1010 CONTINUE
C
      DO 1030 I=1,N
           H = 0
           DO 1020 J=1,N
                H = H + A(I,J) * BM(J)
 1020      CONTINUE
           X(I) = 0.5 * SNGL(H)
 1030 CONTINUE
C
C*****************************************************************
C          STEP2 ITERIERT  X := X + R * (BM - 2*A * X)/2              *
C          R WIRD BEI JEDER ITERATION NEU VON DATEI 26 GELESEN.       *
C          K1 := ANZAHL DER ITERATIONEN                               *
C*****************************************************************
      SUBROUTINE STEP2 (AL,AR,BM,X,R,Y,N,K1)
      DIMENSION AL(N,N),AR(N,N),BM(N),X(N),R(N),Y(N)
      DOUBLE PRECISION BM
C
      DOUBLE PRECISION H
      LOGICAL READY,CONV
      BOUND = 31.622777
      K1 = 0
 2010 CONTINUE
           K1 = K1+1
           REWIND 26
           READY = .TRUE.
           CONV = .TRUE.
           BOUND = BOUND * .31622777
           DO 2030 I=1,N
                H = BM(I)
                DO 2020 J=1,N
                     H = H - (AL(I,J)+AR(I,J)) * X(J)
 2020           CONTINUE
                Y(I) = 0.5*SNGL(H)
 2030      CONTINUE
           DO 2060 I=1,N
                READ (26) R
                S = 0
                DO 2040 J=1,N
                     S = S + R(J) * Y(J)
 2040           CONTINUE
                XNEU = X(I) + S
                IF (X(I) .EQ. 0 .OR. XNEU .EQ. 0) GO TO 2050
                DELTA = MIN ( ABS(S/XNEU) , ABS(XNEU/X(I)) )
                IF (DELTA .GT. 1.0E-7) READY = .FALSE.
                IF (DELTA .GT. BOUND) CONV = .FALSE.
                X(I) = XNEU
 2050
 2060      CONTINUE
      IF ( NOT. READY .AND. CONV) GO TO 2010
C
```

```
C*********************************************************************
C             STEP3 BERECHNET Y := BM - 2*A * X  UND  DX := (R * Y)/2  *
C             R WIRD VON DATEI 26 GELESEN.                             *
C*********************************************************************
      SUBROUTINE STEP3 (AL,AR,BM,X,Y,DX,R,N)
      DIMENSION AL(N,N),AR(N,N),BM(N),X(N),Y(N),DX(N),R(N)
      DOUBLE PRECISION BM
C
      DOUBLE PRECISION H
      REWIND 26
      DO 3020 I=1,N
           H = BM(I)
           DO 3010 J=1,N
                H = H - (AL(I,J)+AR(I,J)) * X(J)
 3010      CONTINUE
           Y(I) = H
 3020 CONTINUE
      DO 3040 I=1,N
           READ (26) R
           H = 0
           DO 3030 J=1,N
                H = H + R(J) * Y(J)
 3030      CONTINUE
           DX(I) = 0.5 * SNGL(H)
 3040 CONTINUE
C
C*********************************************************************
C             STEP4 ITERIERT   DX := DX + R * (Y - 2*A * DX)/2         *
C             R WIRD VON DATEI 26 GELESEN.                             *
C             K2 := ANZAHL DER ITERATIONEN.                            *
C*********************************************************************
      SUBROUTINE STEP4 (AL,AR,Y,X,DX,R,Z,N,K2)
      DIMENSION AL(N,N),AR(N,N),Y(N),X(N),DX(N),R(N),Z(N)
C
      LOGICAL READY,CONV
      DOUBLE PRECISION H
      BOUND = 31.622777
      K2 = 0
 4010 CONTINUE
           K2 = K2+1
           REWIND 26
           READY = .TRUE.
           CONV = .TRUE.
           BOUND = BOUND * .31622777
           DO 4030 I=1,N
                H = Y(I)
                DO 4020 J=1,N
                     H = H - (AL(I,J)+AR(I,J)) * DX(J)
 4020           CONTINUE
                Z(I) = 0.5 * SNGL(H)
 4030      CONTINUE
           DO 4060 I=1,N
                READ (26) R
                S = 0
                DO 4040 J=1,N
                     S = S + R(J) * Z(J)
 4040           CONTINUE
                DXNEU = DX(I) + S
                IF (DX(I) .EQ. 0 .OR. DXNEU .EQ. 0) GO TO 4050
                DELTA = MIN ( ABS(S/DXNEU) , 1.0E9*ABS(DXNEU/X(I)) )
                IF (DELTA .GT. 1.0E-6) READY = .FALSE.
                IF (DELTA .GT. BOUND) CONV = .FALSE.
 4050           DX(I) = DXNEU
 4060      CONTINUE
      IF (.NOT. READY .AND. CONV) GO TO 4010
C
```

```
C*********************************************************************
C             STEP5 BERECHNET INTERVALLMAESSIG                         *
C             Z := DX + R * (B - A * X - A * DX)                       *
C             R WIRD VON DATEI 26 GELESEN                              *
C*********************************************************************
      SUBROUTINE STEP5 (AL,AR,B1,B2,X,DX,Z1,Z2,R,W1,W2,N)
      DIMENSION AL(N,N),AR(N,N)
      DIMENSION B1(N),B2(N),X(N),DX(N),Z1(N),Z2(N),R(N),W1(N),W2(N)
      DOUBLE PRECISION B1,B2
C
      REAL LOW
      DOUBLE PRECISION H1,H2,G1,G2
      DO 5030 I=1,N
           H1 = 0
           H2 = 0
           DO 5010 J=1,N
                IF (DX(J)) 5002,5010,5004
 5002           G1 = DX(J) * AR(I,J)
                G2 = DX(J) * AL(I,J)
                GO TO 5008
 5004           G1 = DX(J) * AL(I,J)
                G2 = DX(J) * AR(I,J)
 5008           CALL DADD(G1,G2,H1,H2,H1,H2)
 5010      CONTINUE
           DO 5020 J=1,N
                IF (X(J)) 5012,5020,5014
 5012           G1 = X(J) * AR(I,J)
                G2 = X(J) * AL(I,J)
                GO TO 5018
 5014           G1 = X(J) * AL(I,J)
                G2 = X(J) * AR(I,J)
 5018           CALL DADD(G1,G2,H1,H2,H1,H2)
 5020      CONTINUE
           CALL DADD (B1(I),B2(I),-H2,-H1,H1,H2)
           W1(I) =  LOW( SNGL(H1))
           W2(I) = -LOW(-SNGL(H2))
 5030 CONTINUE
C
      REWIND 26
      DO 5050 I=1,N
           READ (26) R
           F1 = 0
           F2 = 0
           DO 5040 J=1,N
                CALL MUL1 (R(J),W1(J),W2(J),D1,D2)
                CALL ADD (D1,D2,F1,F2,F1,F2)
 5040      CONTINUE
           CALL ADD (DX(I),DX(I),F1,F2,Z1(I),Z2(I))
 5050 CONTINUE
C
C*********************************************************************
C             STEP6 BERECHNET INTERVALLMAESSIG   B := I - R * A        *
C             R WIRD ZEILENWEISE VON DATEI 25 GELESEN UND B WIRD       *
C             ZEILENWEISE AUF DATEI 25 GESCHRIEBEN                     *
C*********************************************************************
      SUBROUTINE STEP6 (AL,AR,R,B,N)
      DIMENSION AL(N,N),AR(N,N),R(N),B(N,2)
C
      DOUBLE PRECISION H1,H2,F1,F2
      REAL LOW
      REWIND 25
      REWIND 26
      DO 6030 I=1,N
           READ (26) R
           DO 6020 J=1,N
                H1 = 0
```

```
                IF (I .EQ. J) H1 = 1.0D0
                H2 = H1
                DO 6010 K=1,N
                     IF (R(K)) 6002,6010,6004
 6002                F1 = -R(K) * AL(K,J)
                     F2 = -R(K) * AR(K,J)
                     GO TO 6008
 6004                F1 = -R(K) * AR(K,J)
                     F2 = -R(K) * AL(K,J)
 6008                CALL DADD(F1,F2,H1,H2,H1,H2)
 6010           CONTINUE
                B(J,1) =  LOW( SNGL(H1))
                B(J,2) = -LOW(-SNGL(H2))
 6020      CONTINUE
           WRITE (25) B
 6030 CONTINUE
C
C*********************************************************************
C            STEP7 BERECHNET Y := Z + [-EPS,EPS] * DIAM (Z)          *
C            UND ITERIERT INTERVALLMAESSIG  Y  = Z + B * (Y - DX)    *
C            WENN KEINE EINSCHLIESSUNG ERREICHT WERDEN KANN, WIRD    *
C            OK =  FALSE  GESETZT.                                   *
C            ES WIRD MAXIMAL K MAL ITERIERT.                         *
C*********************************************************************
      SUBROUTINE STEP7 (Z1,Z2,Y1,Y2,X,DX,W,B,N,OK,K)
      DIMENSION Z1(N),Z2(N),Y1(N),Y2(N),X(N),DX(N),W(N,2),B(N,2)
      LOGICAL OK
C
      REAL LOW
      LOGICAL READY
      DOUBLE PRECISION F1,F2,G1,G2
      OK = .TRUE.
      EPS = 0.1
      FACTOR = 1./FLOAT(2**26)
      DO 7005 I=1,N
           D = EPS * ( Z2(I) - Z1(I) )
           Y1(I) = Z1(I) - D
           Y1(I) = Y1(I) - FACTOR * ABS(Y1(I)) - 1.469368E-39
           Y2(I) = Z2(I) + D
           Y2(I) = Y2(I) + FACTOR * ABS(Y2(I)) + 1.469368E-39
 7005 CONTINUE
C
      KK = 0
 7010 CONTINUE
           KK = KK+1
           REWIND 25
           READY = .TRUE.
           DO 7020 I=1,N
                CALL ADD (Y1(I),Y2(I),-DX(I),-DX(I),W(I,1),W(I,2))
 7020      CONTINUE
           DO 7040 I=1,N
                READ (25) B
                G1 = 0
                G2 = 0
                DO 7030 J=1,N
                     CALL DMULR (B(J,1),B(J,2),W(J,1),W(J,2),F1,F2)
                     CALL DADD (F1,F2,G1,G2,G1,G2)
 7030           CONTINUE
                CALL DADD (DBLE(Z1(I)),DBLE(Z2(I)),G1,G2,G1,G2)
                IF (G1 .LE. DBLE(Y1(I)) .OR. G2 .GE. DBLE(Y2(I)))
     *               READY = .FALSE.
                IF (ABS(G1) .GT. 1.D+38 .OR. ABS(G2) .GT. 1.D+38)
     *               GO TO 7800
                Y1(I) = LOW(SNGL(G1))
                Y2(I) = -LOW(-SNGL(G2))
                CALL ADD (Y1(I),Y2(I),-DX(I),-DX(I),W(I,1),W(I,2))
 7040      CONTINUE
           IF (READY) GO TO 7900
           IF (KK .EQ. K) GO TO 7800
           IF (KK .EQ. 5) EPS = 1.0
           IF (KK .GT. 5) EPS = EPS+EPS
           DO 7050 I=1,N
                D = EPS * ( Y2(I) - Y1(I) )
                Y1(I) = Y1(I) - D
                Y1(I) = Y1(I) - FACTOR * ABS(Y1(I))
                Y2(I) = Y2(I) + D
                Y2(I) = Y2(I) + FACTOR * ABS(Y2(I))
 7050      CONTINUE
      GO TO 7010
 7800 OK = .FALSE.
 7900 CONTINUE
C
C*********************************************************************
C            STEP8 BERECHNET INTERVALLMAESSIG B := X + Y              *
C*********************************************************************
      SUBROUTINE STEP8 (B1,B2,X,Y1,Y2,N)
      DIMENSION B1(N),B2(N),X(N),Y1(N),Y2(N)
      DOUBLE PRECISION B1,B2
C
      DOUBLE PRECISION H,H1,H2
      DO 8010 I=1,N
           H = X(I)
           H1 = Y1(I)
           H2 = Y2(I)
           CALL DADD (H,H,H1,H2,B1(I),B2(I))
 8010 CONTINUE
      END
```

```
      SUBROUTINE LEIGU ( N , AL, AR , EW , X , E1 , E2 , X1 , X2 , WK )
      DIMENSION AL(N,N) , AR(N,N) , X(N) , X1(N) , X2(N) , WK(N,1)
C WK MUSS MINDESTENS N*(N+3) ELEMENTE HABEN
      DOUBLE PRECISION E1,E2,X1,X2
      CALL LEIGU1 ( N , AL , AR , EW , X , E1 , E2 , X1 , X2 , WK ,
     *              X1 , X2 , WK(1,N+1) , WK(1,N+2) , WK(1,N+3) )
      RETURN
C
C
      SUBROUTINE LEIGU1 (N,AL,AR,EW,X,E1,E2,X1,X2,R,B,Y,Z1,Z2,DX)
      DIMENSION AL(N,N),AR(N,N),X(N),X1(N),X2(N),R(N,N),B(2,N),Y(2,N),
     *     Z1(N),Z2(N),DX(N)
      DOUBLE PRECISION E1,E2,X1,X2
C
      LOGICAL READY,CONV
      DOUBLE PRECISION H1,H2,G1,G2,ERR
      REAL LOW
C
      REAL EPS/0.1/
      REAL FACTOR/0147400000000/
C FACTOR IST DIE KLEINSTE POSITIVE ZAHL, SO DASS 1.*FACTOR .NE. 1.
      REAL MINPOS/0000400000000/
C MINPOS IST DIE KLEINSTE POSITIVE REAL-ZAHL
      DOUBLE PRECISION MXREAL/1.7D38/
C MXREAL IST DIE GROESSTE DARSTELLBARE REAL-ZAHL (IN D.-P.-DARSTELLUNG)
C
      F1 = 8.  * FACTOR
      F2 =64.  * FACTOR
      F3 = 8.  / FACTOR
      ERR = 1.D0
      L = 0
      LL = 0
      CALL EXEC ('@ASG,T 24    ',2,I,$5)
    5 CONTINUE
      REWIND 24
      WRITE (24) AL
      DO 8 I=1,N
         DO 7 J=1,N
            AL(I,J) = 0.5 * (AL(I,J)+AR(I,J))
    7    CONTINUE
    8 CONTINUE
C                                     SUCHE DIE BETRAGSMAESSIG GROESSTE
C                                     KOMPONENTE VON X
      KK = 1
      DO 10 I=2,N
         IF (ABS(X(I)) .GT. ABS(X(KK))) KK = I
   10 CONTINUE
      XKK = X(KK)
C                                     BERECHNE DIE ABLEITUNGSMATRIX UND
C                                     FUEHRE LU-ZERLEGUNG DURCH
      DO 20 J=1,N
         IF (J .EQ. KK) GO TO 20
         DO 15 I=1,N
            R(I,J) = AL(I,J)
   15    CONTINUE
         R(J,J) = R(J,J) - EW
   20 CONTINUE
      DO 25 I=1,N
         R(I,KK) = -X(I)
   25 CONTINUE
      CALL LUDEC ( N , R , B )
C                                     ITERIERE X GLEITPUNKTMAESSIG
      BOUND = 31.622777
   30 CONTINUE
         LL = LL+1
         READY = .TRUE.
         CONV = .TRUE.
         BOUND = BOUND * .31622777
         DO 40 I=1,N
            H1 = DBLE(-EW) * DBLE(X(I))
            DO 35 J=1,N
               H1 = H1 + DBLE(AL(I,J)) * DBLE(X(J))
   35       CONTINUE
            IF (DABS(H1) .GT. MXREAL) GO TO 902
            Z1(I) = H1
   40    CONTINUE
         CALL LUELM (N,R,Z1,Z2,B)
         X(KK) = EW
         DO 45 I=1,N
            XNEU = X(I) - Z2(I)
            IF (X(I) .EQ. 0. .OR. XNEU .EQ. 0.) GO TO 44
            DELTA = AMIN1 (ABS(Z2(I)/XNEU),ABS(XNEU/X(I)))
            IF (DELTA .GT. F1) READY = .FALSE.
            IF (DELTA .GT. BOUND) CONV = .FALSE.
   44       X(I) = XNEU
   45    CONTINUE
   49    EW = X(KK)
         X(KK) = XKK
      IF (.NOT. READY .AND. CONV) GO TO 30
      IF (.NOT. CONV) ERR = 2.D0
C                                     BERECHNE DOPPELTLANG  A - EW * X
      DO 55 I=1,N
         X2(I) = DBLE(-EW) * DBLE(X(I))
         DO 50 J=1,N
            X2(I) = X2(I) + DBLE(AL(I,J)) * DBLE(X(J))
   50    CONTINUE
         DX(I) = 0.
   55 CONTINUE
      DEW = 0.
C                                     ITERIERE GLEITPUNKTMAESSIG DX
      BOUND = 31.622777
   60 CONTINUE
         LL = LL+1
         READY = .TRUE.
         CONV = .TRUE.
         BOUND = BOUND * .31622777
         DO 64 I=1,N
            H1 = - (DBLE(EW) + DBLE(DEW)) * DBLE(DX(I))
            DO 62 J=1,N
               H1 = H1 + DBLE(AL(I,J)) * DBLE(DX(J))
   62       CONTINUE
            H1 = (H1 - DBLE(DEW) * DBLE(X(I))) + X2(I)
            IF (DABS(H1) .GT. MXREAL) GO TO 902
            Z1(I) = H1
   64    CONTINUE
         CALL LUELM (N,R,Z1,Z2,B)
         X(KK) = EW
         DX(KK) = DEW
         DO 68 I=1,N
            DXNEU = DX(I) - Z2(I)
            IF (X(I) .EQ. 0. .OR. DXNEU .EQ. 0.) GO TO 66
            DELTA = AMIN1 (ABS(Z2(I)/DXNEU),F3*ABS(DXNEU/X(I)))
            IF (DELTA .GT. F2) READY = .FALSE.
            IF (DELTA .GT. BOUND) CONV = .FALSE.
   66       DX(I) = DXNEU
   68    CONTINUE
         EW = X(KK)
         DEW = DX(KK)
         X(KK) = XKK
         DX(KK) = 0.
      IF (.NOT. READY .AND. CONV .OR. BOUND .GT. 9.) GO TO 60
C
```

```
C                                       BERECHNE GLEITPUNKTMAESSIG DIE
C                                       INVERSE DER ABLEITUNGSMATRIX
      DO 72 J=1,N
         IF (J .EQ. KK) GO TO 72
         DO 70 I=1,N
            R(I,J) = AL(I,J)
   70    CONTINUE
         R(J,J) = R(J,J) - EW
   72 CONTINUE
      DO 74 I=1,N
         R(I,KK) = -X(I)
   74 CONTINUE
      CALL MATINV ( R , N , B )
C                                       BERECHNE INTERVALLMAESSIG
C                                       B = (A-(EW+DEW)*I) * (X+DX)
C
      CALL OVERFL(I)
      REWIND 24
      READ (24) AL
      DO 83 I=1,N
         H1 = DBLE(-DEW) * DBLE(DX(I))
         H2 = H1
         G1 = DBLE(-DEW) * DBLE(X(I))
         CALL DADD (H1,H2,G1,G1,H1,H2)
         G1 = DBLE(-EW) * DBLE(DX(I))
         CALL DADD (H1,H2,G1,G1,H1,H2)
         G1 = DBLE(-EW) * DBLE(X(I))
         CALL DADD (H1,H2,G1,G1,H1,H2)
         DO 82 J=1,N
            IF (X(J)) 75,78,76
   75       G1 = DBLE(AR(I,J)) * DBLE(X(J))
            G2 = DBLE(AL(I,J)) * DBLE(X(J))
            GO TO 77
   76       G1 = DBLE(AL(I,J)) * DBLE(X(J))
            G2 = DBLE(AR(I,J)) * DBLE(X(J))
   77       CALL DADD (H1,H2,G1,G2,H1,H2)
   78       IF (DX(J)) 79,82,80
   79       G1 = DBLE(AR(I,J)) * DBLE(DX(J))
            G2 = DBLE(AL(I,J)) * DBLE(DX(J))
            GO TO 81
   80       G1 = DBLE(AL(I,J)) * DBLE(DX(J))
            G2 = DBLE(AR(I,J)) * DBLE(DX(J))
   81       CALL DADD (H1,H2,G1,G2,H1,H2)
   82    CONTINUE
         B(1,I) =  LOW ( SNGL(H1))
         B(2,I) = -LOW (-SNGL(H2))
   83 CONTINUE
C                                       BERECHNE INTERVALLMAESSIG Z=-R*B
      DO 90 I=1,N
         H1 = 0.D0
         H2 = 0.D0
         DO 88 J=1,N
            IF (R(I,J)) 84,88,85
   84       G1 = DBLE(-R(I,J)) * DBLE(B(1,J))
            G2 = DBLE(-R(I,J)) * DBLE(B(2,J))
            GO TO 86
   85       G1 = DBLE(-R(I,J)) * DBLE(B(2,J))
            G2 = DBLE(-R(I,J)) * DBLE(B(1,J))
   86       CALL DADD (H1,H2,G1,G2,H1,H2)
   88    CONTINUE
         Z1(I) =  LOW ( SNGL(H1))
         Z2(I) = -LOW (-SNGL(H2))
   90 CONTINUE
C                                       BERECHNE INTERVALLMAESSIG
C                                       B = I - R * (A-EW*I)
C                                       UND SCHREIBE DAS ERGEBNIS
C                                       ZEILENWEISE AUF DATEI 24
      REWIND 24
      DO 140 I=1,N
         DO 130 J=1,N
            IF (J .EQ. KK) GO TO 130
            H1 = 0.D0
            IF (I .EQ. J) H1 = 1.D0
            H2 = H1
            G1 = DBLE(EW) * DBLE(R(I,J))
            CALL DADD (H1,H2,G1,G1,H1,H2)
            G1 = DBLE(DEW) * DBLE(R(I,J))
            CALL DADD (H1,H2,G1,G1,H1,H2)
            DO 120 K=1,N
               IF (R(I,K)) 112,120,114
  112          G1 = DBLE(-R(I,K)) * DBLE(AL(K,J))
               G2 = DBLE(-R(I,K)) * DBLE(AR(K,J))
               GO TO 118
  114          G1 = DBLE(-R(I,K)) * DBLE(AR(K,J))
               G2 = DBLE(-R(I,K)) * DBLE(AL(K,J))
  118          CALL DADD (H1,H2,G1,G2,H1,H2)
  120       CONTINUE
            B(1,J) =  LOW ( SNGL(H1))
            B(2,J) = -LOW (-SNGL(H2))
  130    CONTINUE
         H1 = 0.D0
         IF (I .EQ. KK) H1 = 1.D0
         H2 = H1
         DO 134 K=1,N
            G1 = DBLE(R(I,K)) * DBLE(X(K))
            CALL DADD (H1,H2,G1,G1,H1,H2)
            G1 = DBLE(R(I,K)) * DBLE(DX(K))
            CALL DADD (H1,H2,G1,G1,H1,H2)
  134    CONTINUE
         B(1,KK) =  LOW ( SNGL(H1))
         B(2,KK) = -LOW (-SNGL(H2))
         WRITE (24) B
  140 CONTINUE
C
      DO 150 I=1,N
         Y(1,I) = Z1(I)
         Y(2,I) = Z2(I)
  150 CONTINUE
C                                       ITERIERE INTERVALLMAESSIG
C                                       Y = Z + B * Y
C                                       BIS EINSCHLIESSUNG ERREICHT WIRD
C
  155 CONTINUE
         L = L+1
         IF (L .GT. 7) GO TO 900
         READY = .TRUE.
         DO 160 I=1,N
            W1 = EPS * (Y(2,I)-Y(1,I))
            Y(1,I) = Y(1,I) - W1 - FACTOR * ABS(Y(1,I)) - MINPOS
            Y(2,I) = Y(2,I) + W1 + FACTOR * ABS(Y(2,I)) + MINPOS
  160    CONTINUE
C
         REWIND 24
         DO 215 I=1,N
            READ (24) B
            EW1 = Y(1,KK)
            EW2 = Y(2,KK)
            Y(1,KK) = 0.
            Y(2,KK) = 0.
            H1 = 0.D0
            H2 = 0.D0
            DO 200 J=1,N
               IF (R(I,J)) 170,200,190
```

```
  170          G1 = DBLE(R(I,J)) * DBLE(Y(2,J))
               G2 = DBLE(R(I,J)) * DBLE(Y(1,J))
               GO TO 190
  180          G1 = DBLE(R(I,J)) * DBLE(Y(1,J))
               G2 = DBLE(R(I,J)) * DBLE(Y(2,J))
  190          CALL DADD (H1,H2,G1,G2,H1,H2)
  200       CONTINUE
            CALL DADD (H1,H2,DBLE(B(1,KK)),DBLE(B(2,KK)),H1,H2)
            IF (DABS(H1) .GT. MXREAL  .OR.  DABS(H2) .GT. MXREAL) GO TO 900
            B(1,KK) =  LOW ( SNGL(H1))
            B(2,KK) = -LOW (-SNGL(H2))
            Y(1,KK) = EW1
            Y(2,KK) = EW2
            H1 = 0.DO
            H2 = 0.DO
            DO 210 J=1,N
               CALL DMULR (B(I,J),B(2,J),Y(1,J),Y(2,J),G1,G2)
               CALL DADD (H1,H2,G1,G2,H1,H2)
  210       CONTINUE
            CALL DADD (H1,H2,DBLE(Z1(I)),DBLE(Z2(I)),H1,H2)
            IF (DABS(H1) .GT. MXREAL  .OR.  DABS(H2) .GT. MXREAL ) GO TO 900
            IF(H1 .LE. DBLE(Y(1,I)) .OR. H2 .GE. DBLE(Y(2,I))) READY=.FALSE.
            Y(1,I) =  LOW ( SNGL(H1))
            Y(2,I) = -LOW (-SNGL(H2))
  215    CONTINUE
      IF (.NOT. READY) GO TO 155
C                                        DAS ERGEBNIS IST  X + DX + Y
      X(KK) = EW
      DX(KK) = DEW
      DO 240 I=1,N
         G1 = DX(I)
         CALL DADD (DBLE(Y(1,I)),DBLE(Y(2,I)),G1,G1,H1,H2)
         G1 = X(I)

         CALL DADD (H1,H2,G1,G1,X1(I),X2(I))
C VORSICHT  X2 BELEGT DEN GLEICHEN SPEICHERPLATZ WIE Y
  240 CONTINUE
      E1 = X1(KK)
      E2 = X2(KK)
      X1 (KK) = XKK
      X2 (KK) = XKK
      CALL OVERFL(I)
      IF (I .EQ. 2) GO TO 990
      ERR = 3.DO
      GO TO 900
C                                        FEHLERAUSGANG
  902 ERR = 2.DO
  900 E1 =  ERR
      E2 = -ERR
      DO 910 I=1,N
        X1(I) = ERR
        X2(I) = -ERR
  910 CONTINUE
C                                        NORMALER AUSGANG
  990 R(1,1) = L
      R(2,1) = LL
      RETURN
      END
```

```
      SUBROUTINE LUDEC (N,A,INT)
      DIMENSION A(N,N),INT(N)
C
      DO 10 I=1,N
         INT(I) = I
   10 CONTINUE
C
      N1 = N-1
      DO 60 J=1,N1
         JJ = J
         J1 = J+1
         DO 20 I=J1,N
            IF (ABS(A(I,J)) .GT. ABS(A(JJ,J))) JJ = I
   20    CONTINUE
         IF (JJ .EQ. J) GO TO 35
         DO 30 K=1,N
            X = A(JJ,K)
            A(JJ,K) = A(J,K)
            A(J,K) = X
   30    CONTINUE
         K = INT(JJ)
         INT(JJ) = INT(J)
         INT(J) = K
C
   35    DO 50 I=J1,N
            A(I,J) = A(I,J) / A(J,J)
            DO 40 K=J1,N
               A(I,K) = A(I,K) - A(I,J) * A(J,K)
   40       CONTINUE
   50    CONTINUE
   60 CONTINUE
      RETURN
      END
```

```
      SUBROUTINE LUELM (N,A,B,X,INT)
      DIMENSION A(N,N),B(N),X(N),INT(N)
C
      DO 5 I=1,N
         II = INT(I)
         X(I) = B(II)
    5 CONTINUE
C
      DO 20 I=2,N
         I1 = I-1
         DO 10 K=1,I1
            X(I) = X(I) - A(I,K) * X(K)
   10    CONTINUE
   20 CONTINUE
      X(N) = X(N) / A(N,N)
      DO 40 II=2,N
         I = N-II+1
         I1 = I+1
         DO 30 K=I1,N
            X(I) = X(I) - A(I,K) * X(K)
   30    CONTINUE
         X(I) = X(I) / A(I,I)
   40 CONTINUE
      RETURN
      END
```

```
      SUBROUTINE MATINV (A,N,WK)
      DIMENSION A(N,N),WK(N)
      INTEGER WK
      DO 10 I=1,N
         WK(I)=I
   10 CONTINUE
C
      DO 80 J=1,N
         IF (J .EQ. N) GO TO 40
         I=J
         J1=J+1
         DO 20 K=J1,N
            IF (ABS(A(K,J)) .GT. ABS(A(I,J))) I=K
   20    CONTINUE
         IF (I .EQ. J) GO TO 40
         DO 30 K=1,N
            X=A(J,K)
            A(J,K)=A(I,K)
            A(I,K)=X
   30    CONTINUE
         K=WK(J)
         WK(J)=WK(I)
         WK(I)=K
C
   40    X = 1.0 / A(J,J)
         DO 50 K=1,N
            A(J,K) = A(J,K) * X
   50    CONTINUE
         A(J,J) = X
         DO 70 I=1,N
            IF (I .EQ. J) GO TO 70
            DO 60 K=1,N
               IF (K .EQ. J) GO TO 60
               A(I,K) = A(I,K) - A(I,J) * A(J,K)
   60       CONTINUE
            A(I,J) = -A(I,J) * A(J,J)
   70    CONTINUE
   80 CONTINUE
C
      DO 110 K=1,N
   90    CONTINUE
            J=WK(K)
            IF (J .EQ. K) GO TO 110
            DO 100 I=1,N
               X = A(I,J)
               A(I,J) = A(I,K)
               A(I,K) = X
  100       CONTINUE
            WK(K) = WK(J)
            WK(J) = J
         GO TO 90
  110 CONTINUE
      END
```

```
      SUBROUTINE DMATIN (A,N,WK)
      DIMENSION A(N,N),WK(N)
      DOUBLE PRECISION A
      INTEGER WK
C
      DOUBLE PRECISION X
      DO 10 I=1,N
           WK(I)=I
   10 CONTINUE
C
      DO 90 J=1,N
           IF (J .EQ. N) GO TO 40
           I=J
           J1=J+1
           DO 20 K=J1,N
                IF (ABS(A(K,J)) .GT. ABS(A(I,J))) I=K
   20      CONTINUE
           IF (I .EQ. J) GO TO 40

           DO 30 K=1,N
                X=A(J,K)
                A(J,K)=A(I,K)
                A(I,K)=X
   30      CONTINUE
           K=WK(J)
           WK(J)=WK(I)
           WK(I)=K
C
   40      X = 1.0 / A(J,J)
           DO 50 K=1,N
                A(J,K) = A(J,K) * X
   50      CONTINUE
           A(J,J) = X
           DO 70 I=1,N
                IF (I .EQ. J) GO TO 70
                DO 60 K=1,N
                     IF (K .EQ. J) GO TO 60
                     A(I,K) = A(I,K) - A(I,J) * A(J,K)
   60           CONTINUE
                A(I,J) = -A(I,J) * A(J,J)
   70      CONTINUE
   80 CONTINUE
C
      DO 110 K=1,N
   90      CONTINUE
                J=WK(K)
                IF (J .EQ. K) GO TO 110
                DO 100 I=1,N
                     X = A(I,J)
                     A(I,J) = A(I,K)
                     A(I,K) = X
  100           CONTINUE
                WK(K) = WK(J)
                WK(J) = J
           GO TO 90
  110 CONTINUE
      END
```

```
NT*TEST(1).LOW/ASM
    1         . REAL FUNCTION LOW(X)
    2       $(1)  AXR$
    3       LOW*  LA   A0,*0,X11
    4             JZ   A0,*NULL
    5             JP   A0,2,X11
    6             FM   A0,(0201400000000001)
    7             J    2,X11
    8       NULL  LNA  A0,(0000400000000000)
    9             J    2,X11
   10             END


PRT TEST.OLOW/ASM
```

```
NT*TEST(1).DLOW/ASM
     1          . DOUBLE PRECISION FUNCTION DLOW(X)
     2          . DOUBLE PRECISION X
     3                    AXR$
     4          $(0)
     5          FACTOR    +         0200140000000
     6                    +         0000000000001
     7          MINPOS    +         0000040000000
     8                    +         0
     9          $(1)
    10          DLOW*     DL        A0,*0,X11
    11                    JZ        A0,NULL
    12                    JP        A0,2,X11
    13                    DFM       A0,FACTOR
    14                    J         2,X11
    15          NULL      DLN       A0,MINPOS
    16                    J         2,X11
    17                    END

PRT TEST.ADD/ASM
```

```
NT*TEST(1).ADD/ASM
     1                    AXR$
     2          $(0)
     3          MINPOS    +         0000400000000
     4          FACTR1    +         0201400000001
     5          FACTR2    +         0200777777776
     6          $(1)
     7          ADD*      LA        A2,*0,X11
     8                    LA        A4,*2,X11
     9                    LA        A0,A2
    10                    FA        A0,A4
    11                    JZ        A0,M2
    12                    JP        A0,M3
    13                    FM        A0,FACTR1
    14                    J         M10
    15          M2        LNA       A0,MINPOS
    16                    J         M10
    17          M3        JN        A2,$+2
    18                    JP        A4,M13
    19                    FM        A0,FACTR2
    20          M10       LA        A2,*1,X11
    21                    LA        A4,*3,X11
    22                    SA        A0,*4,X11
    23                    LA        A0,A2
    24                    FA        A0,A4
    25                    JZ        A0,M12
    26                    JP        A0,M13
    27                    JZ        A2,M20
    28                    JP        A2,$+3
    29                    JZ        A4,M20
    30                    JN        A4,M20
    31                    FM        A0,FACTR2
    32                    SA        A0,*5,X11
    33                    J         7,X11
    34          M12       LA        A0,MINPOS
    35                    SA        A0,*5,X11
    36                    J         7,X11
    37          M13       FM        A0,FACTR1
    38          M20       SA        A0,*5,X11
    39                    J         7,X11
    40                    END

PRT TEST.MUL1/ASM
```

```
MT*TEST(1).MUL1/ASM
    1          . SUBROUTINE MUL1 (A,B1,B2,C1,C2)
    2          . (C1,C2) := A * (B1,B2)
    3       $(1)    AXR$
    4       MUL1*   LA      A0,*0,X11
    5               LA      A2,A0
    6               JN      A0,$+4
    7               FM      A0,*1,X11
    8               FM      A2,*2,X11
    9               J       $+3
   10               FM      A0,*2,X11
   11               FM      A2,*1,X11
   12               JZ      A0,M1
   13               JP      A0,$+2
   14               FM      A0,(0201400000001)
   15       M3      JZ      A2,M2
   16               JN      A2,$+2
   17               FM      A2,(0201400000001)
   18       M4      SA      A0,*3,X11
   19               SA      A2,*4,X11
   20               J       6,X11
   21       M1      LNA     A0,(0000400000000)
   22               J       M3
   23       M2      LA      A2,(0000400000000)
   24               J       M4
   25               END

PRT TEST.DADD/ASM
```

```
MT*TEST(1).DADD/ASM
    1                                  . SUBROUTINE(ALOW,AUP,BLOW,BUP,CLOW,CUP)
    2                                  . (CLOW,CUP) := (ALOW,AUP) + (BLOW,BUP)
    3               AXR$
    4       $(3)
    5       MINPOS  +       0000040000000   . KLEINSTE POSITIVE ZAHL
    6               +       0
    7       FACTR1  +       0200140000000   . KLEINSTE ZAHL > 1
    8               +       0000000000001
    9       FACTR2  +       0200077777777   . GROESSTE ZAHL < 1
   10               +       0777777777776
   11       $(1)
   12       DADD*   DL      A2,*0,X11       . A2 := ALOW
   13               DL      A4,*2,X11       . A4 := BLOW
   14               DL      A0,A2
   15               DFA     A0,A4           . A0 := A2 + A4
   16               JZ      A0,M2           . IF A0 = 0 GO TO M2
   17               JP      A0,M3           . IF A0 > 0 GO TO M3
   18               TG      A2,A4
   19               J       $+4             . IF A4 > A2
   20               DFAN    A2,A0
   21               DLN     A2,A2           . THEN A2 := A0 - A2
   22               J       $+3
   23               DFAN    A4,A0           . ELSE A4 := A0 - A4
   24               DLN     A4,A4
   25               TE      A3,A5           . IF A3 =/= A5 OR A2 =/= A4
   26               J       $+3
   27               TNE     A2,A4
   28               J       M10
   29               DFM     A0,FACTR1       . THEN A0 := A0 * FACTR1
   30               J       M10             . GO TO M10
   31                                       .
   32       M2      DA      A2,A4
   33               JP      A2,M10          . IF A2+A4 < 0
   34               DLN     A0,MINPOS       . THEN A0 := -MINPOS
   35               J       M10             . GO TO M10
   36                                       .
   37       M3      JN      A2,$+2          . IF A2 >= 0 AND A4 >= 0 GO TO M10
   38               JP      A4,M10
   39               TG      A4,A2           . IF A2 > A4
   40               J       $+4
   41               DFAN    A2,A0
   42               DLN     A2,A2           . THEN A2 := A0 - A2
   43               J       $+3
   44               DFAN    A4,A0
   45               DLN     A4,A4           . ELSE A4 := A0 - A4
   46               TE      A5,A5           . IF A3 =/= A5 OR A2 =/= A4
   47               J       $+3
   48               TNE     A2,A4
   49               J       M10
   50               DFM     A0,FACTR2       . THEN A0 := A0 * FACTR2
   51                                       .
   52                                       .
   53       M10     DL      A2,*1,X11       . A2 := ALP
   54               DL      A4,*3,X11       . A4 := BUP
   55               DS      A0,*4,X11       . CLOW := A0
   56               DL      A0,A2
   57               DFA     A0,A4           . A0 := A2 + A4
   58               JZ      A0,M12          . IF A0 = 0 GO TO M12
   59               JP      A0,M13          . IF A0 > 0 GO TO M13
   60               JZ      A2,M20          . IF A2 <= 0 AND A4 <= 0 GO TO M20
   61               JP      A2,$+3          .
   62               JN      A4,M20          .
   63               JZ      A4,M20          .
   64               TG      A2,A4           . IF A4 > A2
   65               J       $+4             .
```

```
 66                 DFAN    A2,A0           .
 67                 DLN     A2,A2           . THEN A2 := A0 - A2
 68                 J       $+3             .
 69                 DFAN    A4,A0           .
 70                 DLN     A4,A4           . ELSE A4 := A0 - A4
 71                 TE      A3,A5           . IF A3 =/= A5 OR A2 =/= A4
 72                 J       $+2             .
 73                 TNE     A2,A4           .
 74                 J       $+2             .
 75                 DFM     A0,FACTR2       . THEN A0 := A0 * FACTR2
 76                 DS      A0,*5,X11       . CUP := A0
 77                 J       7,X11           . RETURN
 78                                         .
 79       %12       DA      A2,A4           .
 80                 TG,U    A2,1            . IF A2 * A4 > 0
 81                 DL      A0,MINPOS       . THEN A0 := MINPOS
 82                 DS      A0,*5,X11       . CUP := A0
 83                 J       7,X11           . RETURN
 84                                         .
 85       %13       TG      A4,A2           . IF A2 > A4
 86                 J       $+4             .
 87                 DFAN    A2,A0           .
 88                 DLN     A2,A2           . THEN A2 := A0 - A2
 89                 J       $+3             .
 90                 DFAN    A4,A0           .
 91                 DLN     A4,A4           . ELSE A4 := A0 - A4
 92                 TE      A3,A5           . IF A3 =/= A5 OR A2 =/= A4
 93                 J       $+3             .
 94                 TNE     A2,A4           .
 95                 J       %20             .
 96                 DFM     A0,FACTR1       . THEN A0 := A0 * FACTR1
 97       %20       DS      A0,*5,X11       . CUP := A0
 98                 J       7,X11           . RETURN
 99                 END
```

PRT TEST.DMUL1/ASM

```
NT*TEST(1).DMUL1/ASM
  1                . SUBROUTINE DMUL1 (A,BL,BR,CL,CR)
  2                . DOUBLE PRECISION A,BL,BR,CL,CR
  3                . (CL,CR) := A * (BL,BR)
  4       $(3)     AXA$
  5       FACTOR   +       0200140000000
  6                +       0000000000001
  7       MINPOS   +-      0000040000000
  8                +       0000000000000
  9       $(1)
 10       DMUL1*   DL      A0,*0,X11
 11                DL      A2,A0
 12                JN      A0,$+4
 13                DFM     A0,*1,X11
 14                DFM     A2,*2,X11
 15                J       $+3
 16                DFM     A0,*2,X11
 17                DFM     A2,*1,X11
 18                JZ      A0,M1
 19                JP      A0,$+2
 20                DFM     A0,FACTOR
 21       M3       JZ      A2,M2
 22                JN      A2,$+2
 23                DFM     A2,FACTOR
 24       M4       DS      A0,*3,X11
 25                DS      A2,*4,X11
 26                J       6,X11
 27                .
 28       M1       DLN     A0,MINPOS
 29                J       M3
 30       M2       DL      A2,MINPOS
 31                J       M4
 32                END
```

PRT TEST.DMULR/ASM

```
*T*TEST(1).DMULR/ASY
     1          . SUBROUTINE DMULR (AL,AR,BL,BR,CL,CR)
     2          . DOUBLE PRECISION CL,CR
     3          . (CL,CR) = (AL,AR) * (BL,BR)
     4          $(0)     AXR$
     5          REG      RES      2
     6          $(1)
     7          DMULR*   DS       A6,RLG
     8                   FEL      A3,*0,X11
     9                   JN       A0,M20
    10                   FEL      A2,*1,X11
    11                   FEL      A4,*2,X11
    12                   JN       A4,M11
    13                   DFM      A4,A0
    14                   J        M12
    15          M11      DFM      A4,A2
    16          M12      FEL      A6,*3,X11
    17                   JN       A6,M13
    18                   DFM      A6,A2
    19                   J        M14
    20          M13      DFM      A6,A0
    21          M14      DS       A4,*4,X11
    22                   DS       A6,*5,X11
    23                   DL       A6,REG
    24                   J        7,X11
    25                   .
    26          M20      DL       A2,A0
    27                   FEL      A0,*1,X11
    28                   JP       A0,M100
    29                   FEL      A4,*2,X11
    30                   JN       A4,M21
    31                   DFM      A4,A0
    32                   J        M22
    33          M21      DFM      A4,A2
    34          M22      FEL      A6,*3,X11
    35                   JN       A6,M23
    36                   DFM      A6,A2
    37                   J        M24
    38          M23      DFM      A6,A0
    39          M24      DS       A6,*4,X11
    40                   DS       A4,*5,X11
    41                   DL       A6,REG
    42                   J        7,X11
    43                   .
    44          M100     TP       *2,X11
    45                   J        M101
    46                   FEL      A4,*3,X11
    47                   DFM      A0,A4
    48                   DFM      A2,A4
    49                   DS       A0,*5,X11
    50                   DS       A2,*4,X11
    51                   DL       A6,REG
    52                   J        7,X11
    53                   .
    54          M101     TN       *3,X11
    55                   J        M110
    56                   FEL      A4,*2,X11
    57                   DFM      A0,A4
    58                   DFM      A2,A4
    59                   DS       A0,*4,X11
    60                   DS       A2,*5,X11
    61                   DL       A6,REG
    62                   J        7,X11
    63                   .
    64          M110     FEL      A4,*3,X11
    65                   FEL      A6,*2,X11
    66                   DFM      A4,A2
    67                   DFM      A6,A0
    68                   DS       A4,*4,X11
    69                   DAN      A4,A6
    70                   JN       A4,M111
    71                   DS       A6,*4,X11
    72          M111     FEL      A4,*3,X11
    73                   FEL      A6,*2,X11
    74                   DFM      A4,A0
    75                   DFM      A6,A2
    76                   DS       A4,*5,X11
    77                   DAN      A4,A6
    78                   JP       A4,M112
    79                   DS       A6,*5,X11
    80          M112     DL       A6,REG
    81                   J        7,X11
    82                   END

PRT TEST.LGLG/1
```

```
@F*TEST(1).AKADD
    1       . SUBROUTINE AKADD (AKKU,D)
    2       . INTEGER AKKU(63)
    3       . DOUBLE PRECISION D
    4       $(1)    AXRS
    5       D0      RES     1
    6       D1      RES     1
    7       D2      RES     1
    8       $(1)
    9       AKADD*  DFU     A1,*1,X11
   10               LA,U    A0,3
   11               UI,U    A0,39
   12               TG,U    A1,9
   13               J       M1
   14               LDSL    A2,2,A1
   15               SA      A2,D1
   16               SSA     A2,35
   17               DSA     A2,2,A1
   18               LSSC    A3,0,A1
   19               SA      A3,D0
   20               LA,U    A1,2
   21               J       M2
   22       M1      DL      A4,A2
   23               SSA     A4,35
   24               LDSC    A4,2,A1
   25               SSC     A5,2
   26               SA      A5,D0
   27               TE,U    A1,33
   28               J       $+3
   29               LDSC    A2,1
   30               J       $+3
   31               LNA     A1,A1
   32               DSA     A2,32,A1
   33               SA      A2,D2
   34               SSA     A2,35
   35               DSA     A2,2
   36               SA      A3,D1
   37               LA,U    A1,3
   38       M2      AA,U    A0,*0,X11
   39               LXI,U   A0,1
   40               LA      A2,0,A0
   41               AA      A2,D0
   42               LA      A4,A2
   43               SSA     A4,34
   44               JZ      A4,$+5
   45               JN      A4,$+3
   46               ANA     A2,(0200000000000)
   47               J       $+2
   48               AA      A2,(0200000000000)
   49               SA      A2,0,*A0
   50               LA      A2,0,A0
   51               AA      A2,A4
   52               AA      A2,D1
   53               LA      A4,A2
   54               SSA     A4,34
   55               JZ      A4,$+5
   56               JN      A4,$+3
   57               ANA     A2,(0200000000000)
   58               J       $+2
   59               AA      A2,(0200000000000)
   60               SA      A2,0,*A0
   61               JNB     A1,M20
   62               LA      A2,3,A0
   63               AA      A2,A4
   64               AA      A2,D2
   65               LA      A4,A2
   66               SSA     A4,34
   67               JZ      A4,$+5
   68               JN      A4,$+3
   69               ANA     A2,(0200000000000)
   70               J       $+2
   71               AA      A2,(0200000000000)
   72               SA      A2,0,*A0
   73       M20     JZ      A4,3,X11
   74               LA      A2,0,A0
   75               AA      A2,A4
   76               LA      A4,A2
   77               SSA     A4,34
   78               JZ      A4,$+5
   79               JN      A4,$+3
   80               ANA     A2,(0200000000000)
   81               J       $+2
   82               AA      A2,(0200000000000)
   83               SA      A2,0,*A0
   84               J       M20
   85               END
```

```
VT*TEST(1).AKINT
     1          . SUBROUTINE AKINT (AKKU,AL,AR)
     2          . INTEGER AKKU(L)
     3          S(1)     AXR$
     4          L        EQU      63
     5          AKINT*   LR,U     R1,L
     6                   LA,U     A0,*0,X11
     7                   LXI,XU   A0,-1
     8                   LA,U     A2,0
     9                   SNE      A2,L-1,*A0
    10                   J        NULL
    11                   LA       A2,L,A0
    12          M100     INZ      R1
    13                   J        M600
    14                   LA       A3,L-1,A0
    15                   JN       A2,M105
    16                   JP       A3,M110
    17                   AA,XU    A2,-1
    18                   AA       A3,(020000000000000)
    19                   J        M108
    20          M105     JN       A3,M110
    21                   JZ       A3,M110
    22                   AA,XU    A2,1
    23                   ANA      A3,(020000000000000)
    24          M108     JNZ      A2,M110
    25                   LA       A2,A3
    26                   AA,XU    A0,-1
    27                   LA       A5,R1
    28                   AA,XU    A5,-1
    29                   SA       A5,R1
    30                   J        M100
    31          .
    32          M110     SSC      A3,36
    33                   JNZ      A3,$+3
    34                   JP       A2,$+2
    35                   LNA,U    A3,0
    36                   DLSC     A2,A2
    37                   OSA      A2,3
    38                   LA,U     A1,36
    39                   MSI      A1,R1
    40                   ANA      A1,A4
    41                   ANA,U    A1,721
    42                   JN       A1,M600
    43                   TG,U     A1,256
    44                   J        OVFL
    45                   LCF      A1,A2
    46                   JZ       A3,M400
    47                   JN       A2,M300
    48          M200     SA       A2,*1,X11
    49                   FM       A2,(020140000000001)
    50                   SA       A2,*2,X11
    51                   J        4,X11
    52          .
    53          M300     SA       A2,*2,X11
    54                   FM       A2,(020140000000001)
    55                   SA       A2,*1,X11
    56                   J        4,X11
    57          .
    58          M400     TG,U     A4,7
    59                   J        M410
    60                   LA       A3,L-1,A0
    61                   JZ       A3,M410
    62                   JN       A3,M300
    63                   J        M200
    64          M410     LA       A5,R1
    65                   AA,XU    A5,-1
    66                   SA       A5,R1
    67                   LA,U     A3,0
    68                   SNE      A3,L-2,*A0
    69                   J        EXAKT
    70                   LA       A3,L-1,A0
    71                   JN       A3,M430
    72                   SA       A2,*1,X11
    73                   JN       A2,M415
    74                   FM       A2,(020140000000001)
    75                   SA       A2,*2,X11
    76                   J        4,X11
    77          .
    78          M415     FM       A2,(020037777777776)
    79                   SA       A2,*2,X11
    80                   J        4,X11
    81          .
    82          M430     SA       A2,*2,X11
    83                   JP       A2,M435
    84                   FM       A2,(020140000000001)
    85                   SA       A2,*1,X11
    86                   J        4,X11
    87          .
    88          M435     FM       A2,(020037777777776)
    89                   SA       A2,*1,X11
    90                   J        4,X11
    91          .
    92          NULL     SZ       *1,X11
    93                   SZ       *2,X11
    94                   J        4,X11
    95          .
    96          EXAKT    SA       A2,*1,X11
    97                   SA       A2,*2,X11
    98                   J        4,X11
    99          .
   100          M600     LA       A3,(000004000000000)
   101                   JN       A2,M610
   102                   SZ       *1,X11
   103                   SA       A3,*2,X11
   104                   J        4,X11
   105          .
   106          M610     SNA      A3,*1,X11
   107                   SZ       *2,X11
   108                   J        4,X11
   109          .
   110          OVFL     LA       A3,(037777777777776)
   111                   FA       A3,A3
   112                   END

FIN
```

L e b e n s l a u f

Siegfried M. R u m p

geboren am 3. März in Wuppertal als Sohn des Hubert Rump und seiner Ehefrau Elisabeth Rump geb. Ghiselli

Schulbildung	1961 - 1965 Volksschule Kirn 1965 - 1972 Neusprachliches und Naturwissenschaftliches Gymnasium in Kirn
Reifeprüfung	im Juni 1972 am Neusprachlichen und Naturwissenschaftlichen Gymnasium in Kirn
Studium	der Mathematik an der Universität Kaiserslautern vom Sommersemester 1972 bis zum Wintersemester 1976/77
Diplomprüfung	im Fach Mathematik im Februar 1977
Berufstätigkeit	seit Dezember 1977 am Institut für Angewandte Mathematik der Universität Karlsruhe
Eheschließung	am 6. Juli 1979 mit Angelika, geb. Vierl