Preconditioners for the Interval Gauss-Seidel Method

R. Baker Kearfott

SIAM Journal on Numerical Analysis, Vol. 27, No. 3 (Jun., 1990), 804-822.

Stable URL:
http://links.jstor.org/sici?sici=0036-1429%28199006%2927%3A3%3C804%3APFTIGM%3E2.0.CO%3B2-1

SIAM Journal on Numerical Analysis is currently published by Society for Industrial and Applied Mathematics.

SIAM J. NUMER. ANAL.
Vol. 27, No. 3, pp. 804–822, June 1990

013

PRECONDITIONERS FOR THE INTERVAL GAUSS–SEIDEL METHOD*

R. BAKER KEARFOTT†

Abstract. Interval Newton methods in conjunction with generalized bisection can form the basis of algorithms that find all real roots within a specified box $\mathbf{X} \subset \mathbf{R}^n$ of a system of nonlinear equations $F(X) = 0$ *with mathematical certainty*, even in finite-precision arithmetic. In such methods, the system $F(X) = 0$ is transformed into a linear interval system $0 = F(M) + \mathbf{F}'(\mathbf{X})(\overline{\mathbf{X}} - M)$; if interval arithmetic is then used to bound the solutions of this system, the resulting box $\overline{\mathbf{X}}$ contains all roots of the nonlinear system. The interval Gauss–Seidel method is a reasonable way of finding such solution bounds.

For the overall interval Newton/bisection algorithm to be efficient, the image box $\overline{\mathbf{X}}$ should be as small as possible. To do this, the linear interval system is multiplied by a preconditioner matrix Y before the interval Gauss–Seidel method is applied. In this paper, a technique for computing such preconditioner matrices Y is described. This technique involves optimality conditions expressible as linear programming problems. In many instances, the resulting preconditioners give an $\overline{\mathbf{X}}$ of *minimal* width. They can also be applied when F' approximates a singular matrix, and the optimality conditions can be altered to describe preconditioners with a given structure. This technique is illustrated with some simple examples and with numerical experiments. These experiments indicate that the new preconditioner results in significantly less function and Jacobian evaluations, especially for ill-conditioned problems, but it requires more computation to obtain.

Key words. nonlinear algebraic systems, Newton's method, interval arithmetic, Gauss–Seidel method, global optimization, preconditioners

AMS(MOS) subject classifications. 65H10, 65G10

1. Motivation, introduction, and notation.

The general problem we address is:

Find, with certainty, approximations to all solutions of the nonlinear system

$$F(X) = (f_1(x_1, x_2, \cdots, x_n), \cdots, f_n(x_1, x_2, \cdots, x_n)) = 0, \tag{1.1}$$

where bounds λ_i and υ_i are known such that

$$\lambda_i \leq x_i \leq \upsilon_i \text{ for } 1 \leq i \leq n.$$

We write $X = (x_1, x_2, \cdots, x_n)$, and we denote the box given by the inequalities on the variables x_i by $\mathbf{B}$.

A successful approach to this problem is generalized bisection in conjunction with interval Newton methods, as described in [2], [11], [12], [5], [6], [4], [13], [14], [18], etc. In these methods, we first transform $F(X) = 0$ to the linear interval system

$$\mathbf{F}'(\mathbf{X}_k)(\overline{\mathbf{X}}_k - X_k) = -F(X_k), \tag{1.2}$$

where $\mathbf{F}'(\mathbf{X}_k)$ is a suitable (such as an elementwise) interval extension[1] of the Jacobian matrix over the box $\mathbf{X}_k$ (with $\mathbf{X}_0 = \mathbf{B}$), and where $X_k \in \mathbf{X}_k$. We note that (1.2)

*Received by the editors May 16, 1988; accepted for publication May 30, 1989.

†Department of Mathematics, University of Southwestern Louisiana, Lafayette, Louisiana 70504.

[1]Interval extensions of a function may be defined by simply evaluating the functions in interval arithmetic. The result of such a computation is an interval which contains the range of the function over the interval argument. Readers not knowledgeable in interval arithmetic may consult the introductions in [1], [12] and the recent review [7].

represents the set of all linear systems of equations $A(X - X_k) = -F(X_k)$ as A ranges over all matrices which are contained in the interval matrix $\mathbf{F}'(\mathbf{X}_k)$. If we then formally solve (1.2) using interval arithmetic, the resulting box $\overline{\mathbf{X}}_k$ will contain all solutions to all such systems $A(X - X_k) = -F(X_k)$, and hence, by the mean value theorem, will contain all solutions to $F(X) = 0$ in $\mathbf{X}_k$. We then define the next iterate $\mathbf{X}_{k+1}$ by

$$\mathbf{X}_{k+1} = \mathbf{X}_k \cap \overline{\mathbf{X}}_k. \tag{1.3}$$

This scheme is termed an *interval Newton method.*

If each row of $\mathbf{F}'$ contains all possible vector values that the corresponding row of the scalar Jacobian matrix $F'(X)$ takes on as X ranges over all vectors in $\mathbf{X}_k$, then it follows from the mean value theorem that all solutions of (1.1) in $\mathbf{B}$ must be in $\mathbf{X}_{k+1}$ for all k. If the coordinate intervals of $\mathbf{X}_{k+1}$ are not smaller than those of $\mathbf{X}_k$, then we may bisect one of these intervals to form two new boxes; we then continue the iteration with one of these boxes, and put the other one on a stack for later consideration. As explained in [12], [5], [6], and elsewhere, the following fact (from [13]) allows such a composite generalized bisection algorithm to compute all solutions to (1.1) *with mathematical certainty.* For many methods of solving (1.2),

(1.4) if $\overline{\mathbf{X}}_k \subset \mathbf{X}_k$, then the system of equations in (1.1) has a unique solution in $\mathbf{X}_k$. Conversely, if $\overline{\mathbf{X}}_k \cap \mathbf{X}_k = \emptyset$ then there are no solutions of the system in (1.1) in $\mathbf{X}_k$.

We now present a simplified version of the generalized bisection algorithm in [6].[2]

ALGORITHM 1.1.

1. (*Initialization phase*)
   (a) *Input a tolerance* ϵ *such that no box will have a coordinate width less than* ϵ.
   (b) *Input a tolerance* ϵ_F *such that we do no further computations on an* $\mathbf{X}$ *if* $\|F(X)\|_\infty < \epsilon_F$ *for* $X \in \mathbf{X}$.
   (c) $\mathbf{X}_k \leftarrow \mathbf{B}$.
2. (*Bisection*)
   (a) *If* $\mathbf{X}_k = (\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n)$, *where* $\mathbf{x}_j = [l_j, u_j]$, *then choose a coordinate* i *in which to bisect.*
   (b) *Form two new boxes* $\mathbf{X}_k^1$ *and* $\mathbf{X}_k^2$ *by replacing* $\mathbf{x}_i$ *in* $\mathbf{X}_k$ *by either* $[\omega_i, u_i]$ *or* $[l_i, \omega_i]$, *where* $\omega_i = (l_i + u_i)/2$.
   (c) *Place either* $\mathbf{X}_k^1$ *or* $\mathbf{X}_k^2$ *on a stack* $\mathcal{S}$ *for later consideration, and replace* $\mathbf{X}_k$ *with the other one.*
3. (*Interval Newton method and root storage*)
   (a) (*Test for convergence*)
      (i) *If the width at least one coordinate* $\mathbf{x}_j$ *of* $\mathbf{X}_k$ *is greater than* ϵ, *then compute the interval vector* $\mathbf{F}(\mathbf{X})$ *for use in* (ii) *below.*
      (ii) *If the width of each coordinate* $\mathbf{x}_j$ *of* $\mathbf{X}_k$ *is less than* ϵ, *or if* $\|F(X)\|_\infty < \epsilon_F$ *then*
         (α) *Store* $\mathbf{X}_k$ *in a list* $\mathcal{L}'$ *of small boxes which possibly contain roots.*
         (β) *If the stack* $\mathcal{S}$ *is empty, then stop. Otherwise, pop a box from* $\mathcal{S}$, *let that box become* $\mathbf{X}_k$, *and return to the beginning of this step.*

[2]The algorithms in [6] and [9] contain an "expansion" step to help eliminate redundancies when roots occur near a boundary of a box. However, this step complicates the presentation and is not relevant to the discussion in this paper.

(b) (*Obtain the function and Jacobian for* (1.2).)
  (i) *Compute the interval Jacobian matrix* $\mathbf{F}'(\mathbf{X}_k)$.
  (ii) *Compute* $F(M)$, *using interval arithmetic to bound the roundoff error.*

(c) (*Bound the solution set in* (1.2).) *Use some method to compute an interval enclosure* $\overline{\mathbf{X}}_k$ *to the solution set of the interval linear system* (1.1).

(d) *If* $\overline{\mathbf{X}}_k \subset \text{int}(\mathbf{X}_k)$ *where* $\text{int}(\mathbf{X}_k)$ *denotes the interior of* $\mathbf{X}_k$, *then do the following.*
  (i) *Store* $\mathbf{X}_k$ *in a list* $\mathcal{L}$ *of boxes which contain unique roots.*
  (ii) *If the stack* $\mathcal{S}$ *is empty, then stop. Otherwise, pop a box from* $\mathcal{S}$, *let that box become* $\mathbf{X}_k$, *and return to the beginning of Step* 3(a).
  (e) *If* $\overline{\mathbf{X}}_k \cap \mathbf{X}_k$ *is sufficiently smaller than* $\mathbf{X}_k$, *then replace* $\mathbf{X}_k$ *by* $\overline{\mathbf{X}}_k \cap \mathbf{X}_k$ *and return to Step* 3(a). *Otherwise, replace* $\mathbf{X}_k$ *by* $\overline{\mathbf{X}}_k \cap \mathbf{X}_k$ *and return to Step* 2.
  (f) *If* $\overline{\mathbf{X}}_k \cap \mathbf{X}_k = \emptyset$ *then stop if the stack* $\mathcal{S}$ *is empty; otherwise, pop a box from* $\mathcal{S}$, *let that box become* $\mathbf{X}_k$, *and return to the beginning of Step* 3(a).

In Step 3(e), we may say $\overline{\mathbf{X}}_k$ is sufficiently smaller than $\mathbf{X}_k$ if there is a j such that $u_j - l_j > \epsilon$ and $(\overline{u}_j - \overline{l}_j) \leq (u_j - l_j)/2$. Such a condition will ensure the overall convergence of the algorithm, since it guarantees that each step, whether interval Gauss–Seidel or bisection, will reduce one of the coordinates by at least a factor of 2.

Algorithm 1.1 must eventually complete with (possibly empty) lists of boxes $\mathcal{L}$ and $\mathcal{L}'$, such that all roots of F in $\mathbf{B}$ are contained in boxes in $\mathcal{L}$ or $\mathcal{L}'$, and each box in $\mathcal{L}$ contains a unique root; compare with the convergence analysis in [5]. However, the cost bound in [5] is very pessimistic, and the actual efficiency of Algorithm 1.1 depends on the way we find the solution $\overline{\mathbf{X}}_k$ to (1.2), i.e., the method we choose in Step 3(c). The reason Step 3(c) is important is because the widths of the component intervals of $\overline{\mathbf{X}}_k$ depend on the way in which this step is carried out. If these widths are relatively small, then they are reduced further by iteration of Step 3 (see Step 3(e)), until the condition in Step 3(d) holds. (In fact, this interval Newton method is locally quadratically convergent in the sense that the widths go to zero at that rate.) If not, then bisection reduces the size of $\overline{\mathbf{X}}_k$ slowly, especially when the dimension n is large, and many more boxes must be considered.

The overall functioning of Algorithm 1.1 also depends on the way that the coordinate i is chosen in Step 2(a), since $\overline{\mathbf{X}}_k$ in Step 3 is more sensitive to some coordinates than others. Similarly, most methods used in Step 3(c) will give smaller $\overline{\mathbf{X}}_k$ when the interval extension $\mathbf{F}'$ gives sharper bounds on the range of the entries of the Jacobian matrix. In this paper, however, we will concentrate on the method in Step 3(c), given a fixed $\mathbf{X}_k$ and a fixed interval extension $\mathbf{F}'(\mathbf{X}_k)$. Improvements that we make in this regard will not in general guarantee a quicker theoretical rate of convergence in Algorithm 1.1, but will cause it to complete more rapidly (with less boxes $\mathbf{X}_k$ considered) for many problems.

We will simply write $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n)$ for $\mathbf{X}_k$ and

$$\mathbf{A} = \{\mathbf{A}_{i,j}\}_{i,j=1}^{n} = \{[a_{i,j}, b_{i,j}]\}_{i,j=1}^{n},$$

where $\mathbf{A}_{i,j}$ is the interval in the ith row and jth column of $\mathbf{A} = \mathbf{F}'(\mathbf{X})$. Similarly, we will write[3] $F(X_k) = F = (\mathbf{f}_1, \mathbf{f}_2, \cdots, \mathbf{f}_n)$ and $X_k = M = (m_1, m_2, \cdots, m_n)$, so that

[3]We denote the components of F as boldface intervals, since they must be evaluated in interval arithmetic with directed roundings or else roundoff error may cause Algorithm 1.1 to miss a root.

(1.2) becomes

$$\mathbf{A}(\overline{\mathbf{X}}_k - M) = -F. \tag{1.5}$$

An interval version of the Gauss–Seidel method with starting guess $\mathbf{X}_k$ is a competitive way to compute the bound $\overline{\mathbf{X}}_k$ on the portion of the solution set in (1.5) which is contained in $\mathbf{X}_k$. (See [3], [4], or [13].) It is embodied in the following algorithm.

ALGORITHM 1.2. (*Interval Gauss–Seidel, for Step* 3(c) *of Algorithm* 1.1.) *Do the following for* $i = 1$ *to* n.

1. *Compute*

$$\overline{\mathbf{x}}_i = m_i - \left[\mathbf{f}_i + \sum_{\substack{j=1 \\ j \neq i}}^{n} \mathbf{A}_{i,j}(\mathbf{x}_j - m_j) \right] \Big/ \mathbf{A}_{i,i}$$

using interval arithmetic.

2. *If* $\overline{\mathbf{x}}_i \cap \mathbf{x}_i = \emptyset$, *then return to Step* 3(f) *of Algorithm* 1.1.
3. (*Prepare for the next coordinate.*)
   (a) *Replace* $\mathbf{x}_i$ *by* $\mathbf{x}_i \cap \overline{\mathbf{x}}_i$.
   (b) *Possibly reevaluate* $\mathbf{F}'(\mathbf{X}_k)$ *to replace* $\mathbf{A}$ *by an interval matrix whose corresponding widths are smaller.*

Algorithm 1.2 as stated can be guaranteed to converge without requiring Step 2 of Algorithm 1.1 only for certain problems, such as those for which an interval version of diagonal dominance holds. A simplified version of this condition is

$$\min_{x \in \mathbf{A}_{i,i}} |x| > \max_{x \in \sum_{\substack{j=1 \\ j \neq i}}^{n} \mathbf{A}_{i,j}} |x|. \tag{1.6}$$

See [16], [17], or [19] for an explanation and for examples of the use of Algorithm 1.2 when convergence is assured without bisection.

In other cases, (1.6) does not hold, but we may sometimes cause it to hold by *preconditioning* (1.5). In particular, we multiply by a matrix Y to obtain

$$Y\mathbf{A}(\overline{\mathbf{X}}_k - M) = -YF. \tag{1.7}$$

Let $Y_i = (y_1, y_2, \cdots, y_n)$ denote the ith row of the preconditioner, let $\mathbf{k}_i = Y_i F$, and let

$$\begin{aligned} Y_i\mathbf{A} = \mathbf{G}_i &= (\mathbf{G}_{i,1}, \mathbf{G}_{i,2}, \cdots, \mathbf{G}_{i,n}) \\ &= ([c_{i,1}, d_{i,1}], [c_{i,2}, d_{i,2}], \cdots, [c_{i,n}, d_{i,n}]). \end{aligned}$$

Then the resulting interval Gauss–Seidel algorithm becomes as follows.

ALGORITHM 1.3. (*Preconditioned version of interval Gauss–Seidel, for Step* 3(c) *of Algorithm* 1.1.) *Do the following for* $i = 1$ *to* n.

1. (*Update a coordinate.*)
   (a) *Compute* Y_i.
   (b) *Compute* $\mathbf{k}_i$ *and* $\mathbf{G}_i$.
   (c) *Compute*

$$\overline{\mathbf{x}}_i = m_i - \left[\mathbf{k}_i + \sum_{\substack{j=1 \\ j \neq i}}^{n} \mathbf{G}_{i,j}(\mathbf{x}_j - m_j) \right] \Big/ \mathbf{G}_{i,i} \tag{1.8}$$

using interval arithmetic.

2. (*The new box is empty.*) *If* $\overline{\mathbf{x}}_i \cap \mathbf{x}_i = \emptyset$, *then return to step* 3(f) *of Algorithm* 1.1.
3. (*The new box is nonempty; prepare for the next coordinate.*)
   (a) *Replace* $\mathbf{x}_i$ *by* $\mathbf{x}_i \cap \overline{\mathbf{x}}_i$.
   (b) *Possibly reevaluate* $\mathbf{F}'(\mathbf{X}_k)$ *to replace* $\mathbf{A}$ *by an interval matrix whose corresponding widths are smaller.*

The following theorem shows us that Algorithm 1.3 in conjunction with Algorithm 1.1 will produce lists of boxes $\mathcal{L}$ and $\mathcal{L}'$ which contain all roots of F in $\mathbf{B}$.

THEOREM 1.4. *Let* $\mathbf{X}^+ = (\mathbf{x}_1^+, \mathbf{x}_2^+, \cdots, \mathbf{x}_n^+)$ *denote the new (possibly altered and possibly empty) box* $\mathbf{X}$ *which Algorithm* 1.3 *returns, and refer to the* $\mathbf{X}$ *entering Algorithm* 1.3 *as simply* $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n)$. *Then any roots of* F *in* $\mathbf{X}$ *must also be in the new* $\mathbf{X}^+$.

Proof of Theorem 1.4. Suppose $X^* = (x_1^*, x_2^*, \cdots, x_n^*) \in \mathbf{X}$ is such that $F(X^*) = 0$. Then, since $\mathbf{A}$ is a rowwise interval extension of the Jacobian matrix of F, there is an $A \in \mathbf{A}$ such that

$$0 = F(X^*) = F(M) + A(X^* - M).$$

Denote the element in the ith row and jth column of YA by $G_{i,j}$. Then, when we multiply the above equation by Y and write down the ith row of it, after some rearranging we obtain

$$G_{i,i}(x_i^* - m_i) = -\left[\mathbf{k}_i + \sum_{\substack{j=1 \\ j \neq i}}^{n} G_{i,j}(x_j^* - m_j)\right].$$

If $G_{i,i} \neq 0$, then we may solve the above formula for x_i^*; when we compare the result with (1.8), the fact that the interval arithmetic result contains all possible scalar results we can obtain by selecting elements of the intervals as operands implies $x_i^* \in \mathbf{x}_i^+$. If $\mathbf{G}_{i,i} = 0$, then the numerator in (1.8) and the denominator in (1.8) both contain zero, we may interpret the result of the division in extended interval arithmetic to be $\mathbf{x}_i^+ = \mathbf{x}_i$, and trivially $x_i^* \in \mathbf{x}_i^+$. (But see the discussion above Assumption 1.5 below for this latter case.) □

A Y_i found in the literature is the *inverse midpoint preconditioner*, where we take Y_i to be the ith row of the inverse of the matrix whose (i,j)-th entry is $Y_{i,j}^{-1} = (a_{i,j} + b_{i,j})/2$. This preconditioner results in a matrix $\mathbf{G}$ the centers of whose components form the identity matrix; in [20] it is shown to be optimal in a certain sense and in a different context from here. However, there are instances where multiplication by an inverse of an n-by-n Jacobian matrix is not advisable. For example, if F' is singular somewhere in $\mathbf{X}_k$, then Y may be ill-conditioned, and $\mathbf{G} = Y\mathbf{F}'(\mathbf{X}_k)$ will therefore have very wide entries; this occurs, for example, for Powell's singular function, which appears as problem 3 in [6]. Alternately, one or more of the f_i may be more highly nonlinear than the others, or may be evaluated poorly, in which case the corresponding row or rows of the $\mathbf{F}'(\mathbf{X}_k)$ would have relatively wide entries; this occurs for Brown's almost linear function, which is problem 4 in [6]. Finally, $\mathbf{A}$ may be large and sparse.

In this paper, we develop a technique for computing preconditioners Y which involves solving a linear programming problem to find each row Y_i. The linear programming problems characterize certain simple optimum width-reduction conditions. In contrast to the inverse midpoint preconditioner, the new preconditioners implicitly take account of the relative widths of the entries of $\mathbf{X}$, in addition to both the widths and linear dependencies in $\mathbf{F}'(\mathbf{X})$. Furthermore, computation of each Y_i can take place during the pass through the variables in a Gauss–Seidel iteration (instead of before),

so that we may take full advantage of width reductions which have already taken place during that pass. Also, modifications to the constraints will allow us to compute a preconditioner Y which is optimal subject to structure or sparsity conditions.

As a final introductory observation, we note that (1.8) is defined in the usual interval arithmetic only when $0 \notin \mathbf{G}_{i,i}$. Furthermore, there is a Y_i such that this is so if and only if there is a j such that $0 \notin \mathbf{A}_{j,i}$. For singular systems for which this condition does not hold (and also sometimes when it does), we may use *extended interval arithmetic* to advantage. However, careful consideration of the relevant concepts is outside the scope of this paper, and will appear later. Thus, we make

Assumption 1.5. For each i for which we compute a preconditioner row Y_i by the method in this paper, there is a j such that $0 \notin \mathbf{A}_{j,i}$.

In §2, we introduce additional notation and preliminary results, then present our optimality condition, the linear programming problem, and a proof that the solution to the linear programming problem produces a Y_i which satisfies our optimality condition. In §3, we give examples. In §4, we display numerical results obtained from comparing our linear programming preconditioner to the inverse midpoint preconditioner and to no preconditioner at all. In §5, we summarize and outline future work.

2. Optimality conditions and linear programming. In (2.2) below, and hence in Theorem 2.7 in this section, we will assume that

$$M = (m_1, m_2, \cdots, m_n) = ((l_1 + u_1)/2, (l_2 + u_2)/2, \cdots (l_n + u_n)/2),$$

i.e., M is the midpoint vector of $\mathbf{X} = ([l_1, u_1], [l_2, u_2], \cdots, [l_n, u_n])$. However, in all other results, including Theorem 1.4 above, M can be arbitrary.

Throughout, intervals will be boldface, and vectors and matrices will be uppercase. If $\mathbf{v} = [t, u]$, then $w(v) = (u - t)$ is the width of $\mathbf{v}$. We identify numbers t with zero-width intervals $[t, t]$, and we say $\mathbf{v} > \mathbf{w}$ if every element of $\mathbf{v}$ is greater than every element of $\mathbf{w}$.

Our goal is to choose each $Y_i = (y_1, y_2, \cdots, y_n)$ in Algorithm 1.3 to make $w(\mathbf{x}_i \cap \overline{\mathbf{x}}_i)$ as small as possible. To this end, we first characterize these widths. We consider interval quotients of the form

$$\frac{[p, q]}{[r, s]},$$

where we think of $[p, q]$ as the numerator and $[r, s]$ as the denominator in the quotient in (1.8). We define

$$\mu_d = \min\{|r|, |s|\}. \tag{2.1}$$

If we make Assumption 1.5, then there are two possibilities for the quotient $[p, q]/[r, s]$, which are considered in the following two lemmas.

LEMMA 2.1. *Assume* $0 \notin [r, s]$. *If* $0 \in [p, q]$, *then*

$$w([p, q]/[r, s]) = w([p, q])/\mu_d.$$

If, in addition, $[p, q]/[r, s]$ *is the fraction in* (1.8) *and each* m_j *is the midpoint of* $\mathbf{x}_j$, *then*

$$w([p, q]) = \sum_{\substack{j=1 \\ j \neq i}}^{n} \{\max(|c_{i,j}|, |d_{i,j}|)\, w(\mathbf{x}_j)\}. \tag{2.2}$$

The proof of Lemma 2.1 follows from elementary properties of interval arithmetic.

LEMMA 2.2. *Assume* $[p,q]/[r,s]$ *is the fraction in* (1.8), *assume* $0 \notin [r,s]$, *and assume* $0 \notin [p,q]$, *but* M *is arbitrary. Then, with* l_i *and* u_i *as at the beginning of this section but* m_i *arbitrary,*

$$w(\mathbf{x}_i \cap \overline{\mathbf{x}}_i) < \max\{(m_i - l_i), (u_i - m_i)\}.$$

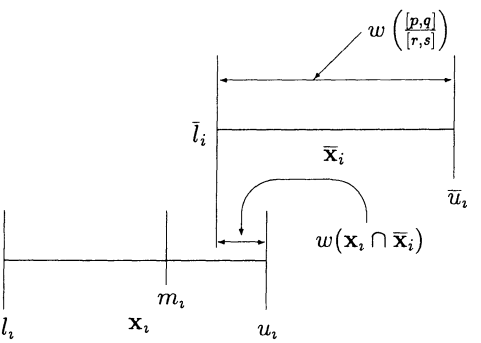

FIG. 1. *In this case,* $[p,q]/[r,s] > 0$, *so every element of* $m_i - [p,q]/[r,s]$ *is less than* m_i.

Proof of Lemma 2.2. First, $[p,q]/[r,s] > 0$ or $[p,q]/[r,s] < 0$. Therefore, $\overline{\mathbf{x}}_i < m_i$ or $\overline{\mathbf{x}}_i > m_i$. The result follows immediately. (See Fig. 1.) □

In what follows, we will understand $[p,q]/[r,s]$ to mean the numerator and denominator in (1.8).

Now choose k according to Assumption 1.5 so that $0 \notin \mathbf{A}_{k,i}$. Then, by choosing $y_j = 0$ if $j \neq k$ and $y_k = 1$, the resulting row $\mathbf{G}_i = Y_i\mathbf{A}$ has $0 \notin \mathbf{G}_{i,i}$. By multiplying Y_i by the appropriate scalar, we may also arrange it so that $r = 1$. Thus, we have the following lemma.

LEMMA 2.3. *Make Assumption* 1.5. *Then there is a row vector* Y_i *such that the denominator* $[r,s]$ *in the resulting Gauss–Seidel iteration* (1.8) *is of the form* $[1,s]$.

We wish to choose Y_i to minimize $w(\mathbf{x}_i^+) = w(\mathbf{x}_i \cap \overline{\mathbf{x}}_i)$. That is a difficult problem which involves several cases. However, if we minimize $w(\overline{\mathbf{x}}_i)$ and it so happens that $\overline{\mathbf{x}}_i \subset \mathbf{x}_i$, then we will also have minimized $w(\mathbf{x}_i^+)$. Furthermore, Theorem 2.6 below indicates that in many cases where $w(\overline{\mathbf{x}}_i)$ would be large, that strategy gives a reasonable $w(\mathbf{x}_i^+)$ anyway. For these reasons, we will concentrate in the remainder of this paper on minimizing $w(\overline{\mathbf{x}}_i)$.

The following lemma and corollary, which follow from the fact that $[p,q]/[r,s]$ is homogeneous in Y_i, indicate that we may minimize $w(\overline{\mathbf{x}}_i)$ by minimizing it subject to the constraint $c = 1$.

LEMMA 2.4. *Let* Y_i *be an arbitrary row-vector in* $\mathbf{R}^n$. *Then the interval* $\overline{\mathbf{x}}_i$ *in* (1.8) *is the same if we replace* Y_i *by* αY_i *for any nonzero scalar* α.

COROLLARY 2.5. *Let* $\Phi(\overline{\mathbf{x}}_i)$ *be a real valued function of* $\overline{\mathbf{x}}_i$, *where* $\overline{\mathbf{x}}_i$ *is as in* (1.8). *Then*

$$\min_{\substack{Y_i \in \mathbf{R}^n \\ 0 \notin [r,s]}} \Phi(\overline{\mathbf{x}}_i) = \min_{\substack{Y_i \in \mathbf{R}^n \\ r=1}} \Phi(\overline{\mathbf{x}}_i),$$

provided the minima exist.

In general, we cannot assume $0 \in [p, q]$, although examination of (1.8) shows that this will be so if $\mathbf{k}_i$, and hence if $\|F(M)\|_\infty$ is small enough. However, the following theorem indicates that a strategy which optimizes $w(\mathbf{x}_i^+)$ under the assumption that the resulting preconditioned system would have $0 \in [p, q]$ is reasonable, even if the assumption turns out to be false.

THEOREM 2.6. *Make Assumption* 1.5. *Then the row vector* Y_i *which minimizes* $w(\overline{\mathbf{x}}_i)$ *in* (1.8) *is a solution to*

$$\min_{\substack{Y_i \\ r=1}} w\left(\frac{[p,q]}{[r,s]}\right). \tag{2.3}$$

Furthermore, suppose we compute a Y_i *which solves the problem*

$$\min_{\substack{Y_i \\ r=1}} w\left\{[p,q]\right\}. \tag{2.4}$$

Then a Y_i *which solves* (2.4) *is a* Y_i *for which* $w(\overline{\mathbf{x}}_i)$ *is minimal, provided the resulting* $[p, q]$ *is such that* $0 \in [p, q]$. *If, on the other hand,* Y_i *solves* (2.4) *but the corresponding* $[p, q]$ *does not contain zero, then, for* M *arbitrary, the resulting* $\mathbf{x}_i^+$ *is such that*

$$w(\mathbf{x}_i^+) < \max\left\{(m_i - l_i), (u_i - m_i)\right\}.$$

Proof of Theorem 2.6. In (1.8), $\overline{\mathbf{x}}_i$ differs from $-[p, q]/[r, s]$ by the scalar m_i. Therefore, $w(\overline{\mathbf{x}}_i) = w([p, q]/[r, s])$; this fact and Corollary 2.5 imply that the minimum of $w(\overline{\mathbf{x}}_i)$ with respect to Y_i occurs at a solution of (2.3).

Suppose now that Y_i is a solution to (2.4). If the resulting $[p, q]$ is such that $0 \in [p, q]$, then

$$w(\overline{\mathbf{x}}_i) = w\left(\frac{[p,q]}{[r,s]}\right) = \frac{q-p}{r} = w([p,q]).$$

However, it follows from the definition of division of intervals that, for $r = 1$ and regardless of whether or not $0 \in [p, q]$, $w([p, q]/[r, s]) \geq w([p, q])$. These last two facts imply that Y_i is also a solution to (1.8), and therefore minimizes $w(\overline{\mathbf{x}}_i)$.

The last assertion in Theorem 2.6 is a direct corollary of Lemma 2.2. □

NOTE. *Solution of* (2.3) *appears to be a nonlinear problem, whereas solution of* (2.4) *can be done via linear programming techniques.*

In the remainder of this section, we will present the linear programming problem based on solution of (2.4) for Y_i. Our objective function will be the right member of (2.2), while we introduce $3n-1$ auxiliary variables and $2n-1$ constraints to transform the absolute values and maximum in (2.2) into linear relationships. We define the variables v_j, $1 \leq j \leq n-1$ such that the intended value of v_j is $\max\{|c_{i,j}|, |d_{i,j}|\}$ if $j < i$ and $\max\{|c_{i,j+1}|, |d_{i,j+1}|\}$ if $j \geq i$. Also, noting that

$$c_{i,j} = \sum_{\substack{t=1 \\ y_t \geq 0}} y_t a_{t,j} + \sum_{\substack{t=1 \\ y_t < 0}} y_t b_{t,j} \tag{2.5a}$$

and

$$d_{i,j} = \sum_{\substack{t=1 \\ y_t \geq 0}} y_t b_{t,j} + \sum_{\substack{t=1 \\ y_t < 0}} y_t a_{t,j}, \tag{2.5b}$$

we define auxiliary variables v_{n-1+t} and v_{2n-1+t}, $1 \le t \le n$, so that the intended value of v_{n-1+t} is $\max\{0, y_t\}$ and the intended value of v_{2n-1+t} is $\min\{0, y_t\}$. Finally, define j' by

$$j' = \begin{cases} j & \text{if } j < i \\ j+1 & \text{if } j \ge i \end{cases}.$$

With j' and the $3n-1$ variables V, the linear programming problem can be written as

$$\text{minimize } C(V) = \sum_{j=1}^{n-1} v_j w(x_{j'}) \tag{2.6a}$$

subject to

$$v_j \ge -\left[\sum_{t=1}^{n} v_{n-1+t} a_{t,j'} + \sum_{t=1}^{n} v_{2n-1+t} b_{t,j'}\right], \qquad 1 \le j \le n-1, \tag{2.6b}$$

$$v_j \ge +\left[\sum_{t=1}^{n} v_{n-1+t} b_{t,j'} + \sum_{t=1}^{n} v_{2n-1+t} a_{t,j'}\right], \qquad 1 \le j \le n-1, \tag{2.6c}$$

$$1 = \left[\sum_{t=1}^{n} v_{n-1+t} a_{t,i} + \sum_{t=1}^{n} v_{2n-1+t} b_{t,i}\right], \tag{2.6d}$$

and

$$v_{n-1+j} \ge 0 \quad \text{and} \quad v_{2n-1+j} \le 0 \quad \text{for } 1 \le j \le n. \tag{2.6e}$$

Once we compute the solution components v_j, $1 \le j \le 3n-1$, we compute the elements of the preconditioner by

$$y_t = v_{n-1+t} + v_{2n-1+t}, \qquad 1 \le t \le n. \tag{2.7}$$

The following theorem underscores the relationship of the solution to (2.6) to the solution to (2.4).

THEOREM 2.7. *Suppose that v_j, $1 \le j \le 3n-1$ form a solution of* (2.6), *and suppose m_i is the midpoint of corresponding $\mathbf{x}_i$. If Y_i is defined by* (2.7), *then Y_i solves* (2.4), *provided that, for each t, $v_{n-1+t} v_{2n-1+t} = 0$.*

Proof of Theorem 2.7. We first note that all solutions Y_i of (2.4) are representable as feasible points of (2.6) in which v_{n-1+t} represents the nonnegative part of y_t and v_{2n-1+t} represents the negative part of y_t. Thus, if V solves (2.6) and every t between 1 and n has the property that at least one of v_{n-1+t} and v_{2n-1+t} is zero, then the bracketed expressions in (2.6b) and (2.6c) represent $[c_{i,j}, d_{i,j}]$, and Lemma 2.1 implies that Y_i defined by (2.7) solves (2.4). □

Since there are $3n-1$ variables and $2n-1$ constraints in (2.6), at most n of the v_j can be nonzero simultaneously; therefore, it should usually happen that one of v_{n-1+t} and v_{2n-1+t} is zero for each t between 1 and n. In fact, computation of the preconditioner based on (2.6) works well in practice. Among all preconditioners computed in our substantial experiments (see §4 below) we have only observed occurrence of $v_{n-1+t} \ne 0$ and $v_{2n-1+t} \ne 0$ in Problem 10, where one of the elements of the pair was much smaller than the other one, and where the occurrence could be due to roundoff error. We thus make the following conjecture.

CONJECTURE. *If* $\{v_i\}_{i=1}^{3n-1}$ *is a solution of* (2.6) *obtained from the simplex method with exact arithmetic, then, for each* t *between* 1 *and* n, $v_{n-1+t}v_{2n-1+t} = 0$.

Finally, we state a corollary of Lemma 2.3.

THEOREM 2.8. *The linear programming problem* (2.5) *has a feasible point if and only if we make Assumption* 1.5.

3. Some examples. Here, we give six examples to compare the preconditioner defined in (2.6) with the inverse midpoint preconditioner, defined as the inverse of the midpoint matrix of $\mathbf{A}$. We denote the preconditioner formed from (2.6) by Y^{lp} and the preconditioner formed from the inverse of the midpoint of the interval Jacobian matrix by Y^{inv}; we denote the corresponding interval matrices $\mathbf{G}$ by $\mathbf{G}^{\text{lp}}$ and $\mathbf{G}^{\text{inv}}$ and the corresponding widths $w([p,q]/[r,s])$ in the ith variable by w_i^{lp} and w_i^{inv}. We will also compare the widths obtained from these two preconditioners with the widths w_i^{none} obtained from Algorithm 1.2, which entails no preconditioner at all.

When giving widths, we assume that $\|F(X_k)\|_\infty$ is small enough so that, after we compute the preconditioner, $0 \in [p,q]$ and Lemma 2.1 holds. We also assume that M is the midpoint of $\mathbf{X}$, so that, for example, if $\mathbf{x}_i = [1,2]$, then $\mathbf{x}_i - m_i = [-.5,.5]$. These examples illustrate that, subject to this assumption on $[p,q]$, Y^{lp} *must* give a better result than Y^{inv} and no preconditioner at all.

EXAMPLE 1. *A nonsingular point matrix; equal widths in all variables.*

$$\mathbf{A} = \begin{bmatrix} [1,1] & [2,2] \\ [3,3] & [4,4] \end{bmatrix}, \qquad \mathbf{X} = \begin{bmatrix} [1,2] \\ [1,2] \end{bmatrix}.$$

Without reordering the equations, we obtain

$$w_1^{\text{none}} = w\left(\frac{[2,2][-.5,.5]}{[1,1]}\right) = w([-1,1]) = 2$$

and

$$w_2^{\text{none}} = w\left(\frac{[3,3][-.5,.5]}{[4,4]}\right) = w\left(\left[-\frac{3}{8},\frac{3}{8}\right]\right) = \frac{3}{4}.$$

The results of applying the inverse preconditioner and (2.6) to problem 1 are identical; we obtain

$$Y^{\text{lp}} = Y^{\text{inv}} = \begin{bmatrix} -2 & 1 \\ 1.5 & -0.5 \end{bmatrix}, \qquad \mathbf{G}^{\text{lp}} = \mathbf{G}^{\text{inv}} = I,$$

$$w_i^{\text{lp}} = w_i^{\text{inv}} = 0 \quad \text{for } i = 1,2.$$

EXAMPLE 2. *A matrix for which the inverse preconditioner leads to* $\overline{\mathbf{x}}_i = \mathbf{R}$.

$$\mathbf{A} = \begin{bmatrix} [1,3] & [2,4] \\ [3,5] & [4,6] \end{bmatrix}, \qquad \mathbf{X} = \begin{bmatrix} [1,2] \\ [1,2] \end{bmatrix}.$$

Algorithm 1.2 gives

$$w_1^{\text{none}} = w\left(\frac{[2,4][-.5,.5]}{[1,3]}\right) = w\left(\frac{[-2,2]}{[1,3]}\right) = 4$$

and $w_2^{\text{none}} = 1.25$. The inverse preconditioner gives

$$Y^{\text{inv}} = \begin{bmatrix} -2.5 & 1.5 \\ 2 & -1 \end{bmatrix}, \qquad \mathbf{G}^{\text{inv}} = \begin{bmatrix} [-3,5] & [-4,4] \\ [-3,3] & [-2,4] \end{bmatrix}.$$

Since $0 \in \mathbf{G}_{i,j}^{\text{inv}}$ for all i and j, w_1^{inv} and w_2^{inv} are undefined in ordinary interval arithmetic, and can be interpreted to be ∞. For the solution of (2.6), we obtain

$$Y^{\text{lp}} \doteq \begin{bmatrix} 0 & .333 \\ 0 & .25 \end{bmatrix}, \qquad \mathbf{G}^{\text{lp}} \doteq \begin{bmatrix} [1,1.67] & [1.33,2] \\ [.75,1.25] & [1,1.5] \end{bmatrix}.$$

Thus, $w_1^{\text{lp}} \doteq 2$, but $w_2^{\text{lp}} \doteq 1.25$. We see that Y^{inv} is useless, but that, depending on $F(M)$, a Gauss–Seidel step with $\mathbf{G}^{\text{lp}}$ may be more successful at reducing the width of $\mathbf{x}_1$ than a Gauss–Seidel step with no preconditioner at all.

EXAMPLE 3. *A matrix for which the inverse preconditioner gives large widths, one of which is finite.*

$$\mathbf{A} = \begin{bmatrix} [1.8, 2.2] & [2, 4] \\ [3.8, 4.2] & [4, 6] \end{bmatrix}, \qquad \mathbf{X} = \begin{bmatrix} [1, 2] \\ [1, 2] \end{bmatrix}.$$

Algorithm 1.2 gives $w_1^{\text{none}} \doteq 2.22$ and $w_2^{\text{none}} \doteq 1.05$. The inverse preconditioner gives

$$Y^{\text{inv}} = \begin{bmatrix} -2.5 & 1.5 \\ 2.0 & -1.0 \end{bmatrix}, \qquad \mathbf{G}^{\text{inv}} = \begin{bmatrix} [.2, 1.8] & [-4, 4] \\ [-.6, .6] & [-2, 4] \end{bmatrix},$$

so $w_1^{\text{inv}} = 20$ and w_2^{inv} can be interpreted to equal ∞. For the preconditioner based on (2.6) we obtain

$$Y^{\text{lp}} \doteq \begin{bmatrix} 0 & .263 \\ 0 & .25 \end{bmatrix}, \qquad \mathbf{G}^{\text{lp}} \doteq \begin{bmatrix} [1, 1.11] & [1.05, 1.58] \\ [.95, 1.05] & [1, 1.5] \end{bmatrix}.$$

Thus, $w_1^{\text{lp}} \doteq 1.58 < w_1^{\text{none}} < w_1^{\text{inv}}$ and $w_2^{\text{lp}} \doteq 1.05 \doteq w_2^{\text{none}} < w_2^{\text{inv}}$. Hence, depending on $F(M)$, a Gauss–Seidel step with $\mathbf{G}^{\text{lp}}$ will be more worthwhile in reducing the width of either coordinate than a Gauss–Seidel step with $\mathbf{G}^{\text{inv}}$. Also, a step with $\mathbf{G}^{\text{lp}}$ may be more effective than a Gauss–Seidel step with $\mathbf{A}$ at reducing the width of $\mathbf{x}_1$.

EXAMPLE 4. *A simple three-dimensional system for which the inverse preconditioner gives reasonable results.*

$$\mathbf{A} = \begin{bmatrix} [9, 11] & [2, 4] & [2, 4] \\ [3.8, 4.2] & [19, 21] & [4, 6] \\ [3.8, 4.2] & [4, 6] & [29, 31] \end{bmatrix}, \qquad \mathbf{X} = \begin{bmatrix} [1, 2] \\ [1, 2] \\ [1, 2] \end{bmatrix}.$$

Algorithm 1.2 gives $w_1^{\text{none}} \doteq .89$, $w_2^{\text{none}} \doteq 0.54$, and $w_3^{\text{none}} \doteq 0.35$. The inverse preconditioner gives

$$Y^{\text{inv}} \doteq \begin{bmatrix} .109 & -.014 & -.009 \\ -.019 & .055 & -.007 \\ -.011 & -.007 & .036 \end{bmatrix},$$

$$\mathbf{G}^{\text{inv}} \doteq \begin{bmatrix} [.886, 1.11] & [-.132, .132] & [-.132, .132] \\ [-.031, .031] & [.919, 1.08] & [-.081, .081] \\ [-.020, .020] & [-.054, .054] & [.946, 1.05] \end{bmatrix}.$$

We thus have $w_1^{\text{inv}} \doteq .298$, $w_2^{\text{inv}} \doteq .122$, and $w_3^{\text{inv}} \doteq .079$. The preconditioner based on (2.6) gives

$$Y^{\text{lp}} \doteq \begin{bmatrix} .123 & -.016 & -.010 \\ -.021 & .060 & -.008 \\ -.012 & -.008 & .038 \end{bmatrix},$$

$$\mathbf{G}^{\text{lp}} = \begin{bmatrix} [1.00, 1.26] & [-.149, .149] & [-.149, .149] \\ [-.034, .034] & [1.00, 1.18] & [-.088, .088] \\ [-.021, .021] & [-.057, .057] & [1.00, 1.11] \end{bmatrix}.$$

Thus, $w_1^{\text{lp}} \doteq .298 \doteq w_1^{\text{inv}} < w_1^{\text{none}}$, $w_2^{\text{lp}} \doteq .122 \doteq w_2^{\text{inv}} < w_2^{\text{none}}$, and $w_3^{\text{lp}} \doteq .079 \doteq w_3^{\text{inv}} < w_3^{\text{none}}$. Hence, depending on $F(M)$, we would expect a Gauss–Seidel step with $\mathbf{G}^{\text{lp}}$ to accomplish about the same significant width reduction as a Gauss–Seidel step with $\mathbf{G}^{\text{inv}}$. A Gauss–Seidel step with Algorithm 1.2 would reduce the widths, but probably not as much.

EXAMPLE 5. *The same as Example* 4 *except the widths of the components of* $\mathbf{X}$ *are not all equal.*

$$\mathbf{A} = \begin{bmatrix} [9,11] & [2,4] & [2,4] \\ [3.8,4.2] & [19,21] & [4,6] \\ [3.8,4.2] & [4,6] & [29,31] \end{bmatrix}, \qquad \mathbf{X} = \begin{bmatrix} [1,2] \\ [-10,10] \\ [.01,.02] \end{bmatrix}.$$

We get $w_1^{\text{none}} \doteq 4.45$, $w_2^{\text{none}} \doteq .22$, and $w_3^{\text{none}} \doteq 2.21$, so that one pass through Algorithm 2.1 would probably not reduce the widths of $\mathbf{x}_1$ and $\mathbf{x}_3$ at all, but would be very effective at reducing the width of $\mathbf{x}_2$. We get the same Y^{inv} and $\mathbf{G}^{\text{inv}}$ as in Example 4, and we get $w_1^{\text{inv}} \doteq .30$, $w_2^{\text{inv}} \doteq .0429$, and $w_3^{\text{inv}} \doteq 1.17$. We obtain

$$Y^{\text{lp}} \doteq \begin{bmatrix} .119 & -.018 & 0 \\ -.021 & .060 & -.008 \\ 0 & -.009 & .036 \end{bmatrix},$$

$$\mathbf{G}^{\text{lp}} \doteq \begin{bmatrix} [1.,1.25] & [-.137,.137] & [.131,.406] \\ [-.034,.034] & [1.,1.18] & [-.088,.088] \\ [.1,.118] & [-.046,.046] & [1,1.09] \end{bmatrix}.$$

Thus, $w_1^{\text{lp}} \doteq .279 < w_1^{\text{inv}} < w_1^{\text{none}}$, $w_2^{\text{lp}} \doteq .0429 \doteq w_2^{\text{inv}} < w_2^{\text{none}}$, and $w_3^{\text{lp}} \doteq 1.03 < w_3^{\text{inv}} < w_3^{\text{none}}$. Hence, depending on $F(M)$, we would expect a Gauss–Seidel step with $\mathbf{G}^{\text{lp}}$ to be better in the first coordinate than a Gauss–Seidel step with $\mathbf{G}^{\text{inv}}$, and the first Gauss–Seidel step without a preconditioner would be ineffective in the first coordinate. Gauss–Seidel with any of $\mathbf{A}$, $\mathbf{G}^{\text{inv}}$, or $\mathbf{G}^{\text{lp}}$ would reduce the width of the second coordinate, and none of them may reduce the width of the third coordinate on the first pass. This example illustrates that, when we use $\mathbf{G}^{\text{lp}}$, we implicitly take advantage of the widths of the components of $\mathbf{X}$.

EXAMPLE 6. *Initial* $\mathbf{X}$ *and Jacobian matrix for Problem* 4 *from* [6] (*Brown's almost linear function*).

$$\mathbf{A} = \begin{bmatrix} [2,2] & [1,1] & [1,1] & [1,1] & [1,1] \\ [1,1] & [2,2] & [1,1] & [1,1] & [1,1] \\ [1,1] & [1,1] & [2,2] & [1,1] & [1,1] \\ [1,1] & [1,1] & [1,1] & [2,2] & [1,1] \\ [-16,16] & [-16,16] & [-16,16] & [-16,16] & [-16,16] \end{bmatrix},$$

$$\mathbf{X} = ([-2,2],[-2,2],[-2,2],[-2,2],[-2,2])^T.$$

The problem from which this example came is difficult for interval Newton methods, which are not able to separate the highly nonlinear behavior of the last function component from the linearity of the remaining $n-1$ function components. The matrices Y^{inv} and $\mathbf{G}^{\text{inv}}$ in this example are not even defined, since the midpoint matrix is singular.

We obtain $w_1^{\text{none}} = w_2^{\text{none}} = w_3^{\text{none}} = w_4^{\text{none}} = 8$, and w_5^{none} is undefined. Hence, we cannot expect to make progress with the interval Gauss–Seidel method without a preconditioner. For $\mathbf{G}^{\text{lp}}$, we obtain

$$Y^{\text{lp}} = \begin{bmatrix} .8 & -.2 & -.2 & -.2 & 0 \\ -.2 & .8 & -.2 & -.2 & 0 \\ -.2 & -.2 & .8 & -.2 & 0 \\ -.2 & -.2 & -.2 & .8 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix},$$

$$\mathbf{G}^{\mathrm{lp}} = \begin{bmatrix} [1,1] & [0,0] & [0,0] & [0,0] & [.2,.2] \\ [0,0] & [1,1] & [0,0] & [0,0] & [.2,.2] \\ [0,0] & [0,0] & [1,1] & [0,0] & [.2,.2] \\ [0,0] & [0,0] & [0,0] & [1,1] & [.2,.2] \\ [2,2] & [1,1] & [1,1] & [1,1] & [1,1] \end{bmatrix},$$

and $w_1^{\mathrm{lp}} = w_2^{\mathrm{lp}} = w_3^{\mathrm{lp}} = w_4^{\mathrm{lp}} = .8$, while $w_5^{\mathrm{lp}} = 20$. Since we can expect to make progress in the ith coordinate in the Gauss–Seidel process if $w(\overline{\mathbf{x}}_i) < w(\mathbf{x}_i)$, use of Y^{lp} as a preconditioner should effectively reduce the widths of the first four coordinates of $\mathbf{X}$. This example thus shows that this preconditioner is of potential value for similar problems.

4. Numerical results. Here, we report results obtained by incorporating the preconditioner computation explained in §2 into a generalized bisection code which is similar to Algorithm 1.1, but more sophisticated. The code is similar to that described in [6], with a few differences, described in [9]. The software in [9] is available from the author upon request. The differences from the algorithm described in [6] include

(i) use of simulated directed roundings for true interval arithmetic;
(ii) use of the Gauss–Seidel method instead of the Krawczyk method;
(iii) use of interval arithmetic to evaluate $F(M)$;
(iv) use of a volume ratio in Step 3(e) of Algorithm 1.1 to determine when to continue iteration of the Gauss–Seidel method; and
(v) use of a special technique to choose coordinate directions in which to bisect, which is based on "maximal smear" in $\mathbf{F}'(\mathbf{X})(\mathbf{X} - M)$.

Items (i) and (iii) are necessary for total reliability, while items (ii), (iv), and (v) are related to efficiency.

The simulated directed roundings are implemented with a subroutine for each of the elementary arithmetic operations. After each such operation, the left endpoint of the interval is decreased and the right endpoint of the interval is increased by a quantity proportional to the number of units in the last place by which an elementary operation can be in error. The result is a machine-representable interval which contains the actual range of values of the elementary operation. The effect is a relatively simple and transportable, yet rigorous interval arithmetic. This is necessary to provide a mathematical guarantee that Algorithm 1.1 will not miss any roots.

It is also for this guarantee that we evaluate $F(M)$ using interval arithmetic.

We have found a volume ratio to be most effective in Step 3(e) of Algorithm 1.1, although the technique described below Algorithm 1.1 also works. In particular, we continue to use the Gauss–Seidel iteration after a sweep of all n coordinates only if

$$\prod_{\substack{i=1 \\ w(\mathbf{x}_i)>\epsilon}}^{n} w(\mathbf{x}_i^+) < \eta \prod_{\substack{i=1 \\ w(\mathbf{x}_i)>\epsilon}}^{n} w(\mathbf{x}_i)$$

for some η with $0 < \eta < 1$, where $\mathbf{x}_i^+ = \overline{\mathbf{x}} \cap \mathbf{x}_i$; in practice we have found $\eta = .6$ works well.

To determine the coordinate in which to bisect, we first compute

$$\sigma_j = \max_{1 \le i \le n} \{|a_{i,j}|, |b_{i,j}|\} (u_j - l_j),$$

where $\mathbf{F}'(\mathbf{X}) = \mathbf{A} = \{[a_{i,j}, b_{i,j}]\}$ is as above (1.5), and where $\mathbf{x}_j = [l_j, u_j]$. We then bisect in the coordinate with index j for which σ_j is maximum. This coordinate direction is, roughly, the one in which the values of the f_i change most rapidly relative

to the individual widths of the present box $\mathbf{X}$, and its choice for bisection introduces an implicit scaling.

Our code differs from that described in [9] in that we do not perform the Gauss–Seidel step for the ith coordinate if $0 \in \mathbf{G}_{i,i}$. Extended interval arithmetic can make the step meaningful in this case, but an analysis is outside the scope of this paper, and we wish to isolate the phenomena presently studied.

We used the linear programming routine we wrote specially to solve (2.6). This routine is eight to ten times faster than a commercial routine we tried earlier for this purpose, but somewhat less reliable on poorly scaled problems.

We will compare the code with the new preconditioner Y^{lp} to the code with the inverse preconditioner Y^{inv}, and the code with no preconditioner at all. The interval Gauss–Seidel algorithm with no preconditioner is Algorithm 1.2, while we use Algorithm 1.3 for Y^{inv}. For Y^{lp}, we modify Algorithm 1.3 to check Assumption 1.5 first. We obtain the following algorithm.

ALGORITHM 4.1. (*Interval Gauss–Seidel for Step* 3(c) *of Algorithm* 1.1, *modified to use* Y^{lp} *efficiently.*) *Do the following for* $i = 1$ *to* n.

0. *Check Assumption* 1.5 *for the present* i.
   (a) *If Assumption* 1.5 *does not hold, then consider the next* i.
   (b) *Otherwise, continue to Step* 1.
1. (*Update a coordinate.*)
   (a) *Compute* Y_i^{lp}.
   (b) *Compute* $\mathbf{k}_i$ *and* $\mathbf{G}_i^{\mathrm{lp}}$.
   (c) *Compute*

$$\overline{\mathbf{x}}_i = m_i - \left[\mathbf{k}_i + \sum_{\substack{j=1 \\ j \neq i}}^{n} \mathbf{G}^{\mathrm{lp}}{}_{i,j} (\mathbf{x}_j - m_j) \right] \Big/ \mathbf{G}^{\mathrm{lp}}{}_{i,i}$$

   using interval arithmetic.
2. (*The new box is empty.*) *If* $\overline{\mathbf{x}}_i \cap \mathbf{x}_i = \emptyset$, *then return to Step* 3(f) *of Algorithm* 1.1.
3. (*The new box is nonempty; prepare for another iteration.*)
   (a) *Replace* $\mathbf{x}_i$ *by* $\mathbf{x}_i \cap \overline{\mathbf{x}}_i$.
   (b) *Possibly reevaluate* $\mathbf{F}'(\mathbf{X}_k)$ *to replace* $\mathbf{A}$ *by an interval matrix whose corresponding widths are smaller.*

In the experiments described here, we do not ever do Step 3 (b) of Algorithm 1.2, Algorithm 1.3, or Algorithm 4.1, but only recompute the interval Jacobian matrix at the beginning of each coordinate sweep in the Gauss–Seidel iteration, i.e., in Step 3(b) of Algorithm 1.1. Our rationale is that, if the coordinates of $\mathbf{X}$ have changed significantly during a sweep, then the present interval Jacobian matrix is serving its purpose.

Similarly, in Algorithm 4.1, we do not adjust M and reevaluate $F(M)$ during a coordinate sweep, but only in Step 3(b) of Algorithm 1.1. As a result, Y_i^{lp} will not necessarily give $\overline{\mathbf{x}}_i$ of minimal width for $i > 1$, but must do so if $\overline{\mathbf{x}}_{i'} = \mathbf{X}_{i'}$ for $i' < i$, and may do so in other cases.

The problem set is that in [6]. We do not include problems 5–8, and 13, since these are linear and hence trivial for interval Newton methods.

In Algorithm 1.1, we used $\epsilon = 10^{-5}$ and $\epsilon_F = 10^{-10}$.

Table 1 gives estimates for the amount of work for each of the three methods. The first column gives the problem number as in [6], the second column gives the

dimension n of the problem, and the third column gives the method, where "none" refers to no preconditioner, Y^{inv} refers to the inverse preconditioner, and Y^{lp} refers to the linear programming preconditioner. Column 4 (NBOX) gives the total number of boxes considered in Algorithm 1.1, column 5 (NFUN) gives the total number of interval function evaluations, column 6 (NJAC) gives the total number of interval Jacobian evaluations, and column 7 gives an estimate W_e for the total amount of work which is computed as

$$W_e = \text{NFUN} + n\,\text{NJAC}. \tag{4.1}$$

In Table 1, the entries "*" mean that the code could not finish without putting more than 50 boxes on the stack $\mathcal{S}$ in Algorithm 1.1. In this case, 28533 boxes were considered, there were 43415 interval Jacobian matrix evaluations, and there were 96183 function evaluations. However, the program did find both roots before stopping.

Similarly, the entries "+" mean that the code could not finish without putting more than 50 entries in the list $\mathcal{L}'$ of boxes which could possibly contain roots, but for which (1.4) does not hold. There are 16 isolated roots to this problem, and with both Y^{inv} and Y^{lp}, the program completed with exactly 16 boxes in $\mathcal{L}$, and no boxes in $\mathcal{L}'$.

We make the following additional observations about this experiment.

1. In only three cases (test problems 1, 9, and 10) did the linear programming preconditioner result in more boxes, function, and Jacobian evaluations than the inverse midpoint preconditioner. In the first two cases, the numbers were almost equal, and the differences could have been due to roundoff effects; it hints that the inverse midpoint preconditioners in these cases were close to optimal. In the third case, the problem was clearly due to the inability of our linear programming solver to handle the extreme poor scaling encountered, since it failed various times. In earlier experiments with a slower but perhaps more robust commercial solver, we obtained 71 boxes, 260 function evaluations, and 117 Jacobian evaluations for the linear programming preconditioner. (The number of boxes considered in the earlier experiments also differed very slightly from that given here in three of the other problems.)
2. In only one case (test problem 17) was no preconditioner at all better than the inverse midpoint preconditioner. However, in that case it was significantly better.
3. Use of no preconditioner at all was impractical for test problems 4 and 11.
4. The linear programming preconditioner resulted in three orders of magnitude less boxes, function, and Jacobian evaluations for test problem 4 (Brown's almost linear function).
5. The linear programming preconditioner resulted in less than a fifth as many boxes and a third as many interval Jacobian evaluations for test problem 3 (a variant of Powell's singular function).

On some problems, producing the linear programming preconditioner takes more computation than producing the inverse midpoint preconditioner. However, this is

TABLE 1
Cost measures for the three methods.

#	n	meth.	NBOX	NFUN	NJAC	Est. Work
1	2	none	23	132	63	258
		Y^{inv}	17	62	30	122
		Y^{lp}	19	73	35	143
2	2	none	231	426	179	784
		Y^{inv}	69	162	69	300
		Y^{lp}	29	104	47	198
3	4	none	3421	5649	1994	13625
		Y^{inv}	1818	3629	1390	9189
		Y^{lp}	368	1064	423	2756
4	5	none	*	*	*	*
		Y^{inv}	24473	42743	16706	126273
		Y^{lp}	33	123	58	413
9	2	none	105	286	137	560
		Y^{inv}	11	41	20	81
		Y^{lp}	13	47	23	93
10	4	none	2780	5969	2376	15473
		Y^{inv}	161	465	203	1277
		Y^{lp}	247	601	247	1589
11	8	none	+	+	+	+
		Y^{inv}	1253	2521	1026	10729
		Y^{lp}	367	1210	525	5410
12	3	none	6766	11340	4130	23730
		Y^{inv}	431	1409	619	3266
		Y^{lp}	255	1050	475	2475
14	2	none	265	456	164	784
		Y^{inv}	17	62	29	120
		Y^{lp}	11	48	23	94
15	2	none	1	4	2	8
		Y^{inv}	1	4	2	8
		Y^{lp}	1	4	2	8
16	4	none	2208	3320	1112	7768
		Y^{inv}	1	4	2	8
		Y^{lp}	1	4	2	8
17	5	none	23	85	39	280
		Y^{inv}	85	197	82	607
		Y^{lp}	23	83	38	273
TOT.		none				*
		Y^{inv}				151899
		Y^{lp}				12569

highly dependent on how we set up the tableau (assuming we are using the simplex method) and on which software we use. In earlier experiments with an off-the-shelf solver, CPU times were eight to ten times larger than those listed here. In Table 2, we list approximate CPU times on an IBM 3090 for the entire program to complete, for each of the three schemes and for problems 3, 4, 10, 11, and 12. These times should be considered estimates, since they are virtual CPU times obtained from the

"START" system command to the next ready message in the VM/CMS time sharing environment.

TABLE 2
Approximate CPU *times in seconds.*

#	none	Y^{inv}	Y^{lp}
3	5.8	6.1	2.6
4	*	139.1	1.5
10	9.5	1.1	3.6
11	+	16.3	20.2
12	24.0	2.7	2.7
ALL	*	166.2	32.0

The specific problems listed in Table 2 are the most nontrivial in the sense that they had singularities, had many roots, were poorly scaled, or generally took a large amount of CPU time. The last entry in Table 2 gives the total CPU time for all twelve problems in Table 1, solved in a single run.

From Table 2, it is clear that the linear programming preconditioner has its place in the design of interval Newton/generalized bisection codes. It is definitely of use in problems like Brown's almost linear function, where it successfully avoids using entries of $\mathbf{F}'$ with excessive widths, and in singular problems, where it can narrow the possible root-containing region even when the inverse midpoint preconditioner is undefined. Improvements in our LP solver will make it better on poorly scaled problems like problem 10.

We note that the ratio of estimated total work is about 12, while the ratio of CPU times is only about 5.2. This is due to the fact that more CPU time is used computing a given LP preconditioner than computing an inverse midpoint preconditioner, on average. This hints at possible additional improvements.

In Problem 11, the LP preconditioner resulted in less estimated total work but more CPU time. In this problem, most of the CPU time was spent in the LP solver. This problem is unusual in that there are 16 roots, and "splitting" preconditioners which employ extended interval arithmetic (not discussed in this paper) are effective. We might also be seeing some effect of dimension here; preliminary experiments indicate that the amount of work on some problems to obtain the linear programming preconditioner for all n coordinates may increase (empirically) like n^4 or n^5, whereas it requires $O(n^3)$ work to obtain the inverse midpoint preconditioner.

5. Summary and future work. We have reviewed the interval Gauss–Seidel method in combination with the method of bisection for reliably finding all roots of a system of nonlinear equations within a box in $\mathbf{R}^n$. We have demonstrated that, in many instances the overall algorithm can be made more efficient by preconditioning the interval systems before applying the Gauss–Seidel iteration. We have formulated optimality conditions for these preconditioners, and have posed a linear programming problem, the solution of which satisfies the optimality conditions. Several simple examples illustrate the resulting preconditioners.

Numerical tests of the new preconditioner in the overall algorithm indicate that it is effective at reducing the number of function and Jacobian evaluations, and often also results in less CPU time.

The preconditioners can probably be made still more practical by implementing formulation and solution of (2.6) more efficiently. Also, it is possible to structure the preconditioner so that (2.2) is only approximately minimized, but computation of the preconditioner requires less work than solution of (2.6).

Second, we may develop a priori criteria for determining when to use the preconditioners. We can view them as being part of a more sophisticated code, which uses heuristics to decide when they should and should not be computed.

We are also presently working on using generalizations of (2.2) with arbitrary M. This is important, since convergence of the interval Gauss–Seidel method is improved if M is taken to be a good approximation to a root. (See [13].)

A fourth line of investigation we are following is development of preconditioners for the case when Assumption 1.5 does not hold. This case typically occurs when the Jacobian matrix is singular at one or more roots, or when there are numerous roots.

Finally, we may compute a sparse preconditioner simply by posing the linear programming problems in low-dimensional subspaces, provided such preconditioners are effective on specific problems. Also, we may force $\mathbf{G}$ to have a number of properties by imposing additional constraints. Ideas from [15] may be useful here.

6. Acknowledgments. I wish to thank the referees for their attention to detail and for their suggestions on how to make the paper more readable. I also wish to thank the production editor for her careful attention.

REFERENCES

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

[2] E. R. Hansen, *On solving systems of equations using interval arithmetic*, Math. Comp., 22 (1968), pp. 374–384.

[3] E. R. Hansen and R. I. Greenberg, *An interval Newton method*, Appl. Math. Comp., 12 (1983), pp. 89–98.

[4] E. R. Hansen and S. Sengupta, *Bounding solutions of systems of equations using interval analysis*, BIT, 21 (1981), pp. 203–211.

[5] R. B. Kearfott, *Abstract generalized bisection and a cost bound*, Math. Comp., 49 (1987), pp. 187–202.

[6] ———, *Some tests of generalized bisection*, ACM Trans. Math. Software, 13 (1987), pp. 197–220.

[7] ———, *Interval arithmetic techniques in the computational solution of nonlinear systems of equations: introduction, examples, and comparisons*, Proceedings of the 1988 AMS-SIAM Summer Seminar in Applied Mathematics, Colorado State University, July 18-29, 1988, to appear.

[8] ———, *On handling singular systems with interval Newton methods*, Proceedings of the Twelfth IMACS World Congress on Scientific Computation, Vol. 4, R. Vichnevetsky, P. Borne, and J. Vignes, eds., pp. 651–653.

[9] R. B. Kearfott and M. Novoa, *A Program for Generalized Bisection*, ACM Trans. Math. Software, to appear.

[10] R. E. Moore, *A test for existence of solutions to nonlinear systems*, SIAM J. Numer. Anal., 14 (1977), pp. 611–615.

[11] ———, *Methods and Applications of Interval Analysis*, Society for Industrial and Applied Mathematics, Philadelphia, 1979.

[12] R. E. MOORE AND S. T. JONES, *Safe starting regions for iterative methods*, SIAM J. Numer. Anal., 14 (1977), pp. 1051–1065.

[13] A. NEUMAIER, *Interval iteration for zeros of systems of equations*, BIT, 25 (1985), pp. 256–273.

[14] K. NICKEL, *On the Newton method in interval analysis*, Tech. Summ. Report 1136, Mathematics Research Center, University of Wisconsin at Madison, 1971.

[15] T. OJIKA, *Structure analysis for large scale nonlinear equations*, Memoirs of Osaka Kyoiku University, Ser. III, 32 (1983), pp. 63–72.

[16] H. SCHWANDT, *An interval arithmetic approach for the construction of an almost globally convergent method for the solution of the nonlinear Poisson equation*, SIAM J. Sci. Statist. Comput., 5 (1984), pp. 427–452.

[17] ———, *The solution of nonlinear elliptic Dirichlet problems on rectangles by almost globally convergent interval methods*, SIAM J. Sci. Statist. Comput., 6 (1985), pp. 617–638.

[18] J. M. SHEARER AND M. A. WOLFE, *Some computable existence, uniqueness, and convergence tests for nonlinear systems*, SIAM J. Numer. Anal., 22 (1985), pp. 1200–1207.

[19] S. THIEL, *Intervalliterationsverfahren für discretisierte elliptische Differentialgleichungen*, Freiburger Intervall-Berichte 86/8, Institut für Angewandte Mathematik der Universität Freiburg, FRG, preprint.

[20] C. XIAOJUN AND W. DEREN, *On the optimal properties of the Krawczyk-type interval operator*, Freiburger Intervall-Berichte 87/5, Institut für Angewandte Mathematik der Universität Freiburg, FRG, preprint.