A Comparison of Some Methods for Bounding Connected and Disconnected Solution Sets of Interval Linear Systems

R. Baker Kearfott*

December 4, 2007

Abstract

Finding bounding sets to solutions to systems of algebraic equations with uncertainties in the coefficients, as well as rapidly but rigorously locating all solutions to nonlinear systems or global optimization problems, involves bounding the solution sets to systems of equations with wide interval coefficients. In many cases, singular systems are admitted within the intervals of uncertainty of the coefficients, leading to unbounded solution sets with more than one disconnected component. This, combined with the fact that computing exact bounds on the solution set is NP-hard, limits the range of techniques available for bounding the solution sets for such systems. However, the componentwise nature and other properties make the interval Gauss–Seidel method suited to computing meaningful bounds in a predictable amount of computing time. For this reason, we focus on the interval Gauss–Seidel method. In particular, we study and compare various preconditioning techniques we have developed over the years but not fully investigated, comparing the results. Based on a study of the preconditioners in detail on some simple, specially-designed small systems, we propose two heuristic algorithms, then study the behavior of the preconditioners on some larger, randomly generated systems, as well as a small selection of systems from the Matrix Market collection.

Keywords: numerical linear algebra, global optimization, validated computing, interval analysis
AMS Subject Classifications: 65F10, 65G20, 65K99

1 Introduction

It is sometimes appropriate to express uncertainties in the coefficients of linear algebraic systems as intervals. Also, it is sometimes appropriate to express the

*Department of Mathematics, University of Louisiana, U.L. Box 4-1010, Lafayette, Louisiana, 70504-1010, USA (rbk@louisiana.edu).

effect of nonlinearities in a system as intervals of possible values[1], also leading to linear systems with interval coefficients. That is, we obtain an interval linear system of equations

$$\boldsymbol{A}x = \boldsymbol{b}, \tag{1}$$

where $\boldsymbol{A}$ is an m by n matrix with interval coefficients and $\boldsymbol{b}$ is an m-vector with interval coefficients. It is desirable to find sharp bounds on the solution set to (1), where the solution set is defined as

$$\Sigma(\boldsymbol{A}, \boldsymbol{b}) = \{x \mid Ax = b \text{ for some } A \in \boldsymbol{A} \text{ and some } b \in \boldsymbol{b}\}. \tag{2}$$

Even when $\boldsymbol{A}$ is a square (n by n) matrix and each $A \in \boldsymbol{A}$ is nonsingular (so that the solution set $\Sigma(\boldsymbol{A}, \boldsymbol{b})$ is bounded), it is known (see, for example [10]) that, in the general case, finding exact bounds $\boldsymbol{x}_i = [\underline{x}_i, \overline{x}_i]$ on the components of the points[2] in $\Sigma(\boldsymbol{A}, \boldsymbol{b})$ is NP-hard. Furthermore, in many cases of interest, such as when the system (1) arises during an interval Newton method, and the underlying nonlinear system has more than one isolated solution with the domain, the solution set $\Sigma(\boldsymbol{A}, \boldsymbol{b})$ contains two or more disjoint unbounded components.

In either case (bounded solution set or unbounded solution set), directly applying an interval version of a solution procedure, e.g. applying interval Gaussian elimination or the interval Gauss–Seidel algorithm directly to (1) in general leads to catastrophic overestimation in the bounds for the components of $\Sigma(\boldsymbol{A}, \boldsymbol{b})$. In such instances, we precondition the system (1), forming[3] the derived system

$$Y\boldsymbol{A}x = Y\boldsymbol{b} \quad \text{i.e. } \tilde{\boldsymbol{A}}x = \tilde{\boldsymbol{b}}, \text{ where } \tilde{\boldsymbol{A}} = YA \text{ and } \tilde{\boldsymbol{b}} = Y\boldsymbol{b}. \tag{3}$$

Although the solution set to (3) in general is not equal to the solution set to (1) (but merely contains it), Y can often be chosen so that the overestimation in the solution process[4] for (3) is not catastrophic, and usable (although not optimal) bounds on the components of $\Sigma(\boldsymbol{A}, \boldsymbol{b})$ can be obtained. (Here, and throughout the paper, we say that computed bounds are *usable* if they do not contain previously known bounds[5]; similarly, we say that a preconditioner is useful if its application results in usable bounds.) This paper deals with designing and choosing preconditioners that give practical results in as wide as possible a range of cases.

The most common preconditioner Y in (3) has been the inverse midpoint matrix, that is $Y = \left(\check{A}\right)^{-1}$, where $\check{A}$ is a floating point approximation to $\mathrm{m}(\boldsymbol{A})$, where $\mathrm{m}(A)$, the "midpoint matrix," is the matrix whose i,j-th entry is $a_{i,j} = (\underline{a}_{i,j} + \overline{a}_{i,j})/2$, where the (i,j)-th entry of $\boldsymbol{A}$ is $\boldsymbol{a}_{i,j} = [\underline{a}_{i,j}, \overline{a}_{i,j}]$. This preconditioner is relatively simple and inexpensive to compute, and is especially effective when the widths in the matrix $\boldsymbol{A}$ are small. However, the

[1]typically, during an interval Newton method

[2]The set $\Sigma(\boldsymbol{A}, \boldsymbol{b})$ is, in general, not a box of the form $\{x \in \mathbb{R}^n : \underline{x}_i \le x_i \le \overline{x}_i, 1 \le i \le n\}$, but is a star-shaped region or an unbounded region; see, for example [13, §3.4].

[3]in theory; in practice, the actual full matrix $Y\boldsymbol{A}$ need not always be formed and stored at once.

[4]usually, interval Gaussian elimination or the interval Gauss–Seidel method

[5]or, in the case no known bounds are available, are not $[-\infty, \infty]$

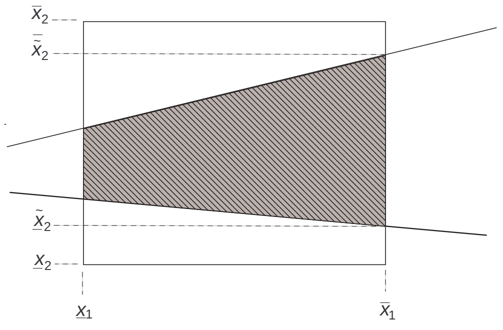

Figure 1: Application of the interval Gauss–Seidel method when the solution set is unbounded.

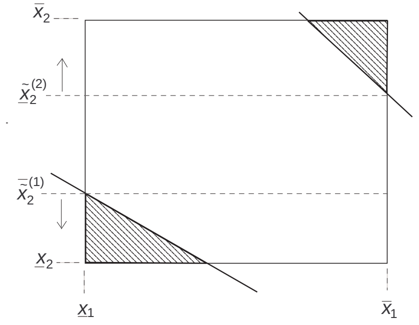

Figure 2: Application of the interval Gauss–Seidel method when the solution set is disconnected.

preconditioner can be ineffective when the entries of $\boldsymbol{A}$ are wide (that is, when the entries of $\boldsymbol{A}$ have much uncertainty). The inverse midpoint preconditioner can be especially ineffective when $\boldsymbol{A}$ contains singular matrices (and $\Sigma(\boldsymbol{A}, \boldsymbol{b})$ is thus unbounded), but we nonetheless desire to compute bounds on some of the coordinates of the solution set that lie within a particular given set of bounds $\boldsymbol{x}$. See Figure 1: there, the solution set lies between the two oblique lines, the original bounds are $\underline{x}_1 \le x_1 \le \overline{x}_1$ and $\underline{x}_2 \le x_2 \le \overline{x}_2$, and it may be possible with the interval Gauss–Seidel method to narrow the bounds on x_2 to $\underline{\tilde{x}}_2 \le x_2 \le \overline{\tilde{x}}_2$.

In contrast, in Figure 2, the solution set is disconnected and unbounded; in this case, the interval Gauss–Seidel method can compute bounds for each component, thus eliminating a portion of the region described by the original bounds. This computation is useful, for example, if an exhaustive search is being used to rigorously bound all solutions to a nonlinear system of equations, the initial bounds represent a portion of that region to (hopefully) be eliminated, and the interval linear system has arisen from an interval Newton method for the nonlinear system of equations.

In previous work ([7] and [8, Ch. 3]), we proposed preconditioners for the interval Gauss–Seidel method for the cases illustrated in Figure 1 and Figure 2. Specifically, if $\tilde{\boldsymbol{A}}$ and $\tilde{\boldsymbol{b}}$ are as in (3), the interval Gauss–Seidel method is given as

$$\begin{array}{lll} \boldsymbol{x}_k & \text{is given} & \text{for } 1 \le k \le n, \\ \tilde{\boldsymbol{x}}_k \leftarrow & \dfrac{1}{\tilde{\boldsymbol{a}}_{k,k}} \left\{ \tilde{\boldsymbol{b}}_k - \sum_{j=1}^{k-1} \tilde{\boldsymbol{a}}_{k,j} \tilde{\boldsymbol{x}}_j - \sum_{j=k+1}^{n} \tilde{\boldsymbol{a}}_{k,j} \boldsymbol{x}_j \right\} & \text{for } k = 1, 2, \ldots, n. \end{array} \tag{4}$$

If $0 \notin \tilde{\boldsymbol{a}}_{k,k}$, then $\tilde{\boldsymbol{x}}_k$ is an interval of the form

$$\tilde{\boldsymbol{x}}_k = [\underline{\tilde{x}}_k, \overline{\tilde{x}}_k], \tag{5}$$

as in Figure 1, whereas, if $0 \in \tilde{\boldsymbol{a}}_{k,k}$ and the numerator in (4) does not contain zero, then $\tilde{\boldsymbol{x}}_k$ will consist of two semi-infinite intervals of the form

$$\tilde{\boldsymbol{x}}_k = \left(-\infty, \overline{\tilde{x}}_k^{(1)}\right] \bigcup \left[\underline{\tilde{x}}_k^{(2)}, \infty\right), \tag{6}$$

as is illustrated in Figure 2.

For the case that $\tilde{\boldsymbol{x}}_k$ is a single connected interval, the preconditioners we described in [8, Ch. 3] are based on devising the k-th row Y_k of the preconditioner to attempt to:

- minimize the width $\mathrm{w}(\tilde{\boldsymbol{x}}_k) = \overline{\tilde{x}}_k - \underline{\tilde{x}}_k$,
- maximize the left end point $\underline{\tilde{x}}_k$, or
- minimize the right end point $\overline{\tilde{x}}_k$

in (5). In [7], we examined in detail "width-optimal" preconditioners (in which we attempt to minimize $\mathrm{w}(\tilde{\boldsymbol{x}}_k)$); in the experiments there, we showed that, for a particular test set, the width-optimal preconditioner resulted in less overall processor time used in a branch and bound algorithm to find all solutions to nonlinear systems of equations, compared to the inverse midpoint preconditioner.

In the case $0 \in \tilde{\boldsymbol{a}}_{k,k}$ (as in Figure 2), the "splitting preconditioners" are based on devising the k-th row Y_k to attempt to:

- maximize the gap $\underline{\tilde{x}}_k^{(2)} - \overline{\tilde{x}}_k^{(1)}$,
- minimize $\overline{\tilde{x}}_k^{(1)}$,
- maximize $\underline{\tilde{x}}_k^{(2)}$, or
- maximize $\min\{|\overline{\tilde{x}}_k^{(1)}|, |\underline{\tilde{x}}_k^{(2)}|\}$.

The last criterion, which we call "mignitude optimality," maximizes the distance from 0 of the image components. This criterion is useful for systems arising from interval Newton methods, in which $\boldsymbol{x}$ is centered about the zero-vector and we may wish to make the volume of $\tilde{\boldsymbol{x}} \cap \boldsymbol{x}$ as small as possible. Furthermore, the corresponding computational preconditioner for mignitude optimality gives a useful preconditioner even if a preconditioner that results in two semi-infinite intervals of the form (6) does not exist; see Lemma 3.7, page 138 of [8].

We refer to preconditioners which lead to two disjoint solution components as "splitting-" or "S-preconditioners." Our early empirical experiments with our "optimal S-preconditioners" gave ambivalent results, but were not complete, and we did not attempt to publish them. However, others (such as [4], in the context of parallel computation) have reported advantages to using "splits" in the interval Gauss–Seidel method[6]. This has prompted us to revisit the issue of optimal splitting preconditioners.

In §2, we review details of our formulations, which are linear programs with complementarity constraints, while in §3, we define the specifics of the extended arithmetic used to separate disconnected components of solutions. In §4, we briefly review or work with illustrative examples designed to test specific aspects of our formulations, while in §5, we propose two heuristically-based algorithms, designed from experience with our illustrative examples. In §6, we give a fairly comprehensive set of experimental results, comparing our two heuristic algorithms to use of the inverse midpoint preconditioner, to the previously studied width-optimal preconditioner, and to no preconditioner. We cite alternate approaches to specialized linear systems in §6.3, and we summarize in §7.

2 Ideas Behind Optimal Preconditioners: Notation, Review and Analysis

Here, we review the derivation in [7] and [8, Ch. 3], placing it in somewhat more general terms.

As in the introduction, we use boldface font for intervals, interval vectors, and interval matrices; we use lower case to denote scalars and vectors and upper case for matrices. Throughout, we formulate computation of the preconditioner in terms of the positive and negative parts of the y_i. In particular,

Definition 1 *(Positive and negative parts) If r is a real number, then*

$$r^+ = \max\{y_i, 0\} \text{ and } r^- = \max\{-r, 0\},$$

so $r = r^+ - r^-$.

In our computations for the optimal preconditioners for (3), we formulate the computation in terms of the positive and negative components of Y. That is,

[6]Splits produce more boxes; however, in a parallel processing context in which each processor is given a box to process, more boxes may not be a disadvantage.

writing the k-th row of the preconditioner Y for (3) as

$$Y_k = (y_1, \ldots, y_m),$$

we rewrite

$$y_i = y_i^+ - y_i^-, \tag{7}$$

for $1 \leq i \leq m$. However, the optimality conditions for our preconditioners lead to an optimization problem with a linear objective, linear constraints, and complementarity constraints $y_i^+ \perp y_i^-$. For some of the preconditioners, we have found it advantageous to include these complementarity conditions as a penalty term in the objective (using a quadratic programming solver). However, if we relax the optimization problem by dropping the complementarity constraints, we obtain a linear program, which can sometimes be solved much more rapidly. When we solve such relaxed problems, we form the preconditioner with

$$y_i = \mathcal{Y}_{i,+} - \mathcal{Y}_{i,-}, \tag{8}$$

where $\mathcal{Y}_{i,+}$ and $\mathcal{Y}_{i,-}$ are components corresponding to y_i^+ and y_i^- of the solution to the (possibly) relaxed problem. Although, in general, $\mathcal{Y}_{i,+} \neq y_i^+$ and $\mathcal{Y}_{i,-} \neq y_i^-$, $\mathcal{Y}_{i,+}$ and $\mathcal{Y}_{i,-}$ often result in good preconditioners; a theoretical basis for this is given in the unpublished work [15], and we give some empirical evidence in the technical report [9].

If we are not computing inverses, we don't need to assume a square system, so we may assume $\boldsymbol{A}$ is a general interval m by n matrix, and $\boldsymbol{b}$ is an interval m vector[7]. We have

Theorem 1 *Assume that no other $\tilde{\boldsymbol{x}}_\ell$ have been computed for $\ell \neq k$ (i.e. that $\tilde{\boldsymbol{x}}_i = \boldsymbol{x}_i$ for $i \neq k$), and assume that the components of the initial guess $\boldsymbol{x}$ are centered at zero, that is, that $\boldsymbol{x}_j = [-x_j, x_j]$, $1 \leq j \leq n$, $j \neq k$. Then the numerator in (4) is*

$$\begin{aligned}
\boldsymbol{\nu}_k &= Y_k\boldsymbol{b} + \sum_{\substack{j=1\\ j\neq k}}^{n} \{Y_k\boldsymbol{A}_{:,j}\}\, \boldsymbol{x}_j \\
&= \sum_{i=1}^{m} y_i \mathrm{m}(\boldsymbol{b}_i) + \frac{1}{2}\sum_{i=1}^{m} |y_i| \mathrm{w}(\boldsymbol{b}_i)[-1,1] \qquad (9)\\
&\qquad + \frac{1}{2}\sum_{\substack{j=1\\ j\neq k}}^{n} \mathrm{w}(\boldsymbol{x}_j)\left|\sum_{i=1}^{m} y_i \boldsymbol{a}_{i,j}\right|[-1,1], \qquad (10)
\end{aligned}$$

and the denominator in (4) is[8]

$$\boldsymbol{d}_k = Y_k\boldsymbol{A}_{:,k} = \sum_{i=1}^{m} y_i\boldsymbol{a}_{i,k} = \sum_{i=1}^{m} y_i \mathrm{m}(\boldsymbol{a}_{i,k}) + \frac{1}{2}\sum_{i=1}^{m} |y_i| \mathrm{w}(a_{i,k})[-1,1]. \tag{11}$$

[7]For certain existence verification arguments, we need $m \leq n$, but this is not even necessary if our interest is merely reducing the size of the set of all possible solutions.

[8]This expression was given incorrectly, omitting the factor of y_i in front of the $\mathrm{m}(\boldsymbol{a}_{i,k})$ in [8, p. 130, (3.19)].

Proof: The proof proceeds from Lemma (3.4) of [8, p. 129], in a manner entirely analogously to Lemma (3.5) of [8]. □

Now we have, as in the development in [8, pp. 130–131],

$$\begin{aligned} \left|\sum_{i=1}^m y_i \boldsymbol{a}_{i,j}\right| &= \left\{\sum_{i=1}^m \big(-y_k^+ \underline{a}_{i,j} + y_k^- \overline{a}_{i,j}\big)\right\} + v_j^+ && (12)\\ &= \left\{\sum_{i=1}^m \big(y_i^+ \overline{a}_{i,j} - y_i^- \underline{a}_{i,j}\big)\right\} + v_j^-, && (13) \end{aligned}$$

where v_j is defined to be equal to

$$v_j = \overline{\sum_{i=1}^m y_i \boldsymbol{a}_{i,j}} + \underline{\sum_{i=1}^m y_i \boldsymbol{a}_{i,j}}, \quad 1 \le j \le n, \quad j \ne k. \tag{14}$$

Thus, if the hypotheses of Theorem 1 hold, then, observing that $|y_i| = y_i^- + y_i^+$ and letting $\delta \in [0,1]$ be a parameter representing an arbitrarily chosen convex combination of (12) and (13), we have

$$\begin{aligned} \boldsymbol{\nu}_k &= \sum_{i=1}^m y_i \mathrm{m}(\boldsymbol{b}_i) + \frac{1}{2}\sum_{i=1}^m (y_i^- + y_i^+)\mathrm{w}(\boldsymbol{b}_i)[-1,1]\\ &\qquad + \frac{1}{2}\sum_{\substack{j=1\\ j\ne k}}^n \mathrm{w}(\boldsymbol{x}_j)\left(\delta\left\{\sum_{i=1}^m \big(-y_i^+ \underline{a}_{i,j} + y_i^- \overline{a}_{i,j}\big) + v_j^+\right\}\right. && (15)\\ &\qquad \left. + (1-\delta)\left\{\sum_{i=1}^m \big(y_i^+ \overline{a}_{i,j} - y_i^- \underline{a}_{i,j}\big) + v_j^-\right\}\right)[-1,1], \end{aligned}$$

and similarly,

$$\boldsymbol{d}_k = \left[\sum_{i=1}^m \big(y_i^+ \underline{a}_{i,k} - y_i^- \overline{a}_{i,k}\big), \sum_{i=1}^m \big(y_i^+ \overline{a}_{i,k} - y_i^- \underline{a}_{i,k}\big)\right]. \tag{16}$$

Now, continuing the derivation on page 133 of [8], the "width-optimal C-preconditioner[9]" of [7] is based on minimizing the width $\mathrm{w}(\tilde{\boldsymbol{x}}_k)$ in (4),

$$\begin{array}{l} \text{find } \min_{Y_k} \mathrm{w}(\boldsymbol{\nu}_k) = \min_{Y_k}\{\overline{\nu}_k - \underline{\nu}_k\}\\ \text{subject to } \underline{d}_k = 1 \end{array} \tag{17}$$

We will now express (17) as an optimization problem with linear objective, linear constraints, and complementarity constraints. In this problem, we let $\mathcal{Y}_{i,+}$ be the variable corresponding to the positive part y_i^+ of y_i, and similarly, we let $\mathcal{Y}_{i,-}$ correspond to y_i^-, $\mathcal{V}_{j,+}$ correspond to v_i^+, and $\mathcal{V}_{j,-}$ correspond to v_i^-. An

[9]We used the designation "C" to denote that the Gauss–Seidel operator "contracts" the box $\boldsymbol{x}$.

optimization problem corresponding to (17) is then

$$\begin{array}{ll}
\text{minimize} & \displaystyle\sum_{i=1}^{m}(\mathcal{Y}_{i,+}+\mathcal{Y}_{i,-})\mathrm{w}(\boldsymbol{b}_i) \\
& \displaystyle + \qquad \delta \sum_{\substack{j=1\\ j\neq k}}^{n} \mathrm{w}(\boldsymbol{x}_j)\left\{\mathcal{V}_{j,+}+\sum_{i=1}^{m}\left(\mathcal{Y}_{i,-}\overline{a}_{i,j}-\mathcal{Y}_{i,+}\underline{a}_{i,j}\right)\right\} \\
& \displaystyle +(1-\delta)\sum_{\substack{j=1\\ j\neq k}}^{n} \mathrm{w}(\boldsymbol{x}_j)\left\{\mathcal{V}_{j,-}+\sum_{i=1}^{m}\left(\mathcal{Y}_{i,+}\overline{a}_{i,j}-\mathcal{Y}_{i,-}\underline{a}_{i,j}\right)\right\} \qquad (18) \\
\text{subject to} & \displaystyle\sum_{i=1}^{m}\left(\mathcal{Y}_{i,+}\underline{a}_{i,k}-\mathcal{Y}_{i,-}\overline{a}_{i,k}\right)=1, \\
& \displaystyle \mathcal{V}_{j,+}-\mathcal{V}_{j,-}-\sum_{i=1}^{m}(\mathcal{Y}_{i,+}-\mathcal{Y}_{i,-})(\underline{a}_{i,j}+\overline{a}_{i,j})=0, \\
& \qquad\qquad\qquad\qquad 1\leq j\leq n, \quad j\neq k, \\
& \mathcal{Y}_{i,+},\mathcal{Y}_{i,-}\geq 0, \quad \mathcal{Y}_{i,+}\perp\mathcal{Y}_{i,-}, \quad 1\leq i\leq m, \\
& \mathcal{V}_{j,+},\mathcal{V}_{j,-}\geq 0, \quad \mathcal{V}_{j,+}\perp\mathcal{V}_{j,-}, \quad 1\leq j\leq n.
\end{array}$$

(cf. page 132 of [8]). Rearranging the objective (15) and the equality constraints, we have the formulation as in Table 1.

Observe now

1. The formulation (18) is equivalent to (17).
2. The formulation (18) is a nonlinear problem (and thus may not be easy to solve), due to the complementarity constraints $\mathcal{Y}_{i,+}\perp\mathcal{Y}_{i,-}$ and $\mathcal{V}_{j,+}\perp\mathcal{V}_{j,-}$.
3. The formulation leads to a linear program if we relax the problem by removing the complementarity constraints; however, then, the solutions for $\mathcal{Y}_{i,+}$ and $\mathcal{Y}_{i,-}$ may not necessarily correspond to the positive and negative parts of y_i, and the linear program may be unbounded. Nonetheless, the preconditioners representing solutions to these relaxed problems are in many cases just as good. (See the unpublished analysis in [15], and the experiments in [9]. In the previous computations for the width-optimal preconditioner in [7], we ignored the complementarity constraints as a heuristic to simplify the problem.)

Summarizing, we have

Definition 2 *A* traditional width-optimal LP *is a linear program consisting of the objective, equality constraints, and nonnegativity constraints in (18) (for an arbitrarily selected* $\delta\in[0,1]$*), but without the complementarity constraints. The linear program thus is as in Table 1, but without the complementarity constraints.*

Table 1: Optimization problem for an optimal width contracting preconditioner

$$\begin{array}{ll}
\text{minimize} & \displaystyle\sum_{i=1}^{m} \mathcal{Y}_{i,+} \left\{ \mathrm{w}(\boldsymbol{b}_i) + \sum_{\substack{j=1\\ j\neq k}}^{n} \mathrm{w}(\boldsymbol{x}_j)\left[(1-\delta)\overline{a}_{i,j} - \delta\underline{a}_{i,j}\right] \right\} \\
& \displaystyle + \sum_{i=1}^{m} \mathcal{Y}_{i,-} \left\{ \mathrm{w}(\boldsymbol{b}_i) + \sum_{\substack{j=1\\ j\neq k}}^{n} \mathrm{w}(\boldsymbol{x}_j)\left[\delta\overline{a}_{i,j} - (1-\delta)\underline{a}_{i,j}\right] \right\} \\
& \displaystyle + \sum_{\substack{j=1\\ j\neq k}}^{n} \mathcal{V}_{j,+}\left\{\delta \mathrm{w}(\boldsymbol{x}_j)\right\} + \sum_{\substack{j=1\\ j\neq k}}^{n} \mathcal{V}_{j,-}\left\{(1-\delta)\mathrm{w}(\boldsymbol{x}_j)\right\} \\
\text{subject to:} & \displaystyle\sum_{i=1}^{m} \mathcal{Y}_{i,+}\left\{\underline{a}_{i,k}\right\} + \sum_{i=1}^{m} \mathcal{Y}_{i,-}\left\{-\overline{a}_{i,k}\right\} = 1, \\
& \displaystyle\sum_{i=1}^{m} \mathcal{Y}_{i,+}\left\{-(\underline{a}_{i,j} + \overline{a}_{i,j})\right\} + \sum_{i=1}^{m} \mathcal{Y}_{i,-}\left\{\underline{a}_{i,j} + \overline{a}_{i,j}\right\} \\
& \qquad\qquad + \mathcal{V}_{j,+} - \mathcal{V}_{j,-} = 0, \quad 1 \le j \le n, \quad j \neq k, \\
\text{and} & \mathcal{Y}_{i,+} \ge 0, \quad \mathcal{Y}_{i,-} \ge 0, \quad \mathcal{Y}_{i,+} \perp \mathcal{Y}_{i,-}, \quad 1 \le i \le m, \\
& \mathcal{V}_{j,+} \ge 0, \quad \mathcal{V}_{j,-} \ge 0, \quad \mathcal{V}_{j,+} \perp \mathcal{V}_{j,-}, \quad 1 \le j \le n, \quad j \neq k.
\end{array}$$

In contrast, for a splitting preconditioner, we assume that the denominator $\boldsymbol{d}_k$ in (4) (and characterized in (11)) contains zero, and we normalize the numerator $\boldsymbol{\nu}_k$ (as characterized in (10)). In particular, the "mignitude-optimal" preconditioner is based on the optimization problem

$$\begin{array}{l} \text{find } \min_{Y_k} |\boldsymbol{d}_k| \\ \text{subject to } \underline{\nu}_k = 1. \end{array} \tag{19}$$

Using a derivation, as in [8] and similar to that above for the width optimal C-preconditioner[10], we obtain an optimization problem as in Table 2.

Definition 3 *A* mignitude-optimal optimization problem *is a problem (for a particular $\delta \in [0,1]$) as in Table 2 (obtained from (19) with Theorem 1 and formulas (15) and (16)). If we drop the complementarity conditions in Table 2, we call the corresponding linear program a* mignitude optimal LP.

Among splitting preconditioners, we have chosen the mignitude optimal preconditioner for further study for the following reason.

Theorem 2 *(Lemma 3.7, page 138 of [8], and originally observed in unpublished work by Manuel Novoa, such as [15]) Suppose a preconditioner row Y_k solves the optimization problem (19). Then the following are true.*

[10]in particular, using (3.28) on page 132 and (3.36) on page 135 of [8]

Table 2: Optimization problem for a mignitude optimal preconditioner

$$
\begin{aligned}
\text{minimize} \quad & \sum_{i=1}^{m} \mathcal{Y}_{i,+}\left\{-\delta\underline{a}_{i,k} + (1-\delta)\overline{a}_{i,k}\right\} \\
& + \sum_{i=1}^{m} \mathcal{Y}_{i,-}\left\{\delta\overline{a}_{i,k} - (1-\delta)\underline{a}_{i,k}\right\} + v_k^{+}\{\delta\} + v_k^{-}\{1-\delta\} \\
\text{subject to:} \quad & \sum_{i=1}^{m} \mathcal{Y}_{i,+}\Bigg\{ \mathrm{m}(\boldsymbol{b}_i) - \frac{1}{2}\mathrm{w}(\boldsymbol{b}_i) \\
& \qquad + \sum_{\substack{j=1\\ j\neq k}}^{n} \mathrm{w}(\boldsymbol{x}_j)\left(\frac{\delta}{2}\underline{a}_{i,j} - \frac{1-\delta}{2}\overline{a}_{i,j}\right)\Bigg\} \\
& + \sum_{i=1}^{m} \mathcal{Y}_{i,-}\Bigg\{ -\mathrm{m}(\boldsymbol{b}_i) - \frac{1}{2}\mathrm{w}(\boldsymbol{b}_i) \\
& \qquad + \sum_{\substack{j=1\\ j\neq k}}^{n} \mathrm{w}(\boldsymbol{x}_j)\left(-\frac{\delta}{2}\overline{a}_{i,j} + \frac{1-\delta}{2}\underline{a}_{i,j}\right)\Bigg\} \\
& + \sum_{\substack{j=1\\ j\neq k}}^{n} \mathcal{V}_{j,+}\left\{-\frac{\delta}{2}\mathrm{w}(\boldsymbol{x}_j)\right\} \\
& + \sum_{\substack{j=1\\ j\neq k}}^{n} \mathcal{V}_{j,-}\left\{-\frac{1-\delta}{2}\mathrm{w}(\boldsymbol{x}_j)\right\} \qquad = 1, \\
\text{and} \quad & \sum_{i=1}^{m} \mathcal{Y}_{i,+}\left\{-(\underline{a}_{i,j} + \overline{a}_{i,j})\right\} + \sum_{i=1}^{m} \mathcal{Y}_{i,-}\left\{\underline{a}_{i,j} + \overline{a}_{i,j}\right\} \\
& \qquad + \mathcal{V}_{j,+} - \mathcal{V}_{j,-} = 0, \quad 1 \le j \le n, \\
\text{and} \quad & \mathcal{Y}_{i,+} \ge 0, \quad \mathcal{Y}_{i,-} \ge 0, \quad \mathcal{Y}_{i,+} \perp \mathcal{Y}_{i,-}, \quad 1 \le i \le m, \\
& \mathcal{V}_{j,+} \ge 0, \quad \mathcal{V}_{j,-} \ge 0, \quad \mathcal{V}_{j,+} \perp \mathcal{V}_{j,-}, \quad 1 \le j \le n.
\end{aligned}
$$

1. *If* $0 \in \boldsymbol{d}_k$*, then* $\tilde{\boldsymbol{x}}_k$ *consists of two semi-infinite intervals, and* Y_k *solves*
$$\max_{\underline{\nu}_k=1} \min \left\{ -\frac{1}{\underline{d}_k}, \frac{1}{\overline{d}_k} \right\},$$
and hence maximizes the minimum distance of $\tilde{\overline{x}}_k^{(1)}$ *and* $\tilde{\underline{x}}_k^{(2)}$ *from* 0 *in (6).*
2. *If* $\boldsymbol{d}_k > 0$*, then* $\tilde{\boldsymbol{x}}_k$ *is a single connected interval as in (5), and* Y_k *maximizes the left end point* $\tilde{\underline{x}}_k$*.*
3. *If* $\boldsymbol{d}_k < 0$*, then* $\tilde{\boldsymbol{x}}_k$ *is a single connected interval as in (5), and* Y_k *minimizes the right end point* $\tilde{\overline{x}}_k$*.*

Thus, if $\boldsymbol{x}_k$ is symmetric about 0, then the solution to the optimization problem for the mignitude optimal preconditioner maximizes, in a sense, the possibility that $\boldsymbol{x}_k \cap \tilde{\boldsymbol{x}}_k$ will contain half or less of the original interval.

In addition to the width-optimal preconditioner and the mignitude-optimal preconditioner, we have formulated, in a way similar to the width-optimal and mignitude-optimal preconditioners, two preconditioners that force the denominator to contain zero (and thus force two disjoint intervals if the numerator does not contain zero). We have formulated a pair of such preconditioners because, for different problems, we expect one of these to be infeasible.

The positive numerator S-preconditioner:

$$\begin{array}{l} \text{find } \max_{Y_k} \underline{\nu}_k \\ \text{subject to } \underline{d}_k \le -1 \text{ and } \overline{d}_k = 1. \end{array} \tag{20}$$

The negative numerator S-preconditioner:

$$\begin{array}{l} \text{find } \min_{Y_k} \overline{\nu}_k \\ \text{subject to } \underline{d}_k = -1 \text{ and } \overline{d}_k \ge 1. \end{array} \tag{21}$$

A formulation for the positive numerator S-preconditioner appears in Table 3, while a formulation for the negative numerator S-preconditioner appears in Table 4.

3 On Extended Arithmetic

In computing the quotient (4), the mignitude-optimal preconditioner and the S-preconditioners lead to denominators that contain 0. Since there have been various and conflicting operational definitions of extended interval arithmetic (when zero is in the denominator of a quotient) in the past, we clarify here what is appropriate in our context.

Our guiding principle in our use of extended intervals is *cset theory* as explained, for example, in [18]. In particular, we want to make certain that no part of the solution set to $\boldsymbol{A}x - \boldsymbol{b}$ contained in the initial bounds $\boldsymbol{x}$ is discarded,

Table 3: Optimization problem for the positive numerator S-preconditioner

$$
\begin{array}{rl}
\text{minimize} & \displaystyle\sum_{i=1}^{m} \mathcal{Y}_{i,+}\left\{-\mathrm{m}(\boldsymbol{b}_i)+\frac{1}{2}\mathrm{w}(\boldsymbol{b}_i)+\sum_{\substack{j=1\\ j\neq k}}^{n}\mathrm{w}(\boldsymbol{x}_j)\left(-\frac{1-\delta}{2}\overline{a}_{i,j}+\frac{\delta}{2}\underline{a}_{i,j}\right)\right\} \\
& \displaystyle +\sum_{i=1}^{m} \mathcal{Y}_{i,-}\left\{\mathrm{m}(\boldsymbol{b}_i)+\frac{1}{2}\mathrm{w}(\boldsymbol{b}_i)+\sum_{\substack{j=1\\ j\neq k}}^{n}\mathrm{w}(\boldsymbol{x}_j)\left(-\frac{\delta}{2}\overline{a}_{i,j}+\frac{1-\delta}{2}\underline{a}_{i,j}\right)\right\} \\
& \displaystyle +\sum_{\substack{j=1\\ j\neq k}}^{n} \mathcal{V}_{j,+}\left\{-\frac{\delta}{2}\mathrm{w}(\boldsymbol{x}_j)\right\}+\sum_{\substack{j=1\\ j\neq k}}^{n} \mathcal{V}_{j,-}\left\{-\frac{1-\delta}{2}\mathrm{w}(\boldsymbol{x}_j)\right\} \\
\text{subject to:} & \displaystyle\sum_{i=1}^{m} \mathcal{Y}_{i,+}\left\{\underline{a}_{i,k}\right\}+\sum_{i=1}^{m} \mathcal{Y}_{i,-}\left\{-\overline{a}_{i,k}\right\}\leq -1, \\
& \displaystyle\sum_{i=1}^{m} \mathcal{Y}_{i,+}\left\{\overline{a}_{i,k}\right\}+\sum_{i=1}^{m} \mathcal{Y}_{i,-}\left\{-\underline{a}_{i,k}\right\}= 1, \\
& \displaystyle\sum_{i=1}^{m} \mathcal{Y}_{i,+}\left\{-(\underline{a}_{i,j}+\overline{a}_{i,j})\right\}+\sum_{i=1}^{m} \mathcal{Y}_{i,-}\left\{\underline{a}_{i,j}+\overline{a}_{i,j}\right\} \\
& \qquad\qquad +\mathcal{V}_{j,+}-\mathcal{V}_{j,-}=0, \quad 1\leq j\leq n, \quad j\neq k, \\
\text{and} & \mathcal{Y}_{i,+}\geq 0, \quad \mathcal{Y}_{i,-}\geq 0, \quad \mathcal{Y}_{i,+}\perp\mathcal{Y}_{i,-}, \quad 1\leq i\leq m, \\
& \mathcal{V}_{j,+}\geq 0, \quad \mathcal{V}_{j,-}\geq 0, \quad \mathcal{V}_{j,+}\perp\mathcal{V}_{j,-}, \quad 1\leq j\leq n, \quad j\neq k.
\end{array}
$$

but we otherwise want the result to be as small a set as possible. This leads to the rules in Table 5.

One property of the cset arithmetic we are using is mathematically elegant but not present in other implementations of extended arithmetic; that is the fact that division of an interval that does not contain zero by any interval that contains zero always results in two disjoint sets. This follows by considering ∞ to be a number, as part of the two-point compactification of the real numbers. Thus, $1/[-\epsilon,\epsilon]=[-\infty,-1/\epsilon]\cup[1/\epsilon,\infty]$, while $1/[0,\epsilon]=[-\infty,-\infty]\cup[1/\epsilon,\infty]$, $1/[0,0]=[-\infty,-\infty]\cup[\infty,\infty]$, etc. For the mathematical underpinning to this, see [18].

4 Experimental Setup and Examples

In this section, we present results from illustrative examples that have been specifically designed, both to test correctness of our implementation and to provide initial evidence of the effectiveness of the underlying ideas. In these “toy” problems, exact bounds on the solution (to within roundout error) can be computed easily enough by solving 2^n linear programs (using the technique explained in [19] and earlier in [16], based on inequalities presented in [17]),

Table 4: Optimization problem for the negative numerator S-preconditioner

$$\begin{aligned}
\text{minimize} \quad & \sum_{i=1}^{m} \mathcal{Y}_{i,+} \left\{ \mathrm{m}(\boldsymbol{b}_i) + \frac{1}{2}\mathrm{w}(\boldsymbol{b}_i) + \sum_{\substack{j=1\\ j\neq k}}^{n} \mathrm{w}(\boldsymbol{x}_j) \left(\frac{1-\delta}{2}\overline{a}_{i,j} - \frac{\delta}{2}\underline{a}_{i,j} \right) \right\} \\
& + \sum_{i=1}^{m} \mathcal{Y}_{i,-} \left\{ -\mathrm{m}(\boldsymbol{b}_i) + \frac{1}{2}\mathrm{w}(\boldsymbol{b}_i) + \sum_{\substack{j=1\\ j\neq k}}^{n} \mathrm{w}(\boldsymbol{x}_j) \left(\frac{\delta}{2}\overline{a}_{i,j} - \frac{1-\delta}{2}\underline{a}_{i,j} \right) \right\} \\
& + \sum_{\substack{j=1\\ j\neq k}}^{n} \mathcal{V}_{j,+} \left\{ \frac{\delta}{2}\mathrm{w}(\boldsymbol{x}_j) \right\} + \sum_{\substack{j=1\\ j\neq k}}^{n} \mathcal{V}_{j,-} \left\{ \frac{1-\delta}{2}\mathrm{w}(\boldsymbol{x}_j) \right\} \\
\text{subject to:} \quad & \sum_{i=1}^{m} \mathcal{Y}_{i,+} \left\{ -\overline{a}_{i,k} \right\} + \sum_{i=1}^{m} \mathcal{Y}_{i,-} \left\{ \underline{a}_{i,k} \right\} \le -1, \\
& \sum_{i=1}^{m} \mathcal{Y}_{i,+} \left\{ \underline{a}_{i,k} \right\} + \sum_{i=1}^{m} \mathcal{Y}_{i,-} \left\{ -\overline{a}_{i,k} \right\} = -1, \\
& \sum_{i=1}^{m} \mathcal{Y}_{i,+} \left\{ -(\underline{a}_{i,j} + \overline{a}_{i,j}) \right\} + \sum_{i=1}^{m} \mathcal{Y}_{i,-} \left\{ \underline{a}_{i,j} + \overline{a}_{i,j} \right\} \\
& \qquad + \mathcal{V}_{j,+} - \mathcal{V}_{j,-} = 0, \quad 1 \le j \le n, \quad j \neq k, \\
\text{and} \quad & \mathcal{Y}_{i,+} \ge 0, \quad \mathcal{Y}_{i,-} \ge 0, \quad \mathcal{Y}_{i,+} \perp \mathcal{Y}_{i,-}, \quad 1 \le i \le m, \\
& \mathcal{V}_{j,+} \ge 0, \quad \mathcal{V}_{j,-} \ge 0, \quad \mathcal{V}_{j,+} \perp \mathcal{V}_{j,-}, \quad 1 \le j \le n, \quad j \neq k.
\end{aligned}$$

so the quality of the bounds obtained with the various preconditioners can be easily studied.

In these studies, we used `MATLAB` version 7 with `INTLAB` ([20]) version 5.2 to do the interval arithmetic. In all of these examples, we systematically tried the optimal width contracting preconditioner, the mignitude optimal preconditioner, the negative numerator S-preconditioner, and the positive numerator S-preconditioner. In various experiments, the details of which appear in our technical report [9], we tried ignoring the complementarity conditions in the formulations of the above four preconditioners, using `linprog` from the `MATLAB` optimization toolbox; we contrasted the results with including the complementarity conditions by explicitly introducing a penalty term in the objective, then using `quadprog` from the `MATLAB` optimization. That is, we converted the linear programs with complementarity constraints given in Tables 1 through 4 to quadratic programs by adding the penalty term

$$Q = \rho \left\{ \sum_{i=1}^{m} y_i^+ y_i^- + \sum_{j} v_j^+ v_j^- \right\} \tag{22}$$

to the linear objectives, for a suitably large but fixed penalty parameter ρ, then used the `MATLAB` optimization toolbox routine `quadprog` to solve the resulting

Table 5: Rules for extended interval division based on cset theory

$$\frac{[\underline{\nu}_k,\overline{\nu}_k]}{[\underline{d}_k,\overline{d}_k]} = \begin{cases} [\underline{\nu}_k,\overline{\nu}_k]/[\underline{d}_k,\overline{d}_k] & \text{in the usual sense if } 0 \notin [\underline{d}_k,\overline{d}_k], \\ [-\infty,\infty] & \text{if } 0 \in [\underline{\nu}_k,\overline{\nu}_k] \text{ and } 0 \in [\underline{d}_k,\overline{d}_k], \\ [-\infty,t_1]\cup[t_2,\infty] & \begin{cases} \text{if } \overline{\nu}_k < 0, \text{ where:} \\ t_1 = \begin{cases} -\infty & \text{if } \overline{d}_k = 0, \\ \overline{\nu}_k/\overline{d}_k & \text{otherwise} \end{cases} \\ \text{and} \\ t_2 = \begin{cases} \infty & \text{if } \underline{d}_k = 0, \\ \overline{\nu}_k/\underline{d}_k & \text{otherwise} \end{cases} \end{cases} \\ [-\infty,t_1]\cup[t_2,\infty] & \begin{cases} \text{if } \overline{\nu}_k \geq 0, \text{ where:} \\ t_1 = \begin{cases} -\infty & \text{if } \underline{d}_k = 0, \\ \underline{\nu}_k/\underline{d}_k & \text{otherwise} \end{cases} \\ \text{and} \\ t_2 = \begin{cases} \infty & \text{if } \overline{d}_k = 0, \\ \underline{\nu}_k/\overline{d}_k & \text{otherwise.} \end{cases} \end{cases} \end{cases}$$

quadratic program.

For each of these eight possibilities (the four preconditioners, with and without the complementarity constraints), we tried the 11 equally spaced values of δ: $\delta = 0$, $\delta = .1$, $\delta = .2$, ..., $\delta = 1$.

We provide detailed descriptions of the results in the technical report [9]. The seven examples detailed in that report consist of

- a simple non-singular point system (ideal for the width optimal and possibly mignitude optimal, but not the S),
- An interval matrix and right-hand-side vector from Brown's almost linear function (a non-regular problem),
- Four simple singular problems, with properties such that different preconditioners worked best with the different problems.

Although coordinate widths can be reduced through multiple sweeps of the Gauss–Seidel method[11], to focus on the details of the mechanisms associated with particular preconditioners, we studied application of the various preconditioners to reduce the width of $\boldsymbol{x}_1$. However, to underscore the fact that preconditioners are necessary, we also did two complete sweeps of the interval Gauss–Seidel method, solving for each variable in each equation, but without

[11]That is, the width of $\boldsymbol{x}_1$, for example, can possibly be reduced by first applying a Gauss–Seidel step to one or more of the other coordinates, then using those reduced coordinate widths in recomputing $\boldsymbol{x}_1$.

preconditioning, and observed whether or not that procedure resulted in a reduction in the coordinate widths; that is, we applied the following algorithm.

Algorithm 1 *(Run through all equations and all variables twice, without preconditioning.)*
INPUT: the m by n interval matrix $\boldsymbol{A}$, the interval m-vector $\boldsymbol{b}$, and the initial bounds $\boldsymbol{x}$.
OUTPUT: new bounds $\tilde{\boldsymbol{x}}^{(1)}$ and $\tilde{\boldsymbol{x}}^{(2)}$ on the coordinates.

```
1. $\tilde{\boldsymbol{x}}^{(1)} \leftarrow \boldsymbol{x}$; $\tilde{\boldsymbol{x}}_i^{(2)} \leftarrow \emptyset$, $1 \le i \le n$. (This value for $\tilde{\boldsymbol{x}}^{(2)}$ is for initialization purposes
   only; $\tilde{\boldsymbol{x}}^{(2)}$ is subsequently possibly set to a finite value later by intersection of
   two semi-infinite intervals with $\tilde{\boldsymbol{x}}^{(1)}$ in step 2b below. Note that intersection of
   this initial $\boldsymbol{x}^{(2)}$ always yields the empty set.)
2. DO for $s = 1$ to 2:
       DO for $j = 1$ to $m$:
           DO for $i = 1$ to $n$:
           (a) Compute new bounds on $x_i$ by solving for $x_i$ in the $j$-the equation
               of $Ax = b$.
           (b) The new bounds may have one or two components $\boldsymbol{z}_i^{(1)}$ and $\boldsymbol{z}_i^{(2)}$.
               Intersect $\boldsymbol{z}_i^{(1)} \cup \boldsymbol{z}_i^{(2)}$ with $\tilde{\boldsymbol{x}}_i^{(1)} \cup \tilde{\boldsymbol{x}}_i^{(2)}$, and store the resulting inter-
               vals[12] back into $\tilde{\boldsymbol{x}}_i^{(1)}$ and $\tilde{\boldsymbol{x}}_i^{(2)}$.
           (c) IF the result from the previous step is the empty set THEN RE-
               TURN .
```

In our tables of numerical experiments below, we will refer to Algorithm 1 as "pivoting," since, using our terminology in [6], Algorithm 1 amounts to applying all possible "pivoting preconditioners."

Although Algorithm 1 does not use heuristics for efficiency as in [4] or even [5] or [6], it represents the best that can be achieved without preconditioning other than by permutations of the identity matrix. For comparison purposes, we just observed the first coordinate of $\tilde{\boldsymbol{x}}^{(1)}$ and $\tilde{\boldsymbol{x}}^{(2)}$ returned from Algorithm 1.

4.1 Some Preliminary Results

Comparing the use of Algorithm 1 to the various preconditioners on the seven examples from [9], we find the following.

- For two of the examples, both running through all coordinates without preconditioner and use of preconditioners gave optimal results, in the sense that the bounds were sharp bounds on the actual solution set to $Ax = b$.

[12]If more than two intervals are produced in this process of intersecting the two intervals $\boldsymbol{z}_i^{(1)}$ and $\boldsymbol{z}_i^{(2)}$ with two intervals $\tilde{\boldsymbol{x}}_i^{(1)}$ and $\tilde{\boldsymbol{x}}_i^{(2)}$, then an error is signalled, $\tilde{\boldsymbol{x}}_i^{(1)}$ and $\tilde{\boldsymbol{x}}_i^{(2)}$ are set to $\boldsymbol{x}_i$, and the routine returns. In principle, this can happen if, during one iteration, $\boldsymbol{x}^{(2)}$ is set to something other than $\emptyset$ because of the intersection of two semi-infinite intervals with $\boldsymbol{x}^{(1)}$, then, on the next step, intersection of these two finite intervals with two semi-infinite intervals from an extended division could produce three finite intervals. However, production of more than two intervals in this way has never been observed to happen.

- For three of the problems, use of no preconditioners gave narrower bounds than those input, but not as narrow as the use of optimal preconditioners.
- For one of the problems, use of optimal preconditioners gave optimally narrow bounds, but use of no preconditioner gave no improvement.
- For one of the problems, no improvement was achieved for any of the preconditioners, as for no preconditioner.

The examples in [9] illustrate that the various preconditioners are complementary, with one working where the others do not. Along these lines, the mignitude optimal preconditioner appears the most versatile, but its performance depends more on δ than the other preconditioners. There is qualitatively no difference between the positive numerator S and negative numerator S preconditioners, since one can be obtained from the other by geometrically reflecting $\boldsymbol{x}_k$ about 0, but a difference in performance, necessarily due to selection of the examples, was observed in [9]. In the tables, we see little difference in effectiveness of the resulting preconditioners between ignoring the complementarity constraints and not ignoring them; however, we have observed slightly more predictability, more normal terminations in `quadprog` versus `linprog` (and presumably if other linear programming or linear programming with complementarity constraints solvers were used) and optima that are better scaled, when the quadratic penalty function is used. Thus, if the computational cost is similar, it is probably better not to ignore the complementarity constraints.

We make additional, more scientific comparisons to no preconditioner and to the inverse midpoint preconditioner in §6 below.

5 Possible Procedures for Use of These Preconditioners

There are various ways that use of these preconditioners can be combined to result in algorithms that may be more effective, and yet do not suffer from exponential complexity when n is large, provided polynomial-time solvers are used to find the preconditioners (such as when the complementarity conditions are dropped and polynomial time solvers are used for the resulting linear programs). In particular, the following procedure will result in an optimal reduction of $\boldsymbol{x}$ in each of the examples in [9] except for Example 5, but the number of linear or quadratic programs to be solved for a particular coordinate is constant with respect to m and n. The choice of δ in each of the steps is based on an analysis of the dependence of the successful computation of particular preconditioners we observed in the seven examples in [9].

Algorithm 2 *(Gives optimal width reductions for each of the examples in [9] except Example 5).*
INPUT: the m by n interval matrix $\boldsymbol{A}$, the interval m-vector $\boldsymbol{b}$, the initial bounds $\boldsymbol{x}$, a subdivision number L for δ, and the coordinate index k to be reduced.
OUTPUT: new bounds $\tilde{\boldsymbol{x}}_k$ on the k-th coordinate.

```
1. $\tilde{\boldsymbol{x}}_k \leftarrow \boldsymbol{x}_k$.
2. Compute $\tilde{\boldsymbol{x}}_k^{(w)}$ using a Gauss–Seidel step with a width-optimal preconditioner
   and a random $\delta \in [0,1]$.
   (a) $\tilde{\boldsymbol{x}}_k \leftarrow \tilde{\boldsymbol{x}}_k \cap \tilde{\boldsymbol{x}}_k^{(w)}$.
   (b) IF $\tilde{\boldsymbol{x}}_k = \emptyset$ THEN RETURN
3. Compute $\tilde{\boldsymbol{x}}_k^{(ns)}$ using the negative numerator S-preconditioner[13] and a random
   $\delta \in [0,1]$.
   (a) $\tilde{\boldsymbol{x}}_k \leftarrow \tilde{\boldsymbol{x}}_k \cap \tilde{\boldsymbol{x}}_k^{(ns)}$.
   (b) IF $\tilde{\boldsymbol{x}}_k = \emptyset$ THEN RETURN
4. Repeat step 3, but with the positive numerator S-preconditioner instead of the
   negative numerator S-preconditioner.
5. DO for $i = 0$ to $L$.
     • Compute $\tilde{\boldsymbol{x}}_k^{(mig,i)}$ using the mignitude-optimal preconditioner with $\delta = ih$.
       (a) $\tilde{\boldsymbol{x}}_k \leftarrow \tilde{\boldsymbol{x}}_k \cap \tilde{\boldsymbol{x}}_k^{(mig,i)}$.
       (b) IF $\tilde{\boldsymbol{x}}_k = \emptyset$ THEN RETURN
   END DO
END ALGORITHM 2
```

The following algorithm will give a successful, but perhaps not optimal, reduction for each of the examples in [9] except for Example 5, but will often complete more quickly than Algorithm 2.

Algorithm 3 *(Gives successful width reductions for each of the examples except Example 5).*
INPUT: the m by n interval matrix $\boldsymbol{A}$, the interval m-vector $\boldsymbol{b}$, the initial bounds $\boldsymbol{x}$, a subdivision number L for δ, and the coordinate index k to be reduced.
OUTPUT: new bounds $\tilde{\boldsymbol{x}}_k$ on the k-th coordinate.
Algorithm 3 is the same as Algorithm 2, except that each test

$$\text{IF } \tilde{\boldsymbol{x}}_k = \emptyset$$

is replaced by

$$\text{IF } \tilde{\boldsymbol{x}}_k \neq \boldsymbol{x}_k.$$

END ALGORITHM 3

Although Algorithm 3 is similar to Algorithm 2 at this level of explanation, Algorithm 3 is somewhat simpler to implement, since, in principle, two or more intervals can be produced from each stage of Algorithm 2, so that the returned value $\tilde{\boldsymbol{x}}_k$ in general would consist of a list of more than two intervals. However, because of the nature of the semi-infinite intervals produced with the extended arithmetic, we think this is unlikely, if not provably impossible.

In the experiments reported here, we used $L = 10$ for both Algorithm 2 and Algorithm 3.

[13] $\tilde{\boldsymbol{x}}_k^{(ns)}$ possibly consists of two disjoint semi-infinite intervals.

6 General Experiments

These general experiments include both randomly generated experiments and experiments on selected matrices from the "Matrix Market" [3].

6.1 Experiments with Random Matrices

For an objective assessment, we designed and carried out some experiments involving random matrices. In particular, we fixed m, n, B, and R, and generated random m by n interval matrices $\boldsymbol{A}$, random right-hand-side vectors $\boldsymbol{b}$, and random initial bounding boxes $\boldsymbol{x}$ as follows:

1. Each entry of $\boldsymbol{A}$ is of the form
$$[a_{i,j} - \beta_{i,j}, a_{i,j} + \beta_{i,j}],$$
where $a_{i,j}$ is pseudo-uniformly distributed in the interval $[-1, 1]$ and $\beta_{i,j}$ is pseudo-uniformly distributed in the interval $[0, B]$.

2. Each entry of $\boldsymbol{b}$ is of the form
$$[\omega_i + b_i - \gamma_i, \omega_i + b_i + \gamma_i],$$
where, as with the entries of $\boldsymbol{A}$, b_i is pseudo-uniformly distributed in the interval $[-1, 1]$, and γ_i is pseudo-uniformly distributed in the interval $[0, B]$, and where the offset ω_i is pseudo-uniformly distributed in the interval $[0, \Omega]$.

3. Each entry of $\boldsymbol{x}$ is of the form $[-r_i, r_i]$, where r_i is pseudo-uniformly distributed in the interval $[0, R]$.

We used the function `rand` from Matlab to generate the pseudo-uniform distribution. With this scheme, we generated sets of problems for analysis; for each such set of problems, we saved the initial state of `rand`, to be able to check and reproduce the results. For each problem in a set, we attempted to use the Gauss–Seidel method to produce narrower bounds[14] on $\boldsymbol{x}_1$. For each set of problems, we gathered statistics for each of the following five schemes:

1. the inverse midpoint preconditioner only;

2. the width-optimal preconditioner only;

3. algorithms 1, 2, and 3.

Based on our observations in [9], we ignored the complementarity constraints for the width-optimal preconditioner, but used the quadratic formulation for each of the other preconditioners[15]. The experimental variables and statistics we gathered for each set of problems, for each of these schemes, are as follows:

[14]Even though we only examine progress with respect to $\boldsymbol{x}_1$ in the statistics, we update all coordinates. For instance, we run through two complete sweeps of Algorithm 1.

[15]both when the width-optimal is used alone and within Algorithms 2 and 3

Table 6: Experiments with random problems, $\Omega = 0$.

m	n	N_t	B	R	Ω	type	N_w	N_s	N_M	$\rho_{\div}$	T
3	3	100	1	1	0	Inv. mid.:	5	2	79	0.974	0.45
						Opt. wid.:	11	0	85	0.959	3.22
						Alg. 2:	22	2	100	0.871	76.48
						Alg. 3:	22	1	92	0.888	55.97
						pivoting:	6	0	82	0.981	9.04
10	10	100	1	1	0	Inv. mid.:	0	0	100	1.000	0.99
						Opt. wid.:	0	0	100	1.000	22.61
						Alg. 2:	0	0	100	1.000	409.63
						Alg. 3:	0	0	100	1.000	406.46
						pivoting:	0	0	100	1.000	200.52
10	10	100	0.1	1	0	Inv. mid.:	1	0	86	0.999	1.05
						Opt. wid.:	11	0	93	0.961	23.66
						Alg. 2:	14	0	100	0.943	492.11
						Alg. 3:	14	0	93	0.956	436.62
						pivoting:	0	0	86	1.000	211.33
10	10	100	0.1	100	0	Inv. mid.:	0	0	99	1.000	1.06
						Opt. wid.:	1	0	100	0.998	24.07
						Alg. 2:	1	0	100	0.998	522.66
						Alg. 3:	1	0	100	0.998	515.92
						pivoting:	0	0	99	1.000	213.75
50	50	10	0.01	10	1	Inv. mid.:	0	0	10	1.000	0.42
						Opt. wid.:	0	0	10	1.000	54.46
						Alg. 2:	0	0	10	1.000	4641.01
						Alg. 3:	0	0	10	1.000	4650.73
						pivoting:	0	0	10	1.000	2067.97

N_t**:** the total number of problems in the set;

N_w**:** the number of problems for which the scheme reduced the measure of the image intersection $\tilde{\boldsymbol{x}}_1 \cap \boldsymbol{x}_1$.

N_s**:** the number of problems for which the scheme returned a disconnected set of two intervals.

N_M**:** The number of problems for which the scheme resulted in the maximum reduction of radius over all possible schemes.

$\rho_{\div}$**:** The average, over all problems in the set, of the ratio of the radius of $\boldsymbol{x}_1$ to the sum of the radii of the (possibly two) components of $\tilde{\boldsymbol{x}}_1 \cap \boldsymbol{x}_1$.

T**:** The total clock time in seconds to execute a Gauss–Seidel step for that particular preconditioner for all of the problems in the set[16].

In our first set of experiments, we set $\Omega = 0$. In these experiments, it is likely that the solution set contains points near the origin in $\mathbb{R}^n$, making it unlikely that the solution set contains disconnected components (and less likely that a preconditioner will produce a split), and also less likely that Algorithm 2 will exit due to $\tilde{\boldsymbol{x}}_1 \cap \boldsymbol{x}_1 = \emptyset$ before all steps have been completed. The results appear in Table 6. We note that all preconditioner schemes seem to have failed when n is large and B is large, or when R is large. Such cases do not give us

[16]We ran the experiments on a dual-processor Dell Optiplex GX-280 with dual 3.2 gigahertz processors and 2 gigabytes of RAM. Since nested loops were used in Matlab at several places, loop overhead may be a significant component of the overall time.

Table 7: Experiments with random problems, $\Omega \neq 0$.

m	n	N_t	B	R	Ω	type	N_w	N_s	N_M	$\rho_\div$	T
10	10	100	1	1	1.0	Inv. mid.:	0	0	100	1.000	0.99
						Opt. wid.:	0	0	100	1.000	22.76
						Alg. 2:	0	0	100	1.000	424.43
						Alg. 3:	0	0	100	1.000	420.02
						pivoting:	0	0	100	1.000	199.35
10	10	100	0.1	1	1.0	Inv. mid.:	6	0	40	0.960	1.01
						Opt. wid.:	19	0	40	0.945	22.88
						Alg. 2:	62	0	100	0.512	395.69
						Alg. 3:	62	0	58	0.708	242.89
						pivoting:	0	0	38	1.000	201.96
10	10	100	1	1	2.0	Inv. mid.:	0	0	99	1.000	0.99
						Opt. wid.:	0	0	99	1.000	22.60
						Alg. 2:	1	0	100	0.998	439.35
						Alg. 3:	1	0	100	0.998	437.89
						pivoting:	0	0	99	1.000	200.52
10	10	100	1	1	5.0	Inv. mid.:	0	0	19	1.000	0.97
						Opt. wid.:	14	0	24	0.915	22.40
						Alg. 2:	81	3	100	0.227	277.58
						Alg. 3:	81	9	54	0.416	168.56
						pivoting:	4	2	19	0.986	146.23
10	10	100	0.1	1	5.0	Inv. mid.:	57	0	49	0.468	0.99
						Opt. wid.:	75	0	50	0.356	22.21
						Alg. 2:	100	0	100	0.000	84.27
						Alg. 3:	100	0	64	0.170	40.35
						pivoting:	1	0	0	0.996	26.26
50	50	10	0.1	1	5.0	Inv. mid.:	0	0	0	1.000	4.14
						Opt. wid.:	0	0	0	1.000	531.42
						Alg. 2:	100	0	100	0.000	6263.93
						Alg. 3:	100	3	11	0.441	2786.64
						pivoting:	0	0	0	1.000	20422.64

much information, since it is possible (but we do not know) in those cases that the exact solution set contains the original bounds $\boldsymbol{x}$ for all of the generated problems. However, the relative performance of the different preconditioner schemes gives us some information in the other cases.

Runs with $\Omega \neq 0$ appear in Table 7. There, we observe more success, especially for the composite algorithms, for larger values of Ω and smaller values of B. This is to be expected, since the solution set to $\boldsymbol{A}x = \boldsymbol{b}$ is more likely to lie outside $\boldsymbol{x}$, and hence splitting preconditioners are more likely to be effective in such cases.

Finally, we did some experiments with no uncertainty in the right-hand-side vector $\boldsymbol{b}$ (that is, with $\boldsymbol{b}$ a point vector); this corresponds to use of these techniques in interval Newton methods. The results appear in Table 8. Comparing Table 8 with Table 7, we see that all preconditioner schemes except using no preconditioner (or using "pivoting" preconditioners) do a better job when there is a point right side vector, although some problems (especially with larger m and n) are still difficult.

The Matlab "m" files for running all of these experiments are available at `http://interval.louisiana.edu/misc/m_files_precond_ivl_lin.zip`.

Table 8: Experiments with random problems, no uncertainty in $\boldsymbol{b}$.

m	n	N_t	B	R	Ω	type	N_w	N_s	N_M	$\rho_\div$	T
10	10	100	0.1	1	0.0	Inv. mid.:	0	0	100	1.000	0.96
						Opt. wid.:	0	0	100	1.000	22.28
						Alg. 2:	0	0	100	1.000	410.73
						Alg. 3:	0	0	100	1.000	407.30
						pivoting:	0	0	100	1.000	193.73
50	50	10	0.1	1	0.0	Inv. mid.:	0	0	10	1.000	0.41
						Opt. wid.:	0	0	10	1.000	52.43
						Alg. 2:	0	0	10	1.000	4801.41
						Alg. 3:	0	0	10	1.000	4814.27
						pivoting:	0	0	10	1.000	2010.78
50	50	10	0.01	1	0.0	Inv. mid.:	5	0	2	0.691	0.41
						Opt. wid.:	6	0	0	0.709	52.26
						Alg. 2:	10	0	10	0.000	434.68
						Alg. 3:	10	0	4	0.363	119.03
						pivoting:	0	0	0	1.000	2005.18
10	10	100	1.0	1	1.0	Inv. mid.:	0	0	99	1.000	0.97
						Opt. wid.:	0	0	99	1.000	22.40
						Alg. 2:	1	1	100	0.997	428.22
						Alg. 3:	1	1	99	1.000	417.46
						pivoting:	0	0	99	1.000	192.63
10	10	100	0.1	1	1.0	Inv. mid.:	59	0	32	0.543	0.97
						Opt. wid.:	79	0	18	0.508	22.07
						Alg. 2:	100	0	100	0.021	200.23
						Alg. 3:	100	0	29	0.358	45.13
						pivoting:	3	0	0	0.991	185.08
10	10	100	1.0	1	2.0	Inv. mid.:	0	0	99	1.000	0.97
						Opt. wid.:	0	0	99	1.000	22.40
						Alg. 2:	1	1	100	0.997	428.22
						Alg. 3:	1	1	99	1.000	417.46
						pivoting:	0	0	99	1.000	192.63
10	10	100	1.0	1	5.0	Inv. mid.:	9	2	14	0.925	0.98
						Opt. wid.:	21	0	22	0.840	22.25
						Alg. 2:	92	1	100	0.094	218.51
						Alg. 3:	92	2	47	0.351	154.11
						pivoting:	2	1	8	0.989	94.90

6.2 Experiments with Selected Matrix Market Matrices

Experiments with random matrices do not always reflect problems that occur in practice, such as problems with high condition numbers. To study an example with a high condition number and examples of problems occurring in practice, we have selected several problems from the Matrix Market collection [3].

In these experiments, we formed matrices with midpoints the corresponding matrix in the Matrix Market set, then perturbed it randomly with the parameter B as above chosen with various values consistent with the magnitudes of the entries in the midpoint matrix; in the case of non-regular interval matrices (present with larger values of B, depending on the condition number of the midpoint matrix), larger values of Ω (leading to entries of $\boldsymbol{b}$ with larger magnitudes), combined with larger values of R (leading to larger initial guess boxes), make it more likely that the intersection of the solution set with the original $\boldsymbol{x} \in [0, R]^n$ will have more than one disjoint component. This is because, in a Gauss–Seidel step (4), the numerator will not contain zero provided $\tilde{\boldsymbol{b}}_k$ is sufficiently large, so that a split (the union of two disjoint intervals as in Table 5) will occur when $\tilde{\boldsymbol{a}}_{k,k}$ contains 0.

Table 9: Experiments with the Hilbert matrix.

n	κ	N_t	B	R	Ω	type	N_w	N_s	N_M	$\rho_\div$	T
10	$1.6 \cdot 10^{13}$	100	0.1	1	10	Inv. mid.:	7	2	5	0.943	1.23
						Opt. wid.:	50	0	8	0.713	26.74
						Alg. 2:	100	0	100	0.000	276.22
						Alg. 3:	100	0	23	0.383	97.50
						pivoting:	6	0	0	0.992	98.22
10	$1.6 \cdot 10^{13}$	100	0.1	1	10^2	Inv. mid.:	99	0	99	0.010	1.44
						Opt. wid.:	98	0	97	0.023	26.41
						Alg. 2:	100	0	100	0.000	28.19
						Alg. 3:	100	0	100	0.000	27.00
						pivoting:	0	0	0	1.000	1.50
10	$1.6 \cdot 10^{13}$	100	0.1	1	10^4	Inv. mid.:	100	0	100	0.000	1.48
						Opt. wid.:	100	0	100	0.000	26.78
						Alg. 2:	100	0	100	0.000	26.95
						Alg. 3:	100	0	100	0.000	26.98
						pivoting:	0	0	0	1.000	1.27
50	$> 1.6 \cdot 10^{16}$	10	0.1	1	1	Inv. mid.:	0	0	2	1.000	0.50
						Opt. wid.:	0	0	2	1.000	61.33
						Alg. 2:	8	0	10	0.261	812.36
						Alg. 3:	8	1	5	0.472	474.47
						pivoting:	0	0	2	1.000	1486.48

In all cases, for simplicity in the experiments, we chose to not to utilize the parameter γ, so each right-hand-side $\boldsymbol{b}$ is actually a point vector, dependent only on the random numbers chosen according to B and Ω.

In the tables for the results, we did not list m since all matrices were square, but we listed the condition number κ (or estimate thereof).

The chosen matrices are as follows:

the Hilbert matrix, to probe the effects of ill-conditioning in the midpoint matrix;

IMPCOL B, Cavett's process, from the chemical engineering plant models collection, included as a small instance of a practical problem;

WEST0067, the Cavett problem with five components from the chemical engineering plant models collection;

BCSSTRUC1, a small test problem in structural engineering;

Although these test problems include sparse problems and symmetric problems, we handled the systems as general dense systems. However, when adding interval uncertainty to the matrix entries, we only added uncertainties to those entries that were not exactly zero.

We see the results for the Hilbert matrix in Table 9. For matrix perturbations on the order of the size of the entries ($B = .1$), and relatively small R compared to the probable size of the actual solution to $Ax = b$, where A is the midpoint matrix, we see that the inverse midpoint matrix led to a reduction in only 9% of the cases, the best possible result using no preconditioner (also known as "pivoting preconditioners" in [5]) led to a reduction of width in only 6% of the cases, and the optimal width preconditioner alone led to a reduction of width

Table 10: Experiments with IMPCOL B.

n	κ	N_t	B	R	Ω	type	N_w	N_s	N_M	$\rho_\div$	T
59	$1.6 \cdot 10^5$	10	0.1	10	1	Inv. mid.:	5	0	5	0.500	0.59
						Opt. wid.:	9	0	7	0.246	85.42
						Alg. 2:	10	0	10	0.000	175.23
						Alg. 3:	10	0	7	0.207	125.93
						pivoting:	0	0	0	1.000	474.39
59	$1.6 \cdot 10^5$	10	1.0	10	1	Inv. mid.:	0	0	0	1.000	0.61
						Opt. wid.:	4	0	1	0.727	85.75
						Alg. 2:	10	0	10	0.000	367.14
						Alg. 3:	10	0	6	0.177	248.27
						pivoting:	0	0	0	1.000	1125.14
59	$1.6 \cdot 10^5$	10	1.0	10	100	Inv. mid.:	5	0	5	0.500	0.63
						Opt. wid.:	9	0	8	0.123	86.29
						Alg. 2:	10	0	10	0.000	109.48
						Alg. 3:	10	0	9	0.023	97.91
						pivoting:	0	0	0	1.000	7.38

in only 50% of the cases. In contrast, both Algorithm 2 and Algorithm 3 gave width reductions in all cases, but Algorithm 2 gave wider reductions in 77% of the cases. For larger Ω, every preconditioner scheme except the "pivoting preconditioners" gave optimal results; this is probably because the numerator in the Gauss–Seidel equation (4) is so large due to the elements of $\boldsymbol{b}$ that the iteration proves that the solution lies outside the initial box $[-R, R]^n$; note that, in this case, Algorithm 2 and Algorithm 3 can dispatch with the problem quickly, using no more time than a single application of the optimal width preconditioner. (Increasing R may change the results.)

To get a hint of how the results depend on dimension, everything else being the same, we tried the 50-dimensional Hilbert matrix, and adjusted Ω until the results corresponded roughly to the $B = 0.1$, $n = 10$, $\Omega = 10$ case. With $\Omega = 10$, the inverse midpoint preconditioner was dominant, while the results for $\Omega = 1$ are seen in the last block in Table 9. Because the behaviors of Algorithm 2 and Algorithm 3 depend so strongly on the problem, it is not possible to get a precise measure of dimension dependence. However, the last block of Table 9 corresponds roughly to the first block in the sense that the inverse midpoint preconditioner and optimal width preconditioner are relatively ineffective. If the time increased cubically (as should be the case for the inverse midpoint preconditioner), then the times in the last block should be the times in the first block multiplied by $12.5 = (10/100 \cdot (50/10)^3)$. In fact, for the inverse midpoint preconditioner, the ratio is less than 1, whereas, for the optimal width preconditioner, the ratio is $61.33/26.74 \approx 2.3 < 12$. Algorithm 2 and Algorithm 3 similarly correspond to ratios less than 12, and the relative difference between Algorithm 2 and Algorithm 3 is less for $n = 50$; perhaps Algorithm 2 is more desirable for larger, more uncertain problems.

Results for IMPCOL B appear in Table 10. For uncertainties on the order of 10% of the matrix elements, we se that, although the inverse midpoint preconditioner resulted in a width reduction in 5/10 of the cases, the optimal width preconditioner resulted in a reduction in 9/10 of the cases, and only Algorithm 2 and Algorithm 3 resulted in a width reduction for all cases, while Algorithm 2

Table 11: Experiments with WEST0067.

n	κ	N_t	B	R	Ω	type	N_w	N_s	N_M	$\rho_{\div}$	T
67	$1.6 \cdot 10^2$	10	0.1	10	0	Inv. mid.:	0	0	9	1.000	0.71
						Opt. wid.:	1	0	10	0.998	109.86
						Alg. 2:	1	0	10	0.998	6209.82
						Alg. 3:	1	0	10	0.998	5607.79
						pivoting:	0	0	9	1.000	5853.95
67	$1.6 \cdot 10^2$	10	0.1	10	1	Inv. mid.:	0	0	8	1.000	0.69
						Opt. wid.:	1	0	9	0.968	109.67
						Alg. 2:	2	0	10	0.868	4724.11
						Alg. 3:	2	0	9	0.918	4166.21
						pivoting:	0	0	8	1.000	5296.61
67	$1.6 \cdot 10^2$	10	0.1	10	10	Inv. mid.:	1	0	1	0.900	0.66
						Opt. wid.:	10	0	10	0.000	109.31
						Alg. 2:	10	0	10	0.000	109.36
						Alg. 3:	10	0	10	0.000	110.39
						pivoting:	0	0	0	1.000	411.95
67	$1.6 \cdot 10^2$	10	0.01	10	1	Inv. mid.:	8	0	2	0.523	0.66
						Opt. wid.:	8	0	0	0.569	109.42
						Alg. 2:	10	0	10	0.000	922.15
						Alg. 3:	10	0	1	0.388	161.92
						pivoting:	0	0	0	1.000	5505.00
67	$1.6 \cdot 10^2$	10	0.1	10	10	Inv. mid.:	10	0	9	0.040	0.66
						Opt. wid.:	10	0	10	0.000	109.26
						Alg. 2:	10	0	10	0.000	110.87
						Alg. 3:	10	0	10	0.000	109.72
						pivoting:	0	0	0	1.000	197.56

did better than Algorithm 3 in 3/10 of the cases. Using all possible "pivoting" preconditioners did not lead to *any* width reduction, so a more advanced preconditioner scheme is necessary here. When the uncertainty intervals were on the order of the size of the matrix entries, results were similar for Algorithm 2 and Algorithm 3, but there was less success with the optimal width preconditioner, and the inverse midpoint preconditioner was useless. With results similar to those for the Hilbert matrix, for larger Ω but fixed R, the inverse midpoint matrix becomes better, (but not as good as the optimal width or Algorithm 2 and Algorithm 3); pivoting preconditioners were useless in all cases tried.

The experiments with WEST0067 appear in Table 11. We for both uncertainties, we observe a phenomenon in common with the results for the Hilbert matrix and IMPCOL B: the inverse midpoint preconditioner becomes more useful when Ω (determining the size of the right-hand-side vector b) becomes larger, but the pivoting preconditioner is useless in all cases. For uncertainty intervals on the order of 10% of the largest matrix element and small Ω, there was limited success with any algorithm, although the width-optimal preconditioner gave a width reduction in one case ($\Omega = 0$), and Algorithm 2 and Algorithm 3 did better than the width-optimal preconditioner in one case for $\Omega = 1$. For smaller uncertainty intervals ($B = .01$, representing about 1% of the largest matrix element), the inverse midpoint preconditioner worked in 80% of the cases, but Algorithm 2 worked in all cases and gave a much better average width reduction for $\Omega = 1$; for $\Omega = 10$, all preconditioner schemes except pivoting preconditioners worked, and there was only a minor difference in the average width reduction between the inverse midpoint preconditioner and other preconditioners.

Table 12: Experiments with BCSSTRUC1.

n	κ	N_t	B	R	Ω	type	N_w	N_s	N_M	$\rho_\div$	T
48	$8.8 \cdot 10^5$	10	1	10	10	Inv. mid.:	10	0	10	0.012	0.48
						Opt. wid.:	10	0	10	0.012	56.48
						Alg. 2:	10	0	9	0.018	1266.26
						Alg. 3:	10	0	10	0.012	84.05
						pivoting:	0	0	0	1.000	2186.22
48	$8.8 \cdot 10^5$	10	1	10	10^2	Inv. mid.:	10	0	10	0.002	0.48
						Opt. wid.:	10	0	10	0.002	57.42
						Alg. 2:	10	0	10	0.002	1184.69
						Alg. 3:	10	0	10	0.002	85.29
						pivoting:	0	0	0	1.000	2208.58
48	$8.8 \cdot 10^5$	10	10^5	10	10^2	Inv. mid.:	0	0	4	1.000	0.49
						Opt. wid.:	6	0	10	0.626	59.94
						Alg. 2:	3	0	6	0.846	1418.66
						Alg. 3:	2	0	4	0.932	1306.35
						pivoting:	1	0	4	0.985	2191.72
48	$8.8 \cdot 10^5$	10	10^5	10	10^5	Inv. mid.:	0	0	6	1.000	0.48
						Opt. wid.:	4	0	10	0.775	57.46
						Alg. 2:	1	0	6	0.993	1308.53
						Alg. 3:	1	0	6	0.993	1211.14
						pivoting:	0	0	6	1.000	2190.85

In Table 12, we see the results for the small structural analysis test problem. For a very small level of uncertainty (with $B = 1$, on the order of 10^{-9} of the size of the entries of the midpoint matrix), we see that the inverse midpoint, optimal width, Algorithm 2, and Algorithm 3 do equally well, for small Ω, but the pivoting preconditioners are useless. For larger uncertainty ($B = 10^5$, corresponding to 0.01% uncertainty in relation to the matrix entry with maximum magnitude), the inverse midpoint preconditioner is useless, but the width-optimal preconditioner is successful in half of the tries. Surprisingly, the width-optimal preconditioner seems to do better than either Algorithm 2 or Algorithm 3. This is probably because, first, width-optimal preconditioners are probably most appropriate for this problem and, second, because we chose $\delta = 0.5$ (a "classic" choice) when only the width-optimal preconditioner was used, but we chose random δ in the width-optimal preconditioners for Algorithm 2, and Algorithm 3; $\delta = 0.5$ may lead to more easily solvable linear programs (such as well-posed ones) for this problem. (The results are similar for larger Ω.)

6.3 Comparison with Alternate Techniques

Most general-purpose solvers do not handle the case when $\boldsymbol{A}$ is not regular. For example, `verifylss`, distributed with `INTLAB`, gives an optimal enclosure[17] to x_1 for Example 1 of [9], but gives all components of $\boldsymbol{x}_1$ equal to $[\mathtt{NaN}, \mathtt{NaN}]$ for the other six examples.

The techniques described in this work are for general, non-structured, nonsymmetric systems. In many cases, much better results can be obtained if properties of a particular problem are taken into account. For example, techniques from [1] and [2] might be used (possibly in conjunction with the optimal

[17]to within rounding error

preconditioner formulations in this work), to enable more efficient bounding of solution sets with symmetric matrices or matrices with other structure. Also, in certain applications, alternate techniques can yield very good results in the presence of large uncertainties. An example of this is in the analysis of truss structures [14] or the earlier work of Muhanna and Mullen, e.g. [12]. However, the work [14] and that of Muhanna and Mullen do not address uncertainties that are so large that the solution set is disconnected.

7 Summary and Future Work

Sharply bounding the solution sets of linear systems with large uncertainties in the coefficients and right-hand-side vectors is an NP-hard problem, but various heuristics, including preconditioning, can be used. We have presented details of linear programming formulations with complementarity constraints for such preconditioners for cases where the solution set is unbounded or contains more than one component. Based on detailed studies of examples specially designed to illustrate the contexts in which each of these are advantageous, we have proposed two composite polynomial time algorithms which incorporate all of these preconditioners. We have tested these algorithms with randomly generated matrices, as well as with selected matrices from the Matrix Market collection.

Our results illustrate that preconditioning other than merely choosing a pivot variable and equation is in general necessary, that the inverse midpoint preconditioner is inadequate in many cases, but our optimal preconditioners sometimes perform well or adequately. In particular, our linear programming (or linear programming with complementarity constraint) based splitting preconditioners and the mignitude optimal preconditioner have been formulated but not previously examined from a practical point of view[18]; here, we have seen that these formulations complement the width-optimal preconditioner, and are worth incorporating into solution algorithms, in various contexts.

It may be possible to improve the performance, particularly for specific classes of problems, by heuristically identifying which preconditioners may be most appropriate and only using those, rather than a sequence of all preconditioners[19]. Furthermore, theory as observed in [5] can be used to inexpensively determine a priori non-existence of S-preconditioners, obviating the need to formulate and solve the linear programs in such cases. Moreover, use of recent developments in the solution of complementarity constrained problems, such as in [11] and the references therein, may help. Also, to solve many larger problems occurring in practice, the implementation should be modified to take account of sparsity, and the efficiency of the overall implementation can be improved by pre-compiling important parts of the *m*-files.

The various steps of Algorithm 2 are completely independent, and thus can be done in parallel, with the intersection of the various $\tilde{\boldsymbol{x}}_k$ with $\boldsymbol{x}_k$ computed

[18]Splitting not involving linear programming was also examined in [5], [6], and in practice in [4].

[19]This has been done, for example, in [4].

afterwards.

The techniques in this work can possibly be combined with branch and bound techniques (even in the linear case), or change of basis techniques, to obtain sharp bounds on solution sets.

Finally, it is noteworthy that these techniques, in their basic form, are for general matrices; alternate techniques proposed by others are superior for specific classes of problems.

Acknowledgement

The author wishes to acknowledge Siripawn Winter, whose preliminary experiments and discussions were quite helpful. I also wish to thank both of the referees; each of them read the paper with care and understanding, resulting in numerous minor corrections and overall suggestions that have made the work correct, stronger and easier to comprehend. Finally, I wish to thank the editor, Dietmar Saupe, for his patience and diligence.

References

[1] G. Alefeld, V. Kreinovich, and G. Mayer, *The shape of the symmetric solution set*, in Applications of interval computations: Papers presented at an international workshop in El Paso, Texas, February 23–25, 1995, R. B. Kearfott and V. Kreinovich, eds., vol. 3 of Applied Optimization, Norwell, MA, USA, and Dordrecht, The Netherlands, 1996, Kluwer Academic Publishers Group, pp. 61–80.

[2] ——, *On the solution sets of particular classes of linear interval systems*, J. Comput. Appl. Math., 152 (2003), pp. 1–15.

[3] R. F. Boisvert, R. Pozo, K. Remington, R. Barrett, and J. J. Dongarra, *The Matrix Market: A web resource for test matrix collections*, in Quality of Numerical Software, Assessment and Enhancement, R. F. Boisvert, ed., London, 1997, Chapman and Hall, pp. 125–137.

[4] C.-Y. Gau and M. A. Stadtherr, *Reliable high performance computing strategies for chemical process modelling*, 1999. `http://www.nd.edu/~markst/dallas99/slides213c.pdf`.

[5] C.-Y. Hu, *Splitting Preconditioners for the Interval Newton Method*, PhD thesis, University of Southwestern Louisiana, 1990.

[6] C.-Y. Hu and R. B. Kearfott, *A pivoting scheme for the interval Gauss–Seidel method: Numerical experiments*, in Approximation, Optimization and Computing: Theory and Applications, Amsterdam, 1990, Elsevier Science Publishers, pp. 97–100.

[7] R. B. Kearfott, *Preconditioners for the interval Gauss–Seidel method*, SIAM J. Numer. Anal., 27 (1990), pp. 804–822.

[8] ——, *Rigorous Global Search: Continuous Problems*, Kluwer, Dordrecht, Netherlands, 1996.

[9] R. B. Kearfott and S. Hongthong, *On preconditioners and splitting in the interval Gauss–Seidel method*, tech. report, University of Louisiana at Lafayette, 2005. `http://interval.louisiana.edu/preprints/2005_new_S_preconditioner.as_submitted.pdf`.

[10] V. Kreinovich, A. Lakeyev, J. Rohn, and P. Kahl, *Computational Complexity and Feasibility of Data Processing and Interval Computations*, Kluwer, Dordrecht, Netherlands, 1998.

[11] S. Leyffer, G. Lopez-Calva, and J. Nocedal, *Interior methods for mathematical programs with complementarity constraints*, SIAM J. Optim., 17 (2006), pp. 52–77.

[12] R. L. Muhanna and R. L. Mullen, *Uncertainty in mechanics problems*, Journal of Engineering Mechanics, 127 (2001), pp. 557–566.

[13] A. Neumaier, *Interval Methods for Systems of Equations*, Cambridge University Press, Cambridge, England, 1990.

[14] A. Neumaier and A. Pownuk, *Linear systems with large uncertainties, with applications to truss structures*, Reliable Computing, 13 (2007), pp. 149–172.

[15] M. Novoa, *Theory of preconditioners for the interval Gauss-Seidel method and existence/uniqueness theory with interval newton methods.*, technical report, Dept. Mathematics, Univ. Southwestern Louisiana, 1993.

[16] W. Oettli, *On the solution set of a linear system with inaccurate coefficients*, SIAM J. Numer. Anal., 2 (1965), pp. 115–118.

[17] W. Oettli and W. Prager, *Compatibility of approximate solution of linear equations with given error bounds for coefficients and right-hand sides*, Numer. Math., 6 (1964), pp. 405–409.

[18] J. D. Pryce and G. F. Corliss, *Interval arithmetic with containment sets*, Computing, 78 (2006), pp. 251–276.

[19] J. Rohn, *Solving systems of linear interval equations*, in Reliability in Computing, Perspectives in Computing, New York, 1988, Academic Press, pp. 171–182.

[20] S. M. Rump, *INTLAB – INTerval LABoratory (INTLAB home page)*, 2005. `http://www.ti3.tu-harburg.de/~rump/intlab/`.