An Interval Algorithm for Solving Systems of Linear Equations to Prespecified Accuracy - Ein Intervalalgorithmus für die Lösung von linearen Gleichungssystemen mit vorausgewählter Genauigkeit

James W. Demmel
University of California, Berkeley

Fritz Krückeberg
Gesellschaft für Mathematik und Datenverarbeitung

ABSTRACT

We describe an interval arithmetic algorithm for solving a special class of simultaneous linear equations. This class includes but is not limited to systems $Ax=b$ where A and b have integer entries. The algorithm uses fixed point arithmetic, and has two properties which distinguish it from earlier algorithms: given the absolute accuracy ϵ desired, the algorithm uses only as much precision as needed to achieve it, and the algorithm can adjust its own parameters to minimize computation time.

Wir beschreiben einen Intervalalgorithmus, der eine gewisse Klasse von linearen Gleichungssystemen löst. Diese Klasse enthält u. a. Systeme $Ax=b$, bei denen A und b ganzzahlige Komponenten haben. Dieser Algorithmus verwendet Festpunktarithmetik und unterscheidet sich von früheren Algorithmen wie folgt. Erstens: Bei Vorgabe der gewünschten absoluten Genauigkeit ϵ des Ergebnisses, benötigt der Algorithmus nur so viel Zwischengenauigkeit wie notwendig, um die Fehlerschranke ϵ zu erreichen. Zweitens kann der Algorithmus selbststeuernd seine eigenen Parameter dynamisch ändern, um die Rechenzeit zu minimieren.

AMS classification: 65G10 (primary), 65F05 (secondary)
Keywords: systems of linear equations, interval arithmetic, fixed point arithmetic

July 6, 1983

1. Introduction

In this paper we describe an interval arithmetic algorithm for solving a special class of systems of simultaneous linear equations. This class includes but is not limited to systems $A\mathbf{x}=\mathbf{b}$ where A and $\mathbf{b}$ have integer entries. (Capital italic letters denote matrices and lower case bold letters denote vectors.) The algorithm uses fixed point arithmetic where the precision may be chosen by the program. Our algorithm has two properties which distinguish it from previous interval linear system solvers:

(1) Given the absolute accuracy ϵ desired in the solution, the algorithm uses only as much precision as needed to achieve it.

(2) The algorithm can adjust its own parameters to minimize computation time.

Our research was motivated by ongoing work in Petri nets [1] at the Gesellschaft für Mathematik und Datenverarbeitung[2]. Several decision problems in Petri nets can be reduced to deciding if a particular linear system $A\mathbf{x}=\mathbf{b}$ with integer A and $\mathbf{b}$ has a nonnegative integer solution vector [2]. By solving $A\mathbf{x}=\mathbf{b}$ with absolute accuracy $\epsilon<1$, we produce an interval vector $\mathbf{x}$ such that $A^{-1}\mathbf{b} \equiv \hat{\mathbf{x}} \in \mathbf{x}$ and the width of each component interval x_i is less than 1. Thus at most one vector $\mathbf{x}_I$ of integer entries can lie within x, and by testing to see if $\mathbf{x}_I$ is nonnegative and satisfies $A\mathbf{x}=\mathbf{b}$, we may answer our decision problem.

By properly scaling we may convert the Hilbert matrix $H_{ij} = 1/(i+j-1)$ to a matrix of integer entries. The Hilbert matrix is well known to be very ill-conditioned for inversion [3], and so provides a good test case for our algorithm. We will present numerical results later in Section 5. Needless to say, any problem $A\mathbf{x}=\mathbf{b}$ where A and $\mathbf{b}$ have rational entries can be scaled so they have integer entries.

Our assumptions and limitations in this paper are as follows:

(1) Our underlying arithmetic delivers results of the operations add/subtract and multiply to within an *absolute* precision chosen by the program (the program will choose the precision once at the beginning of the computation and not change it). Thus addition/subtraction may be performed without rounding error if the precision of the result

is no smaller than the precision of the operands. Fixed point arithmetic has these properties, and for our numerical tests we used a variable precision fixed point format (see section 5). This requirement implies that we can compute inner products to a given absolute accuracy.

(2) We consider only problems $A\mathbf{x}=\mathbf{b}$ where we can find an approximate (nonsingular) inverse matrix B of A with the following properties:

2.1 The matrix $R \equiv I - BA$ and the vector $\mathbf{c} \equiv B\mathbf{b}$ are exactly representable (i.e. without roundoff) in our number system.

2.2 $\| R \|_\infty \equiv \max_i \sum_j |R_{ij}|$ = max-row-sum norm <1.

Condition 2.1 holds if A and $\mathbf{b}$ have integer entries because if we round B to fit in our fixed point number system (and if integers are exactly representable) then we may then compute BA, $I-BA$, and $B\mathbf{b}$ without error by assumption (1). Condition 2.2 is needed to guarantee that $|R|$ is a contraction, that is $|R|^n \rightarrow 0$ as $n \rightarrow \infty$. A condition similar to 2.2 is required for convergence by virtually all interval arithmetic algorithms, as we will discuss below.

We do not care how B is computed. It may, for example, be computed using standard floating point library routines and then rounded to fit in our fixed point format.

The benefits of these assumptions are the following:

(1) The condition number of the problem ($\|A\|_\infty \|A^{-1}\|_\infty$) does not determine the limiting accuracy of the algorithm. The condition number will determine if we can find an approximate inverse B so that $R = I - BA$ has norm less than 1, but as long as we can find such a B, we can compute $A^{-1}\mathbf{b}$ to as much absolute accuracy ϵ as desired (limited, of course, by the accuracy available in the underlying arithmetic). This is in contrast to other interval arithmetic algorithms using a fixed, problem independent amount of precision in which the accuracy achievable degrades as the condition number grows [5, 6].

(2) We may decide ahead of time (that is after computing B, R, and c but before starting the algorithm proper) exactly how much precision we need to use to achieve the desired accuracy ϵ. In fact, there are some parameters in the algorithm which may be chosen to minimize the precision needed as a function of $\| R \|_{\infty}$ and ϵ. By tuning the algorithm to the problem this way we can save time and possibly memory if a variable width fixed point format is used.

Let us put this algorithm into historical perspective. Almost all interval linear system solvers, including ours, convert the original problem $A\mathbf{x}=\mathbf{b}$ into an iteration of the form

$$\mathbf{x}_{n+1} = R\mathbf{x}_n + \mathbf{c} \tag{1}$$

for some matrix R and vector $\mathbf{c}$. In our case we obtain (1) by multiplying $A\mathbf{x}=\mathbf{b}$ by B on both sides and adding $R\mathbf{x}$ to both sides to obtain the equivalent system

$$\mathbf{x} = R\mathbf{x} + \mathbf{c} \tag{2}$$

(equivalent, that is, as long as B is nonsingular). A necessary and sufficient condition in exact arithmetic for the intervals $\mathbf{x}_n$ defined in (1) to converge to the solution $\hat{\mathbf{x}}$ of (2) for any $\mathbf{x}_0$ is that $|R|$ be a contraction, that is that every eigenvalue of $|R|$ be less than 1 in absolute value. In practice we use the sufficient condition that some norm of $|R|$ be less than 1 (in our case we use the max-row-sum norm for reasons that will be clear later), and indeed all iterative linear equation solvers make such an assumption explicitly or implicitly, even if the form of (1) used is slightly different [7].

A general linear system solver does not assume that R and c can be computed exactly, as we do. Thus, in general, they are intervals. The width of c is a lower bound on the width of all $\mathbf{x}_{n+1}$. Typically, the width of the intervals in R will depend on the condition number of A [6]. Thus, the width of $\mathbf{x}_{n+1}$ will depend on the condition number of the problem, and, it turns out, of the size of the solution $\mathbf{x}$ itself. Wongwises [5] shows that naive use of (1) does indeed produce solution intervals whose width is proportional to the condition number of A, a performance limitation shared by Gaussian elimination without any iterative improvement at all. Cleverer use of (1) can deliver the answer to an accuracy equivalent to roundoff error in the largest components of the solution in most cases, but there is still a decline in accuracy as the condition number of A gets too large [6, 8]. This inescapable dependence of accuracy on condition number is reflected in our algorithm by requiring higher precision to be able to deliver a desired accuracy ϵ if $||R||_\infty$ is close to 1, but by assuming R and $\mathbf{c}$ to be exact, we eliminate the complicated dependence of the achievable accuracy on the size the answer (which we do not, after all, know ahead of time) and on the width of the intervals in R and $\mathbf{c}$. In fact, the proof of our algorithm exploits the ease of determining the width of $\mathbf{x}_{n+1}$ from $\mathbf{x}_n$.

As a final historical comment, we note that almost every iterative interval algorithm has assumed that the initial interval vector $\mathbf{x}_0$ contains the solution, and then iterates as follows:

$$\mathbf{x}_{n+1} = (R\mathbf{x}_n + c) \bigcap \mathbf{x}_n \quad . \tag{3}$$

$\mathbf{x}_0$ is often determined by using a very coarse approximate interval inverse matrix guaranteed to contain the actual inverse of A, and then multiplying this matrix by $\mathbf{b}$ [5, 6]. This kind of iteration is shown in Figure 1.

Our algorithm, in contrast, makes no assumption about the accuracy of $\mathbf{x}_0$, but only about its width. Our algorithm works by taking an $\mathbf{x}_n$ of width less than ϵ, artificially expanding it by adding a carefully chosen constant to the right endpoints and subtracting it from the left endpoints of each component interval, and then contracting this expanded interval using (1) several times to generate an $\mathbf{x}_{n+1}$ also of width less than ϵ. This kind of iteration is shown in Figure 2. This artificial expansion combined with the contraction (1) guarantees that the solution $\hat{\mathbf{x}}$ eventually lies in some $\mathbf{x}_n$. The algorithm terminates when $\mathbf{x}_{n+1}$ lies within the expanded version of $\mathbf{x}_n$. Choosing the parameters such as precision, how much to artificially expand $\mathbf{x}_n$, and how many times to repeat (1) to guarantee both $\text{span}(\mathbf{x}_{n+1}) < \epsilon$ and that the algorithm terminates precisely when it has found the solution constitutes the proof of the algorithm, given in Section 3.

This expansion technique is also used by Rump [9]. He expands his intervals by a certain fraction of their size, where he determined the fraction empirically as the one which gave best convergence [4]. We must choose the amount by which we expand subject to mathematical constraints given below; this careful choice of expansion leads to the guaranteed performance described in the theorem below. Our algorithm also differs from Rump's in its restriction to integer problems and its exploitation of variable precision arithmetic, using only as much precision as required to solve the problem to the required accuracy.

We have not attempted to compare our algorithm to other ones designed specifically for solving integer problems, since these algorithms may operate on entirely different principles than seeking a contraction and iterating with it. Integer Gaussian Elimination [10], for example, consists of a triangular factorization of A scaling the factors to keep them integer and exact. The forward and backward substitutions can also be done exactly by scaling. Alternatively, one may solve $A\mathbf{x}=\mathbf{b}$ in modular arithmetic using several relatively prime moduli, and retrieve the solution using the Chinese remainder theorem [11]. Our intention in this paper was to explore the benefits of taking a standard interval arithmetic approach and tuning it for integer problems.

In Section 2 we define our notation, in Section 3 we state our algorithm and prove it has the stated properties, in Section 4 we show how to choose certain parameters of the algorithm to minimize the computation time, and in Section 5 we present numerical results.

2. Notation

$\mathbf{F}_d$ denotes the set of fixed point numbers $\{nd: n$ is an integer and $d \leq 1$ is the distance between adjacent numbers$\}$ over which we perform our computations. Typically d will be 1 over a power of the radix (e.g. $1/2^j$ or $1/10^j$). We assume that $1/d$ is an integer so that that the integers are exactly representable in $\mathbf{F}_d$. $\mathbf{I}_d$ denotes the set of intervals over $\mathbf{F}_d$. Scalars in $\mathbf{F}_d$ and $\mathbf{I}_d$ will be represented by lower case italic letters. $w \in \mathbf{I}_d$ can be represented $w = [\underline{w}, \overline{w}]$ where $\underline{w}$, $\overline{w} \in \mathbf{F}_d$ and $\underline{w} \leq \overline{w}$. The span of an interval w is $\text{span}(w) = \overline{w} - \underline{w}$. $\mathbf{F}_d^n$, $\mathbf{F}_d^{n,m}$, $\mathbf{I}_d^n$, and $\mathbf{I}_d^{n,m}$ denote n-vectors and n by m matrices over $\mathbf{F}_d$ and $\mathbf{I}_d$. Vectors will be written as lower case bold letters, and matrices as capital italic letters. If $\mathbf{w} \in \mathbf{I}_d^n$, then $\text{span}(\mathbf{w}) = \max_i \text{span}(w_i)$, and similarly for $\text{span}(R)$, $R \in \mathbf{I}_d^{n,m}$. All quantities will be assumed to consists of intervals, unless we wish to emphasize that a certain variable is a point interval, i.e. an interval of span zero. Such variables will be written with a point over them, like $\dot{R}$ and $\dot{\mathbf{c}}$.

$\dot{\mathbf{x}}$ will denote the solution of $bx = \dot{R}\, bx + \dot{\mathbf{c}}$.

$||\mathbf{x}||_\infty$ will denote the infinity norm of the vector $\mathbf{x}$:

$$||\mathbf{x}||_\infty \equiv \max_i |x_i|$$

and $||R||_\infty$ will denote the matrix norm induced by this vector norm, the max-row-sum norm:

$$||R|| \equiv \sup_{\mathbf{x}\neq 0} \frac{||R\mathbf{x}||_\infty}{||\mathbf{x}||_\infty} = \max_i \sum_j |R_{ij}| \ .$$

The definitions of addition, subtraction, and multiplication of intervals can be found in the literature [12]. Since we are interested in controlling the amount of precision used in our calculations, we use a notation for interval arithmetic which explicitly displays the precision: int('expression',d) denotes the result of evaluating the 'expression' in interval arithmetic over $\mathbf{I}_d$. In our application 'expression' will only contain additions, subtractions, and multiplications, so since we are assuming unbounded range in $\mathbf{F}_d$, the value of 'expression' is always well defined. 'Expression' may contain scalars, vectors, and matrices. If 'expression' is a single variable, the operation int('expression',d) only involves rounding outward.

We will need one more piece of notation to denote the artificial expansion of an interval mentioned above. If $w = [\underline{w}\ , \overline{w}] \in \mathbf{I}_d$, then

$$\text{roundout}(w, d, r) \equiv [\underline{w} - rd\ , \overline{w} + rd] \in \mathbf{I}_d \ .$$

r is the integer number of interpoint distances d to round out.

3. The Algorithm

The algorithm is as follows:

```
4.1   Input $\epsilon$, $\dot{\mathbf{c}}$, $\dot{R}$, $\mathbf{x}_0$ ($\mathrm{span}(\mathbf{x}_0) \le \epsilon$ and $n$ is the dimension of $R$)
4.2   Compute parameters $m$, $t$, $d$, and $e$ according to the theorem
4.3   $i := 0$
4.4   repeat
4.5       $i := i+1$
4.6       $\mathbf{w} := \mathrm{roundout}(\mathbf{x}_{i-1}, t, d)$
4.7       $\mathbf{x}_i := \mathbf{w}$
4.8       for $j := 1$ to $m$ do
4.9           $\mathbf{x}_i := \mathrm{int}(\mathrm{int}(\dot{R}\mathbf{x}_i + \dot{\mathbf{c}}, de), d)$
4.10  until $\mathbf{x}_i \subseteq \mathbf{w}$
4.11  Output $\mathbf{x}_i$
```

The subscript i is only needed for stating the theorem below; $\mathbf{x}_i$ may be written over $\mathbf{x}_{i+1}$. The expression in (4.9) indicates that $\dot{R}\mathbf{x}_i + \dot{\mathbf{c}}$ is to be computed in $\mathbf{I}_{de}$ ($e \trianglelefteq 1$) and the result rounded out to fit in $\mathbf{I}_d$.

The parameters m, t, d, and e have the following meanings:

d determines the precision ($\mathbf{F}_d$) in which we will represent our data ($\dot{R}$, $\dot{\mathbf{c}}$, and $\mathbf{x}$),

e is the extra precision used to compute the inner products in (4.9) ($e \trianglelefteq 1$ means $\mathbf{F}_{de}$ is at least as precise as $\mathbf{F}_d$; if d is a power of the radix (e.g. $(1/2)^j$ or $(1/10)^j$ for some integer j) then we may also take e to be a power of the radix,

t (integer) determines by how many units of d we round $\mathbf{x}_{i-1}$ out in (4.6), and

m (integer) is the iteration count in (4.8).

The following theorem describes how the algorithm works.

Theorem: Given an arbitrary $\epsilon > 0$, an arbitrary starting vector $\mathbf{x}_0$ (such that $\mathrm{span}(\mathbf{x}_0) < \epsilon$), $\dot{R}$ (such that $r \equiv \|\dot{R}\|_\infty < 1$), and $\dot{\mathbf{c}}$, it is possible to choose the parameters d, e, t, and m so that the algorithm generates a sequence of interval vectors $\{\mathbf{x}_i\}$ with the following properties:

(5.1) $\operatorname{span}(\mathbf{x}_i)<\epsilon$ for all i. Thus, as soon as we know $\dot{\mathbf{x}}\in\mathbf{x}_i$, $\mathbf{x}_i$ is our answer.

(5.2) $\dot{\mathbf{x}}\in\mathbf{x}_i$ must occur for some finite i. Thus, the algorithm must eventually find $\dot{\mathbf{x}}$.

(5.3) $\dot{\mathbf{x}}\in\mathbf{x}_i$ implies the algorithm terminates on the next iteration. Combined with property (2) above, this property means the algorithm is finite.

(5.4) If the algorithm terminates, the final $\mathbf{x}_i$ must contain $\dot{\mathbf{x}}$.

In short, it is possible to choose the parameters to compute an arbitrarily narrow interval vector $\mathbf{x}_i$ containing the solution $\dot{\mathbf{x}}$. In addition, we will see how to choose the parameters to minimize the cost of computation in a problem dependent way.

Proof: More succinctly, the four properties above are:

(5.1') $\operatorname{span}(\mathbf{x}_i) < \epsilon \rightarrow \operatorname{span}(\mathbf{x}_{i+1}) < \epsilon$,

(5.2') $\dot{\mathbf{x}} \in \mathbf{x}_i$ occurs for some finite i,

(5.3') $\dot{\mathbf{x}} \in \mathbf{x}_i \rightarrow \mathbf{x}_{i+1} \subseteq \mathbf{w}$, and

(5.4') $\mathbf{x}_i \subseteq \mathbf{w} \rightarrow \dot{\mathbf{x}} \in \mathbf{x}_i$.

We will prove the four properties in the order 5.4', 5.2', 5.1', and 5.3'. 5.4' is an application of Brouwer's fixed point theorem [13] standard in interval analysis, and 5.2' is an application of the contraction mapping theorem.

We will then derive two inequalities in the parameters d, e, t, and m that are sufficient conditions for the validity of 5.1' and 5.3'. By solving these inequalities simultaneously for the parameters, we will prove the theorem. Since the two inequalities do not determine the four parameters uniquely, we can choose the parameters to minimize the cost of computation.

To prove 5.4', note that if F is a continuous mapping of a compact interval in $\mathbf{R}^n$ to a subset of itself, then it must have a fixed point in that subset by Brouwer's theorem. In our case F is the mapping $\mathbf{x}\rightarrow R\mathbf{x} + \mathbf{c}$ composed with itself m times, the compact interval is $\mathbf{w}$, and the subset is $\mathbf{x}_i$. Since the $\mathbf{x}_i$ computed in (4.8 - 4.9) must contain $F(\mathbf{w})$ by the properties of interval arithmetic, (4.10) does indeed guarantee that $F(\mathbf{w}) \subseteq \mathbf{x}_i \subseteq \mathbf{w}$, so the conditions of Brouwer's theorem are satisfied, implying F has a fixed point. It is easy to see that this fixed point is the solution of the equation $\mathbf{x} = R\mathbf{x} + \mathbf{c}$, proving 5.4'.

To prove 5.2$'$, we use the contraction mapping theorem [14]. Let $\dot{\mathbf{x}}_0$ be any point in $\mathbf{x}_0$, and let $\dot{\mathbf{x}}_i$ be the point image of $\dot{\mathbf{x}}_{i-1}$ under F, where F was defined in the last paragraph. Then since

$$\dot{\mathbf{x}}_i = R^m \dot{\mathbf{x}}_{i-1} + \dot{\mathbf{c}}' \tag{6}$$

for some constant vector $\dot{\mathbf{c}}'$ we have

$$\|\dot{\mathbf{x}}_i - \dot{\mathbf{x}}\|_\infty \le r^m \|\dot{\mathbf{x}}_{i-1} - \dot{\mathbf{x}}\|_\infty \le r^{mi} \|\dot{\mathbf{x}}_0 - \dot{\mathbf{x}}\|_\infty \ . \tag{7}$$

Since we use interval arithmetic, the point vector $\dot{\mathbf{x}}_i$ is a member of the interval vector $\mathbf{x}_i$.

Thus, since we assumed $r<1$, some point in $\mathbf{x}_i$ is eventually closer to the solution $\dot{\mathbf{x}}$ than td (here we use the fact that t is an integer ≥ 1). Then, the next time through the main loop 4.4 - 4.10, $\mathbf{x}_i$ will be rounded out far enough so that $\mathbf{w}$ contains $\dot{\mathbf{x}}$. Since $\dot{\mathbf{x}} \in \mathrm{F}(\mathbf{w}) \subseteq \mathbf{x}_i$ ($\dot{\mathbf{x}}$ being a fixed point of F), 5.2$'$ is satisfied. It is easy to see that the i for which 5.2$'$ is true is bounded by

$$i \le \frac{\log(td / \|\dot{\mathbf{x}}_0 - \dot{\mathbf{x}}\|_\infty)}{m \log r} + 2 \ . \tag{8}$$

We will now derive an inequality in ϵ, d, e, t and m which is a sufficient condition for 5.1$'$ to be true. Since the coordinates of $\mathbf{x}_i$ are all multiples of d, we introduce an integer variable l which satisfies

$$ld \le \epsilon \tag{9}$$

and replace 5.1$'$ by

$$\mathrm{span}(\mathbf{x}_i) < ld \rightarrow \mathrm{span}(\mathbf{x}_{i+1}) < ld \ .$$

Our inequality will be in terms of l rather than d. We take l to be an integer variable since $\mathit{WIDTH}(\mathbf{x}_i)$ must be an integer multiple of d.

Now $\mathrm{span}(\mathbf{x}_i) < ld$ means

$$\mathrm{span}(\mathbf{w}) < (l + 2t)\, d \tag{10}$$

by the definition of roundout used in 4.6.

To compute the span of $\mathbf{x}_i$ after one iteration of 4.9, we exploit the fact that R and $\mathbf{e}$ contain only point intervals, and so contribute to span($\mathbf{x}_i$) in a particularly simple way. We also exploit the properties of fixed point arithmetic which guarantee that addition and subtraction introduce no error, and bounds the absolute error in multiplication. Taken together, these facts mean

$$\mathrm{span}(\mathrm{int}(R_{ij}w_j\ ,\ de)) \le |R_{ij}|\mathrm{span}(w_j) + 2de$$

and

$$\mathrm{span}(\mathrm{int}(R_{ij}w_j + R_{ij'}w_{j'}\ ,\ de)) = \mathrm{span}(\mathrm{int}(R_{ij}w_j\ ,\ de)) + \mathrm{span}(\mathrm{int}(R_{ij'}w_{j'}\ ,\ de))\ ,$$

so after some manipulation

$$\mathrm{span}(\mathrm{int}(R\mathbf{w} + \mathbf{e}\ ,\ de)) \le r\ \mathrm{span}(\mathbf{w}) + 2nde\ . \qquad (11)$$

Rounding out again yields

$$\mathrm{span}(\mathbf{x}_i \text{ after } 4.9\ ) \le r\ \mathrm{span}(\mathbf{x}_i \text{ before } 4.9\ ) + 2d(ne+1)\ . \qquad (12)$$

After m iterations of 4.9 we get

$$\mathrm{span}(\mathbf{x}_{i+1}) \le r^m\ \mathrm{span}(\mathbf{w}) + (1 + r + \cdots + r^{m-1}) \cdot 2d(ne+1)\ . \qquad (13)$$

Substituting $(l+2t)d$ for span($\mathbf{w}$) in the R.H.S. of (13) and requiring this quantity to be less than ld (which is a sufficient condition for 5.1′) yields (after some manipulation)

$$l > \frac{2r^m}{1-r^m}t + \frac{2(ne+1)}{1-r}\ . \qquad (14)$$

(14) is our first inequality relating l, e, t and m.

We will derive our second inequality from 5.3′. $\dot{\mathbf{x}} \in \mathbf{x}_i$ means that the distance from $\dot{\mathbf{x}}$ to the edge of $\mathbf{w}$ is at least td and no more than $(l+t)d$ (see 4.6). We seek a relationship among l, d, e, t, and m that guarantees that every point of $\mathbf{x}_{i+1}$ is no farther from $\dot{\mathbf{x}}$ than td, because this will imply that $\mathbf{x}_{i+1} \subseteq \mathbf{w}$ as desired. We expect to be able to find such a relationship because the contractive property of F implies all points in $\mathbf{w}$ will approach $\dot{\mathbf{x}}$ under the action of F.

Thus, if $\dot{\mathbf{x}} \in \mathbf{x}_i$ and $\mathbf{w}$=roundout($\mathbf{x}_i$, t , d), we have

$$\sup_{\dot{y} \in \mathbf{w}} \|\dot{\mathbf{y}} - \dot{\mathbf{x}}\|_\infty \le (l+t)d \quad . \tag{15}$$

After every iteration of 4.9 we have from an analysis similar to the one leading to (14)

$$\sup_{\dot{y} \in \mathbf{x}_{i+1} \text{ after } 4.9} \|\dot{\mathbf{y}} - \dot{\mathbf{x}}\|_\infty \le r \cdot \sup_{\dot{y} \in \mathbf{x}_{i+1} \text{ before } 4.9} \|\dot{\mathbf{y}} - \dot{\mathbf{x}}\|_\infty + nde + d \quad , \tag{16}$$

so after m iterations of 4.9 we get

$$\sup_{\dot{y} \in \mathbf{x}_{i+1}} \|\dot{\mathbf{y}} - \dot{\mathbf{x}}\|_\infty \le r^m \; (l+t)d + (\frac{1-r^m}{1-r})d(ne+1) \quad . \tag{17}$$

We require that the R.H.S. of (17) be no larger than td:

$$l \le \frac{1-r^m}{r^m} t - \frac{1-r^m}{r^m(1-r)}(ne+1) \quad . \tag{18}$$

(18) is our second inequality relating l, e, t, and m.

Now we have to solve the inequalities (14) and (18) simultaneously in l, e, t, and m (note that d does not appear; we will deal with it later). We will only show here that a solution does exist, deferring to the next section a discussion of finding the best solution. We may choose any e such that $0 < e \le 1$. Considering (14) and (18) only as inequalities in l and t, we see they are both linear, and so both determine half planes in the t,l plane. It is easy to see that (14) and (18) can only have a common, positive solution in t and l if

$$\frac{1-r^m}{r^m} > \frac{2r^m}{1-r^m} \quad , \tag{19}$$

or

$$0 > (r^m)^2 + 2r^m - 1 \quad , \tag{20}$$

where $(1-r^m)/(r^m)$ is the slope of the half plane boundary line determined by (18) and $(2r^m)/(1-r^m)$ is the slope of the half plane boundary line determined by (14). We may simplify (20) and combine it with the condition $0 \le r < 1$ to get

$$m > \frac{\log(\sqrt{2} - 1)}{\log r} \tag{21}$$

($\sqrt{2}-1$ is a root of the quadratic in (20)). Having chosen m subject to this last constraint, the region of common solution of (14) and (18) is a sector in the t,l plane, and so must contain points with integer (t,l) coordinates, any of which are candidate solutions for (14) and (18). The smallest possible values of t and l are the (not necessarily integer) coordinates of the apex of the sector, yielding the lower bounds:

$$t > (ne+1)\,\frac{1-r^m}{1-r}\,\frac{1+r^m}{1-2r^m-r^{2m}}\,, \tag{22}$$

and

$$l > (ne+1)\,\frac{1-r^m}{1-r}\,\frac{2}{1-2r^m-r^{2m}}\,. \tag{23}$$

So far we have shown how to choose e, t, l and m in order to simultaneously satisfy (14) and (18). It remains to choose d. d may be any number satisfying $ld < \epsilon$, or $d < \epsilon / l$. The lower bound on l in (23) translates into an upper bound on d:

$$d < \frac{\epsilon\,(1-2r^m-r^{2m})}{2(ne+1)}\,\frac{1-r}{1-r^m}\,. \tag{24}$$

This completes the proof of the theorem. Q.E.D.

4. Choosing the parameters to minimise cost

In the proof of the theorem we showed that subject to certain inequality constraints, we could choose the parameters m, t, d, and e to make the algorithm behave as claimed. It became clear in the discussion that the constraints left some freedom in the choice of these parameters. In this section we will exploit that freedom and show how the parameters may be chosen to minimize the cost of the algorithm. In particular, we will see that the naive choice of parameters suggested by the proof of the theorem results in a cost function with poles at a countable number of values of r, so that some care really must be exercised in choosing parameters.

The cost is proportional to the product of the following four factors:

(25.1) i_M = the number of iterations of the main loop (4.4 - 4.10) of the algorithm. A bound for i_M is $i+1$, where i is bounded in (8).

(25.2) m = the number of iterations of the inner loop (4.8 - 4.9). A lower bound for m is given in (21).

(25.3) The cost of a multiplication in the inner loop 4.9. This cost is a function, mult, of the number of places, p, used in the computation. p is proportional to $\log(1/de)$, and mult can grow as slowly as $p \log p \log\log p$ [15] or as quickly as p^2 depending on the implementation of multiplication.

(25.4) The number of multiplies in the inner loop, $2n^2$.

Thus, we model the cost as follows:

$$\text{Cost} = 2n^2 \times \left\{ \frac{\log(\|\dot{\mathbf{x}}_0 - \dot{\mathbf{x}}\|_\infty / \, td)}{\log 1/r} + 3m \right\} \times \text{mult}(\log(\frac{1}{de})) \ , \tag{26}$$

where K is a constant of proportionality. We do not include the cost of computing R and e, which is n^3 (fixed point) multiplies no more expensive than those in the inner loop above, as well as $O(n^3)$ (floating point) multiplies to compute the approximate inverse B. The cost in (26) above may be less or more than the cost of obtaining R and e, depending on the initial error $\|\dot{\mathbf{x}}_0 - \dot{\mathbf{x}}\|_\infty$ and the desired precision ϵ. Here we address only the problem of minimizing the cost in (26) since naive choices of m, t, d, and e may make it much larger than necessary.

To minimize (26) exactly, we would need to know the initial error $\|\dot{\mathbf{x}}_0 - \dot{\mathbf{x}}\|_\infty$ the function mult, and the set of discrete values to which d is restricted. Since we do not know all these things in general, we just show that the following algorithm makes a reasonable choice of m, t, d, and e:

```
27.1)  $e := 1$ (i.e. no extra precision)
27.2)  Choose the smallest $m$ such that $r^m < 1/3$
27.3)  $d :=$ largest value less than the upper bound in (24)
           (recall $d$ must be a negative power of the radix)
27.4)  $l :=$ largest integer such that $l < \epsilon / d$
27.5)  given $l$, find the largest integer $t$ satisfying both (14) and (18)
27.6)  if no such $t$ exists, or if $t/l < \frac{1}{2}\frac{1-r^m}{2r^m}$,
         decrease $d$ to the next smaller value and goto (27.4)
```

The rationale behind this algorithm is as follows. From (26) we see that the cost decreases as the precision decreases (de increases) and the amount of round out in 4.6 increases (td increases). Approximating d by ϵ / l (see (9)) we see that the precision de is

$$de = \frac{\epsilon}{2} \cdot \frac{e}{ne+1} \cdot \frac{1-r}{1-r^m} \cdot (1 - 2r^m - r^{2m}) \ . \tag{28}$$

The term depending on the extra precision e is maximized at $e = 1$ (no extra precision). This justifies 27.1.

The $1-r$ factor reflects the inescapable effect of condition number on precision, since the larger the condition number of A, the closer to one r is likely to be, and so the more precision needed. ($1/(1-r^m)$ is bounded between 1 and $1/(2-\sqrt{2})$ and does not effect the cost appreciably.)

The interesting factor is $1 - 2r^m - r^{2m}$. If we choose m to be the smallest integer greater than its lower bound $\log(\sqrt{2}-1)/\log r$ (see (21)), then as a function of r, $1 - 2r^m - r^{2m}$ has a countable number of points where it approaches zero (from the left): $\{(\sqrt{2}-1)^{1/j}\}_{j=1,\infty}$ This means that if we choose m as small as possible for all r, then there are a countable number of values of r where the required precision goes to infinity! We avoid this strange behavior by simply increasing m if r^m is too close to $\sqrt{2}-1$. Increasing m by a factor of 1.25 above the minimum value, for example, guarantees that $r^m < 1/3$ and so $1 - 2r^m - r^{2m} > .22$. On examining (26), we see that this can increase the second factor by at most 1.25 while possibly decreasing the third factor much more. This justifies 27.2.

To justify the rest of the algorithm, we consider $td \approx \epsilon\, t/l$. Maximizing td implies maximizing the ratio t/l, which means choosing the point (t,l) among those points in the solution sector as close to the lower boundary as possible, in particular as far to the right (in the t,l plane) as possible. The maximum value of t/l is approached as both t and l approach $+\infty$: $(1-r^m)/2r^m$ (the reciprocal of the slope of the lower boundary; see (14)). Therefore, in line 27.4 we pick the largest possible l subject to $d < \epsilon/l$, and in 27.5 we pick the largest t for that l. Such a t may not exist if the solution sector for (14) and (18) is unfortunately located; even if it does exist t/l may be far from its maximum value $(1-r^m)/2r^m$. In either case we decrease d in 27.6 (by at least a factor of 2 in binary arithmetic, in general by a factor of the radix) and recompute l and t. The constraint $r^m < 1/3$ and the looseness of the test

$$t/l < \frac{1}{2}\,\frac{1-r^m}{2r^m}$$

guarantees that t and l will be recomputed by 27.4-27.5 at most once. This justifies the rest of the algorithm.

We may now substitute the values of m, t, d, and ϵ computed by this algorithm into our cost function (26). The upper bound on $d\epsilon$ obtained from (24) will be within a factor of $1/(6(n+1))$ of $\epsilon(1-r)$, and td will be within a factor of $1/12$ of ϵ/r^m, so (26) will be close to

$$\mathrm{Cost} \approx 2n^2 \times \left(\frac{\log(\|\dot{\mathbf{x}}_0 - \dot{\mathbf{x}}\|_\infty/\epsilon)+K_1}{\log(\frac{1}{r})}\right) \times \mathrm{mult}[\log(\frac{1}{\epsilon}) + \log(\frac{1}{1-r}) + K_2] \qquad (0$$

for modest constants K_1 and K_2. The cost goes to ∞ as either the desired precision ϵ goes to zero or the residual norm r goes to 1, as expected.

We mention the case when $r \ll 1$ (for which $m = 1$) which is likely when A is not too ill-conditioned. In this case we may pick t very large while keeping l small since the lower boundary of the solution sector is almost horizontal (slope $= 2r/(1-r)$). If we can pick t large enough so that td is larger than the initial error $\|\dot{\mathbf{x}}_0 - \dot{\mathbf{x}}\|_\infty$ then the first $\mathbf{w}$ will contain the solution $\dot{\mathbf{x}}$, and our algorithm will terminate after at most two iterations of the outer loop. If our initial guess $\dot{\mathbf{x}}_0$ is good enough (and many good interval algorithms start with a reasonably accurate solution obtained cheaply using noninterval arithmetic), then this quick termination is likely.

5. Numerical Results

To test our algorithm we tried to invert the Hilbert matrix scaled to have integer entries. Specifically, we multiplied $H_{ij} = 1/(i+j-1)$ by the least common multiple of $1, \ldots, 2n-1$ (n is the dimension of the matrix). The Hilbert matrix is known to be very ill-conditioned and so is a good test of our algorithm.

The algorithm was written in FORTRAN and executed on an IBM 370/158. The arithmetic used was also written in FORTRAN (and accessed via subroutine calls). It used a fixed point format, with 80 decimal places before and 80 places after the decimal point. Decimal digits were available in blocks of 4, making the radix effectively 10^4.

The approximate inverse B was computed using the NAG [16] scientific subroutine library. The double precision versions of F01ADF and F01ACF were used. F01ADF computes the approximate inverse of a symmetric, positive-definite matrix, such as the Hilbert matrix, and F01ACF computes an accurate inverse using iterative refinement.

In all cases, ϵ was $10^{-15} \| H^{-1} \|_\infty$ (which means that the largest element in H^{-1} was computed to a relative accuracy of at least 10^{-15}), e was taken to be 10^{-4}, m turn out to be 1 (since r was $\triangleleft 1/3$ is all cases), and $\dot{\mathbf{x}}_0$ was taken from the approximate inverse supplied by the NAGLIB routines. Our results are shown in Table 1. Column 1 (labelled n) gives the dimension of the Hilbert matrix, column 2 (labelled Routine) indicates which NAGLIB routine was used to obtain B (C for F01ACF and D for F01ADF), column 3 (labelled Cond) gives the condition number $\| H \|_\infty \| H^{-1} \|_\infty$ of the Hilbert matrix, columns 4, 6, 7, and 8 give the values of the parameters r, d, t, and l, column 5 gives $\epsilon = 10^{-15} \| H^{-1} \|_\infty$ and column 9 (labelled i_M) gives the maximum number of iterations of the algorithm's main loop used by any of the columns of the inverse.

n	Routine	Cond	r	ϵ	d	t	l	i_M
10	D	$3.5 \cdot 10^{13}$	.0084	$5.2 \cdot 10^{-11}$	10^{-12}	3	2	5
11	C	$1.2 \cdot 10^{15}$	.023	$1.8 \cdot 10^{-9}$	10^{-12}	3	2	2
12	C	$4.1 \cdot 10^{16}$	.23	$2.5 \cdot 10^{-9}$	10^{-12}	5	3	3
13	C	$1.3 \cdot 10^{18}$	20.1	$2.5 \cdot 10^{-8}$	-	-	-	-

All the inequalities and inclusions predicted in the theorem were automatically tested and verified. The values of t and l were not chosen according to algorithm 27 but as the smallest solutions of (14) and (18); our concern was not to minimize time but to verify the conclusions of the theorem. F01ADF (approximate inverse) could not be used for $n > 10$ because the B it produced was so inaccurate that r was greater than 1. For $n = 13$ r was greater than 1 even with the more accurate library routine, so we could not use the algorithm. The number of iterations i_M did not grow with n because the initial approximation $\dot{x}_0$ came from the NAGLIB inverse B and so was rather accurate to start with. Indeed, most good interval algorithms wisely attempt to attain as accurate a solution as possible using noninterval arithmetic, requiring only a few iterations of the relatively expensive interval arithmetic at the end to refine the result, and ours is no exception.

Authors' Addresses

James Demmel, Computer Science Division, EECS Department, University of California, Berkeley, CA 94707, U.S.A.

Prof. Fritz Krückeberg, Gesellschaft für Mathematik und Datenverarbeitung, Schloss Birlinghoven, D-2500 St. Augustin, Federal Republic of Germany

Acknowledgement

The authors acknowledge the kind assistance of several of Prof. Krückeberg's students who implemented the fixed point arithmetic and assisted in programming the algorithm. The authors also acknowledge the financial support of the Gesellschaft für Mathematik und Datenverarbeitung and the U. S. Department of Energy, Contract DE-AM03-76SF00034, Project Agreement DE-

AS03-79ER10358.

Bibliography

[1] Genrich, H.J., Lautenbach, K., Thiagarajan, P.S., Elements of General Net theory, in Net Theory and Applications (Lecture Notes in Comuter Science, vol 84), pp 121-164, Berlin, Springer, 1980

[2] Memmi, G., Roucairol, G., Linear Algebra in Net Theory, in Net Theory and Applications (Lecture Notes in Comuter Science, vol 84), pp 213-224, Berlin, Springer, 1980

[3] Wilkinson, J.H., Reinsch, C., Linear Algebra (Handbook for Automatic Computation, vol 2), pp 41-44, Berlin, Springer, 1971.

[4] Rump, S. M. private communication, 1982

[5] Wongwises, P., Experimentelle Untersuchungen zur Numerische Auflösung von linearen Gleichunssystemen mit Fehlererfassung, pp 316-325, in Interval Mathematics (Lecture Notes in Computer Science, vol 29), editor K. Nickel, Berlin, Springer, 1975

[6] Krier, N., Spelluci, P., Untersuchungen der Grenzgenauigkeit von Algorithmen zur Auflösung linearer Gleichungssysteme mit Fehlererfassung, pp 288-297, in Interval Mathematics (Lecture Notes in Computer Science, vol 29), editor K. Nickel, Berlin, Springer, 1975

[7] Krawczyk, R., Newton-Algorithmen zur Bestimmung von Nullstellen mit Fehlerschranken, Computing 4 , pp 187-201, (1969)

[8] Wilkinson, J.H., Rounding Errors in Algebraic Processes, Englewood Cliffs, N.J., Prentice-Hall, 1963

[9] Rump, S.M., Kleine Fehlerschranken bei Matrixproblemen, Dissertation, Universität Karlsruhe, 1980

[10] Bareiss, E.H., Sylvester's Identity and Multistep Integer-preserving Gaussian Elimination, Math. Comp., 22 , 103, pp 565-578, (July 1968)

[11] Howell, J.A., Gregory, R.T., An Algorithm for Solving Linear Algebraic Equations using Residue Arithmetic, BIT, 9 , pp 200-234, 324-337, (1969)

[12] Moore, R.E., Methods and Applications of Interval Analysis, Philadelphia, SIAM, 1979

[13] Dunford, N., Schwartz, J.T., Linear Operators, Part I, p 453, New York, Interscience, 1957

[14] Apostol, T., Mathematical Analysis, 2nd ed., p 92, Reading, Addison-Wesley, 1974 1974

[15] Aho, A.V., Hopcroft, J.E., Ullman, J.D., The Design and Analysis of Computer Algorithms, pp 270-274, Reading, Mass., Addison-Wesley, 1974

[16] NAG Reference Manual, Oxford, Numerical Algorithms Group, LTD., 1976