æ

A Review of Preconditioners for the Interval Gauss–Seidel Method

R. Baker Kearfott,
Chenyi Hu,
and
Manuel Novoa III

Department of Mathematics
University of Southwestern Louisiana

Abstract. Interval Newton methods in conjunction with generalized bisection can form the basis of algorithms that find all real roots within a specified box $\mathbf{X} \subset \mathbb{R}^n$ 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 $0 = F(M) + \mathbf{F}'(\mathbf{X})(\tilde{\mathbf{X}} - M)$; if interval arithmetic is then used to bound the solutions of this system, the resulting box $\tilde{\mathbf{X}}$ contains all roots of the nonlinear system. We may use the interval Gauss–Seidel method to find these solution bounds.

In order to increase the overall efficiency of the interval Newton / generalized bisection algorithm, the linear interval system is multiplied by a preconditioner matrix Y before the interval Gauss–Seidel method is applied. Here, we review results we have obtained over the past few years concerning computation of such preconditioners. We emphasize importance and connecting relationships, and we cite references for the underlying elementary theory and other details.

1. Introduction and Goals

The general problem we address is:

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

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

where bounds $\underline{x}_i$ and $\overline{x}_i$ are known such that

$$\underline{x}_i \leq x_i \leq \overline{x}_i \text{ for } 1 \leq i \leq n.$$

We write $X = (x_1, x_2, \ldots, x_n)$, and we denote the box given by the inequalities on the variables x_i by $\mathbf{B}$.[1]

[1] Throughout the paper, we will denote interval quantities with boldface letters. Vectors and matrices will be denoted with capital letters.

Typeset by $\mathcal{A}_{\mathcal{M}}\mathcal{S}$-TEX

A successful approach to this problem is generalized bisection in conjunction with interval Newton methods. Interval Newton / generalized bisection methods for (1.1) are described in [**3**], [**9**], [**10**], [**15**], [**17**], [**20**], [**22**], etc. For an introduction to the interval arithmetic underlying these methods, see [**1**], [**14**], etc. Also, the book [**19**] will contain an overview of interval methods for linear and nonlinear systems of equations.

In these methods, we first transform $F(X) = 0$ to the linear interval system

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

where $\mathbf{F}'(\mathbf{X}_k)$ is a suitable (such as an elementwise) interval extension of the Jacobian matrix over the box $\mathbf{X}_k$ (with $\mathbf{X}_0 = \mathbf{B}$), and where $X_k \in \mathbf{X}_k$ represents a predictor or initial guess point. If we then formally solve (1.2) using interval arithmetic, the resulting box $\tilde{\mathbf{X}}_k$, which actually just satisfies

$$\mathbf{F}'(\mathbf{X}_k)(\tilde{\mathbf{X}}_k - X_k) \supset -F(X_k), \tag{1.2(b)}$$

will contain all solutions to every possible system $A(X - X_k) = -F(X_k)$ with $A \in \mathbf{F}'(\mathbf{X}_k)$. Furthermore, under suitable (usually satisfied) assumptions on the interval extension $\mathbf{F}'$, it follows from the mean value theorem that $\tilde{\mathbf{X}}_k$ will contain all solutions to $F(X) = 0$ in $\mathbf{X}_k$. We may then iterate $\mathbf{X}_{k+1}$ with the formula

$$\mathbf{X}_{k+1} = \mathbf{X}_k \cap \tilde{\mathbf{X}}_k, \tag{1.3}$$

to obtain tighter bounds on all possible roots.

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 (1.3) with one of these boxes, and put the other one on a stack for later consideration. As explained in [**9**], [**10**], [**15**], and elsewhere, the following fact (from [**17**, p. 263]) allows such a composite generalized bisection algorithm to compute all solutions to (1.1(b)) *with mathematical certainty*. For many methods of solving (1.2),

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

The portion of the overall generalized bisection algorithm treated here (and given completely in [**13**] and elsewhere) is the *preconditioned*

interval Gauss–Seidel method, which we describe with the following notation. We write $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n)$ for $\mathbf{X}_k$ and we write $\mathbf{f}'_{i,j}$ for the interval in the i-th row and j-th column of $\mathbf{F}' = \mathbf{F}'(\mathbf{X})$. Similarly, we write[2] $F(X_k) = F = (\mathbf{f}_1, \mathbf{f}_2, \ldots, \mathbf{f}_n)$, and $X_k = (x_1, x_2, \ldots, x_n)$, so that (1.2) becomes

$$\mathbf{F}'(\tilde{\mathbf{X}}_k - X_k) = -F. \tag{1.5}$$

In order to make some measure of $\mathbf{X}_{k+1}$ in the (1.3) small, we precondition (1.5); i.e., we multiply by a matrix Y to obtain

$$Y\mathbf{F}'(\tilde{\mathbf{X}}_k - X_k) = -YF. \tag{1.6}$$

Such preconditioning usually lessens the work required to complete the generalized bisection algorithm, and accelerates convergence of the iterates of the interval Gauss-Seidel method to a point in cases where generalized bisection is not required. Let

$$Y_i = (y_1, y_2, \ldots, y_n)$$

denote the i-th row of the preconditioner, let

$$\mathbf{k}_i = Y_i F, \tag{1.7(a)}$$

and let

$$\begin{aligned} Y_i\mathbf{F}' = \mathbf{G}_i &= (\mathbf{g}_{i,1}, \mathbf{g}_{i,2}, \cdots, \mathbf{g}_{i,n}) \\ &= ([\underline{g}_{i,1}, \overline{g}_{i,1}], [\underline{g}_{i,2}, \overline{g}_{i,2}], \cdots, [\underline{g}_{i,n}, \overline{g}_{i,n}]). \end{aligned} \tag{1.7(b)}$$

With the above notation, we have

ALGORITHM 1.1. *(Preconditioned version of interval Gauss–Seidel) Do the following for $i = 1$ to n.*

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

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

[2] 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.

using interval arithmetic.

2. *(The new box is empty.) If* $\tilde{\mathbf{x}}_i \cap \mathbf{x}_i = \emptyset$*, then signal that there is no root of* F *in* $\mathbf{X}_k$*, and continue the generalized bisection algorithm.*
3. *(The new box is non-empty; prepare for the next coordinate)*
   (a) *Replace* $\mathbf{x}_i$ *by* $\mathbf{x}_i \cap \tilde{\mathbf{x}}_i$*.*
   (b) *Possibly re-evaluate* $\mathbf{F}'(\mathbf{X}_k)$ *to replace* $\mathbf{F}'$ *by an interval matrix whose corresponding widths are smaller.*

Moore and Jones proposed in [**15**] that Y be taken to be the inverse of the midpoint matrix of $\mathbf{F}'$ in the Krawczyk method for bounding $\tilde{\mathbf{X}}_k$ in (1.2). Also, Hansen proposed the identical in preconditioner Y for the interval Gauss-Seidel method in using this same inverse midpoint matrix in [**2**]. In fact, this particular preconditioner has been generally used and has been thought to be usually optimal (cf. eg. [**24**]). However, the examples in Sect. 3 of [**13**] show that this preconditioner can be far from optimal in the sense just described.

Here, we survey results we have obtained over the last several years on preconditioners for the interval Gauss-Seidel method. Though our purpose is not to give an elementary introduction, we will give references for such a study. We do not present proofs in their entirety, but cite the works where they originally appeared. This paper is meant to be a guide for this body of knowledge, as well as a formula reference for machine implementation.

In §2, we describe our general approaches to finding optimal preconditioners, and we present the resulting classification scheme. At the highest level of this classification, we have the "C" or "contraction" and the "S" or "splitting" preconditioners. In §3, we review our C-preconditioner results, while we review the S-preconditioner results in §4. We discuss numerical experiments in §5, while we point to future work in §6.

2. Classification of Preconditioners

In one dimension, all scalar non-zero preconditioners for the interval Newton method are equivalent. However, even in one dimension, we may illustrate the two possibilities which lead to the C-preconditioner and the S-preconditioner.

Example 2.1. *Suppose* $n = 1$ *and that*

$$f(x) = x^2 - 4,$$

so that (1.8) *becomes*

$$\tilde{\mathbf{x}}_i = \tilde{\mathbf{x}} = x_i - \frac{f(x_i)}{2\mathbf{x}_i} = x - \frac{f(x)}{2\mathbf{x}}.$$

If, in Example 2.1, the initial interval and point are

$$\mathbf{X}_k = \mathbf{x}_i = \mathbf{x} = [1, 2.5], \qquad x_i = x = 1.75,$$

then

$$\tilde{\mathbf{x}} = 1.75 - \frac{-.9375}{[2,5]} = 1.75 - [.1875, .46875] = [1.9375, 2.21875] \subset \mathbf{x},$$

so we may conclude that there is a unique root of f in $\mathbf{x}$, Furthermore, interval Newton iteration will converge to that root.

When $n > 1$, a C-preconditioner replaces f_i by a linear combination of the components of F for which, when (1.8) is applied, the width $w(\tilde{\mathbf{x}}_i \cap \mathbf{x}_i)$ of the interval $\tilde{\mathbf{x}}_i \cap \mathbf{x}_i$ is small. Formally, we have

DEFINITION 2.1. *A preconditioner row Y_i as in* (1.7) *is called a C-preconditioner, provided, in* (1.7(b)), $0 \notin \mathbf{g}_{i,i}$.

NOTE. *With C-preconditioners, the denominator in the interval Gauss-Seidel iteration* (1.8) *does not contain zero, so the image interval $\tilde{\mathbf{x}}_i$ is a single connected interval.*

On the other hand, if we took

$$\mathbf{x} = \mathbf{B} = [-3, 3], \qquad x = 0,$$

then we may use *extended interval arithmetic* (cf. [**14**], or [**3**] and [**8**] for the context here) to obtain

$$\tilde{\mathbf{x}} = 0 - \frac{-4}{[-6,6]} = \left(-\infty, -\frac{2}{3}\right] \cup \left[\frac{2}{3}, \infty\right),$$

so

$$\mathbf{x} \cap \tilde{\mathbf{x}} = \left[-3, -\frac{2}{3}\right] \cup \left[\frac{2}{3}, 3\right].$$

In this case, the result is two intervals, which isolate the two roots of f in $\mathbf{B}$. The total width of these two intervals is smaller than the total width of the two intervals $[-3, 0] \cup [0, 3]$ which we would have obtained from bisection.

When $n > 1$, an S-preconditioner replaces f_i by a linear combination of the components of F for which, when (1.8) is applied, the sum of the widths $w(\tilde{\mathbf{x}}_i \cap \mathbf{x}_i)$ of the interval $\tilde{\mathbf{x}}_i \cap \mathbf{x}_i$ is small. Formally, we have

DEFINITION 2.2. *A preconditioner row Y_i as in* (1.7) *is called an S-preconditioner, provided, in* (1.7(b)), $0 \in \mathbf{g}_{i,i}$ *and* $0 \neq \mathbf{g}_{i,i}$.

NOTE. *With S-preconditioners, the denominator in the interval Gauss-Seidel iteration* (1.8) *contains zero, so the image $\tilde{\mathbf{x}}_i$ is an extended interval consisting of either one or two semi-infinite real intervals.*

We will note a duality between C-preconditioners and S-preconditioners in §4.

Ideally, in cases for which a C-preconditioner is appropriate, we would use that Y_i in (1.7) which made the width $w(\mathbf{x}_i \cap \tilde{\mathbf{x}}_i)$ in Step 3(a) of Algorithm 1.1 minimal. However, computation of such a preconditioner in general appears to be a fairly difficult optimization problem, whereas, if certain possibilities are considered separately, the optimal preconditioner is accessible as the solution of a reasonably small linear programming problem. In particular, we consider three cases, depending upon whether we expect

(i) $\tilde{\mathbf{x}}_i \subseteq \mathbf{x}_i$,
(ii) the left endpoint $\underline{\tilde{x}}_i$ of $\tilde{\mathbf{x}}_i$ to be greater than the left endpoint $\underline{x}_i$ of $\mathbf{x}_i$, or
(iii) the right endpoint $\overline{\tilde{x}}_i$ of $\tilde{\mathbf{x}}_i$ to be less than the right endpoint $\overline{x}_i$ of $\mathbf{x}_i$.

Corresponding to these three cases, we have the following three types of C-preconditioners.

DEFINITION 2.3. *A C-preconditioner $Y_i^{C_W}$ is a W-optimal C-preconditioner if it minimizes the width $w(\overline{\tilde{x}}_i - \underline{\tilde{x}}_i)$ of the image $\tilde{\mathbf{x}}_i$ in* (1.8) *over all C-preconditioners.*

DEFINITION 2.4. *A C-preconditioner $Y_i^{C_L}$ is an L-optimal C-preconditioner if it maximizes the left endpoint $\underline{\tilde{x}}_i$ of the image $\tilde{\mathbf{x}}_i$ in* (1.8) *over all C-preconditioners.*

DEFINITION 2.5. *A C-preconditioner $Y_i^{C_R}$ is an R-optimal C-preconditioner if it minimizes right endpoint $\overline{\tilde{x}}_i$ of the image $\tilde{\mathbf{x}}_i$ in* (1.8) *over all C-preconditioners.*

We have an analogous situation for S-preconditioners. In particular, if we apply an S-preconditioner and the numerator in (1.8) does not contain zero, and a is the left endpoint of the numerator when we apply the minus sign, then $\tilde{\mathbf{x}}_i$ in (1.8) will consist of the union of two disjoint semi-infinite intervals

$$\tilde{\mathbf{x}}_i = (-\infty, a/\underline{g}_{i,i} + x_i] \cup [a/\overline{g}_{i,i} + x_i, \infty)$$

if neither $\underline{g}_{i,i}$ nor $\overline{g}_{i,i}$ equals zero. (The image $\tilde{\mathbf{x}}_i$ will consist of one such semi-infinite interval if one of $\underline{g}_{i,i}$ and $\overline{g}_{i,i}$ equals zero.) Thus, an interval

$$\mathbf{z}_i = (a/\underline{g}_{i,i} + x_i, a/\overline{g}_{i,i} + x_i) \tag{2.1}$$

will be excluded from $\mathbf{x}_i \cap \tilde{\mathbf{x}}_i$. (See [**7**] or Lemma 5.1 in [**8**].)

DEFINITION 2.6. *The interval* $\mathbf{z}_i$ *in* (2.1) *which is excluded in* $\mathbf{x}_i \cap \tilde{\mathbf{x}}_i$ *when we apply an S-preconditioner is termed a solution complement.*

Our three cases for the S-preconditioner depend on whether we expect

(i) $\mathbf{z}_i \subseteq \mathbf{x}_i$,
(ii) The right endpoint $a/\overline{g}_{i,i} + x_i$ of $\mathbf{z}_i$ to be greater than the right endpoint $\underline{x}_i$ of $\mathbf{x}_i$.
(iii) The left endpoint $a/\underline{g}_{i,i} + x_i$ of $\mathbf{z}_i$ to be less than the left endpoint $\overline{x}_i$ of $\mathbf{x}_i$.

DEFINITION 2.7. *An S-preconditioner* $Y_i^{S_W}$ *is a W-optimal S-preconditioner if it maximizes the width* $a/\overline{g}_{i,i} - a/\underline{g}_{i,i}$ *of the solution complement, subject to* $a > 0$ *and* $\underline{g}_{i,i} < 0 < \overline{g}_{i,i}$.

DEFINITION 2.8. $Y_i^{S_L}$ *is an L-optimal S-preconditioner if it minimizes the left endpoint* $a/\underline{g}_{i,i} + x_i$ *of the solution complement, subject to* $a > 0$ *and* $\underline{g}_{i,i} < 0 \leq \overline{g}_{i,i}$.

DEFINITION 2.9. $Y_i^{S_R}$ *is an R-optimal S-preconditioner if it maximizes the right endpoint* $a/\overline{g}_{i,i} + x_i$ *of the solution complement, subject to* $a > 0$ *and* $\underline{g}_{i,i} \leq 0 < \overline{g}_{i,i}$.

At times we may not need to form a linear combination of the rows of the interval Jacobian matrix for adequate convergence properties, but may enable or prove convergence by selecting the appropriate row of the Jacobian matrix. For these cases, we have

DEFINITION 2.10. *We say* Y_i *is a pivoting preconditioner provided* Y_i *is a multiple of a coordinate vector. We will speak of pivoting C-preconditioners and pivoting S-preconditioners as well as W-optimal, L-optimal, and R-optimal pivoting preconditioners.*

Finally, we clarify notation in

DEFINITION 2.11. *If we speak of a preconditioner* Y_i*, we will mean a preconditioner row for the* i*-th variable, where* i *is as in* (1.8).

We summarize results related to the three types of C-preconditioners in §3, while we do the same for the S-preconditioners in §4.

3. C-Preconditioners

First, we note

THEOREM 3.1. *([13]) for a given interval Jacobian matrix $\mathbf{F}'$, there exists a C-preconditioner Y_i if and only if at least one entry of the i-th column of $\mathbf{F}'$ does not contain 0.*

In what follows, we will assume that a C-preconditioner exists.

In our first published results in [**13**], we assumed that the guess point X_k was the midpoint of $\mathbf{X}_k$, i.e., that

$$x_j = (\underline{x}_j + \overline{x}_j)/2 \qquad \text{for } j = 1 \text{ to } n. \tag{3.1}$$

There, we formulated computation of the W-optimal C-preconditioner as solution of a linear programming problem. That linear programming problem was based on the following

THEOREM 3.2. *(Theorem 2.6 in [**13**]) Assume a C-preconditioner exists. Then the row vector Y_i which minimizes $w(\tilde{\mathbf{x}}_i)$ in (1.8) is a solution to*

$$\min_{\substack{Y_i \\ \underline{g}_{i,i}=1}} w\left(\frac{[a,b]}{[\underline{g}_{i,i}, \overline{g}_{i,i}]}\right),$$

where $[a,b]$ is the numerator in the quotient in (1.8), including the minus sign. Furthermore, suppose we compute a Y_i which solves the problem

$$\min_{\substack{Y_i \\ \underline{g}_{i,i}=1}} w\left\{[a,b]\right\}. \tag{3.2}$$

Then a Y_i which solves (3.2) is a Y_i for which $w(\tilde{\mathbf{x}}_i)$ is minimal, provided the resulting $[a,b]$ is such that $0 \in [a,b]$. If, on the other hand, Y_i solves (3.2) but the corresponding $[a,b]$ does not contain zero, then, for X_k arbitrary, the resulting $\mathbf{x}_i \cap \tilde{\mathbf{x}}_i$ is such that

$$w(\mathbf{x}_i \cap \tilde{\mathbf{x}}_i) < \max\left\{(x_i - \underline{x}_i), (\overline{x}_i - x_i)\right\}.$$

If we define j' by

$$j' = \begin{cases} j & \text{if} \quad j < i \\ j+1 & \text{if} \quad j \geq i, \end{cases}$$

then the linear programming problem presented in [**13**] for solving (3.2) is essentially

$$\text{minimize } W(V) = \sum_{j=1}^{n-1} v_j w(\mathbf{x}_{j'}) \tag{3.3a}$$

subject to

(3.3b)
$$v_j \geq -\left[\sum_{l=1}^{n} v_{l+(n-1)} \underline{f}'_{l,j'} - \sum_{l=1}^{n} v_{l+(2n-1)} \overline{f}'_{l,j'}\right], \qquad 1 \leq j \leq n-1,$$

(3.3c)
$$v_j \geq +\left[\sum_{l=1}^{n} v_{l+(n-1)} \overline{f}'_{l,j'} - \sum_{l=1}^{n} v_{l+(2n-1)} \underline{f}'_{l,j'}\right], \qquad 1 \leq j \leq n-1,$$

(3.3d)
$$1 = \left[\sum_{l=1}^{n} v_{l+(n-1)} \underline{f}'_{l,i} - \sum_{l=1}^{n} v_{l+(2n-1)} \overline{f}'_{l,i}\right],$$

and

(3.3e) $$v_j \geq 0 \qquad \text{for } 1 \leq j \leq 3n-1.$$

Once we compute the solution components v_j, $1 \leq j \leq 3n-1$, we compute the elements of the preconditioner $Y_i^{Cw} = (y_1, y_2, \ldots, y_n)$ by

(3.4) $$y_l = v_{l+(n-1)} - v_{l+(2n-1)}, \qquad 1 \leq l \leq n.$$

The following theorem appeared in [**13**].

THEOREM 3.3. *(Theorem 2.7 in [**13**]) Assume* (3.1), *and suppose that* v_j, $1 \leq j \leq 3n-1$ *form a solution of* (3.3). *If* Y_i *is defined by* (3.4), *then* Y_i *solves* (3.2), *provided that, for each* l *between* 1 *and* n, $v_{l+(n-1)} v_{l+(2n-1)} = 0$.

The following stronger theorem, observed by Novoa and stated as a conjecture in [**13**], shows that the solution to (3.3) combined with (3.4) *always* solves (3.2).

THEOREM 3.4. *(to appear in [**21**]) If* v_j, $1 \leq j \leq 3n-1$ *represent an optimum of* (3.3(a)) *subject to* (3.3(b)), (3.3(c)), (3.3(d)), *and* (3.3(e)), *and if* y_l, $1 \leq l \leq n$ *is defined through* (3.4), *then the resulting preconditioner* Y_i *solves the optimization problem* (3.2).

Though numerical results based on (3.3) in [**13**] show an improvement over the inverse midpoint preconditioner, we have developed several additional formulations for solving (3.2), and continue to investigate. Preliminary experiments indicate that the following formulation (whose derivation is explained in [**21**]) is a less costly way of obtaining the solution to (3.2) when the simplex method is used.

(3.5a) minimize $W(V) =$

$$\sum_{j=1}^{n-1}\left(-\sum_{l=1}^{n} v_{l+(n-1)}\underline{f}'_{l,j'} + \sum_{l=1}^{n} v_{l+(2n-1)}\overline{f}'_{l,j'} + v_j\right) w(\mathbf{x}_{j'})$$

subject to

(3.5b)

$$v_j \geq \sum_{l=1}^{n}\left(v_{l+(n-1)} - v_{l+(2n-1)}\right)\left(\underline{f}'_{l,j'} + \overline{f}'_{l,j'}\right), \qquad 1 \leq j \leq n-1,$$

(3.5c)

$$1 = \sum_{l=1}^{n} v_{l+(n-1)}\underline{f}'_{l,i} - \sum_{l=1}^{n} v_{l+(2n-1)}\overline{f}'_{l,i},$$

and

(3.5d) $$v_j \geq 0 \qquad \text{for } 1 \leq j \leq 3n-1.$$

As for (3.3), we then define $Y_i^{C_W} = (y_1, y_2, \ldots, y_n)$ by (3.4).

Results corresponding to Theorem 3.3 and Theorem 3.4 will be stated and proven in [**21**].

If we assume (3.1), we may formulate linear programming problems analogous to (3.5) for computing the L-optimal and R-optimal C-preconditioners. In particular, the following linear programming problem is solved to compute $Y_i^{C_L}$.

(3.6) minimize $L(V) =$ $$\sum_{l=1}^{n}\left(v_{l+(n-1)} - v_{l+(2n-1)}\right) f_i(X_k)$$

$$+ \tfrac{1}{2}\left\{\sum_{j=1}^{n-1}\left(-\sum_{l=1}^{n} v_{l+(n-1)}\underline{f}'_{l,j'} + \sum_{l=1}^{n} v_{l+(2n-1)}\overline{f}'_{l,j'} + v_j\right) w(\mathbf{x}_{j'})\right\}$$

subject to

(3.5(b)), (3.5(c)), and (3.5(d)).

Similarly, the following linear programming problem is solved to compute $Y_i^{C_R}$.

(3.7) minimize $-R(V) = -\sum_{l=1}^{n}\left(v_{l+(n-1)} - v_{l+(2n-1)}\right) f_i(X_k)$

$$+ \frac{1}{2}\left\{\sum_{j=1}^{n-1}\left(-\sum_{l=1}^{n} v_{l+(n-1)}\underline{f}'_{l,j'} + \sum_{l=1}^{n} v_{l+(2n-1)}\overline{f}'_{l,j'} + v_j\right) w(\mathbf{x}_{j'})\right\}.$$

subject to

$$(3.5(\mathrm{b})),\ (3.5(\mathrm{c})),\ \text{and}\ (3.5(\mathrm{d})).$$

The solutions to (3.6) and (3.7) give approximations to $Y_i^{C_L}$ and $Y_i^{C_R}$, respectively, and give $Y_i^{C_L}$ and $Y_i^{C_R}$ exactly when the resulting numerator for (1.8) contains zero. Such theory will appear in [**21**].

The following assertion is proved in a manner analogous to Lemma 5.6 in [**8**].

LEMMA 3.5. *Suppose we compute* $\tilde{\mathbf{x}}_i^{\mathrm{L}}$ *from* $\mathbf{X}_k$ *via* (1.8) *using an L-optimal C-preconditioner, and independently (with the same* $\mathbf{F}'$ *and* $\mathbf{X}_k$*) compute* $\tilde{\mathbf{x}}_i^{\mathrm{U}}$ *using a R-optimal C-preconditioner. Then the width of*

$$\tilde{\mathbf{x}}_i^{\mathrm{L}} \cap \tilde{\mathbf{x}}_i^{\mathrm{U}} \cap \mathbf{x}_i$$

is at least as small as the width of

$$\tilde{\mathbf{x}}_i^{\mathrm{A}} \cap \mathbf{x}_i,$$

where $\tilde{\mathbf{x}}_i^{\mathrm{A}}$ *is obtained through* (1.8) *by application of an arbitrary C-preconditioner.*

In practice, whether it is less costly to apply a single W-optimal preconditioner or to apply an L-optimal and R-optimal pair of preconditioners probably depends on the problem.

If we do not assume (3.1), i.e., if we do not assume that the guess point X_k is the midpoint vector for the interval vector $\mathbf{X}_k$, then we may still formulate solution of (3.2) as a linear programming problem; an explanation is given in [**21**]. However, such an LP problem would have far more variables and constraints than even (3.3), so we feel the cost to obtain it would outweigh the possible advantages of using a theoretically optimal preconditioner for off-center guess points.

We *may* compute L-optimal and R-optimal preconditioners via moderately sized linear programming problems analogous to (3.4), (3.5), and (3.6) when we allow the coordinates of X_k to (arbitrarily) be left endpoints, right endpoints, or midpoints of the corresponding coordinates of $\mathbf{X}_k$. Since such X_k may approximate the Neumaier predictor points (see [**17**]), these *endpoint preconditioners* may be useful in some contexts. See [**21**] for their formulation.

An interesting preconditioner may also be obtained if we ignore subdistributivity in computing the width of the numerator in (1.8). In particular, if we replace the intervals

$$\mathbf{g}_{i,j}(\mathbf{x}_j - x_j)$$

by the containing intervals

$$\sum_{l=1}^{n} y_{i,l}\mathbf{f}'_{l,j}(\mathbf{x}_j - x_j)$$

in the formula for computing $w(\tilde{\mathbf{x}}_i)$ (see [**6**]), we obtain the following LP problem, whose solution will approximate the W-optimal C-preconditioner.

$$\text{(3.8a)} \qquad \text{minimize } W(V) = \sum_{l=1}^{n} \left[(v_l + v_{l+n}) \sum_{\substack{j=1 \\ j\neq i}}^{n} w\left(\mathbf{f}'_{l,j}(\mathbf{x}_j - x_j)\right) \right]$$

subject to

$$\text{(3.8b)} \qquad 1 = \sum_{l=1}^{n} v_l \underline{f}'_{l,i} - \sum_{l=1}^{n} v_{l+n} \overline{f}'_{l,i},$$

and

$$\text{(3.8c)} \qquad v_j \geq 0 \qquad \text{for } 1 \leq j \leq 2n.$$

Analogous to (3.4), we then compute the y_l by

$$\text{(3.9)} \qquad y_l = v_l - v_{l+n}, \qquad 1 \leq l \leq n.$$

THEOREM 3.6. *(See [7]) The preconditioner corresponding to solution of* (3.8) *is a W-optimal pivoting preconditioner.*

Theorem 3.6 is due to the fact that (3.8) is a linear programming problem in $2n$ variables and only one constraint.

Pivoting preconditioners are obtainable with less computation than general optimal preconditioners, especially when the number of equations and variables n is large. For this reason, we conclude this section with a summary of such preconditioners. The following width characterization gives insight and is useful for computing pivoting preconditioners for arbitrary guess points X_k.

THEOREM 3.7. *(Theorem 2.4 in [5] or Theorem 4.1.2 in [7]) In* (1.8), *if* $x_j \in \mathbf{x}_j$ *for* $1 \leq j \leq n$,
if $\mathbf{k}_i \in -\sum_{\substack{j=1 \\ j\neq i}}^{n} \mathbf{g}_{i,j}(\mathbf{x}_j - x_j)$, *and if*
$0 \notin \mathbf{g}_{i,i}$, *then*

$$\text{(3.10)} \qquad w(\tilde{\mathbf{x}}_i) = \sum_{\substack{j=1 \\ j\neq i \\ \sigma_j = 1}}^{n} \max\{|\underline{g}_{i,j}|, |\overline{g}_{i,j}|\} w(\mathbf{x}_j^1) + \sum_{\substack{j=1 \\ j\neq i \\ \sigma_j = 0}}^{n} \lambda_j^* w(\mathbf{g}_{i,j}) w(\mathbf{x}_j^1),$$

where

$$\lambda_j^* = \begin{cases} \max\left\{(x_j - \underline{x}_j), (\overline{x}_j - x_j)\right\} / w(\mathbf{x}_j^1) & \text{if} \quad w(\mathbf{x}_j^1) \neq 0, \\ 0 & \text{otherwise,} \end{cases}$$

where

$$\sigma_j = \begin{cases} 0 & \text{if} \quad |\underline{g}_{i,j}|(1/\lambda_j^* - 1) < |\overline{g}_{i,j}| < |\underline{g}_{i,j}|\lambda_j^*/(1 - \lambda_j^*) \\ & \text{and} \quad 0 \in \mathbf{g}_{i,j} = [\underline{g}_{i,j}, \overline{g}_{i,j}], \\ 1 & \text{otherwise,} \end{cases}$$

and where

$$\mathbf{x}_j^1 = \begin{cases} \mathbf{x}_j \cap \tilde{\mathbf{x}}_j & \text{if} \quad j \leq i, \\ \mathbf{x}_j & \text{if} \quad j > i. \end{cases}$$

In [**5**], we give details on using (3.10) in computing the pivoting C-preconditioner which is W-optimal over all pivoting C-preconditioners. Though such preconditioners are not adequately general to be effective on all problems, on many they work significantly better than no preconditioner at all, and they require two orders of magnitude (a proven $O(n^2)$ versus an empirically observed $O(n^3)$) less arithmetic operations than preconditioners based on linear programming problems or solution of linear systems when the size n of the system is large. (See the tables in Section 4 of [**5**].) They may thus be used in a hybrid algorithm in which a pivoting preconditioner is applied first, followed by an LP preconditioner only if the pivoting preconditioner does not work.

The following algorithm selects the W-optimal pivoting C-preconditioner.

ALGORITHM 3.11. *(Algorithm 3.9 in [**5**]; determination of the W-optimal pivoting C-preconditioner)*

1. *Check* $w(\mathbf{x}_j^1)$ *for* $j = 1, 2, \ldots, i-1, i+1, \ldots, n$ *to find* $\mathcal{N}_{\text{col}}$, *where*

$$\mathcal{N}_{\text{col}} = \left\{ j \in \{1, 2, \ldots, i-1, i+1, \ldots, n\} \mid w(\mathbf{x}_j^1) \neq 0 \right\}.$$

2. *Check the* i*-th column of the interval Jacobian matrix* $\mathbf{F}'$ *to determine* $\mathcal{N}_{\text{row}}$, *where*

$$\mathcal{N}_{\text{row}} = \left\{ j \in \{1, 2, \ldots, n\} \mid 0 \notin \mathbf{f}'_{j,i} \right\}.$$

3. *Pick* $m_0 \in \mathcal{N}_{\text{row}}$ *which minimizes* $w(\tilde{\mathbf{x}}_i)$, *over all* $m \in \mathcal{N}_{\text{row}}$, *where*

$w(\tilde{\mathbf{x}}_i)$ is defined as in (3.10) *by*

$$w(\tilde{\mathbf{x}}_i) = \left[\sum_{\substack{j \in \mathcal{N}_{\mathrm{col}} \\ \sigma_j = 1}} \max\{|\underline{f}'_{m,j}|, |\overline{f}'_{m,j}|\} w(\mathbf{x}_j^1) + \sum_{\substack{j \in \mathcal{N}_{\mathrm{col}} \\ \sigma_j = 0}} \lambda_j^* w(\mathbf{f}'_{m,j}) w(\mathbf{x}_j^1) \right] \Big/ \min\{|\underline{f}'_{m,i}|, |\overline{f}'_{m,i}|\}.$$

4. *Choose $Y_i = e_{m_0}^T$, where $e_{m_0}^T$ is that unit row vector with* 1 *in the m_0-th coordinate and* 0 *in the other coordinates.*

4. S-Preconditioners

The S-preconditioners depend on the concept of extended interval arithmetic; see [**14**, pp. 66–68] for a definition.

We introduce the section with an examination of conditions under which S-preconditioners exist.

THEOREM 4.1. *(Theorem 4.3 in [8] or Theorem 2.5.3 in [7]) An S-preconditioner Y_i exists provided i-th column of the interval Jacobian matrix $\mathbf{F}'(\mathbf{X}_k)$ satisfies one of the following conditions.*

(1) One or more entries contains zero but is not zero, or

(2) at least two entries do not contain zero, and at least one of these has nonzero width.

Furthermore, if neither (1) nor (2) is valid, then there does not exist an S-preconditioner.

The conditions in Theorem 4.1 complement the conditions under which we would apply an S-preconditioner in the sense of

REMARK 4.2. *If the i-th column of the interval Jacobian matrix contains only one nonzero entry and that entry does not contain zero, then a C-preconditioner row exists for the i-th variable. On the other hand, when the Jacobian matrix is a scalar nonsingular matrix, then not only does a C-preconditioner exist, but the width characterization* (3.10) *shows that either an optimal C-preconditioner or the inverse midpoint preconditioner will result in $w(\tilde{\mathbf{x}}_i) = 0$.*

There is an interesting duality between L-optimal C-preconditioners and R-optimal S-preconditioners, and between R-optimal C-preconditioners and L-optimal S-preconditioners ([**7**]). In particular, if we let

$[a, b]$ denote the numerator in (1.8) (including the minus sign), and $0 \in [a, b]$ but $0 \notin \mathbf{g}_{i,i}$, then the left endpoint of the image interval is

$$x_i + a/\underline{g}_{i,i}.$$

On the other hand, if $[a, b] > 0$ but $0 \in \mathbf{g}_{i,i}$ (which can be arranged if an S-preconditioner exists), then the right endpoint of the solution complement is

$$x_i + a/\overline{g}_{i,i}$$

In Theorem 3.2, as well as in (3.3d), and (3.5c), we normalized the C-preconditioner through the constraint

$$\underline{g}_{i,i} = 1.$$

If the right endpoint $\overline{g}_{i,i}$ of the denominator is nonzero (so that $\tilde{\mathbf{x}}_i$ contains the semi-infinite interval $[x_i + a/\overline{g}_{i,i}, \infty)$, then we may similarly normalize the S-preconditioner via the constraint

$$\overline{g}_{i,i} = 1.$$

The analogue to (3.2) for the L-optimal C-preconditioner (upon which (3.6) is based; see [**21**]) then becomes

$$\min_{\substack{Y_i \\ \underline{g}_{i,i}=1}} -a, \tag{4.1}$$

where we obtain an L-optimal C-preconditioner if $a < 0$. In contrast, the corresponding analogue for the R-optimal S-preconditioner is

$$\min_{\substack{Y_i \\ \overline{g}_{i,i}=1}} -a, \tag{4.2}$$

where we obtain an S-preconditioner if $\underline{g}_{i,i} \leq 0$ at the optimal point, and where we obtain a nonempty solution complement if $a > 0$ at the optimal point.

The above duality is exploited in [**7**, Chapter 6] to unify and simplify theory of C-preconditioners and S-preconditioners.

In (4.2), we may use $\underline{g}_{i,i} \leq 0$ as an additional constraint in constructing the linear programming problem, to ensure that the resulting preconditioner is an S-preconditioner. However, we may instead construct the LP problem from (4.2) exclusively, and simply use the resulting C-preconditioner if $\underline{g}_{i,i} > 0$ at the optimal point. The latter

scheme is preferable in practice, since the right endpoint of the resulting C-preconditioner is at least as large as the left endpoint of the solution complement in the S-preconditioner which would have been obtained had the constraint $\underline{g}_{i,i} \leq 0$ been used.

A similar duality between the R-optimal C-preconditioner and the L-optimal S-preconditioner holds.

If, analogously to derivation of (3.8), we ignore subdistributivity in the expression for a in (4.2), we obtain a pivoting preconditioner which allows for this splitting. It takes $O(n^2)$ operations per variable to determine this preconditioner. On the other hand, we may use a special subtraction technique pointed out independently by Novoa (and similar to that in [**18**]) which avoids interval dependencies to apply all possible pivoting preconditioners for all variables in $O(n^2)$ total operations. This technique is embodied in the following algorithm.

ALGORITHM 4.3. *([**7**] or [**8**]; solving for each variable in each row, with a special subtraction step) Do all arithmetic with directed roundings.*

Do the following for for $m = 1$ to n.

1. *Compute*

$$[\alpha, \beta] = \mathbf{f}_m + \sum_{j=2}^{n} \mathbf{f'}_{m,j}(\mathbf{x}_j - x_j)$$

Do the following for $i = 1$ to n.

2. *Compute*

$$\tilde{\mathbf{x}}_i = x_i - \frac{[\alpha, \beta]}{\mathbf{f'}_{m,i}}$$

using interval arithmetic.

3. *If $\tilde{\mathbf{x}}_i \cap \mathbf{x}_i = \emptyset$, then signal that there is no root of F in $\mathbf{X}$, and continue the generalized bisection algorithm.*
4. *(Prepare for the next coordinate)*
   (a) *Replace $\mathbf{x}_i$ by $\mathbf{x}_i \cap \tilde{\mathbf{x}}_i$.*[3]
   (b) *Possibly re-evaluate $\mathbf{F'}(\mathbf{X}_k)$ to replace $\mathbf{F'}$ by an interval matrix whose corresponding widths are smaller.*

Next i.

4. *(Special subtraction)*

$$\alpha \leftarrow \alpha + u_i - u_{i+1},$$
$$\beta \leftarrow \beta + v_i - v_{i+1},$$

[3] We may not do this step if $\mathbf{x}_i \cap \tilde{\mathbf{x}}_i$ consists of two intervals and the solution complement is not sufficiently large, where "sufficiently" is determined by a heuristic parameter. Also, note that in the case when $\mathbf{x}_i \cap \tilde{\mathbf{x}}_i$ consists of two boxes, we will push one of them onto a stack at this point.

where

$$[u_i, v_i] = \mathbf{f}'_{m,i}(\mathbf{x}_i - x_i),$$

where we use the values of $\mathbf{F}'$ *and* $(\mathbf{x}_i - x_i)$ *before Step 4a in computing* $[u_i, v_i]$ *and the values after Step 4a in computing* $[u_{i+1}, v_{i+1}]$

Next m.

In [**21**] and [**8**], an alternate optimization problem which leads to a similar hybrid S-preconditioner or C-preconditioner is presented. This optimization problem is

$$\min_{\substack{Y_i \\ a=1}} \left(\max \left\{ \underline{g}_{i,i}, \overline{g}_{i,i} \right\} \right), \tag{4.3}$$

where a is the left endpoint of the numerator in (1.8).

This minimization problem is related to C- and S-preconditioners in the following way.

THEOREM 4.4. *([**21**]) Let*

$$D = \max \left\{ -\underline{g}_{i,i}, \overline{g}_{i,i} \right\}$$

correspond to the solution of (4.3). *The following are true.*

(1) (4.3) *has a feasible point (so that* D *exists) if and only if there exists a preconditioner for which the numerator in* (1.8) *does not contain zero.*

(2) If $D > 0$, *and the resulting preconditioner is an S-preconditioner, then the solution complement (and hence, up to the boundaries, the portion of* $\mathbf{x}_i$ *which is eliminated in* $\tilde{\mathbf{x}}_i$*) is of width at least* $2/D$.

(3) If $D > 0$ *and the resulting preconditioner is a C-preconditioner, then* $w(\tilde{\mathbf{x}}_i)$ *differs from* $w(\mathbf{x}_i)$ *by at least* $\frac{1}{2}w(\mathbf{x}_i) + 1/D$.

(4) If (4.3) *does not have a feasible point, then the numerator in* (1.8) *will contain zero when we use the preconditioner obtained from* (3.5), (3.6), *or* (3.7). *Therefore, the preconditioner will be W-optimal, L-optimal, or R-optimal, respectively.*

(5) If $D = 0$, *then the original nonlinear system of equations* (1.1) *has no solution in* $\mathbf{X}_k$.

Theorem 4.4 asserts that, if (4.3) has a solution, then the resulting preconditioner maximizes a measure of the size of the complement of $\mathbf{x}_i$ in $\tilde{\mathbf{x}}_i$, and that, if (4.3) does not have a solution, then any of the linear programming problems for C-preconditioners will give optimal preconditioners.

DEFINITION 4.5. *The preconditioner defined by* (4.3) *will be called the* S^N, *or normalized numerator preconditioner.*

Novoa has exhibited a generalization of the following linear programming problem for (4.3) in [**21**].

$$\text{(4.4a)}\quad \text{minimize } D(V) = v_i - \sum_{l=1}^{n} \Big[(v_{l+n} - v_{l+2n})\, \overline{f}'_{l,i}(\mathbf{X}_k) - v_{l+n} \left(\overline{f}'_{l,i}(\mathbf{X}_k) - \underline{f}'_{l,i}(\mathbf{X}_k) \right) \Big] .$$

subject to

$$\text{(4.4b)}\qquad v_j \geq - \sum_{l=1}^{n} (v_{l+n} - v_{l+2n}) \left(\underline{f}'_{l,j'} + \overline{f}'_{l,j'} \right), \qquad 1 \leq j \leq n$$

and

$$\text{(4.4c)}\quad 1 = \sum_{l=1}^{n} (v_{l+n} - v_{l+2n})\, f_l(X_k) - \frac{1}{2} \sum_{\substack{j=1 \\ j \neq i}}^{n} w(\mathbf{x}_j) \left\{ v_j + \sum_{l=1}^{n} \Big[- (v_{l+n} - v_{l+2n})\, \overline{f}'_{l,j'}(\mathbf{X}_k) + v_{l+n} \left(\overline{f}'_{l,j'}(\mathbf{X}_k) - \underline{f}'_{l,j'}(\mathbf{X}_k) \right) \Big] \right\} ,$$

and subject to the natural constraints

$$\text{(4.4d)}\qquad v_j \geq 0 \qquad \text{for } 1 \leq j \leq 3n.$$

In contrast to the linear programming problem corresponding to (4.2), we (Novoa) have been able to prove

THEOREM 4.6. *([**21**]) If we set* $y_l = v_{l+n} - v_{l+2n}$ *for* $1 \leq l \leq n$, *then any solution to* (4.4) *is a solution to* (4.3). *Conversely, if we set* $v_{l+n} = \max\{y_l, 0\}$ *and* $v_{l+2n} = \max\{-y_l, 0\}$, *then any solution of* (4.3) *is a solution to* (4.4).

5. NUMERICAL EXPERIMENTS

We will not exhaustively reproduce results here which have appeared or will appear elsewhere, but will summarize, interpret, and give references as appropriate.

In most of the experiments, we use modifications of the portable Fortran 77 interval Newton / generalized bisection code in [**12**], and we employ cost measures such as the total number of boxes, total number of interval Jacobian matrices, total number of evaluations of F, and total CPU time for completion of the entire generalized bisection algorithm.

Our experiments are carried out on various machines, including on an IBM3090 on a single processor in scalar mode with the VS-Fortran compiler and on an 80286-based IBM PC-compatible with a numeric co-processor and Microsoft Fortran. Though CPU times varied on these machines, other cost measures such as number of boxes are almost identical for a given program.

The test set includes a set of seventeen problems originally assembled in [**10**]. We also have tried our techniques on additional problems, such as the unreduced system from a heart dipole model ([**16**]), a vision problem ([**4**]), and some non-polynomial systems such as the Layne Watson exponential cosine function of [**23**].

Original experiments with the inverse midpoint preconditioner, in which we allowed zero in the denominator of (1.8) (and hence, extended interval arithmetic), appear in [**10**].

Experiments with the prototypical linear programming problem (3.3) appear in [**13**]. In those experiments, only (3.3) was used to obtain preconditioners (without ever allowing a splitting preconditioner), and the results were compared to the inverse midpoint preconditioner, in which we did not allow zero in the denominator of (1.8), and to use of no preconditioner at all. In all cases, (3.3) led to significantly less function and Jacobian evaluations than the inverse midpoint preconditioner and no preconditioner at all, and it also resulted in less CPU time for most problems. For the five-dimensional version of Brown's almost linear function, approximately 300 *times* less Jacobian evaluations and 100 *times* less CPU time were required to complete the generalized bisection.

In [**13**], a dramatic improvement also occurred with a variant of Powell's singular function when we used (3.3). However, situation like that can be handled even more efficiently by taking advantage of the fact that the classical Newton's method usually converges even to roots at which the Jacobian matrix is singular, and by appropriately subdividing the region. See [**11**].

The way the simplex tableau is set up and the method of solution are critical. In the results in [**13**], the amount of CPU time in the linear programming problems was roughly a factor of 10 smaller than that in our first experiments.

In [**13**] (where we used (3.3)), the portion of the CPU time spent in the linear programming problem varied from somewhat less than half to

over 90%, with proportionally more time spent in linear programming in problems with more variables n. Experiments to appear in [**21**] and elsewhere use (3.5) instead of (3.3), and use a different simplex method. In these experiments, all cost measures except CPU time were identical to the experiments with (3.3), except in certain cases where the linear programming for (3.3) failed due to roundoff error, but that based on (3.5) did not. However, the CPU times spent in linear programming varied from between roughly a factor of 2/3 to a factor of 1/6 of those corresponding to (3.3).

We have done some experiments with linear programming C-preconditioners based on taking the guess point X_k to be the left or right endpoint of the interval $\mathbf{X}_k$ instead of the midpoint; see [**21**]. Such *endpoint preconditioners* were initially considered because it was thought we could formulate smaller linear programming problems for them. However, these experiments did not reveal any particular advantage to endpoint preconditioners. We are presently working on some ideas for combining endpoint preconditioners to get a better hybrid algorithm.

We examine pivoting C-preconditioners in [**5**], and in summary form in [**6**]. In [**5**], we presented a table of the numbers of operations, broken down by interval and scalar operations, as a function of the dimension n, the number of nonzeros ν in the interval Jacobian matrix, and the number ω of entries of the interval Jacobian matrix which do not contain zero. That table reveals that, for general dense systems, $O(n^2)$ total operations are required for the interval Gauss-Seidel method with the pivoting preconditioner, while $O(n^3)$ are required for the inverse midpoint preconditioner, or by applying (1.8) n^2 times, once for each variable in each equation.[4]

Numerical experiments in [**5**] with Broyden's banded function (treated as a dense system), for n between 5 and 40, corroborate these operations counts, and show that the inverse midpoint preconditioner rapidly becomes impractical for this problem for $n > 10$; similarly, unpublished experiments on Brown's almost linear function (treated as a dense system) for n between 5 and 20 indicate that the amount of time required to complete generalized bisection when (3.3) is used goes up at a rate of between $O(n^4)$ and $O(n^5)$. Thus, pivoting preconditioners should be preferable, if applicable, for large problems.

Experiments in [**5**] on those problems in the test set for which pivoting preconditioners are practical indicate the potential importance of such preconditioners in practice. In practice, however, we will probably

[4] This was before discovery of the special subtraction technique in [**8**], in which only $O(n^2)$ operations are required for each variable and each row.

use Algorithm 4.3.

Experiments comparing pivoting preconditioners in conjunction with the Neumaier predictor X_k as in [**17**] to the same preconditioners with X_k equal to the midpoint have given mixed results. See [**7**].

Comparison of the inverse midpoint preconditioner results when we allow $0 \in \mathbf{g}_{i,i}$ in (1.8) to the results of similar experiments in which we do not apply (1.8) when $0 \in \mathbf{g}_{i,i}$ indicate that use of extended interval arithmetic (and splitting preconditioners) improves performance. (See also the discussion in [**2**].) However, preliminary experiments with pivoting S-preconditioners or with indiscriminate application of linear programming S-preconditioners have shown equivocal results. In [**8**] and in [**21**], respectively, we report results from the following hybrid algorithms, which are intended to be applicable for general systems.

ALGORITHM 5.1. *(All pivoting preconditioners possibly followed by a W-optimal C-preconditioner; [**8**] and [**7**])*

1. *Input a heuristic parameter* τ, $0 \leq \tau \leq 1$, *a zero tolerance* ϵ_w *and* $\mathbf{X}_k = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n)$.
2. *Apply Algorithm 4.3 to* $\mathbf{X}_k$*; call the new* $\mathbf{X}_k$

$$\mathbf{X}_k^+ = (\mathbf{x}_1^+, \mathbf{x}_2^+, \ldots, \mathbf{x}_n^+).$$

3. *Apply* (1.8) *for those* i *for which* $w(x_i^+) \geq \epsilon_w |x_i^+|$ *and* $w(x_i^+) \geq \tau w(x_i)$*; use the preconditioner computed through* (3.5).
4. *Repeat steps 2 and 3 of this algorithm or else apply generalized bisection, as appropriate.*

ALGORITHM 5.2. *(The preconditioner based on* (4.4) *followed by a W-optimal C-preconditioner as appropriate; see [**21**].)*

1. *Input a heuristic parameter* τ, $0 \leq \tau \leq 1$, *a zero tolerance* ϵ_w $\mathbf{X}_k = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n)$, *and an additional preconditioner selection parameter* η, $0 \leq \eta \leq 1$.
2. *Compute the quantities* $\mathcal{N}_i$, $1 \leq i \leq n$ *through the formula*

$$\mathcal{N}_i = \max_{1 \leq m \leq n} \left\{ \sum_{\substack{j=1 \\ j \neq i}} \max \left\{ |\underline{f}'_{m,j}|, |\overline{f}'_{m,j}| \right\} \right\} w(\mathbf{x}_i).$$

(Note: These quantities can be computed in $O(n^2)$ *operations, for dense systems, using a technique similar to that in Algorithm 5.1.) Do the following for* $i = 1$ *to* n.

3. *(Application of the S^{N}-preconditioner if appropriate)*
   If $\|F(X_k)\|_\infty > \eta \mathcal{N}_i$, *then apply* (1.8), *where the preconditioner is computed through* (4.4); *call the new* $\mathbf{X}_k$ $\mathbf{X}_k^+ = (\mathbf{x}_1^+, \mathbf{x}_2^+, \ldots, \mathbf{x}_n^+)$.
4. *If* $w(x_i^+) \geq \epsilon_w |x_i^+|$ *and* $w(x_i^+) \geq \tau w(x_i)$ *then apply* (1.8); *use the preconditioner computed through* (3.5).

Next i.

5. *Repeat the loop on* i *in this algorithm or else apply generalized bisection, as appropriate.*

Our assumption in the above two algorithms is that the advantage implied by Lemma 3.5 of (3.6) and (3.7) over (3.5) is outweighed by the cost of computing the solution to two linear programming problems instead of just one.

Here, to indicate the progress we have made since [**10**] and [**13**], we give some experimental results for generalized bisection (similar to Algorithm 1.1, pp. 805-806 of [**13**]) in conjunction with an algorithm similar (but somewhat more sophisticated than) Algorithm 5.1. Here, we took $\epsilon_w = 10^{-5}/16$, ϵ_F (from Algorithm 1.1 in [**13**]) to be 10^{-10}; we took $\tau = .8$. Some modifications are

1. reuse of the computed linear programming preconditioner rows for sibling boxes in the generalized bisection algorithm, provided the preconditioner resulted in a sufficent width reduction for the parent box, and
2. iteration of portions of step 2 of Algorithm 5.1 twice before trying step 3.

These results appear in Table 1. As in [**13**], the problem numbers refer to problems from [**10**]; also as in [**13**], we have reported only on those problems which are nontrivial for the algorithm. The column labelled "n" gives the dimension of the problem, the column labelled "NBOX" gives the number of boxes considered in the generalized bisection algorithm, the column labelled "NFUN" gives the total number of interval function evaluations, and the column labelled "NJAC" gives the total number of interval Jacobian evaluations. The column labelled "CPU sec." gives CPU time on an IBM 3090 with a single processor, using Fortran 77 code compiled with VS-Fortran, and measuring the time with the routine "CPUTIME"; the time represents all functions excluding input and output.

A comparison of these results with those in [**13**] reveals improvements we have made during the past several years.

TABLE 1.
Some results with generalized bisection and a variant of Algorithm 5.2

#	n	NBOX	NFUN	NJAC	CPU sec.
1	2	10	75	37	0.048934
2	2	15	74	36	0.087997
3	4	87	334	137	0.467594
4	5	35	107	50	0.587334
9	2	11	33	16	0.032724
10	4	35	195	94	0.508553
11	8	103	434	203	2.250024
12	3	187	1022	487	2.140635
14	2	14	58	27	0.037991
15	2	3	5	2	0.003180
16	4	1	4	2	0.006693
17	5	19	63	29	0.297805
Totals		520	2404	1120	6.469464

We have also experimented with Algorithm 5.2, but we have not observed significant advantages of that algorithm in its present form over Algorithm 5.1.

6. Summary and Future Work

During the past several years, we have been studying preconditioners based on various optimality conditions. These preconditioners implicitly take account of scaling, differences in nonlinearity, etc. Significant improvements can be obtained in interval Newton methods based on the interval Gauss-Seidel method if such preconditioners are used in place of the inverse midpoint preconditioner . Use of appropriate heuristics to combine various preconditioners in an overall algorithm will give general root-finding software whose range of applicability exceeds that of software based exclusively on the inverse midpoint preconditioner or any other particular preconditioner .

We have studied general root-finding extensively, and have developed good general algorithms. One possible improvement would be to develop an appropriate heuristic to specify when to use the inverse midpoint preconditioner instead of a linear-programming based preconditioner. The inverse midpoint preconditioner is appropriate when the problem is well-scaled and when the Jacobian matrix is well-conditioned and approximately a scalar matrix. This is because, for dense larger systems, obtaining the inverse midpoint preconditioners seems to require roughly at least a factor of n less work than solving the linear programming problems.

Possibly our most significant future work will entail application of our knowledge to specific problems. For example, the numerous low-

dimensional problems which would arise in a robust geometric computation algorithm embedded in a CAD system would undoubtedly require different types of preconditioners than a large, sparse system arising from discretization of a nonlinear elliptic partial differential equation. It is unreasonable to try to design general software which will perform optimally in all such cases, and significant work can be done to take advantage of problem-specific properties.

Another line of research involves algorithmic design on parallel architectures. If efficiency is measured by total clock time, then the relative merits of the various preconditioners may differ, depending on the architecture. For example, if each box is allocated to a processor and there are a large number of processors, then there may be less of a penalty in forming two boxes via a splitting preconditioner.

References

1. Alefeld, Götz, and Herzberger, Jürgen, "Introduction to Interval Computations," Academic Press,, 1983.
2. Hansen, E. R., and Greenberg, R. I., *An Interval Newton Method*, Appl. Math. Comput. **12** (1983), 89–98.
3. Hansen, E., and Sengupta, S., *Bounding Solutions of Systems of Equations Using Interval Analysis*, BIT **21** (1981), 203–211.
4. Holt, Robert. J., *A 9-Dimensional Polynomial System Related to Computer Vision*, private communication.
5. Hu, C.-Y., and Kearfott, R. B., *A Width Characterization and Row Selection Strategy for the Interval Gauss–Seidel Method*, manuscript.
6. Hu, C.-Y., and Kearfott, R. B., *A Pivoting Seheme for the Interval Gauss-Seidel Method: Numerical Experiments*, to appear, in "Approximation , Optimization, and Computing, ed. A. G. Law and C.-L. Wang," Elsevier Science Publishers, Amsterdam, 1990.
7. Hu, C.-Y., *Preconditioners for Interval Newton Methods*, Ph.D. Dissertation, University of Southwestern Louisiana.
8. Hu, C.-Y., Kearfott, R. B., and Novoa, M., *Splitting Preconditioners for the Interval Gauss–Seidel Method*, to be submitted, SIAM J. Numer. Anal. (1990).
9. Kearfott, R. B., *Abstract Generalized Bisection and a Cost Bound*, Math. Comp. **49** 179 (1987), 187–202.
10. Kearfott, R. B., *Some Tests of Generalized Bisection*, ACM Trans. Math. Software **13** 3 (1987), 197–220.
11. Kearfott, R. B., *Interval Newton / Generalized Bisection When There are Singularities near Roots*, accepted for publication, Annals of Operations Research (1990).
12. Kearfott, R. B., and Novoa, M., *A Program for Generalized Bisection*, ACM Trans. Math. Software (1990) (to appear).
13. Kearfott, R. B., *Preconditioners for the Interval Gauss-Seidel method*, SIAM J. Numer. Anal. **27** 3 (1990), 804–822.
14. Moore, Ramon E., "Methods and Applications of Interval Analysis," SIAM, Philadelphia, 1979.

15. Moore, R. E., and Jones, S. T., *Safe Starting Regions for Iterative Methods*, SIAM J. Numer. Anal. **14** 6 (1977), 1051–106.
16. Morgan, A. P., Sommese, A. J., and Watson, L. T., *The Mathematical Reduction of a System of Equations for a Heart Dipole Problem*, IEEE Trans. Biomed. Eng. (1989) (to appear).
17. Neumaier, A., *Interval Iteration for Zeros of Systems of Equations*, BIT **25** 1 (1985), 256–273.
18. Neumaier, A., *The enclosure of solutions of parameter-dependent systems of equations*, in "Reliability in Computing: The Role of Interval Methods in Scientific Computing," Academic Press, New York, 1988, pp. 269-286.
19. Neumaier, A., "Interval Methods for Systems of Equations," in press, Cambridge University Press, 1990.
20. Nickel, K., *On the Newton Method in Interval Analysis*, MRC Technical Summary Report no. 1136, Mathematics Research Center, the University of Wisconsin at Madison, Madison, WI.
21. Novoa, M., *Linear Programming Preconditioners for the Interval Gauss–Seidel Method and their Implementation in Generalized Bisection*, Ph.D. Dissertation, University of Southwestern Louisiana.
22. Shearer, J. M., and Wolfe, M. A., *Some Computable Existence, Uniqueness, and Convergence Tests for Nonlinear Systems*, SIAM J. Numer. Anal. **22** 6 (1985), 1200–1207.
23. Watson, L. T., *A Globally Convergent Algorithm for Computing Fixed Points of C2 Maps*, Appl. Math. Comput. **5** (1979), 297–311.
24. Xiaojun, Chen, and Deren, Wang, *On the optimal properties of the Krawczyk type interval operator*, preprint, Freiburger Intervall-Berichte **87** 5 (1987), 1–5.

Keywords. nonlinear algebraic systems, Newton's method, interval arithmetic, Gauss–Seidel method, preconditioners
1980 *Mathematics subject classifications*: Primary: 65H10; Secondary: 65G10

U.S.L Box 4-1010, Lafayette, LA 70504