ON PRECONDITIONERS AND SPLITTING IN THE INTERVAL GAUSS–SEIDEL METHOD

R. BAKER KEARFOTT* AND SIRIPORN HONGTHONG†

Abstract. Finding bounding sets to solutions to systems of algebraic equations with uncertainties in the coefficients, as well as finding mathematically rigorous but rapid location of all solutions to nonlinear systems or finding global optima, involves bounding the solution sets to systems of equations with wide interval coefficients. The interval Gauss–Seidel algorithm has various properties that make it suited to this task. However, the system must in general be preconditioned for the interval Gauss–Seidel method to be effective. The most common preconditioner has been the "inverse midpoint" preconditioner; however, we have proposed other classes of preconditioners that obey certain optimality conditions, and have shown empirically advantages of their use. In this paper, we revisit similar preconditioners, previously applied only in the context of interval Newton methods, that are appropriate when the solution set may have more than one semi-infinite component. We first review our previous work, describing it in a more general context than interval Newton methods. We then carefully study the effectiveness of various preconditioners, compared to the inverse midpoint preconditioner, our previously studied "width-optimal" preconditioner, exact bounding of the systems of equations, and compared to variants based on a quadratic penalty function. We first compare the preconditioners in detail on some simple, specially-designed small systems, then we study the behavior of the preconditioners on some larger, randomly generated systems.

Key words. 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 effect of nonlinearities in a system as intervals of possible values[1], also leading to linear systems with interval coefficients. In either case, this leads to an interval linear system of equations

$$\boldsymbol{A}x = \boldsymbol{b}, \tag{1.1}$$

where $\boldsymbol{A}$ is a matrix with interval coefficients and $\boldsymbol{b}$ is a vector with interval coefficients. It is desirable to find sharp bounds on the solution set to (1.1), where the solution set is defined as

$$\Sigma(\boldsymbol{A},\boldsymbol{b}) = \{x \mid Ax = b \text{ for some } A \in \boldsymbol{A} \text{ and some } b \in \boldsymbol{b}\}, \tag{1.2}$$

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 [**?**]) 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 (1.1) 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.

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

†Department of Mathematics, University of Louisiana, U.L. Box 4-1010, Lafayette, Louisiana, 70504-1010, USA (`sxh1113@hotmail.com`).

[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 [**?**, §3.4].

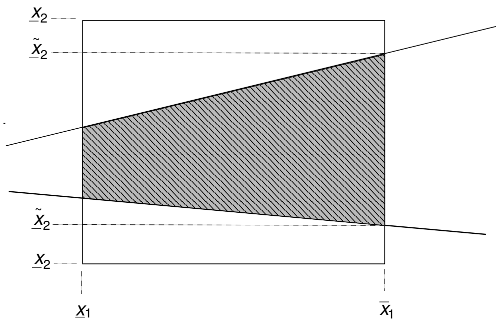

FIG. 1.1. *Application of the interval Gauss–Seidel method when the solution set is unbounded.*

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 (1.1) 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 (1.1), forming[3] the derived system

$$YAx = Y\boldsymbol{b} \quad \text{i.e. } \tilde{\boldsymbol{A}}x = \tilde{\boldsymbol{b}}. \tag{1.3}$$

Although the solution set to (1.3) in general is not equal to the solution set to (1.1) (but merely contains it), Y can often be chosen so that the overestimation in the solution process[4] for (1.3) is not catastrophic, and usable (although not optimal) bounds on the components of $\Sigma(\boldsymbol{A}, \boldsymbol{b})$ can be obtained. 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 (1.3) has been the inverse midpoint matrix, that is $Y = (\mathrm{m}(\boldsymbol{A}))^{-1}$, where $\mathrm{m}(\boldsymbol{A})$ is the matrix whose i, j-th entry is a floating point approximation to $a_{i,j} = (\underline{a}_{i,j} + \overline{a}_{i,j})/2$, 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 of the matrix $\boldsymbol{A}$ are small. However, the preconditioner can be ineffective when the 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 1.1: there, the solution set lies between the two oblique lines, the original bounds are $\underline{x}_1 \leq x_1 \leq \overline{x}_1$ and $\underline{x}_2 \leq x_2 \leq \overline{x}_2$, and it may be possible with the interval Gauss–Seidel method to narrow the bounds on x_2 to $\tilde{\underline{x}}_2 \leq x_2 \leq \tilde{\overline{x}}_2$.

In contrast, in Figure 1.2, the 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.

[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

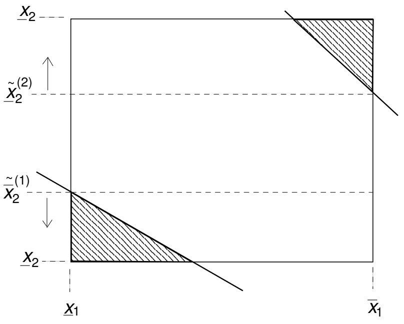

FIG. 1.2. *Application of the interval Gauss–Seidel method when the solution set is disconnected.*

In previous work ([**?**] and [**?**, Ch. 3]), we proposed preconditioners for the interval Gauss–Seidel method for the cases illustrated in Figure 1.1 and Figure 1.2. These preconditioners are designed for the interval Gauss–Seidel method. Specifically, if $\tilde{\boldsymbol{A}}$ and $\tilde{\boldsymbol{b}}$ are as in (1.3), the interval Gauss–Seidel method is given as

$$\begin{array}{lll} \boldsymbol{x}_k & \text{is given} & \text{for } 1 \le i \le n, \\ \tilde{\boldsymbol{x}}_k & \leftarrow \dfrac{1}{\tilde{\boldsymbol{a}}_{k,k}} \left\{ \tilde{\boldsymbol{b}}_k - \displaystyle\sum_{j=1}^{k-1} \tilde{\boldsymbol{a}}_{k,j} \tilde{\boldsymbol{x}}_j - \sum_{j=k+1}^{n} \tilde{\boldsymbol{a}}_{k,j} \boldsymbol{x}_j \right\} & \text{for } k = 1, 2, \ldots, n. \end{array} \tag{1.4}$$

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

$$\tilde{\boldsymbol{x}}_k = [\underline{\tilde{x}}_k, \overline{\tilde{x}}_k], \tag{1.5}$$

as in Figure 1.1, whereas, if $0 \in \tilde{\boldsymbol{a}}_{i,i}$ and the numerator in (1.4) does not contain zero, then $\tilde{\boldsymbol{x}}_k$ will consist of two semi-infinite intervals of the form

$$\tilde{\boldsymbol{x}}_k = \left(-\infty, \overline{\tilde{x}}_k^{(1)}\right] \bigcup \left[\underline{\tilde{x}}_k^{(2)}, \infty\right), \tag{1.6}$$

as is illustrated in Figure 1.2.

For the case that $\tilde{\boldsymbol{x}}_k$ is a single connected interval, the preconditioners we described in [**?**, 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 (1.5). In [**?**], we examined in detail "width-optimal" preconditioner (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 $0 \in \tilde{\boldsymbol{a}}_{i,i}$ (as in Figure 1.2), the "splitting preconditioners" are based on devising the i-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 $\min\{|\tilde{\overline{x}}_k^{(1)}|, |\tilde{\underline{x}}_k^{(2)}|\}$.

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 (1.6) does not exist; see Lemma 3.7, page 138 of [**?**].

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, and were never published. However, others (such as [**?**], in the context of parallel computation) have reported advantages to using "splits" in the interval Gauss–Seidel method[5]. This has prompted us to revisit the issue of optimal splitting preconditioners.

In §2, we review details of our linear programming formulations, while in §3, we give a penalty function formulation that leads to a quadratic program. In §5, we give examples designed to test specific aspects of our formulations. In §7, we give a more comprehensive set of experimental results, while we summarize in §8.

2. Ideas Behind Optimal Preconditioners: Review and Analysis.

Here, we review the derivation in [**?**] and [**?**, Ch. 3], placing it in somewhat more general terms and pointing out an underlying heuristic assumption that, when not true, can cause the solution of the linear program to fail to give a good preconditioner.

We assume the i-th row of the preconditioner Y for (1.3) is

$$Y_k = (y_1, \dots y_n)^T,$$

and we rewrite

$$y_i = y_i^+ - y_i^-, \quad \text{where } y_i^+ = \max\{y_i, 0\} \text{ and } y_i^- = \max\{-y_i, 0\}, \tag{2.1}$$

for $1 \le i \le m$. 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[6]. We have

THEOREM 2.1. *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 $\boldsymbol{x}_j = [-x_j, x_j]$, $1 \le j \le m$, $j \neq k$. Then the numerator in (1.4) is*

$$\boldsymbol{\nu}_k = Y_k\boldsymbol{b} + \sum_{\substack{j=1\\ j\neq k}}^{n} \{Y_k \boldsymbol{A}_{:,j}\}\, \boldsymbol{x}_j$$

$$= \sum_{i=1}^{m} y_i \mathrm{m}(\boldsymbol{b}_i) + \frac{1}{2}\sum_{i=1}^{m} |y_i| \mathrm{w}(\boldsymbol{b}_i)[-1,1] + \frac{1}{2}\sum_{\substack{j=1\\ j\neq k}}^{n} \mathrm{w}(\boldsymbol{x}_j)\left|\sum_{i=1}^{m} y_i \boldsymbol{a}_{i,j}\right|[-1,1], \tag{2.2}$$

and the denominator in (1.3) is

$$\boldsymbol{d}_k = Y_k \boldsymbol{A}_{:,k} = \sum_{i=1}^{m} y_i \boldsymbol{a}_{i,k} = \sum_{i=1}^{m} \mathrm{m}(\boldsymbol{a}_{i,k}) + \frac{1}{2}\sum_{i=1}^{m} |y_i| \mathrm{w}(a_{i,k})[-1,1]. \tag{2.3}$$

[5]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.

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

Proof. The proof proceeds from Lemma (3.4) of [**?**, p. 129], in a manner entirely analogously to Lemma (3.5) of [**?**]. ☐

Now, *assuming* that y_i^+ and y_i^- are the positive and negative parts of y_i (as in (2.1)), we have, as in the development in [**?**, pp. 130–131],

$$\left|\sum_{i=1}^{m} y_i \boldsymbol{a}_{i,j}\right| = \left\{\sum_{i=1}^{m} \big(- y_k^+ \underline{a}_{i,j} + y_k^- \overline{a}_{i,j}\big)\right\} + v_j^+. \tag{2.4}$$

and

$$\left|\sum_{i=1}^{m} y_i \boldsymbol{a}_{i,j}\right| = \left\{\sum_{i=1}^{m} \big(y_i^+ \overline{a}_{i,j} - y_i^- \underline{a}_{i,j}\big)\right\} + v_j^-, \tag{2.5}$$

where v_j^+ and v_j are the positive and negative parts of v_j, as defined for y in (2.1), and where v_j is defined to be equal to

$$v_j = \overline{\sum_{i=1}^{m} y_i \boldsymbol{a}_{i,j}} + \underline{\sum_{i=1}^{m} y_i \boldsymbol{a}_{i,j}}, \quad 1 \le j \le n, \quad j \ne i. \tag{2.6}$$

Thus, if the hypotheses of Theorem 2.1 hold, then, observing that $|y_i| = y_i^- + y_i^+$, we have

$$\begin{aligned}\boldsymbol{\nu}_k = {} & \sum_{i=1}^{m} y_i \mathrm{m}(\boldsymbol{b}_i) + \frac{1}{2}\sum_{i=1}^{m} (y_i^- + y_i^+)\mathrm{w}(\boldsymbol{b}_i)[-1,1] \\ & + \frac{1}{2}\sum_{\substack{j=1\\ j\ne k}}^{n} \mathrm{w}(\boldsymbol{x}_j) \left(\delta \left\{ \sum_{i=1}^{m} \big(- y_i^+ \underline{a}_{i,j} + y_i^- \overline{a}_{i,j}\big) + v_j^+ \right\}\right. \\ & \left. + (1-\delta) \left\{ \sum_{i=1}^{m} \big(y_i^+ \overline{a}_{i,j} - y_i^- \underline{a}_{i,j}\big) + v_j^- \right\}\right) [-1,1],\end{aligned} \tag{2.7}$$

and similarly

$$\boldsymbol{d}_k = \left[\sum_{i=1}^{m} \big(y_i^+ \underline{a}_{i,k} - y_i^- \overline{a}_{i,k}\big), \sum_{i=1}^{m} \big(y_i^+ \overline{a}_{i,k} - y_i^- \underline{a}_{i,k}\big) \right]. \tag{2.8}$$

Now, continuing the derivation on page 133 of [**?**], the "width-optimal C-preconditioner[7]" of [**?**] is based on minimizing the width $\mathrm{w}(\tilde{\boldsymbol{x}}_k)$ in (1.4),

$$\begin{aligned} & \text{find } \min_{Y_k} \mathrm{w}(\boldsymbol{\nu}_k) = \min_{Y_k} \{\overline{\nu}_k - \underline{\nu}_k\} \\ & \text{subject to } \underline{d}_k = 1, \end{aligned} \tag{2.9}$$

which can be expressed in terms of the positive and negative parts y_i^+, y_i^-, v_i^+ and v_i^- defined above as

$$\text{minimize} \quad \sum_{i=1}^{m} (y_i^+ + y_i^-)\mathrm{w}(\boldsymbol{b}_i)$$

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

$$
\begin{aligned}
+ \qquad & \delta \sum_{\substack{j=1\\ j\neq k}}^{n} \mathrm{w}(\boldsymbol{x}_j) \left\{ v_j^+ + \sum_{i=1}^{m} \left(y_i^- \overline{a}_{i,j} - y_i^+ \underline{a}_{i,j} \right) \right\} \\
& +(1-\delta) \sum_{\substack{j=1\\ j\neq k}}^{n} \mathrm{w}(\boldsymbol{x}_j) \left\{ v_j^- + \sum_{i=1}^{m} \left(y_i^+ \overline{a}_{i,j} - y_i^- \underline{a}_{i,j} \right) \right\} \qquad (2.10)\\
\text{subject to} \quad & \sum_{i=1}^{m} \left(y_i^+ \underline{a}_{i,k} - y_i^- \overline{a}_{i,k} \right) = 1.
\end{aligned}
$$

Observe now

1. The formulation (2.10) is equivalent to (2.9), *provided that the y_i^+ and y_i^- are the positive and negative parts of y_i and that the v_j^+ and v_j^- are the positive and negative parts of v_j, respectively.*
2. The formulation (2.10) is linear in y_i^+, y_i^-, v_i^+, and v_j^-.

In formulating our linear program, we use the linear objective and linear constraint in (2.10). To this objective and constraint, we append the following.

1. To force the v_j to be as defined in (2.6), we append the constraints

$$
v_j^+ - v_j^- - \sum_{i=1}^{m} (y_i^+ - y_i^-)(\underline{a}_{i,j} + \overline{a}_{i,j}) = 0, \quad 1 \le j \le n, \quad j \neq k
$$

   (cf. (3.29) on page 132 of [?]).
2. To (hopefully) force the y_i^+, y_i^-, v_j^+ and v_j^- to correspond to the positive and negative parts of the y_i and v_j at the solution of the linear program, we append the natural constraints $y_i^+ \ge 0$, $y_i^- \ge 0$, $v_j^+ \ge 0$, and $v_i^- \ge 0$ for $1 \le i \le m$ and $1 \le j \le n$, $j \neq k$.

Summarizing, we have

Definition 2.2. *A* traditional width-optimal LP *is a linear program consisting of the objective and equality constraint in (2.10) (for an arbitrarily selected $\delta \in [0,1]$), along with the equality constraints in item 1 above and the nonnegativity constraints in item 2 above. Rearranging the objective (2.7) and the equality constraints, the linear program thus is as in Table 2.1.*

In contrast, for a splitting preconditioner, we assume that the denominator $\boldsymbol{d}_k$ in (1.4) (and characterized in (2.3)) contains zero, and we normalize the numerator $\boldsymbol{\nu}_k$ (as characterized in (2.2)). In particular, the “mignitude-optimal” preconditioner is based on the optimization problem

$$
\begin{aligned}
&\text{find } \min_{Y_k} |\boldsymbol{d}_k| \\
&\text{subject to } \underline{\nu}_k = 1.
\end{aligned} \qquad (2.11)
$$

Using a derivation, as in [?] and similar to that above for the width optimal C-preconditioner[8], we obtain a linear program as in Table 2.2.

Definition 2.3. *A* mignitude-optimal LP *is a linear program (for a particular $\delta \in [0,1]$) as in Table 2.2 (obtained from (2.11) with Theorem 2.1 and formulas (2.7) and (2.8)).*

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

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

Table 2.1
A linear program for an optimal width contracting preconditioner

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

Theorem 2.4. *(Lemma 3.7, page 138 of [?], and originally observed in unpublished work by Manuel Novoa) Suppose a preconditioner row Y_k solves the optimization problem (2.11). Then the following are true.*

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 $\overline{\tilde{x}}_k^{(1)}$ and $\underline{\tilde{x}}_k^{(2)}$ from 0 in (1.6).
2. *If $\boldsymbol{d}_k > 0$, then $\tilde{\boldsymbol{x}}_k$ is a single connected interval as in (1.5), and Y_k maximizes the left end point$\underline{\tilde{x}}_k$.*
3. *If $\boldsymbol{d}_k < 0$, then $\tilde{\boldsymbol{x}}_k$ is a single connected interval as in (1.5), and Y_k minimizes the right end point$\overline{\tilde{x}}_k$.*

Thus, if $\boldsymbol{x}_k$ is symmetric about 0, then the solution to the optimization problem corresponding to the mignitude optimal LP maximizes, in the 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.

Table 2.2
A linear program for a mignitude optimal preconditioner

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

The positive numerator S-preconditioner:

$$
\begin{array}{l}
\text{find } \max_{Y_k} \underline{\nu}_k \\
\text{subject to } \underline{d}_k \le -1 \text{ and } \overline{d}_k = 1.
\end{array}
\tag{2.12}
$$

The negative numerator S-preconditioner:

$$
\begin{array}{l}
\text{find } \min_{Y_k} \overline{\nu}_k \\
\text{subject to } \underline{d}_k = -1 \text{ and } \overline{d}_k \ge 1.
\end{array}
\tag{2.13}
$$

A linear program for the positive numerator S-preconditioner appears in Table 2.3,

TABLE 2.3
A linear program for the positive numerator S-preconditioner

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

while a linear program for the negative numerator S-preconditioner appears in Table 2.4.

3. A Penalty Function Formulation.

The linear programs exemplified in Tables 2.1 through 2.4 do not specify the variables sufficiently enough to force $y_i^+ y_i^- = 0$ and $v_j^+ v_j^- = 0$. In fact, the traditional width-optimal LP often gives solutions for which both $y_i^+ > 0$ and $y_i^- > 0$ for many i; however, defining the preconditioner by $y_i = y_i^+ - y_i^-$, $1 \le i \le m$ often leads to a good preconditioner. However, in the case of splitting preconditioners or the mignitude optimal preconditioner, ambivalent initial results leads us to question whether or not forcing $y_i^+ y_i^- = 0$ and $v_i^+ v_i^- = 0$ is advantageous. To do so, we simply add the quadratic penalty function

$$Q = \rho \left\{ \sum_{i=1}^{m} y_i^+ y_i^- + \sum_{j} v_j^+ v_j^- \right\} \tag{3.1}$$

for some large ρ, to the linear objectives in Table 2.1 or Table 2.2. This results in a quadratic program[9].

4. On Extended Arithmetic.

In computing the quotient (1.4), the mignitude-optimal preconditioner and the S-preconditioners lead to denominators that contain

[9]which is not necessarily positive-definite

TABLE 2.4
A linear program for the negative numerator S-preconditioner

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

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 [**?**]. 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, but we otherwise want the result to be as small a set as possible. This leads to the rules in Table 4.1.

5. Examples. In this section, we present results from problems that have been specifically designed, both to test correctness of our implementation and provide initial evidence of the effectiveness of the underlying ideas. In most of 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 [**?**] and earlier in [**?**], based on inequalities presented in [**?**]), 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` ([**?**]) to do the interval arithmetic, while we used `linprog` and `quadprog` from the MATLAB optimization toolbox to approximately solve the linear and quadratic programs, respectively. In all of the examples in this section, we systematically tried

1. the optimal width contracting preconditioner,
2. the mignitude optimal preconditioner,

TABLE 4.1
Rules for extended interval division based on cset theory

$$\frac{[\underline{\nu}_k,\overline{\nu}_k]}{[\underline{d}_k,\overline{d}_k]} = \begin{cases} [\underline{\nu}_k,\overline{\nu}_k]/[\underline{d}_k,\overline{d}_k] & \text{in the usual sense if } 0 \notin [\underline{d}_k,\overline{d}_k], \\[1ex] [-\infty,\infty] & \text{if } 0 \in [\underline{\nu}_k,\overline{\nu}_k] \text{ and } 0 \in [\underline{d}_k,\overline{d}_k], \\[1ex] [-\infty,t_1]\cup[t_2,\infty] & \begin{cases} \text{if } \overline{\nu}_k < 0, \text{ where:} \\ t_1 = \begin{cases} -\infty & \text{if } \overline{d}_k = 0, \\ \overline{\nu}_k/\overline{d}_k & \text{otherwise} \end{cases} \\ \text{and} \\ t_2 = \begin{cases} \infty & \text{if } \underline{d}_k = 0, \\ \overline{\nu}_k/\underline{d}_k & \text{otherwise} \end{cases} \end{cases} \\[1ex] [-\infty,t_1]\cup[t_2,\infty] & \begin{cases} \text{if } \overline{\nu}_k \geq 0, \text{ where:} \\ t_1 = \begin{cases} -\infty & \text{if } \underline{d}_k = 0, \\ \underline{\nu}_k/\underline{d}_k & \text{otherwise} \end{cases} \\ \text{and} \\ t_2 = \begin{cases} \infty & \text{if } \overline{d}_k = 0, \\ \underline{\nu}_k/\overline{d}_k & \text{otherwise.} \end{cases} \end{cases} \end{cases}$$

3. the negative numerator S-preconditioner, and
4. the positive numerator S-preconditioner

with

1. the MATLAB optimization toolbox routine `linprog` to solve approximately for the LP preconditioners, and
2. the MATLAB optimization toolbox routine `quadprog` to solve approximately the quadratic program involving the penalty term (3.1), with $\rho = 1000$.

For each of these eight possibilities, we tried the 11 equally spaced values of δ: $\delta = 0$, $\delta = .1$, $\delta = .2$, …, $\delta = 1$.

EXAMPLE 1. *The simple non-singular point system:*

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 10 \end{pmatrix}, \quad b = \begin{pmatrix} -1 \\ 0 \\ 1 \end{pmatrix}, \quad \text{and} \quad x = \begin{pmatrix} [-10,10] \\ [-10,10] \\ [-10,10] \end{pmatrix}.$$

The point solution to this point system is $x = (5/3, -4/3, 0)^T$. The width-optimal preconditioner and mignitude optimal preconditioner should both give Y_k a multiple of the k-th row of the identity matrix, while the LP for the S-preconditioners should be infeasible.

The results for Example 1 appear in Table 5.1. Summarizing Table 5.1, we see that, for Example 1, the width-optimal preconditioner was reliable, regardless of δ and regardless of whether or not a quadratic penalty function was used, while the S-preconditioner LP and QP were identified as infeasible for all δ tried. (This is a satisfactory result, since no S-preconditioners exist for this problem.) The mignitude optimal preconditioner behaved similarly with or without the penalty function, but only gave a satisfactory result for $\delta = .3$, $\delta = .4$, and $\delta = .5$. Further experimentation indicated that successful mignitude optimal preconditioners could be computed only for $\delta \in [0.25, 0.5]$.

TABLE 5.1
Results for Example 1

preconditioner	linear or quadratic	Comments
width-optimal	linear	Different solutions were obtained for different δ, but in all cases, the preconditioner obtained by setting $y_i = y_i^+ - y_i^-$ was within five significant digits of $(-2/3, -4/3, 1)$, and the image of the Gauss–Seidel step was a correct enclosing interval contained in $[1.6666, 1.6667]$.
width-optimal	quadratic	The computation succeeded for all δ, all preconditioners so computed were within five significant digits of $(-2/3, -4/3, 1)$, and, as without the penalty function, the image of the Gauss–Seidel step was a correct enclosing interval contained in $[1.6666, 1.6667]$.
mignitude-optimal	linear	`linprog` indicated an unbounded solution or primal and dual infeasibility, except for $\delta = .3$, $.4$, and $.5$. For $\delta = .3$ and $\delta = .4$, the corresponding preconditioner was $(-.4, -.8, .6)$, and the Gauss–Seidel step gave results similar to the width-optimal preconditioner, while a worse preconditioner resulted for $\delta = .5$.
mignitude-optimal	quadratic	Although `quadprog` indicated success, the returned y_i^+ and y_i^- were on the order of 10^{12} (indicating possibly an unbounded LP), except for $\delta = .3$, $.4$, and $.5$. For those values of δ, the corresponding preconditioner was $Y_1 = (-.4, -.8, .6)$, and the Gauss–Seidel step gave results similar to the width-optimal preconditioner.
neg. num.-S	linear	No feasible point was found for any value of δ. However, except for $\delta = 1$, $\tilde{\boldsymbol{x}}_1$ was within rounding error of the exact point result, and $\tilde{\boldsymbol{x}}_1$, though somewhat wider, was close to that.
neg. num.-S	quadratic	No feasible point was found for any value of δ, but useable preconditioners were returned for each δ.
pos. num.-S	linear	Either no feasible point was found or both primal and dual were reported to be infeasible, for all values of δ. No useable preconditioners were returned.
pos. num.-S	quadratic	No feasible point was found for any value of δ, but the point `quadprog` returned resulted in $\tilde{\boldsymbol{x}}_1 \subseteq [-1.6380, 4.2304]$, a useful result.

EXAMPLE 2. *(The Brown's almost linear function example from [?, p. 121])*

$$\boldsymbol{A} = \begin{pmatrix} 2 & 1 & 1 & 1 & 1 \\ 1 & 2 & 1 & 1 & 1 \\ 1 & 1 & 2 & 1 & 1 \\ 1 & 1 & 1 & 2 & 1 \\ [-18.1, 17.7] & [-17.3, 16.9] & [-19.0, 18.5] & [-18.5, 19.0] & [-8.5, 17.7] \end{pmatrix},$$

$$\boldsymbol{b} = \begin{pmatrix} 5.925 \\ 5.925 \\ 5.925 \\ 5.825 \\ -1.00000015625 \end{pmatrix}, \quad \textit{and} \quad \boldsymbol{x} = \begin{pmatrix} [-2.0, 2.1] \\ [-2.1, 2.2] \\ [-1.9, 2.0] \\ [-2.0, 1.9] \\ [-2.0, 2.05] \end{pmatrix}.$$

Since the components of $\boldsymbol{x}$ are not centered about 0 in Example 2, the linear programs for the width-optimal, mignitude-optimal, and S preconditioners do not correspond exactly to solutions to the corresponding nonlinear optimization problems, but still may give reasonable results.

Results for Example 2 appear in Table 5.2. Summarizing Table 5.2, we observe

TABLE 5.2
Results for Example 2

preconditioner	linear or quadratic	Comments
width-optimal	linear	Although different values for δ gave different solutions y_i^+ and y_i^- with $y_i^+ y_i^- \neq 0$, in all cases the corresponding preconditioner Y_1 formed by $y_i = y_i^+ - y_i^-$ was equal to $(.8, -.2, -.2, -.2, 0)$ to at least 5 significant digits, with $\tilde{\boldsymbol{x}}_1 \subseteq [.7949, 1.6051]$.
width-optimal	quadratic	All values of δ gave approximately the same solution, with $Y_1 \approx (.8, -.2, -.2, -.2, 0)$, and with $\tilde{\boldsymbol{x}}_1 \subseteq [.7949, 1.6051]$.
mignitude-optimal	linear	`linprog` indicated successful termination only for $\delta = 0, .1, .2, .3, .4$, and $.5$. For $\delta = 0$ and $\delta = .4$, the resulting preconditioner was $(1, -.25, -.25, -.25, 0)$ to 4 or more significant figures, while the preconditioners were different for $\delta = 0, .1, .2, .3$, and $.5$. The Gauss–Seidel step gave $\tilde{\boldsymbol{x}}_1 \subseteq [.7949, 1.6051]$ (similar to the width-optimal preconditioner) for $\delta = 0, .1, .2, .3$, and $.4$, while $\delta = .5$ gave[10] $\tilde{\boldsymbol{x}}_1 \subseteq [.7065, 1.6198]$. Useable $\tilde{\boldsymbol{x}}_1$ were obtained for all δ.
mignitude-optimal	quadratic	`quadprog` indicated success for all values of δ. For $\delta = 0$, 0.1, 0.2, 0.3, and 0.4, the preconditioners were identical to those obtained with `linprog` without a penalty function, while, for $\delta = 0.5$, the preconditioner in this case was identical to that for $\delta = 0$ and $\delta = 0.4$. For $\delta = 0.6$, 0.7, 0.8, 0.9, and 1.0, the preconditioners had entries of magnitudes on the order of 10^{12} or greater; however, the Gauss–Seidel steps gave $\tilde{\boldsymbol{x}}_1 \subseteq [-0.0434, 4.0434]$, $\tilde{\boldsymbol{x}}_1 \subseteq [-0.0377, 3.8007]$, $\tilde{\boldsymbol{x}}_1 \subseteq [-0.0757, 4.0100]$, $\tilde{\boldsymbol{x}}_1 \subseteq [-3.5815, 14.3347]$, and $\tilde{\boldsymbol{x}}_1 \subseteq [0.0109, 1.7200]$, respectively. (This represents useful width reductions, except for $\delta = 0.9$, but inferior to the reductions for smaller δ and for the width-optimal preconditioner.
neg. num.-S	linear	`linprog` reported success in all cases, but the resulting preconditioner with $y_i = y_i^+ - y_i^-$ had entries with $\|Y_1\|_\infty < 10^{-10}$ in all cases, resulting in $0 \in \boldsymbol{\nu}_1$, $0 \in \boldsymbol{d}_1$, and $\tilde{\boldsymbol{x}}_1 \subseteq [-\infty, \infty]$.
neg. num.-S	quadratic	Results were identical to the case without the penalty function. (The penalty function was not sufficient to force $y_5^+ y_5^- \approx 0$, even when increased to $\rho = 10^6$.)
pos. num.-S	linear	`linprog` reported an unbounded problem for all values of δ except for $\delta = .8$, where it gave $Y_1 \approx (0, 0, 0, 0, 0)$. No useable preconditioners were produced.
pos. num.-S	quadratic	`quadprog` reported success for all values of δ, but $\|Y_1\|_\infty > 10^{10}$ in all cases except $\delta = .8$, in which case $\|Y_1\|_\infty < 10^{-16}$. No useable preconditioners were produced.

that the width-optimal preconditioner does well, while the mignitude-optimal preconditioner does well at reducing the interval, but not at producing splits, and the mignitude-optimal preconditioner depends on a good guess for δ. The S-preconditioners were unsuccessful at producing splits.

EXAMPLE 3.

$$\boldsymbol{A} = \begin{pmatrix} 1 & 1 \\ [-2, 0] & 1 \end{pmatrix}, \quad \boldsymbol{b} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \quad \textit{and} \quad \boldsymbol{x} = \begin{pmatrix} [0, .5] \\ [-1, 0] \end{pmatrix}.$$

Subtracting the first equation from the second, we obtain $[-1, 1]x_1 = 1$, whence $x_1 \in [-\infty, -1] \cup [1, \infty]$ and $x_2 \in [-\infty, -1] \cup [1, \infty]$, and the intersection of this set with $\boldsymbol{x}$ is the empty set.

The results appear in Table 5.3. Examining Table 5.3, we surmise that there is

TABLE 5.3
Results for Example 3

preconditioner	linear or quadratic	Comments
width-optimal	linear	`linprog` reported success for all δ, the resulting preconditioner was $Y_1 \approx (1, 0)$ for all δ, and this resulted in $\tilde{\boldsymbol{x}}_1 \subseteq [-1, 0]$, whence $\tilde{\boldsymbol{x}}_1 \cap \boldsymbol{x}_1 = [0, 0]$, for all δ.
width-optimal	quadratic	Although the y_i^+ and y_i^- differed from the linear case (without penalty function), the resulting preconditioners were virtually the same, for all δ.
mignitude-optimal	linear	`linprog` reported an unbounded problem for $\delta = 0$, 0.1, 0.2, 0.8, 0.9, and 1.0, and reported success for other values of δ. For $\delta = 0.3$, 0.4, 0.5, 0.6, and 0.7, the resulting preconditioner was $Y_1 = (-1, 1)$, resulting in $\tilde{\boldsymbol{x}} \subseteq [\infty, -1] \cup [1, \infty]$, and $\tilde{\boldsymbol{x}} \cap \boldsymbol{x} = \emptyset$. In the unbounded cases $\delta = 0$¡ 0.1, and 0.2, useable preconditioners were returned, with $\tilde{\boldsymbol{x}}_1 \subseteq [-3.0001, -0.3332]$, $\tilde{\boldsymbol{x}}_1 \subseteq [-2.5473, -0.3036]$, and $\tilde{\boldsymbol{x}}_1 \subseteq [-3.0001, -0.3333]$, respectively; a useable preconditioner was not produced for $\delta = 0.8$, 0.9, and 1.0.
mignitude-optimal	quadratic	`quadprog` reported an unbounded problem for $\delta = 0$ and $\delta = 0.1$, the number of iterations exceeded the maximum for $\delta = 1$, and `quadprog` reported success for all other values of δ. However, the resulting preconditioner was $Y_1 = (1, 1)$ only for $\delta = 0.3$, 0.4, 0.5, 0.6, and 0.7; for $\delta = 0.2$, 0.8, 0.9, preconditioners with $\|Y_1\|_\infty > 10^{10}$ were produced. Useable preconditioners, similar to those in produced without the penalty functions, were obtained for the same δ that usable preconditioners were obtained without the quadratic penalty function; in addition, a useable preconditioner was produced for $\delta = 0.8$.
neg. num.-S	linear	`linprog` reported an unbounded problem for each δ. However, interestingly, in all cases, a useable preconditioner was returned, giving $\boldsymbol{d}_1 \approx [-1, M]$, $\boldsymbol{\nu}_1 = [-N, -.5M]$ for large numbers M and N, so $\tilde{\boldsymbol{x}}_1 \approx [-\infty, -.5] \cup [N, \infty]$, so $\tilde{\boldsymbol{x}}_1 \cap \boldsymbol{x}_1 = \emptyset$.
neg. num.-S	quadratic	`quadprog` reported an unbounded problem for each δ, but, for each δ, gave useable preconditioners of the same form as without the quadratic penalty function.
pos. num.-S	linear	`linprog` reported an unbounded problem for $0 \leq \delta \leq 0.6$, and reported success for $0.7 \leq \delta \leq 1$. In all of the unbounded cases, the preconditioner produced $\tilde{\boldsymbol{x}}_1 \subseteq [-\infty, -1] \cup [0, \infty]$, not useable in this case, while $Y_1 = (-1, -1)$, $\tilde{\boldsymbol{x}}_1 \subseteq [-\infty, -1] \cup [1, \infty]$, and $\tilde{\boldsymbol{x}}_1 \cap \boldsymbol{x}_1 = \emptyset$ in the bounded cases.
pos. num.-S	quadratic	`quadprog` reported success for all δ, but gave a preconditioner $Y_1 = (-1, -1)$ only for those δ for which `linprog` reported success. For the other δ, virtually the same $\tilde{\boldsymbol{x}}_1$ was produced as when no quadratic penalty function was used.

no practical gain[11] in use of the penalty function, even though not using it results in solutions with $y_i^+ y_i^- \neq 0$. The most successful preconditioner was the negative numerator S, giving an optimal $\tilde{\boldsymbol{x}}_1 \cap \boldsymbol{x}_1 = \emptyset$ for each δ tried; the fact that unbounded problems were nonetheless produced suggests that the positive numerator S-preconditioner defining optimization problem (2.12) be modified with the additional constraint $\underline{d}_k \geq -M$ and that the corresponding problem (2.13) for the negative numerator preconditioner be modified with the additional constraint $\overline{d}_k \leq M$, for some large number M.

[11] at least for this problem

Example 4.

$$\boldsymbol{A} = \begin{pmatrix} [-1,1] & 1 & 3 \\ 4 & [-5,5] & 6 \\ [-1,1] & 8 & 10 \end{pmatrix}, \quad \boldsymbol{b} = \begin{pmatrix} 100 \\ 200 \\ 300 \end{pmatrix} \quad \textit{and} \quad \boldsymbol{x} = \begin{pmatrix} [-1,1] \\ [-1,1] \\ [-1,1] \end{pmatrix}.$$

Table 5.4
Results for Example 4

preconditioner	linear or quadratic	Comments
width-optimal	linear	`linprog` reported success for all δ. For $0 \le \delta \le 0.4$, the resulting preconditioner was $Y_1 \approx (0, 0.29412, -0.17647)$ and $\tilde{\boldsymbol{x}}_1 \subseteq [2.2174, 8.7647]$, while, for $0.5 \le \delta \le 1$, the resulting preconditioner was $Y_1 \approx (0, 0.25, 0)$ and $\tilde{\boldsymbol{x}}_1 \subseteq [47.25, 52.75]$. Thus, for all δ, $\tilde{\boldsymbol{x}}_1 \cap \boldsymbol{x}_1 = \emptyset$. Also, for all δ for this example, $y_i^+ y_i^- = 0$, $i = 1, 2, 3$.
width-optimal	quadratic	Results in this case were virtually identical to results without the quadratic penalty function.
mignitude-optimal	linear	`linprog` reported an unbounded problem for all δ. For $\delta = 0$ and $\delta = 0.1$, $\tilde{\boldsymbol{x}}_1 \subseteq [-7.5677, 0]$, and for $0.2 \le \delta \le 0.4$, $\tilde{\boldsymbol{x}}_1 \subseteq [-7.1044, 0]$, useful but not optimal. For $\delta = 0.5$, $\tilde{\boldsymbol{x}}_1 \subseteq [47.25, 52.75]$, and for $\delta = 0.6$, $\tilde{\boldsymbol{x}}_1 \subseteq [9.4159, 17.7697]$ optimal since $\tilde{\boldsymbol{x}}_1 \cap \boldsymbol{x}_1 = \emptyset$ in these cases. For $0.7 \le \delta \le 1$, the corresponding preconditioners gave $\tilde{\boldsymbol{x}}_1$ a single interval with $\inf \tilde{x}_1 > 0$, not optimal.
mignitude-optimal	quadratic	`quadprog` reported success for $0 \le \delta \le 0.8$, but reported "maximum number of iterations exceeded" for $\delta = 0.9$ and $\delta = 1$. The resulting $\tilde{\boldsymbol{x}}_1$ varied more with respect to δ than when no penalty function was used, with useable preconditioners for all δ except $\delta = 0.9$ and $\delta = 1$, and $\tilde{\boldsymbol{x}}_1 \cap \boldsymbol{x}_1 = \emptyset$ for $\delta = 0.3$, 0.4, and 0.5.
neg. num.-S	linear	`linprog` reported an unbounded problem for each δ, but produced a useable preconditioner with $\tilde{\boldsymbol{x}}_1 \approx [-M, -23.11]$ in all cases, with M large and slightly dependent on δ. Thus, $\tilde{\boldsymbol{x}}_1 \cap \boldsymbol{x}_1 = \emptyset$ for all δ.
neg. num.-S	quadratic	`quadprog` reported success for all 11 values of δ. In each case, a splitting preconditioner was produced, with $\tilde{\boldsymbol{x}}_1 \subseteq [-\infty, -28.5108] \cup [M, \infty]$, with M large and slightly dependent on δ. Thus, $\tilde{\boldsymbol{x}}_1 \cap \boldsymbol{x}_1 = \emptyset$ for all δ.
pos. num.-S	linear	`linprog` reported an unbounded problem for each δ, but, for each δ, a preconditioner was produced that gave a single interval $\tilde{\boldsymbol{x}}_1 \subseteq [M, N]$, with M and N large and dependent on δ. Thus, in all cases, $\tilde{\boldsymbol{x}}_1 \cap \boldsymbol{x}_1 = \emptyset$.
pos. num.-S	quadratic	`quadprog` reported success for all δ; for each δ, a splitting preconditioner was produced which gave $\tilde{\boldsymbol{x}}_1 \subseteq [-\infty, -M] \cup [164.625, \infty]$, for M large and somewhat dependent on δ; hence, for all δ, $\tilde{\boldsymbol{x}}_1 \cap \boldsymbol{x}_1 = \emptyset$.

For this small ($m = n = 3$) but singular problem, the techniques from [?] (and reviewed in [?]) indicate that no component of the solution set to $\boldsymbol{A}x = \boldsymbol{b}$ lies in $\boldsymbol{x}$. The preconditioner results appear in Table 5.4. For Example 4, use of the quadratic penalty function was apparently advantageous in forcing S-preconditioners, and the S-preconditioners were apparently superior to the mignitude-optimal preconditioners. However, the width-optimal preconditioners, although apparently corresponding to unbounded problems, worked just as well in this case.

EXAMPLE 5.

$$\boldsymbol{A} = \begin{pmatrix} [0,1] & [-1,0] & 0 \\ 0 & [0,1] & [-1,0] \\ -1 & 0 & 1 \end{pmatrix}, \quad \boldsymbol{b} = \begin{pmatrix} -.25 \\ -.25 \\ 0 \end{pmatrix} \quad \textit{and} \quad \boldsymbol{x} = \begin{pmatrix} [-.5,.5] \\ [-.5,.5] \\ [-.5,.5] \end{pmatrix}.$$

Using the techniques from [**?**], we compute the exact interval bounds on the portion

TABLE 5.5
Results for Example 5

preconditioner	linear or quadratic	Comments
width-optimal	linear	`linprog` reported success and gave preconditioner $Y_1 \approx (0,0,-1)$ for all δ, but $\tilde{\boldsymbol{x}}_1 = [-.5,.5]$, a useless computation.
width-optimal	quadratic	Results were virtually identical to the case without a quadratic penalty function.
mignitude-optimal	linear	`linprog` reported success for all δ, with preconditioners of the form $Y_1 \approx (0,0,a)$, with a depending on δ. For each δ, $\tilde{\boldsymbol{x}}_1 = [-.5,.5]$, not useful.
mignitude-optimal	quadratic	`quadprog` reported success for all δ except for $\delta = 0.5$; preconditioners produced were virtually identical to those without the quadratic penalty function, except for $\delta = 0.5$, in which case the preconditioner was the unusable $Y_1 \approx (0,0,0)$.
neg. num.-S	linear	`linprog` reported an unbounded problem for $\delta = 0$, and reported success for all other values of δ. In all cases, $0 \in \boldsymbol{\nu}_1$, so the result $\tilde{\boldsymbol{x}}_1 = [-\infty,\infty]$ is unusable.
neg. num.-S	quadratic	`quadprog` reported success for all 11 values of δ, but $\tilde{\boldsymbol{x}}_1 = [-\infty,\infty]$ for every δ, an unusable result.
pos. num.-S	linear	`linprog` reported an unbounded problem for each δ, and the preconditioners returned gave $0 \notin \boldsymbol{d}_1$ and $\tilde{\boldsymbol{x}}_1 = [-a,b]$, with a and b depending on δ, but $a > .5$ and $b > .5$ for each δ.
pos. num.-S	quadratic	`quadprog` reported success for all δ; for each δ, a splitting preconditioner was produced, but $0 \in \boldsymbol{\nu}_1$, so $\tilde{\boldsymbol{x}}_1 = [-\infty,\infty]$, an unusable result for each δ.

of the solution set lying in $\boldsymbol{x}$ to be

$$([0.25,0.5],[0.25,0.5],[0.25,0.5]) \cup ([-0.5,-0.25],[-0.5,-0.25],[-0.5,-0.25]),$$

whence the best possible bounds $\tilde{\boldsymbol{x}}_1$ are $\tilde{\boldsymbol{x}}_1 = [-0.5,-0.25] \cup [0.25,0.5]$. An interesting aspect of this problem is that a preconditioner without $0 \in \boldsymbol{d}_1$ cannot be effective. The results for the preconditioners appear in Table 5.5. We see that, despite the fact that the actual intersection of the solution set with $\boldsymbol{x}$ could be bounded more narrowly than by $\boldsymbol{x}$, no preconditioner gave a useable result for this example, for any value of δ.

EXAMPLE 6.

$$\boldsymbol{A} = \begin{pmatrix} [-.5,.5] & [-.25,.25] & 0 \\ 0 & [0,.25] & [-.25,0] \\ -1 & 0 & 1 \end{pmatrix},$$

$$\boldsymbol{b} = \begin{pmatrix} -.25 \\ -.25 \\ 0 \end{pmatrix} \quad \textit{and} \quad \boldsymbol{x} = \begin{pmatrix} [-.5,.5] \\ [-.5,.5] \\ [-.5,.5] \end{pmatrix}.$$

TABLE 5.6
Results for Example 6

preconditioner	linear or quadratic	Comments
width-optimal	linear	`linprog` reported success and gave preconditioner $Y_1 \approx (0, 0, -1)$ for each δ, but $\tilde{\boldsymbol{x}}_1 = [-.5, .5]$, a useless computation.
width-optimal	quadratic	`quadprog` reported success for each δ and the values returned resulted in the same preconditioners as without the quadratic penalty function.
mignitude-optimal	linear	`linprog` reported success for $0 \leq \delta \leq 0.5$, and reported an unbounded problem for the other values of δ. For $0 \leq \delta \leq .5$, the preconditioner returned had $0 \notin \boldsymbol{d}_1$, and $\tilde{\boldsymbol{x}}_1 \subseteq [0.5, a]$, where a depended on δ; this gives an optimal result $\tilde{\boldsymbol{x}}_1 \cap \boldsymbol{x}_1 = \{0.5\}$, for $0 \leq \delta \leq 0.5$. For $0.6 \leq \delta \leq 0.8$ and $\delta = 1$, $\tilde{\boldsymbol{x}}_1 = [0, 2]$, useable but not optimal, while for $\delta = 0.9$, $\tilde{\boldsymbol{x}}_1 \subseteq [-0.039, 1.885]$, also useable but not optimal.
mignitude-optimal	quadratic	`quadprog` reported success for each δ. However, for each δ except $\delta = 0.9$, the resulting preconditioner gave $\tilde{\boldsymbol{x}}_1$ of the same form and with the same usefulness as without the quadratic penalty function. For $\delta = 0.9$, $\tilde{\boldsymbol{x}}_1 \subseteq [-0.517, 0.589]$, not useful.
neg. num.-S	linear	`linprog` reported success for each δ. For each δ, $\tilde{\boldsymbol{x}}_1 = [-\infty, -0.25] \cup [0.25, \infty]$, useful but not optimal.
neg. num.-S	quadratic	`quadprog` reported success for all δ, and gave the same $\tilde{\boldsymbol{x}}_1$ as without the quadratic penalty function, despite the fact that the returned optima and the resulting preconditioners differed.
pos. num.-S	linear	`linprog` reported an unbounded problem for each δ, the preconditioners returned had $0 \notin \boldsymbol{d}_1$, and, for each δ $[-.5, .5] \subset \tilde{\boldsymbol{x}}_1$, not useful.
pos. num.-S	quadratic	`quadprog` reported success for each δ; for each δ, $\boldsymbol{d}_1 \approx [-1, 1]$, but $0 \in \boldsymbol{\nu}_1$; thus $\tilde{\boldsymbol{x}}_1 = [-\infty, \infty]$, not useful for any δ.

In Example 6, the intersection of the solution set with $\boldsymbol{x}$ is $\{(0.5, 0.5, 0.5)\}$. Preconditioner results appear in Table 5.6. We see that the width-optimal preconditioner is useless for Example 6, and the only useful preconditioners were the negative numerator S and the mignitude optimal; interestingly, the mignitude optimal preconditioner was the only preconditioner that resulted in optimally narrow bounds $\tilde{\boldsymbol{x}}_1$.

EXAMPLE 7.

$$\boldsymbol{A} = \begin{pmatrix} [-.5, .5] & [-.25, .25] & 0 \\ 0 & [0, .25] & [-.25, 0] \\ [-1, 0] & 0 & 1 \end{pmatrix},$$

$$\boldsymbol{b} = \begin{pmatrix} -.25 \\ -.25 \\ 0 \end{pmatrix} \quad \textit{and} \quad \boldsymbol{x} = \begin{pmatrix} [-.5, .5] \\ [-.5, .5] \\ [-.5, .5] \end{pmatrix}.$$

Using the techniques in [?], we see that, as in Example 6, the intersection of the solution set for Example 7 with $\boldsymbol{x}$ is the single point $\{(0.5, 0.5, 0.5)\}$, while the preconditioner results appear in Table 5.7.

For one summary of these seven examples, we assign a score to each preconditioner computation according to the number of δ for which the maximum width reduction in $\tilde{\boldsymbol{x}}_1 \cap \boldsymbol{x}_1$ versus $\boldsymbol{x}_1$ is achieved; these results appear in Table 5.8.

In Table 5.9, we list the number of times each particular preconditioner led to successful reduction of the measure of $\boldsymbol{x}$, regardless of whether or not the amount of re-

TABLE 5.7
Results for Example 7

preconditioner	linear or quadratic	Comments
width-optimal	linear	`linprog` reported (correctly) that the problem is infeasible, for all values of δ; the returned preconditioner resulted in $\tilde{\boldsymbol{x}}_1 = [-\infty, \infty]$, unusable in all cases.
width-optimal	quadratic	`quadprog` also reported infeasibility, and $\tilde{\boldsymbol{x}}_1 = [-\infty, \infty]$, as without the quadratic penalty function.
mignitude-optimal	linear	`linprog` reported success and returned preconditioners with $0 \in \boldsymbol{d}_1$, $0 \notin \boldsymbol{\nu}_1$, for each δ. Except for $\delta = 1$, $\tilde{\boldsymbol{x}}_1 = [-\infty, -m] \cup [0.5, \infty]$, with M depending on δ but large, so that $\tilde{\boldsymbol{x}}_1 \cap \boldsymbol{x}_1 = \{0.5\}$, an optimal result. For $\delta = 1$, $\tilde{\boldsymbol{x}}_1 \subseteq [-\infty, -M] \cup [0.2660, \infty]$, useful but not optimal.
mignitude-optimal	quadratic	`quadprog` reported success for each δ, and gave $\tilde{\boldsymbol{x}}_1 \cap \boldsymbol{x}_1 = \{0.5\}$, an optimal result for each δ.
neg. num.-S	linear	`linprog` reported success, and gave $\tilde{\boldsymbol{x}}_1 = [-\infty, -0.25] \cup [0.25, \infty]$, useful but not optimal, for each δ.
neg. num.-S	quadratic	`quadprog` reported success for each δ. Although the actual preconditioners differed from those obtained without the quadratic penalty function, the effect was the same, namely: $\tilde{\boldsymbol{x}}_1 = [-\infty, -0.25] \cup [0.25, \infty]$, useful but not optimal, for each δ.
pos. num.-S	linear	`linprog` reported an unbounded problem for each δ, the preconditioners returned had $0 \in \boldsymbol{d}_1$ and $0 \in \boldsymbol{\nu}_1$, and hence $\tilde{\boldsymbol{x}}_1 - [-\infty, \infty]$, not useful for any δ.
pos. num.-S	quadratic	`quadprog` reported success for each δ, but, as without the quadratic penalty function, $\tilde{\boldsymbol{x}}_1 - [-\infty, \infty]$, not useful for any δ.

TABLE 5.8
Summary of when the preconditioners gave optimally sharp results

Ex.#	width opt. lin. / quad.	mig. opt. lin. / quad.	neg. num. S lin. / quad.	pos. num. S lin. / quad.
1	11 / 11	2 / 3	10 / 0	0 / 0
2	11 / 11	5 / 5	0 / 0	0 / 0
3	0 / 0	8 / 8	11 / 11	4 / 4
4	11 / 11	2 / 3	11 / 11	11 / 11
5	0 / 0	0 / 0	0 / 0	0 / 0
6	0 / 0	6 / 6	0 / 0	0 / 0
7	0 / 0	10 / 11	0 / 0	0 / 0
Totals	33 / 33	33 / 36	22 / 22	15 / 15

duction was optimal. Finally, in Table 5.10, we list, for each preconditioner, the number of δ for which `linprog` or `quadprog` reported successful termination. Table 5.10 illustrates that successful termination depended on δ most for the mignitude optimum preconditioner, and use of the quadratic penalty function[12] forced successful termination (and, presumably, forced the corresponding preconditioner computation problem to be well-posed) in a number of cases.

One infers from these small examples that the various preconditioners are comple-

[12]that is, using `quadprog` rather than `linprog`

TABLE 5.9
Summary of when the preconditioners successfully reduced $\boldsymbol{x}$

Ex.#	width opt. lin. / quad.	mig. opt. lin. / quad.	neg. num. S lin. / quad.	pos. num. S lin. / quad.
1	11 / 11	11 / 10	11 / 11	0 / 11
2	11 / 11	11 / 10	0 / 0	0 / 0
3	11 / 8	8 / 9	11 / 11	4 / 4
4	11 / 11	11 / 9	11 / 11	11 / 11
5	0 / 0	0 / 0	0 / 0	0 / 0
6	0 / 0	11 / 10	11 / 11	0 / 0
7	0 / 0	11 / 11	11 / 11	0 / 0
Totals	44 / 41	63 / 59	55 / 55	15 / 26

TABLE 5.10
Summary of when the LP or QP solver reported successful termination

Ex.#	width opt. lin. / quad.	mig. opt. lin. / quad.	neg. num. S lin. / quad.	pos. num. S lin. / quad.
1	11 / 11	3 / 11	0 / 0	0 / 0
2	11 / 11	6 / 10	11 / 11	1 / 11
3	11 / 11	5 / 8	0 / 0	4 / 11
4	11 / 11	1 / 8	0 / 11	0 / 11
5	11 / 11	11 / 11	11 / 11	0 / 11
6	11 / 11	6 / 11	11 / 11	0 / 11
7	0 / 0	11 / 11	11 / 11	0 / 11
Totals	66 / 66	43 / 60	44 / 44	5 / 66

mentary, 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, so the difference in performance seen in the tables must be due to selection of the examples. In the tables, we see little difference in effectiveness of the resulting preconditioners between using the quadratic penalty function and not using it; however, we have observed slightly more predictability, more normal terminations in `quadprog` versus `linprog` (and presumably if other linear programming and quadratic programming solvers were used) and optima that are better scaled, when the quadratic penalty function is used. Thus, if the computational cost is similar, use of the quadratic penalty function is probably preferable.

5.1. 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[13] to x_1 for Example 1, but gives all components of $\boldsymbol{x}_1$ equal to [`NaN`, `NaN`] for Examples 2 through 7.

6. Possible Procedures for Use of These Preconditioners. There is a number of ways that use of these preconditioners can be combined to result in algo-

[13]to within rounding error

rithms that may be more effective, and yet do not suffer from exponential complexity when n is large. In particular, the following procedure will result in an optimal reduction of $\boldsymbol{x}$ in each of the examples in §5 except for Example 5, but will not result in exponential complexity with respect to m and n, if linear programming solvers or quadratic programming solvers that execute in polynomial time are used. (That is, the number of linear or quadratic programs to be solved for a particular coordinate is constant with respect to m and n.)

ALGORITHM 1. *(Gives optimal 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.

```
 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 $\delta \in [0,1]$.
 3. $\tilde{\boldsymbol{x}}_k \leftarrow \tilde{\boldsymbol{x}}_k \cap \tilde{\boldsymbol{x}}_k^{(w)}$.
 4. IF $\tilde{\boldsymbol{x}}_k = \emptyset$ THEN RETURN
 5. Compute $\tilde{\boldsymbol{x}}_k^{(ns)}$ using the negative numerator S-preconditioner[14] and a random
    $\delta \in [0,1]$.
 6. $\tilde{\boldsymbol{x}}_k \leftarrow \tilde{\boldsymbol{x}}_k \cap \tilde{\boldsymbol{x}}_k^{(ns)}$.
 7. IF $\tilde{\boldsymbol{x}}_k = \emptyset$ THEN RETURN
 8. Compute $\tilde{\boldsymbol{x}}_k^{(ps)}$ using the positive numerator S-preconditioner[15] and a random
    $\delta \in [0,1]$.
 9. $\tilde{\boldsymbol{x}}_k \leftarrow \tilde{\boldsymbol{x}}_k \cap \tilde{\boldsymbol{x}}_k^{(ps)}$.
10. IF $\tilde{\boldsymbol{x}}_k = \emptyset$ THEN RETURN
11. $h \leftarrow 1/L$.
12. DO for $i = 0$ to $L$.
    (a) Compute $\tilde{\boldsymbol{x}}_k^{(mig,i)}$ using the mignitude-optimal preconditioner with $\delta =$
        $ih$.
    (b) $\tilde{\boldsymbol{x}}_k \leftarrow \tilde{\boldsymbol{x}}_k \cap \tilde{\boldsymbol{x}}_k^{(mig,i)}$.
    (c) IF $\tilde{\boldsymbol{x}}_k = \emptyset$ THEN RETURN
    END DO
END ALGORITHM 1
```

The following algorithm will give a successful, but perhaps not optimal, reduction for each of the examples except Example 5, but will often complete more quickly than Algorithm 1.

ALGORITHM 2. *(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 2 is the same as Algorithm 1, except that each test

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

is replaced by

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

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

[15] $\tilde{\boldsymbol{x}}_k^{(\mathrm{ps})}$ possibly consists of two disjoint semi-infinite intervals

END ALGORITHM 2

Although Algorithm 2 is similar to Algorithm 1 at this level of explanation, Algorithm 2 is somewhat simpler to implement, since, in principle, two or more intervals can be produced from each stage of Algorithm 1, 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.

7. General Experiments. For a more 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 version 7.04 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 on $\boldsymbol{x}_1$. For each set of problems, we gathered statistics for each of the following four preconditioner schemes:

1. the inverse midpoint preconditioner only;
2. the width-optimal preconditioner only;
3. Algorithm 1;
4. Algorithm 2.

We used the linear formulation for the width-optimal preconditioner and the quadratic formulation for each of the other preconditioners, both when the width-optimal is used alone and within Algorithms 1 and 2. The experimental variables and statistics we gathered for each set of problems, for each of these schemes, are as follows:

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 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 less likely that Algorithm 1 will exit due to $\tilde{\boldsymbol{x}}_1 \cap \boldsymbol{x}_1 = \emptyset$ before all steps have been completed. The results appear in Table 7.1. We note that

TABLE 7.1
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.:	8	3	71	0.946	0.89
						Opt. width:	17	0	80	0.917	5.20
						Alg. 1:	29	1	99	0.775	95.97
						Alg. 2:	29	5	86	0.827	63.62
10	10	100	1	1	0	Inv. mid.:	0	0	100	1.000	1.62
						Opt. width:	0	0	100	1.000	33.44
						Alg. 1:	0	0	100	1.000	563.49
						Alg. 2:	0	0	100	1.000	561.48
10	10	100	0.1	1	0	Inv. mid.:	1	0	85	0.999	1.59
						Opt. width:	11	0	92	0.961	33.29
						Alg. 1:	15	0	100	0.942	620.72
						Alg. 2:	15	0	93	0.956	539.49
10	10	10	0.1	100	0	Inv. mid.:	0	0	100	1.000	1.63
						Opt. width:	0	0	100	1.000	32.96
						Alg. 1:	0	0	100	1.000	552.29
						Alg. 2:	0	0	100	1.000	547.78
50	50	10	0.01	10	1	Inv. mid.:	0	0	10	1.000	0.68
						Opt. width:	0	0	10	1.000	81.27
						Alg. 1:	0	0	10	1.000	3772.12
						Alg. 2:	0	0	10	1.000	3765.33

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 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 7.2. 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 7.3. Comparing Table 7.3 with Table 7.2, we see that all preconditioner schemes do a better job when there is a point right side vector, although some problems (especially with larger m and n) are still difficult.

8. 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 for such preconditioners for cases where the solution set is unbounded or contains more than one component,

[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 7.2
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	1.63
						Opt. width:	0	0	100	1.000	33.35
						Alg. 1:	0	0	100	1.000	566.42
						Alg. 2:	0	0	100	1.000	563.16
10	10	100	0.1	1	1.0	Inv. mid.:	3	0	36	0.984	1.60
						Opt. width:	16	0	38	0.931	33.22
						Alg. 1:	64	0	100	0.476	495.51
						Alg. 2:	64	0	52	0.681	318.04
10	10	100	1	1	2.0	Inv. mid.:	0	0	99	1.000	1.59
						Opt. width:	0	0	99	1.000	33.20
						Alg. 1:	1	0	100	0.998	593.34
						Alg. 2:	1	0	100	0.998	585.54
10	10	100	1	1	5.0	Inv. mid.:	2	0	25	0.984	1.60
						Opt. width:	12	0	28	0.916	33.60
						Alg. 1:	76	5	100	0.305	405.30
						Alg. 2:	76	11	52	0.478	237.36
10	10	100	0.1	1	5.0	Inv. mid.:	64	0	54	0.406	1.61
						Opt. width:	71	0	44	0.430	33.69
						Alg. 1:	100	0	100	0.000	113.19
						Alg. 2:	100	0	65	0.168	56.19
50	50	10	0.1	1	5.0	Inv. mid.:	0	0	0	1.000	0.67
						Opt. width:	0	0	0	1.000	80.42
						Alg. 1:	10	0	10	0.000	671.95
						Alg. 2:	10	0	3	0.350	332.57

TABLE 7.3
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.:	49	0	22	0.688	1.66
						Opt. width:	68	0	14	0.667	34.21
						Alg. 1:	94	0	99	0.133	347.50
						Alg. 2:	93	1	22	0.511	102.57
50	50	10	0.1	1	0.0	Inv. mid.:	0	0	10	1.000	0.67
						Opt. width:	0	0	10	1.000	79.98
						Alg. 1:	0	0	10	1.000	4849.18
						Alg. 2:	0	0	10	1.000	4856.83
50	50	10	0.01	1	0.0	Inv. mid.:	6	0	3	0.514	0.68
						Opt. width:	7	0	0	0.715	80.30
						Alg. 1:	10	0	10	0.000	475.97
						Alg. 2:	10	0	4	0.388	111.13
10	10	100	1.0	1	1.0	Inv. mid.:	0	0	100	1.000	1.60
						Opt. width:	0	0	100	1.000	33.20
						Alg. 1:	0	0	100	1.000	566.26
						Alg. 2:	0	0	100	1.000	562.78
10	10	100	0.1	1	1.0	Inv. mid.:	57	0	30	0.572	1.63
						Opt. width:	74	0	20	0.536	33.33
						Alg. 1:	98	0	100	0.033	244.84
						Alg. 2:	98	0	33	0.373	68.73
10	10	100	1.0	1	2.0	Inv. mid.:	0	0	93	1.000	1.59
						Opt. width:	2	0	93	0.998	33.48
						Alg. 1:	7	0	100	0.946	585.66
						Alg. 2:	7	0	97	0.965	564.11
10	10	100	1.0	1	5.0	Inv. mid.:	10	6	11	0.948	1.61
						Opt. width:	17	0	17	0.867	33.23
						Alg. 1:	92	3	100	0.099	299.71
						Alg. 2:	92	8	47	0.361	183.40

and have posed related penalty-function formulations that result in a quadratic program. We have used specially designed examples to illustrate the contexts in which each of these are advantageous, and we have proposed two composite polynomial time algorithms which incorporate all of these preconditioners. We have tested these algorithms with randomly generated matrices.

Our results illustrate that the inverse midpoint preconditioner is inadequate in such cases, but our optimal preconditioners sometimes perform well or adequately. In particular, our linear programming- or quadratic programming-based splitting preconditioners, previously examined only[17] [?], are worth incorporating into solution algorithms, in various contexts.

The performance of our computations may be improved, 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[18]. Furthermore, theory as observed in [?] can be used to inexpensively determine a priori non-existence of S-preconditioners, obviating the need to formulate and solve the linear or quadratic programs in such cases.

We observe that the various steps of Algorithm 1 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 afterwards.

Finally, the execution time depends heavily on the linear programming solver or quadratic programming solver used to compute the preconditioners; choices of solver, including choice of either the linear programming or quadratic programming formulation, appropriate for the problem size and structure, will undoubtedly improve performance.

[17]Splitting not involving linear programming was also examined in [?] and [?].
[18]This has been done, for example, in [?].