Preconditioners for the Interval Gauss-Seidel Method

by

R. Baker Kearfott

Department of Mathematics
U.S.L.
Lafayette, LA 70504

ABSTRACT

Interval Newton methods can form the basis of algorithms for reliably finding all real roots of a system of nonlinear equations which lie within a specified box in R^n. If X represents such a box in R^n, and $M \in X$, then a step of an interval Newton method can be viewed as solving the linear interval system

$$(1) \qquad 0 \in F(M) + F'(X)\ (\overline{X} - M)$$

for the box $\overline{X}$, where $F'(X)$ is a suitable interval extension of the Jacobian matrix.

We wish to make the coordinate widths of $\overline{X}$ as small as possible. To this end, for a wide range of methods, one multiplies (1) by a real preconditioner matrix Y so that $Y\ F'(X)$ approximates the identity matrix. A common choice of Y which has been shown to be optimal under certain conditions is the inverse of the Jacobian matrix $F'(M)$.

The interval Gauss-Seidel method is competitive for solving (1). We describe techniques for computing preconditioners for the interval Gauss-Seidel method which involve optimality conditions expressable as linear programming problems. In many instances, these preconditioners result in an $\overline{X}$ of *minimal* width. They can also be applied when F' approximates a singular matrix, and the optimality conditions can be altered to describe preconditioners with a given structure. We illustrate the technique with some simple examples and with a battery of test problems, a generalized bisection code, and an off-the-shelf linear programming problem solver. These second experiments indicate that use of these preconditioners results in significantly less function and Jacobian evaluations, especially for singular problems; however, considerable time is spent solving the linear programming problems. The method may be made more practical by adapting the linear programming solution technique to this case.

AMS Subject Classification: Primary: 65H10; Secondary: 65G10

Key Words: nonlinear algebraic systems, Newton's method, interval arithmetic, Gauss-Seidel method, global optimization, preconditioners

1. *Introduction and Motivation.*

The general problem we address is:

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

$$f_i(x_1, x_2, \ldots, x_n) = 0, \quad 1 \leq i \leq n, \tag{1.1}$$

where bounds l_i and u_i are known such that:

$$l_i \leq x_i \leq u_i \text{ for } 1 \leq i \leq n.$$

We write $F(X) = 0$, where $F = (f_1, f_2, \ldots, f_n)$ and $X = (x_1, x_2, \ldots, x_n)$. We denote the box given by the inequalities on the variables x_i by $\boldsymbol{B}$.

A successful approach to this problem is generalized bisection in conjunction with interval Newton methods, as described in [2], [10], [5], [6], [4], [12], [13], [17], etc. These methods can be thought of as finding a box $\boldsymbol{X}_k$ which contains all solutions of the interval linear system

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

where $\boldsymbol{F}'(\boldsymbol{X}_k)$ is a suitable interval extension of the Jacobian matrix of F over the box $\boldsymbol{X}_k$ (with $\boldsymbol{X}_0 = \boldsymbol{B}$), and where X_k is some point in $\boldsymbol{X}_k$. We then define the next iterate $\boldsymbol{X}_{k+1}$ by

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

This scheme is termed an *interval Newton* method.

If each row of $\boldsymbol{F}'$ contains all possible vector values that that row of the scalar Jacobian matrix takes on as X ranges over all vectors in $\boldsymbol{X}_k$, then it follows from the mean value theorem that all solutions of (1.1) in $\boldsymbol{B}$ must be in $\boldsymbol{X}_{k+1}$ for all k. If the coordinate intervals of $\boldsymbol{X}_{k+1}$ are not smaller than those of $\boldsymbol{X}_k$, then we may bisect one of these intervals to form two new boxes; we then continue the iteration with one of these boxes, and put the other one on a stack for later consideration. As is explained in [10], [5], and elsewhere, such a composite *generalized bisection* algorithm will reliably compute all solutions to (1.1) to within a specified tolerance.

The efficiency of the generalized bisection algorithm depends on

(1) the sharpness of the interval extension to the rows of the Jacobian matrix; and

(2) the way we find the solution $\overline{X}_k$ to (1.2).

In particular, iteration with formulas (1.2) and (1.3) should exhibit the quadratic local convergence properties of Newton's method, and repeated bisections are to be avoided if possible. We are thus interested in arranging the computations so that $\overline{X}_k$ has coordinate intervals which are as narrow as possible.

One method of solving (1.2) is the Krawczyk method, in which $\overline{X}_k$ is given by

$$(1.4) \qquad \overline{X}_k = K(X_k) = X_k - Y_k F(X_k) + (I - Y_k F'(X_k))(X_k - X_k).$$

where Y_k is a *preconditioner matrix* which is an approximation to $[F'(X_k)]^{-1}$. Moore observed in [9] that the Krawczyk method converged (without bisection), provided

$$|| I - Y_k F'(X_k) || < 1,$$

where the norm is the usual one for interval matrices. (See [1] or [11].) This condition, which also serves as part of a computational existence and uniqueness test, was subsequently weakened. Chen Xiaojun has shown in [19] that, for $F'(X_k)$ with certain properties, choosing Y_k to be the inverse of the midpoint matrix of F' minimizes $|| I - Y_k F'(X_k) ||$. Thus, that choice of Y_k will tend to produce an interval vector $\overline{X}_k$ whose components have relatively small widths.

An interval version of the Gauss-Seidel method, with extended interval arithmetic, is also a popular way of solving (1.2) (cf. eg. [3], or [4].). When we first multiply both sides of (1.2) by Y_k as above, we obtain a method which is usually superior to the Krawczyk method. However, for large, banded or sparse systems, multiplication by an inverse is impractical. For example Schwandt (in [15], [16], etc.) has used the Gauss-Seidel method without a preconditioner to solve instances of Poisson's equation, where diagonal dominance obviates its need. An excellent exposition and improvements of this technique appear in [18].

There are other instances where multiplication by an inverse of an n by n Jacobian matrix may not be advisable. For example, if the Jacobian matrix is singular somewhere in X_k, then Y_k may be ill-conditioned, and $Y_k F'(X_k)$ will therefore have very wide entries; this occurs, for example, for Powell's singular function, which appears as problem 3 in [6]. Alternately, one or

more of the f_i may be more highly nonlinear than the others, or may be evaluated poorly, in which case the corresponding row or rows of the Jacobian matrix would have relatively wide entries; this occurs for Brown's almost linear function, which is problem 4 in [6].

In this paper, we will develop a technique for computing preconditioners Y for the Gauss-Seidel method. This technique involves solving a linear programming problem to find each row of Y. The linear programming problems characterize certain simple optimum width-reduction conditions. In contrast to previous preconditioners (as mentioned above), the new preconditioners implicitly take account of the relative widths of the entries of $\boldsymbol{X}$, in addition to both the widths and linear dependencies in $\boldsymbol{F}'(\boldsymbol{X})$. Furthermore, computation of each row of Y can take place during the pass through the variables in a Gauss-Seidel iteration (instead of before), so that we may take full advantage of width reductions which have already taken place during that pass. Also, modifications to the constraints will allow us to compute a preconditioner Y which is optimal subject to structure or sparsity conditions.

In Section 2, we give notational preliminaries and we present the relevant underlying interval arithmetic results. We then present two types of preconditioners: In Section 3, we present the C-Preconditioner, and we present the S-preconditioner in Section 4. We prove certain optimality results for each type of preconditioner. In Section 5, we explain some examples highlighting the C-preconditioner. In Section 6, we display numerical results obtained from incorporating both the S-preconditioner and the C-preconditioner into a generalized bisection code. In Section 7, we summarize and draw conclusions.

2. Notation and Underlying Results.

We assume the reader is already familiar with interval mathematics in general, and we merely highlight some basic results here. See [1] or [11] for careful and thorough treatments.

As in the introduction, we denote vectors and matrices by upper-case letters, and scalars by lower-case letters. We denote interval vectors, matrices, and scalars by boldface. In what follows, we will speak in terms of a single step of (1.2): subscripts will denote components of vectors and matrices. In (1.2), the box $\boldsymbol{X}_k$ will simply be denoted $\boldsymbol{X} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n)$. We will understand $\boldsymbol{X}$ to represent the most current box in the case that we have completed part of a Gauss-Seidel iteration. We will assume that the point X_k is such that $X_k \in \boldsymbol{X}$, and we will denote it by $\boldsymbol{M} = (m_1, m_2, \ldots m_n)$. (Simplications in our formulas result if we take $\boldsymbol{M}$ to be the midpoint vector of $\boldsymbol{X}$; also, Lemma 2.1 and Theorem 3.6 require this to be so. However, ensuring this during an entire Gauss-Seidel sweep would require repeated evaluation of $F(\boldsymbol{M})$.) The interval derivative matrix $\boldsymbol{F}'(\boldsymbol{X}_k)$ will be denoted by $\boldsymbol{A}$, and will have element

$$\boldsymbol{A}_{i,j} = [a_{i,j}, b_{i,j}]$$

in its i-th row and j-th column. We will denote the vector $F(X_k)$ simply by F.

We will denote a hypothetical preconditioner matrix by Y, and the preconditioned matrix $Y\,\boldsymbol{A}$ will be denoted by $\boldsymbol{C}$; it will have entry

$$\boldsymbol{C}_{i,j} = [c_{i,j}, d_{i,j}]$$

in its i-th row and j-th column. We will denote the vector $Y\,F$ by $\boldsymbol{K}$, with entries $\boldsymbol{k}_i$, $1 \leq i \leq n$. We will deal with only the i-th row of the matrix Y at a time, and we will denote it Y_i; we will denote the n entries of that row by y_j, $1 \leq j \leq n$.

If $\boldsymbol{v} = [t, u]$, then $w(\boldsymbol{v}) = (u-t)$ is the *width* of $\boldsymbol{v}$. If $\boldsymbol{V}$ is an interval vector or matrix, then $w(\boldsymbol{V})$ is understood to be the vector or matrix whose elements are corresponding widths of the elements of $\boldsymbol{V}$.

Throughout, if $\boldsymbol{v}$ and $\boldsymbol{w}$ are intervals, we understand $\boldsymbol{v} > \boldsymbol{w}$ to mean that each element of $\boldsymbol{v}$ is greater than the corresponding element of $\boldsymbol{w}$.

We will identify real numbers with intervals both of

whose endpoints are equal.

With this notation, solution for the i-th component in a Gauss-Seidel iteration becomes

$$\overline{x}_i = m_i - \left[k_i - \sum_{\substack{j=1 \\ j \neq i}}^{n} G_{i,j} \, (x_j - m_j) \right] \Big/ G_{i,i} ; \tag{2.1}$$

$$x_i^{\dagger} \leftarrow x_i \cap \overline{x}_i .$$

(It is understood that, if $i > 1$, we replace x_j by $x_j^{\dagger}$ for $j < i$ when we define x_i.)

Regardless of the choice of Y, if

$$X^{\dagger} = (x_1^{\dagger}, x_2^{\dagger}, \ldots, x_n^{\dagger}),$$

and if the quotient in (2.1) is defined in extended interval arithmetic, then any $X \in \boldsymbol{X}$ with $F(X) = 0$ is also contained in $\boldsymbol{X}^{\dagger}$ (cf. Theorem 2.4 *infra*). Our goal is therefore to choose y_j, $1 \leq j \leq n$ to make $w(x_i \cap \overline{x}_i)$ as small as possible. There are two cases:

Case 1. there are elements of the i-th column of A which do not contain zero; and

Case 2. each element of the i-th column of A contains zero.

We will consider Case 1 in Section 3 with the help of the following lemmas. We will consider interval quotients of the form

$$\frac{[p,q]}{[c,d]},$$

where we think of $[p,q]$ as the numerator in the quotient in (2.1), and where we think of $[c,d]$ as the denominator in that quotient. We define

$$m_d = \min \{ |c| , |d| \} \tag{2.2}$$

We thus have

Lemma 2.1. Assume $0 \notin [c,d]$. If $0 \in [p,q]$, then

$$w\ (\ [p,q]/[c,d]\ ) = w\ ([p,q])\ /\ m_d .$$

If, in addition, $[p,q]$ is the numerator in (2.1), and each m_j is the midpoint of x_j, then

$$(2.3) \qquad w([p,q]) = \sum_{\substack{j=1 \\ j \neq i}}^{n} \left\{ \max\ (\ |c_{i,j}|\ ,\ |d_{i,j}|\ )\ w(x_j) \right\}.$$

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

In the case $0 \notin [c,d]$ and $0 \notin [p,q]$, we have the following lemma.

Lemma 2.2. Assume $[p,q]$ is the numerator in (2.1), assume $0 \notin [c,d]$, and assume $0 \notin [p,q]$. Then

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

Proof of Lemma 2.2. First, $[p,q]/[c,d] > 0$ or $[p,q]/[c,d] < 0$. Therefore, $x_i < m_i$ or $x_i > m_i$. The result follows immediately. (See fig. 1.)

In Case 2, it is not possible to choose Y_i so that $0 \notin [c,d]$. If, in addition, $0 \in [p,q]$, then x_i in (2.1) is the entire real line. However, if $0 \notin [p,q]$, then x_i in (2.1) consists of two disjoint semi-infinite intervals. We then hope to choose y_j, $1 \leq j \leq n$, to make the intersection of each of these semi-infinite intervals with x_j as small as possible. We consider this case in Section 4 with the aid of the following lemmas.

Lemma 2.3. Suppose $[p,q]$ represents the numerator in (2.1), suppose $G_{i,i} = [c,d]$, suppose $0 \in [c,d]$, and suppose $[p,q] > 0$. Then x_i consists of two disjoint intervals $x_{i,1}$ and $x_{i,2}$ such that

(i) $w\ (\ x_i \cap \overline{x}_{i,l}\ ) < \max\ \{\ (m_i - l_i),\ (u_i - m_i)\ \}$ for $l = 1,2$;

(ii) if $x_i \cap \overline{x}_{i,1}$ is non-empty, then

$$w(x_i \cap \overline{x}_{i,1}) = (m_i - l_i) - p/d;\ \text{and}$$

(iii) if $x_i \cap \overline{x}_{i,2}$ is non-empty, then

$$w(x_i \cap \overline{x}_{i,2}) = (u_i - m_i) - (p/(-c)).$$

Proof of Lemma 2.3. See fig. 2 and refer to [4] for an earlier discussion of extended interval arithmetic in this context. Elementary properties of extended interval arithmetic imply that

$$\overline{x}_i = (-\infty, m_i - \frac{p}{d}] \cup [m_i - \frac{p}{c}, +\infty).$$

Furthermore, we also have

$$x_i = [m_i - (m_i - l_i), m_i + (u_i - m_i)].$$

Thus, $x_i \cap \overline{x}_i = x^{\dagger}_{i,1} \cup x^{\dagger}_{i,2}$, where

$$x^{\dagger}_{i,1} = [m_i - (m_i - l_i), m_i - \frac{p}{d}]$$

and where

$$x^{\dagger}_{i,2} = [m_i + \frac{p}{-c}, m_i + (u_i - m_i)].$$

The conclusions of Lemma 2.3 follow directly from these characterizations.

We conclude this section with a known observation about general preconditioners.

Theorem 2.4. Let X be a box in R^n, let F: $X \rightarrow R^n$, let $M \in X$, and let A be a rowwise interval extension of the Jacobian matrix of F. Let Y be an arbitrary n by n matrix, let $G = Y A$, and let $k = Y F(M)$. Then any roots of F in X must also be in $X^{\dagger}$.

Remark 1: The set $X^{\dagger}$ is always defined when using extended interval arithmetic, since all quotients $[p,q]/[c,d]$ are defined. Note that $x_i^{\dagger}$ will consist of two disjoint intervals if $0 \in [c,d]$, $[c,d] \neq 0$, and $0 \notin [p,q]$, and that $x_i^{\dagger} = R$ if $0 \in [p,q]$ and $0 \in [c,d]$.

Remark 2: By a rowwise interval extension, we mean an interval extension such that the i-th row of A is an interval extension of gradient of the i-th component of F.

Remark 3: In particular, as long as the extended interval arithmetic is defined, the inclusion property (and hence the ability of the method to reliably find all roots) holds regardless of whether Y is singular or not.

Proof of Theorem 2.4. Suppose $X^* = (x_1^*, x_2^*, \ldots, x_n^*) \in \boldsymbol{X}$ is such that $F(X^*) = 0$. Then, since $\boldsymbol{A}$ is a rowwise interval extension of the Jacobian matrix of F, there is an $A \in \boldsymbol{A}$ such that

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

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

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

If $G_{i,i} \neq 0$, then we may solve the above formula for x_i^*; when we compare the result with (2.1), the fact that the interval arithmetic result contains all possible scalar results we can obtain by selecting elements of the intervals as operands implies $x_i^* \subseteq \overline{x}_i$. If $G_{i,i} = 0$, then the numerator in (2.1) and the denominator in (2.1) both contain zero, so $\overline{x}_i = \Omega$, and trivially $x_i^* \subseteq \overline{x}_i$. This proves Theorem 2.4.

3. *The C-Preconditioner.*

In this section, we discuss *contraction* or C-*preconditioners*, the i-th row of which are defined when not all of the entries in the i-th column of $\boldsymbol{A}$ contain zero. Our discussion will center on computation of the i-th row of the preconditioner matrix $Y_i = (y_1, y_2, \ldots, y_n)$. Let $\boldsymbol{x}_i$, $\overline{\boldsymbol{x}}_i$, $\boldsymbol{x}_i^{\dagger}$, $\boldsymbol{G}$, and k be as in (2.1), let $[p,q]$ denote the numerator in (2.1), and let $\boldsymbol{G}_{i,i} = [c,d]$. We will denote the i-th row of $\boldsymbol{G}$ by $\boldsymbol{G}_i$.

Let k be such that $0 \notin \boldsymbol{A}_{k,i}$. Then, by choosing $y_j = 0$ if $j \neq k$ and $y_k = 1$, the resulting row $\boldsymbol{G}_i = Y_i \boldsymbol{A}$ has $0 \notin \boldsymbol{G}_{i,i}$. By multiplying Y_i by the appropriate scalar, we may also arrange it so that $c = 1$. Thus, we have

Lemma 3.1. Suppose there is an index j such that $0 \notin \boldsymbol{A}_{i,j}$. Then there is a row vector Y_i such that the denominator $[c,d]$ in the resulting Gauss-Seidel iteration (2.1) is of the form $[1,d]$.

We wish to choose Y_i to minimize $w(\boldsymbol{x}_i^{\dagger}) = w(\boldsymbol{x}_i \cap \overline{\boldsymbol{x}}_i)$. In the C-preconditioners, we attempt to minimize with respect to Y_i, subject to the constraint $c = 1$, or similar constraint which forces $0 \notin [c,d]$ and which somehow normalizes $[c,d]$. The following lemma and corollary show that this is a reasonable simplification.

Lemma 3.2. Let Y_i be an arbitrary row-vector in $\boldsymbol{R}^n$. Then the interval $\overline{\boldsymbol{x}}_i$ in (2.1) defined by Y_i is the same if we replace Y_i by αY_i for any nonzero scalar α.

Corollary 3.3. Let $\Phi(\overline{\boldsymbol{x}}_i)$ be a real valued function of $\overline{\boldsymbol{x}}_i$, where $\boldsymbol{x}_i$ is as in (2.1). Then

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

provided the minima exist.

There are three strategies for producing C-preconditioners, depending upon how we assume $\overline{\boldsymbol{x}}_i$ to intersect $\boldsymbol{x}_i$. Define l_i and u_i by

$$\overline{\boldsymbol{x}}_i = [\overline{l}_i, \overline{u}_i], \text{ where}$$

$$\boldsymbol{x}_i = [l_i, u_i].$$

Then the three possibilities are illustrated in figure 3. They

are

Strategy 1. (*cf.* fig. 3(a)) Choose Y_i to maximize $\overline{l}_i$ subject to a condition on $[c,d]$.

Strategy 2. (*cf.* fig 3(b)) Choose Y_i to minimize $\overline{u}_i$ subject to a condition on $[c,d]$.

Strategy 3. (*cf.* fig 3(c)) Choose Y_i to minimize $w(x_i) = w([p,q])$ subject to $c = 1$.

If we assume *a priori* that $0 \notin [p,q]$, then the optimization problems posed in each of these strategies are solutions to moderately sized linear programming problems, as we show below. If, however, the numerator $[p,q]$ corresponding to the solution of the linear programming problem for Strategy 3 does not contain zero, then that solution is not an optimal one. However, Lemma 2.2 states that we still obtain an adequate reduction of width, provided m_i is not too near the edge of x_i.

We do not presently know how to embody all three strategies in a single simple optimization problem. Also, the conditions on $[c,d]$ in Strategy 1 and Strategy 2 are different depending on whether we assume *a priori* that $0 \in [p,q]$, $[p,q] > 0$, or $[p,q] < 0$. The following lemma, which is essentially a restatement of elementary properties of interval arithmetic, hints at why this is so.

Lemma 3.4. Assume $[c,d] > 0$.

(i) If $[p,q] > 0$, then

$$\overline{l}_i = m_i - \frac{q}{c} \quad \text{and} \quad \overline{u}_i = m_i - \frac{p}{d} .$$

(ii) If $[p,q] < 0$, then

$$\overline{l}_i = m_i - \frac{q}{d} \quad \text{and} \quad \overline{u}_i = m_i - \frac{p}{c} .$$

(iii) If $0 \in [p,q]$, then

$$\overline{l}_i = m_i - \frac{q}{c} \quad \text{and} \quad \overline{u}_i = m_i - \frac{p}{c} .$$

On a given iteration, we may compute three separate Y_i which are optimal according to Strategy 1, Strategy 2, and Strategy 3 and perform the Gauss-Seidel iteration three times.

This may result in a smaller $w(x_i^{\dagger})$ (and must result in the $x_i^{\dagger}$ of minimal width, since all cases would have been exhaustively considered), but we would incur some extra cost. This additional overhead may be worthwhile in many cases to the overall algorithmic efficiency, but adds conceptual complexity to the algorithm. The following theorem indicates that Strategy 3 would be a reasonable choice to use by itself.

Theorem 3.5. Suppose that for some j between 1 and n, $0 \notin A_{j,j}$. Then the row vector Y_i which minimizes $w(x_i)$ in (2.1) is a solution to

$$\min_{\substack{Y_i \\ c = 1}} w\left\{ \frac{[p,q]}{[c,d]} \right\} \tag{3.1}$$

Furthermore, suppose we compute a Y_i which solves the problem

$$\min_{\substack{Y_i \\ c = 1}} w\{[p,q]\}. \tag{3.2}$$

Then a Y_i which solves (3.2) is a Y_i for which $w(\overline{x}_i)$ is minimal, provided the resulting $[p,q]$ is such that $0 \in [p,q]$. If, on the other hand, Y_i solves (3.2) but the corresponding $[p,q]$ does not contain 0, then the resulting $x_i^{\dagger}$ is such that

$$w(x_i^{\dagger}) < \max \{ (m_i - l_i), \ (u_i - m_i) \}.$$

Proof of Theorem 3.5. In (2.1), $\overline{x}_i$ differs from $-[p,q]/[c,d]$ by the scalar m_i. Therefore, $w(\overline{x}_i) = w([p,q]/[c,d])$; this fact and Corollary 3.3 imply that the minimum of $w(x_i)$ with respect to Y_i occurs at a solution of (3.1).

Suppose now that Y_i is a solution to (3.2). Then, if the resulting $[p,q]$ is such that $0 \in [p,q]$, then

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

However, it follows from the definition of division of intervals that, for $c = 1$, $w([p,q]/[c,d]) \geq w([p,q])$. These last two facts imply that Y_i is also a solution to (2.1), and therefore minimizes $w(x_i)$.

The last assertion in Theorem 3.5 is a direct corollary of Lemma 2.2.

Note: Solution of (3.1) appears to be a nonlinear problem, whereas solution of (3.2) can be done via linear programming techniques, in the case that each m_i is the midpoint of the correspoonding x_i.

In the remainder of this section, we will present the method of computing Y_i based on Strategy 3 and solution of (3.2). More careful consideration of solution of (3.1), as well as consideration of methods based on Strategy 1 and Strategy 2, should and will be presented in another work.

The linear programming problem we pose to solve (3.2) will be based on a set of $3n - 1$ auxiliary variables and $2n - 1$ constraints. We base the structure on Lemma 2.1; our cost function will be the right member of (2.3). We define the auxiliary variables v_j, $1 \leq j \leq n-1$ such that the intended value of v_j is $\max\{|c_{i,j}|, |d_{i,j}|\}$ if $j < i$ and $\max\{|c_{i,j+1}|, |d_{i,j+1}|\}$ if $j \geq i$. We first note that

$$(3.3a) \quad c_{i,j} = \sum_{\substack{j=1 \\ y_t \geq 0}}^{n} y_t\, a_{t,j} + \sum_{\substack{j=1 \\ y_t < 0}}^{n} y_t\, b_{t,j}$$

and

$$(3.3b) \quad d_{i,j} = \sum_{\substack{j=1 \\ y_t \geq 0}}^{n} y_t\, b_{t,j} + \sum_{\substack{j=1 \\ y_t < 0}}^{n} y_t\, a_{t,j}.$$

Based on (3.3), we define auxiliary variables v_{n-1+t} and v_{2n-1+t}, $1 \leq j \leq n$, so that the intended value of v_{n-1+t} is $\max\{0, y_t\}$ and the intended value of v_{2n-1+t} is $\min\{0, y_t\}$. Finally, define j' by

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

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

$$\text{(3.4a)}\quad \text{minimize}\quad C(V) = \sum_{j=1}^{n-1} v_j\, w(x_{j,}) $$

subject to

$$\text{(3.4b)}\quad v_j \geq -\left[\sum_{t=1}^{n} v_{n-1+t}\; a_{t,j'} + \sum_{t=1}^{n} v_{2n-1+t}\; b_{t,j'} \right],\quad 1 \leq j \leq n-1,$$

$$\text{(3.4c)}\quad v_j \geq +\left[\sum_{t=1}^{n} v_{n-1+t}\; b_{t,j'} + \sum_{t=1}^{n} v_{2n-1+t}\; a_{t,j'} \right],\quad 1 \leq j \leq n-1,$$

$$\text{(3.4d)}\quad 1 = \left[\sum_{t=1}^{n} v_{n-1+t}\; a_{t,i} + \sum_{t=1}^{n} v_{2n-1+t}\; b_{t,i} \right],$$

and

$$\text{(3.4e)}\quad v_{n-1+j} \geq 0 \quad \text{and} \quad v_{2n-1+j} \leq 0 \quad \text{for} \quad 1 \leq j \leq n.$$

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

$$\text{(3.5)}\quad y_t = v_{n-1+t} + v_{2n-1+t}, \text{ for } 1 \leq t \leq n.$$

There are several interesting questions concerning properties of the solution to (3.4). In particular, we viewed v_{n-1+t} and v_{2n-1+t} as the positive and negative parts of y_t, respectively; the bracketed expressions in (3.4b) and (3.4c) are in general equal to the endpoints $c_{i,j}$ and $d_{i,j}$ only in the case that, for each t, at most one of v_{n-1+t} and v_{2n-1+t} is nonzero (and thus, solutions of (3.4) would correspond to solutions of (3.2) only in that case). Since there are $3n-1$ variables and (3.4b), (3.4c), and (3.4d) represent only $2n-1$ constraints, we would expect at least n of the $2n$ nonnegativity constraints (3.4e) to be binding at the solution to (3.4). In fact, in actual computational tests, we have seldom observed v_{n-1+t} and v_{2n-1+t} to be simultaneously non-zero. Also, the following theorem shows that this would not be a concern.

Theorem 3.6. Suppose that v_j, $1 \leq j \leq 3n-1$ form a solution of (3.4), and suppose each m_i is the midpoint of the corresponding x_i. If Y_i is defined by (3.5), then Y_i solves (3.2), regardless of whether or not there is a t for which $v_{n-1+t} \neq 0$ and $v_{2n-1+t} \neq 0$ simultaneously.

Proof of Theorem 3.6. We first note that all solutions Y_i of (3.2) are representable as feasible points of (3.4) in which v_{n-1+t} represents the nonnegative part of y_t and v_{2n-1+t} represents the negative part of y_t. Thus, if V solves (3.4) and for every t between 1 and n has the property that at least one of v_{n-1+t} and v_{2n-1+t} is zero, then the bracketed expressions in (3.4b) and (3.4c) represent $[c_{i,j}, d_{i,j}]$, and Lemma (2.1) implies that Y_i defined by (3.5) solves 3.2.

In the case that, in the solution of (3.4) there is a t for which $v_{n-1+t} \neq 0$ and $v_{2n-1+t} \neq 0$ simultaneously, we may still form Y_i from (3.5). The subdistributivity of interval arithmetic then implies

$$G_{i,j} = [c_{i,j}, d_{i,j}] = \sum_{t=1}^{n} (v_{n-1+t} + v_{2n-1+t})\,[a_{t,j}, b_{t,j}]$$

$$\underline{C} \left[\left[\sum_{t=1}^{n} v_{n-1+t}\, a_{t,j} + \sum_{t=1}^{n} v_{2n-1+t}\, b_{t,j} \right], \left[\sum_{k=1}^{n} v_{n-1+t}\, b_{t,j} + \sum_{k=1}^{n} v_{2n-1+t}\, a_{t,j} \right] \right]. \tag{3.6}$$

However, the left and right endpoints of the larger interval in (3.6) are simply the bracketed expressions in (3.4b) and (3.4c), respectively. Therefore, Lemma 2.1 implies that $w([p,q]) \leq C(V)$. Since $C(V)$ was a solution of (3.4) and since the set over which we are minimizing in (3.2) corresponds to a subset of the set of feasible points V of (3.4) for which $w[p,q] = C(V)$, Y_i must minimize (3.2) even in this case.

Finally in this section, we state the following corollary

of Lemma 3.1.

Theorem 3.7. The linear programming problem has a feasible point if and only if the i-th column of the interval matrix $\boldsymbol{A}$ has an entry $[a_{t,i}, b_{t,i}]$ for which $0 \notin [a_{t,i}, b_{t,i}]$.

4. The S-Preconditioner.

In this section, we discuss the *splitting* or *S-preconditioner*, which can be applied when the i-th column of the interval matrix $\boldsymbol{A}$ has an entry $[a_{t,i}, b_{t,i}]$ for which $0 \in [a_{t,i}, b_{t,i}]$.

The S-preconditioners will make use of Lemma 2.3. When $\boldsymbol{x}_i$ consists of two semi-infinite intervals, to minimize the volume of $\boldsymbol{x}_i \cap \boldsymbol{x}_i$ we need to minimize the sum of two widths. In particular, we have the following characterization, which is a direct corollary of Lemma 2.3.

Theorem 4.1. Suppose $[p,q]$ and $[c,d]$ obey the hypotheses of Lemma 2.3, set $w_i = w(\boldsymbol{x}_i)$, and define

$$v_{i,1} = \min \left\{ \frac{p}{d} , (m_i - l_i) \right\} \text{ and } v_{i,2} = \min \left\{ \frac{p}{-c} , (u_i - m_i) \right\}.$$

Then the volume of $\boldsymbol{x}_i \cap \overline{\boldsymbol{x}}_i$ is

$w_i - (v_{i,1} + v_{i,2})$, where

w_i is the width of $\boldsymbol{x}_i$. More specifically, the volume of $\boldsymbol{x}_i \cap \overline{\boldsymbol{x}}_{i,1}$ is

$(m_i - l_i) - v_{i,1}$

and the volume of $\boldsymbol{x}_i \cap \overline{\boldsymbol{x}}_{2,i}$ is

$(u_i - m_i) - v_{i,2}$.

In the case $0 \in [c,d]$, $\overline{\boldsymbol{x}}_i = \boldsymbol{R}$ unless we also assume $0 \notin [p,q]$. Under this assumption, we may also assume $p = 1$, and we have analogues of Lemma 3.2 and Corollary 3.3, with p replacing c. However, even with this normalization, it does not appear that minimization of $w_i - (v_{i,1} + v_{i,2})$ can be posed as a single linear problem. However, we can probably pose maximization of either $v_{i,1}$ or $v_{i,2}$ as a single linear programming problem. We may obtain an optimum reduction of width by computing $\boldsymbol{x}_i^{\dagger}$ first from the preconditioner obtained by minimizing $v_{i,1}$; we then replace $\boldsymbol{x}_i$ by $\boldsymbol{x}_i^{\dagger}$, then compute a new $\boldsymbol{x}_i^{\dagger}$ from the preconditioner obtained by maximizing $v_{i,2}$. (Note: in some cases the width may be reduced more by maximizing $v_{i,2}$ first.)

In the remainder of this section, we will propose two methods of computing Y_i which are intended to give Y_i's which

maximize $v_{i,1}$ and $v_{i,2}$, respectively. The methods are based on Theorem 4.1 and Lemma 3.2.

Method 1. To hopefully obtain Y_i for which $w(x_i \cap \overline{x}_{i,1})$ is minimum, we solve

$$\max_{Y_i} \{ p \}. \quad d = 1 \tag{4.1}$$

Method 2. To hopefully obtain the Y_i for which $w(x_i \cap \overline{x}_{i,2})$ is minimum, we solve

$$\max_{Y_i} \{ p \}. \quad c = -1 \tag{4.2}$$

Remark 1: We do not require $c \leq 0$ in (4.1) because, if $c > 0$ then Method 1 will hopefully lead to a preconditioner for which u_i is minimized according to Strategy 2 of Section 3 (*cf.* Lemma 3.4). Similarly, we do not require $d \geq 0$ in (4.2) because $d < 0$ will hopefully lead to a preconditioner for which l_i is maximized.

Remark 2: These conditions do not allow $[c,d] = 0$. If this condition occurs simultaneously with $0 \notin [p,q]$, then we may conclude there are no roots in $\boldsymbol{X}$. However, this special case is probably most easily checked separately.

Just as in Section 3, we introduce $3n-1$ auxiliary variables to pose the linear programming problem. The variables v_{n-1+j} and v_{2n-1+j} will denote the positive and negative parts of y_j, $1 \leq j \leq n$, just as in Section 3. However, we define the auxiliary variables v_j, $1 \leq j \leq n-1$ somewhat differently. As in Section 3, we define

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

and define l_i and u_i by

$$x_i = [l_i, u_i].$$

The intended value of v_j is then

$$v_j \simeq \max \{ -c_{i,j'}(l_{j'} - m_{j'}), \; -c_{i,j'}(u_{j'} - m_{j'}), \; -d_{i,j'}(l_{j'} - m_{j'}), \; -d_{i,j'}(u_{j'} - m_{j'}) \}.$$

The linear programming problem corresponding to (4.1) can thus be posed as

$$(4.3a)\quad \text{maximize } C(V) = \sum_{k=1}^{n} (v_{n-1+k} + v_{2n-1+k})\, f_k(M) - \sum_{j=1}^{n-1} v_j$$

subject to

$$(4.3b)\quad \left\{ \begin{array}{lll} v_j \geq 0 & \text{for} & 1 \leq j \leq n-1 \\ v_{n-1+j} \geq 0 & \text{for} & 1 \leq j \leq n \\ v_{2n-1+j} \leq 0 & \text{for} & 1 \leq j \leq n \end{array} \right\},$$

$$(4.3c)\quad v_j \geq -\left[\sum_{k=1}^{n} v_{n-1+k}\, a_{k,j'} + \sum_{k=1}^{n} v_{2n-1+k}\, b_{k,j'} \right] (l_{j'} - m_{j'})$$

for $1 \leq j \leq n-1$,

$$(4.3d)\quad v_j \geq -\left[\sum_{k=1}^{n} v_{n-1+k}\, a_{k,j'} + \sum_{k=1}^{n} v_{2n-1+k}\, b_{k,j'} \right] (u_{j'} - m_{j'})$$

for $1 \leq j \leq n-1$,

$$(4.3e)\quad v_j \geq -\left[\sum_{k=1}^{n} v_{n-1+k}\ b_{k,j'} + \sum_{k=1}^{n} v_{2n-1+k}\ a_{k,j'}\right](l_{j'} - m_{j'})$$

for $1 \leq j \leq n-1$,

$$(4.3f)\quad v_j \geq -\left[\sum_{k=1}^{n} v_{n-1+k}\ b_{k,j'} + \sum_{k=1}^{n} v_{2n-1+k}\ a_{k,j'}\right](u_{j'} - m_{j'})$$

for $1 \leq j \leq n-1$,

and

$$(4.3g)\quad \left[\sum_{k=1}^{n} v_{n-1+k}\ b_{k,i} + \sum_{k=1}^{n} v_{2n-1+k}\ a_{k,i}\right] = 1.$$

Similarly, the linear programming problem corresponding to (4.2) can be posed as

(4.4a)- maximize $C(V)$ as in (4.3a), subject to (4.3b), (4.3c),
(4.4f) (4.3e), (4.3f), and

$$(4.4g)\quad \left[\sum_{k=1}^{n} v_{n-1+k}\ a_{k,i} + \sum_{k=1}^{n} v_{2n-1+k}\ b_{k,i}\right] = -1.$$

A thorough analysis of these methods will occur elsewhere.

5. *Some Examples of the C-Preconditioner.*

In this section, we compare the C-preconditioner defined in (3.4) with the preconditioner defined as the inverse of the midpoint matrix of A; we will refer to the latter as the inverse preconditioner. We denote the C-preconditioner by Y^C and the inverse preconditioner by Y^I; we denote the corresponding interval matrices G by G^C and G^I and the corresponding widths $w([p,q]/[c,d])$ in the i-th variable by $w_i{}^C$ and $w_i{}^I$. To clearly see the possibilities, we use the following six simple cases.

When giving widths, we assume $0 \in [p,q]$ so that Lemma 2.1 holds. (The validity of this assumption depends on the values of F, which we leave unspecified here.)

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

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

The results of applying the inverse preconditioner and (3.4) to problem 1 were identical; we obtained

$$Y^C \simeq Y^I \simeq \begin{bmatrix} -2 & 1 \\ 1.5 & -0.5 \end{bmatrix}, \quad G^C \simeq G^I \simeq I, \text{ and } w_i^C \simeq w_i^I \simeq 0 \text{ for } i = 1,2.$$

to within roundoff error.

Problem 2. A matrix for which the inverse preconditioner leads to $x_i = R$.

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

The inverse preconditioner gives

$$Y^I \simeq \begin{bmatrix} -2.5 & 1.5 \\ 2 & -1 \end{bmatrix}; \quad G^I \simeq \begin{bmatrix} [-3,5] & [-4,4] \\ [-3,3] & [-2,4] \end{bmatrix}.$$

Since $0 \in G^I{}_{i,j}$ for all i and j, $w_1{}^I = w_2{}^I = \infty$. For the solution of (3.4), we obtain

$$Y^C \simeq \begin{bmatrix} 0 & .333 \\ 0 & .25 \end{bmatrix}; \quad G^C \simeq \begin{bmatrix} [1,1.67] & [1.33,2] \\ [.75,1.25] & [1,1.5] \end{bmatrix}.$$

Thus, $w_1{}^C = 2$, but $w_2{}^C \simeq 1.25$. Therefore, depending on $F(M)$, a Gauss-Seidel step with G^C may be worthwhile, whereas a step with the inverse preconditioner has no chance of reducing the widths.

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

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

The inverse preconditioner gives

$$Y^I \simeq \begin{bmatrix} -2.5 & 1.5 \\ 2.0 & -1.0 \end{bmatrix}; \quad G^I \simeq \begin{bmatrix} [.2,1.8] & [-4,4] \\ [-.6,.6] & [-2,4] \end{bmatrix}.$$

$w_1{}^I = 20$ and $w_2{}^I = \infty$. For the preconditioner based on (3.4) we obtain

$$Y^C \simeq \begin{bmatrix} 0 & .263 \\ 0 & .25 \end{bmatrix}; \quad G^C \simeq \begin{bmatrix} [1,1.11] & [1.05,1.58] \\ [.95,1.05] & [1,1.5] \end{bmatrix}.$$

Thus, $w_1{}^C = 1.58 < 20$ and $w_2{}^C \simeq 1.05 < \infty$; hence, depending on $F(M)$, a Gauss-Seidel step with G^C will be more worthwhile in reducing the width of the either coordinate than a Gauss-Seidel step with G^I.

Problem 4. A simple three-dimensional problem for which Y^I gives reasonable results.

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

The inverse preconditioner gives

$$Y^I \simeq \begin{bmatrix} .109 & -.014 & -.009 \\ -.019 & .055 & -.007 \\ -.011 & -.007 & .036 \end{bmatrix}; \quad \text{and}$$

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

We thus have $w_1{}^I \simeq .298$, $w_2{}^I \simeq .122$, and $w_3{}^I \simeq .079$. The preconditioner based on (3.4) gives

$$Y^C \simeq \begin{bmatrix} .123 & -.016 & -.010 \\ -.021 & .060 & -.008 \\ -.012 & -.008 & .038 \end{bmatrix}; \quad \text{and}$$

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

Thus, $w_1{}^C \simeq .298 \simeq w_1{}^I$, $w_2{}^C \simeq .122 \simeq w_2{}^I$, and $w_3{}^C \simeq .079 \simeq w_3{}^I$. Hence, depending on $F(M)$, we would expect a Gauss-Seidel step with G^C to accomplish about the same width reduction as a Gauss-Seidel step with G^I.

Problem 5. The same as problem 4 except the widths of the components of X *are not all equal.*

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

We get the same Y^I and G^I for the inverse preconditioner as in problem 4, but the widths $w_i{}^I$ are different, and are listed below. For the C-preconditioner, we obtain

$$Y^C \simeq \begin{bmatrix} .119 & -.018 & 0 \\ -.021 & .060 & -.008 \\ 0 & -.009 & .036 \end{bmatrix}; \quad \text{and}$$

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

Thus, $w_1{}^C \simeq .279 < w_1{}^I \simeq .299$, $w_2{}^C \simeq .0429 \simeq w_2{}^I$, and $w_3{}^C \simeq 1.03 < 1.17 \simeq w_3{}^I$. Hence, depending on $F(M)$, we would expect a Gauss-Seidel step with G^C to be better in the first and third coordinates than a Gauss-Seidel step with G^I. This problem illustrates that the C-preconditioner can take advantage of width

Problem 6. Initial X and Jacobian matrix for Problem 4 from [6] *(Brown's almost linear function).*

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

and $X = ([-2,2], [-2,2], [-2,2], [-2,2], [-2,2])^T$.

The problem from which this example came is difficult for interval Newton methods which are not able to separate the highly nonlinear behavior of the last function component from the linearity of the remaning n-1 function components. The matrices Y^I and G^I in this example are not even defined, since the midpoint matrix is singular. For the C-preconditioner, we obtain

$$Y^C \simeq \begin{bmatrix} .8 & -.2 & -.2 & -.2 & 0 \\ -.2 & .8 & -.2 & -.2 & 0 \\ -.2 & -.2 & .8 & -.2 & 0 \\ -.2 & -.2 & -.2 & .8 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix};$$

$$G^C \simeq \begin{bmatrix} [1,1] & [0,0] & [0,0] & [0,0] & [.2,.2] \\ [0,0] & [1,1] & [0,0] & [0,0] & [.2,.2] \\ [0,0] & [0,0] & [1,1] & [0,0] & [.2,.2] \\ [0,0] & [0,0] & [0,0] & [1,1] & [.2,.2] \\ [2,2] & [1,1] & [1,1] & [1,1] & [1,1] \end{bmatrix};$$

and $w_1{}^C \simeq w_2{}^C \simeq w_3{}^C \simeq w_4{}^C \simeq .8$, while $w_5{}^C \simeq 20$. We note that we can expect to make progress in the i-th coordinate in the Gauss-Seidel process if $w_i{}^C < 1$. This example thus shows that the C-preconditioner is of great potential value for such problems.

6. Numerical Results from an Actual Interval Newton / Bisection Code.

Here, we report results obtained by incorporating the C-preconditioner and the two S-preconditioners mentioned above into a generalized bisection / interval Newton code. The code is similar to that described in [6], with a few differences, to be described in [7]. These differences include: (i) use of simulated directed roundings for true interval arithmetic; (ii) use of the Gauss/Seidel method instead of the Krawczyk method; (iii) use of interval arithmetic to evaluate $F(M)$; (iv) use of 'volume change' to determine when to stop iteration of the Gauss-Seidel method; and (v) use of a special technique to choose coordinate directions in which to bisect, which is based on "maximal smear" in $F'(X)(X-M)$, where $F'(X)$ denotes the interval extension of the Jacobian matrix. Items (i) and (iii) are necessary for total reliability, while items (ii), (iv), and (v) are related to efficiency.

We used the IMSL routine DDLPRS to solve the linear programming problems associated with the new preconditioners.

We will compare the code which includes the new preconditioners to the code which includes only the inverse preconditioner.

The problem set is that in [6]. We do not include problems 5, 6, 7, 8, and 13, since these are linear and hence trivial for interval Newton methods.

The preconditioners are applied row by row. For each row, we perform the following steps.

Algorithm 6.1. (Applied to the i-th row, $1 \leq i \leq n$) The following steps are done first for the C-preconditioner, then for the first S-preconditioner, then for the second S-preconditioner

1. If $M \notin X$, then recompute $F(M)$.

2. Attempt to compute the preconditioner.

3. If successful (that is, if (3.4), (4.3), or (4.4), respectively, has a solution), then compute k_i and $G_{i,*}$ using it.

4. If not successful, but the inverse preconditioner is defined, then compute k_i and $G_{i,*}$ using the inverse preconditioner.

5. Otherwise, use $k_i = F(M)$ and $G = F'(X)$.

6. Apply (2.1).

In Table 1, we attempt to compare the amount of work for the method with just the inverse preconditioner to the amount of work for the method which includes Algorithm 6.1. The first column gives the problem number from [6], while the second column gives the number of variables. The column labeled NFUN gives the number of evaluations of $F(X)$ (i.e. the number of interval function evaluations), the column labeled NSCALF gives the number of evaluations of $F(M)$ (i.e. the number of scalar function evaluations), and the column labeled NJAC gives the number of evaluations of the interval Jacobian matrix $F'(X)$.

The total work is estimated to be

(6.1) WORK $\simeq$ NFUN + NSCALF + n * NJAC.

We ran the code on an IBM 3090 after having compiled it with the VS-Fortran compiler. The column labelled "Tot. CPU" gives the total amount of CPU time, excluding input and output. The column labelled "CPU in LP" is the total amount of CPU time spent solving (3.4), (4.3), and (4.4) via DDLPRS, while the column labelled "CPU TOT/LP" gives the percentage of total CPU time spent solving these problems. We observe solving (3.4), (4.3), and (4.4) is the predominant computation when the C- and S-preconditioners are used. Future algorithms may include more sophisticated techniques for weighing this expense against possibly fewer function and Jacobian evaluations. We may also be able to implement a linear programming solver which is specifically more efficient for (3.4), (4.3), and (4.4).

The last column represents the ratio of estimated total work per unit of CPU time. If the numbers in this column, for a given method, are constant over all problems, then the estimated total work is a good estimate. We see that this is only very roughly so.

The method with the new preconditioners performed better on all problems except problem 11. It performed impressively better on problems 3 and 4.

The last few rows of the table compare the code with just the inverse preconditioner with the code which employs Algorithm 6.1. The two rows labelled "TOT" give the totals for each column, while the row labelled "Ratio" gives the ratios of these totals. We see that, with regard to numbers of Jacobian evaluations and estimated total work, the code with the linear

programming preconditioners was approximately six times better than the code using just the inverse preconditioner. In fact, with regard to these measures, the code with the new preconditioners performed better on all problems except problem 11, and performed impressively better on problems 3 and 4.

We do not presently know precisely why the new preconditioners performed worse on problem 11. However, this problem is special in that it has independent subsystems of equations and variables. Iteration of an interval Newton method on this problem typically results in widths in certain coordinate directions which are almost zero, while widths in other coordinate directions are still large. We suspect that the problems on this problem are due to effects of the machine arithmetic. They may possibly be ameliorated via a technique for analyzing subsystems such as that in [19]. Also, a more sophisticated way of determining when to stop the interval Newton method may help.

Problem 10 is difficult perhaps because of the large differences in scales of the variables and equations. The new preconditioners implicitly take account of these differences by including $(X-M)$ in the computations.

Problem 3 is a variant of Powell's singular function, which is doubly singular at the solution. Hence, near the solution, we could expect the inverse preconditioner to lead to very large widths. (See [8].) The new preconditioners are based on minimizing widths.

Problem 4 is Brown's almost linear function. As explained in Section 5, the new preconditioners result in linear combinations which do not include the excessively wide entries in the last row of the Jacobian matrix.

Since we apply all three new preconditioners, and possibly also apply rows of the inverse preconditioner, we need to present more data to illustrate where the gains in efficiency are. We do this in Table 2. The column labeled NROTOT gives the total number of rows to which Gauss/Seidel was applied to each preconditioner. (This is the total number of times Algorithm 6.1 was applied to each preconditioner.) The column labeled %C gives the percentage of these times that the C-preconditioner gave a reduction in width, the column labeled %S1 gives the percentage of the time that the first S-preconditioner gave a reduction in width, and the column labeled %S2 gives the percentage of the time that the second S-preconditioner gave a reduction in width. The column labeled %GEC gives the percentage of the time that the C-preconditioner could not be computed, but the inverse

preconditioner gave a reduction in width. The column labeled %GES gives the percentage of the time that one of the S-preconditioners could not be computed, but the inverse preconditioner gave a reduction in width.

The results in Table 2 seem to indicate that the increased performance with regard to numbers of Jacobian evaluations is due to the preconditioners, as opposed to the fact that we iterate three times in each coordinate before going on to the next. The worth of the two S-preconditioners may be underestimated because they were applied after the C-preconditioner was. We note that there are very few cases when the inverse preconditioner is successful but the new preconditioners are not.

Our final table in this section gives the number of "splits" which resulted from each type of preconditioner, i.e. the number of times x_i consisted of two disjoint intervals. The S-preconditioners are meant to cause such splits, which are desirable when the initial X contains more than one root. (Such is the case with problems 1, 4, 9, 11, and 12.) The first column gives the problem number in [6], the column labeled "C" gives the number of splits due to the C-preconditioner, the column labeled "S1" gives the number of splits due to the first S-preconditioner, and the column labeled "S2" gives the number of splits due to the second S-preconditioner. Splits could also occur when one of the preconditioners is not successfully computed and a row of the inverse preconditioner is used. The labeled IPC gives the number of splits when the inverse preconditioner was used instead of the C-preconditioner; the column labeled IPS1 gives the number of splits when the inverse preconditioner was used instead of the first S-preconditioner, and the column labeled IPS2 gives the number of splits when the inverse preconditioner was used instead of the second S-preconditioner.

The table indicates that these preconditioners do indeed result in splits, and that this mechanism probably adds significantly to the efficiency. However even if x_i contains two semi-infinite components, $x_i^{\dagger}$ may only contain one component, and it may happen that $w(x_i^{\dagger}) = w(x_i)$. Even so, we have reason to believe that the S-preconditioners are effective. The process appears to be a major one in problem 3, for which there is a single singular root, and in problem 12, where there are 12 separate roots. It is not as effective as it should be in problem 11; possible reasons are cited above.

Since the constraints defining the C-preconditioner preclude any splits when it is used, the second column of Table 3

merely corroborates the fact that the C-preconditioner is being computed properly. (There were, however, possibly some splits from the inverse preconditioner when the C-preconditioner was not defined.)

7. *Summary, Conclusions, and Future Work*

We have described examples of two types of preconditioners for the interval Gauss-Seidel method. These preconditioners are meant to optimize quantities specifically associated with the interval Gauss-Seidel method. In certain cases, they significantly decrease the total number of required Jacobian evaluations over preconditioners based on the inverse of the midpoint matrix of the interval Jacobian matrix, at the expense of solving large numbers of small linear programming problems. This includes cases where there is a singularity near the root or where there is a component function which is more highly nonlinear than the others.

Computation of a row of one of these preconditioners requires solution of a linear programming problem with $3n-1$ variables, where n is the order of the system. These linear programming problems have a special structure which can perhaps be exploited. Additional study will enable us to develop criteria for deciding when to use each of the preconditioners.

Finally, we note that we may compute a sparse preconditioner simply by posing the linear programming problems in low-dimensional subspaces. Also, we may force G to have a number of properties by imposing additional constraints. These prospects are exciting, and will be investigated.

Bibliography

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

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

3. Hansen, E. R., and Greenberg, R. I. An Interval Newton Method, *Appl. Math. Comput. 12* (1983), 89-98.

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

5. Kearfott, R. B. Abstract generalized bisection and a cost bound, *Math. Comput. 49*, 179 (July, 1987), 187-202.

6. Kearfott R. B. Some tests of generalized bisection, *ACM Trans. Math. Software 13*, 3 (Sept., 1987), 197-220.

7. Kearfott, R. B., and Novoa, M. A program for generalized bisection, manuscript.

8. Kearfott, R. B. On handling singular systems with interval Newton methods, in the proceedings of the Twelfth IMACS World Congress on Scientific Computation, 1988.

9. Moore, R. E. A test for existence of solutions to nonlinear systems, *SIAM J. Numer. Anal. 14*, 4 (Sept. 1977), 611-615.

10. Moore, R. E., and Jones, S. T. Safe starting regions for iterative methods, *SIAM J. Numer. Anal. 14*, 6 (Dec. 1977), 1051-1065.

11. Moore, R. E. *Methods and Applications of Interval Analysis*, SIAM, Philadelphia, 1979.

12. Neumaier, A. Interval iteration for zeros of systems of equations, *BIT 25*, 1 (1985), 256-273.

13. Nickel, K. On the Newton method in interval analysis, Technical Summary Report no. 1136, Mathematics Research Center, University of Wisconsin at Madison, 1971.

14. Ojika, T. Structure analysis for large scale nonlinear equations, *Memoirs of Osaka Kyoiku University, Ser. III*

32, 1 (1983), 63-72.

15. Schwandt, H. An interval arithmetic approach for the construction of an almost globally convergent method for the solution of the nonlinear Poisson equation, *SIAM J. Sci. Statist. Comput.* **5**, 2 (June, 1984), 427-452.

16. Schwandt, H. The solution of nonlinear elliptic Dirichlet problems on rectangles by almost globally convergent interval methods, *SIAM J. Sci. Statist. Comput.* **6**, 3 (July, 1985), 617-638.

17. Shearer, J. M., and Wolfe, M. A. Some computable existence, uniqueness, and convergence tests for nonlinear systems, *SIAM J. Numer. Anal. 22*, 6 (Dec., 1985), 1200-1207.

18. Thiel, S. Intervalliterationsverfahren für discretisierte elliptische Differentialgleichungen, preprint, *Freiburger Intervall-Berichte* 86/8, Institut für Angewandte Mathematik der Universität Freiburg, 1-72.

19. Xiaojun, C., and Deren, W. On the optimal properties of the Krawczyk-type interval operator, preprint, *Freiburger Intervall-Berichte* 87/5, Institut für Angewandte Mathematik der Universität Freiburg, 1-15.

(Interval preconditioners -- Kearfott) (Bibliography)

#	n	meth.	NFUN	NSCALF	NJAC	est. total work	Tot. CPU	CPU in LP	CPU TOT/ LP	work/ CPU
1	2	Old	22	21	21	85	0.03			2833.33
		New Pre.	14	42	13	82	1.49	1.45	97%	54.90
2	2	Old	68	50	50	218	0.14			1557.14
		New Pre.	23	52	23	121	1.00	0.89	89%	121.18
3	4	Old	1963	1276	1301	8443	11.23			751.83
		New Pre.	258	619	200	1677	251.11	247.95	99%	6.68
4	5	Old	8628	6420	6440	47248	125.09			377.71
		New Pre.	60	98	55	433	63.97	63.08	99%	6.77
9	2	Old	31	12	28	115	0.02			4791.67
		New Pre.	25	7	25	122	0.31	0.28	91%	394.41
10	4	Old	145	116	116	725	1.41			514.18
		New Pre.	62	156	56	442	32.96	32.33	98%	13.41
11	8	Old	457	276	366	3661	9.44			387.82
		New Pre.	668	1286	521	6122	1130.93	1090.33	96%	5.41
12	3	Old	732	589	590	3091	5.21			593.28
		New Pre.	416	874	375	2415	84.15	79.28	94%	28.70
14	2	Old	31	28	28	115	0.05			2346.94
		New Pre.	25	47	25	122	0.69	0.62	91%	178.06
15	2	Old	2	2	2	8	.00			3200.00
		New Pre.	1	3	1	6	1.72	1.71	100%	3.50
16	4	Old	2	2	2	12	0.01			1348.31
		New Pre.	2	9	2	19	1.20	1.18	98%	15.85
17	5	Old	108	79	80	587	1.56			376.28
		New Pre.	29	45	27	209	30.59	30.06	98%	6.83
TOT.		Old	12189	8871	9024	64308	154.19			417.06
		New Pre.	1583	3238	1323	11770	1600.10	1549.16	97%	7.36
Ratio:										
		Old/New:	7.70	2.74	6.82	5.46	0.10			56.70

Table 1. Comparison of the efficiencies of the composite method to the method with just the inverse preconditioner.

#	n	NRTOT	%C	%S1	%S2	%GEC	%GES
1	2	25	72%	68%	100%	8%	24%
2	2	42	62%	38%	31%	0%	7%
3	4	739	46%	39%	17%	0%	9%
4	5	234	56%	18%	12%	0%	6%
9	2	13	77%	38%	23%	8%	15%
10	4	210	43%	25%	8%	2%	15%
11	8	4115	40%	0%	0%	2%	0%
12	3	1071	71%	34%	27%	1%	5%
14	2	33	64%	18%	21%	6%	30%
15	2	2	100%	50%	0%	0%	0%
16	4	8	100%	0%	0%	0%	0%
17	5	121	41%	26%	11%	0%	0%
Tot.:		6492	47%	12%	8%	2%	3%

Table 2. Comparison of the relative efficiencies of the inverse, C-, and S-preconditioners in the composite method.

#	C	S1	S2	IPC	IPS1	IPS2
1	0	2	2	2	0	0
2	0	3	1	0	2	2
3	0	134	56	0	4	33
4	0	2	0	0	0	1
9	0	2	2	1	0	0
10	0	14	6	0	2	4
11	0	0	0	79	0	0
12	0	37	24	12	9	8
14	0	5	5	0	0	0
15	0	0	0	0	0	0
16	0	0	0	0	0	0
17	0	0	1	0	0	0
Tot.	0	199	96	94	17	48

Table 3. Number of "splits" produced by each type of preconditioner.

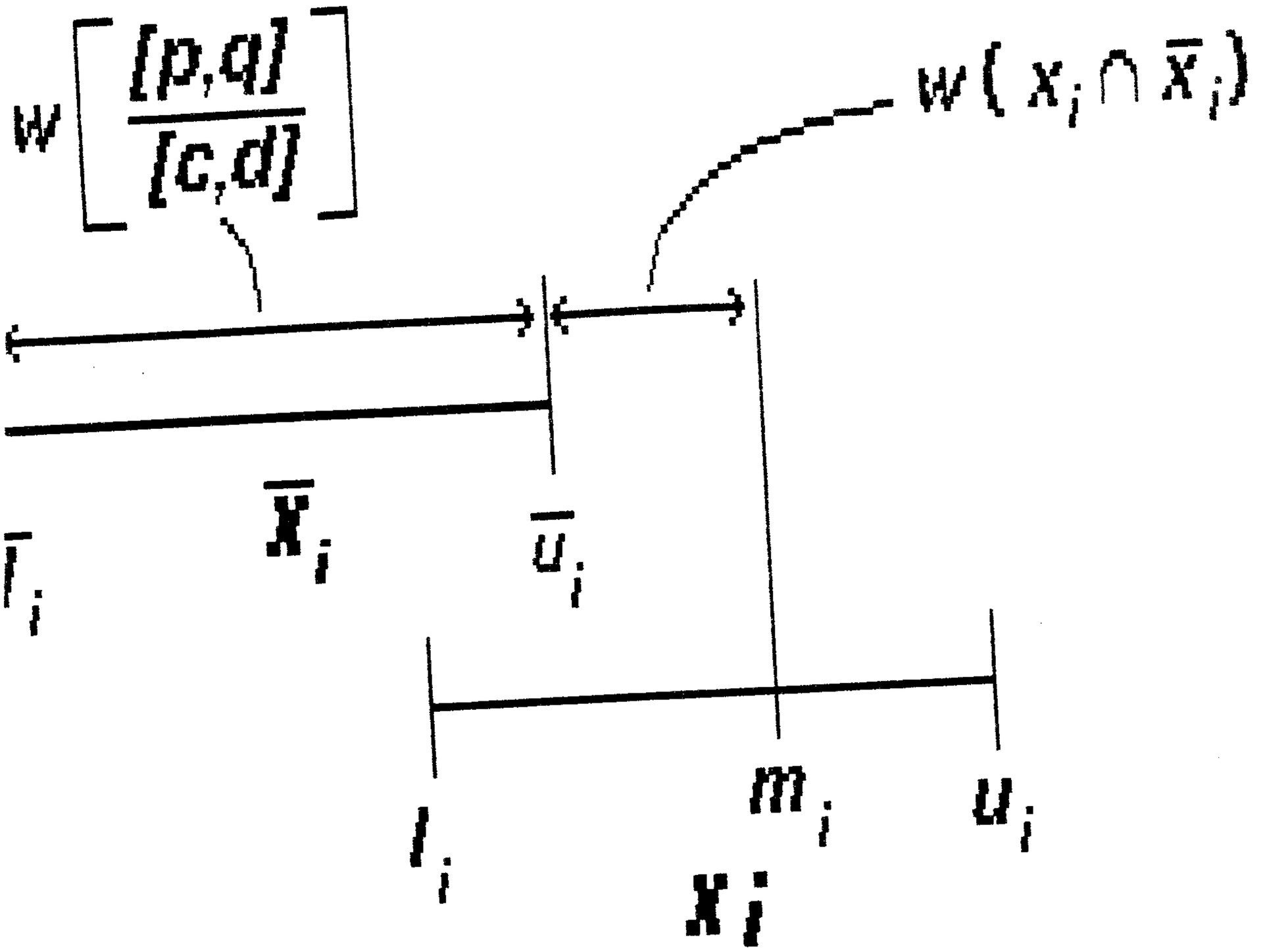

Figure 1a . In this case , [p,q] / [c,d] > 0, so every element of m_i — [p,q]/[c,d] is less than m_i .

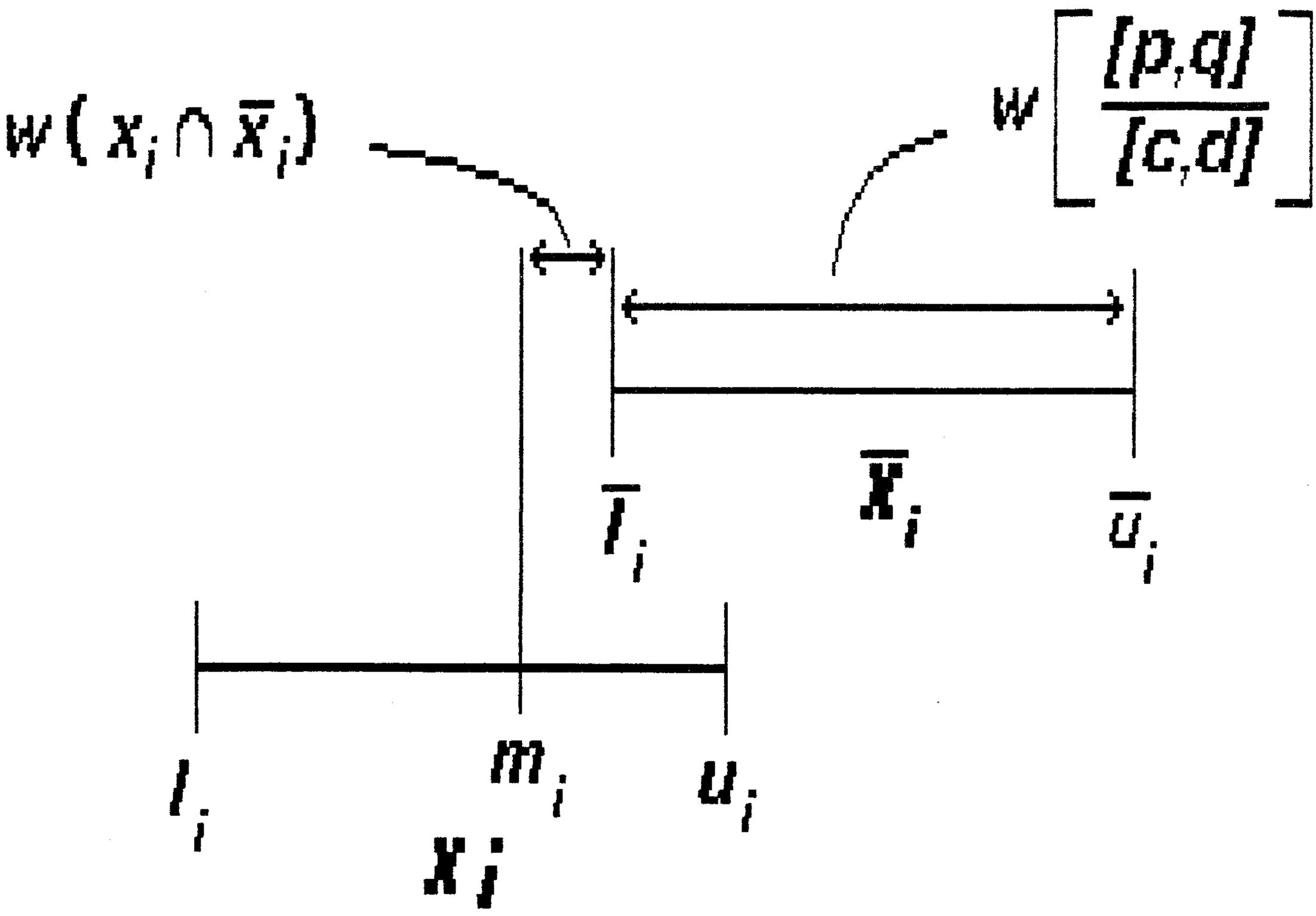

Figure 1 b. In this case , [p,q] / [c,d] < 0, so every element of m_i — [p,q]/[c,d] is greater than m_i .

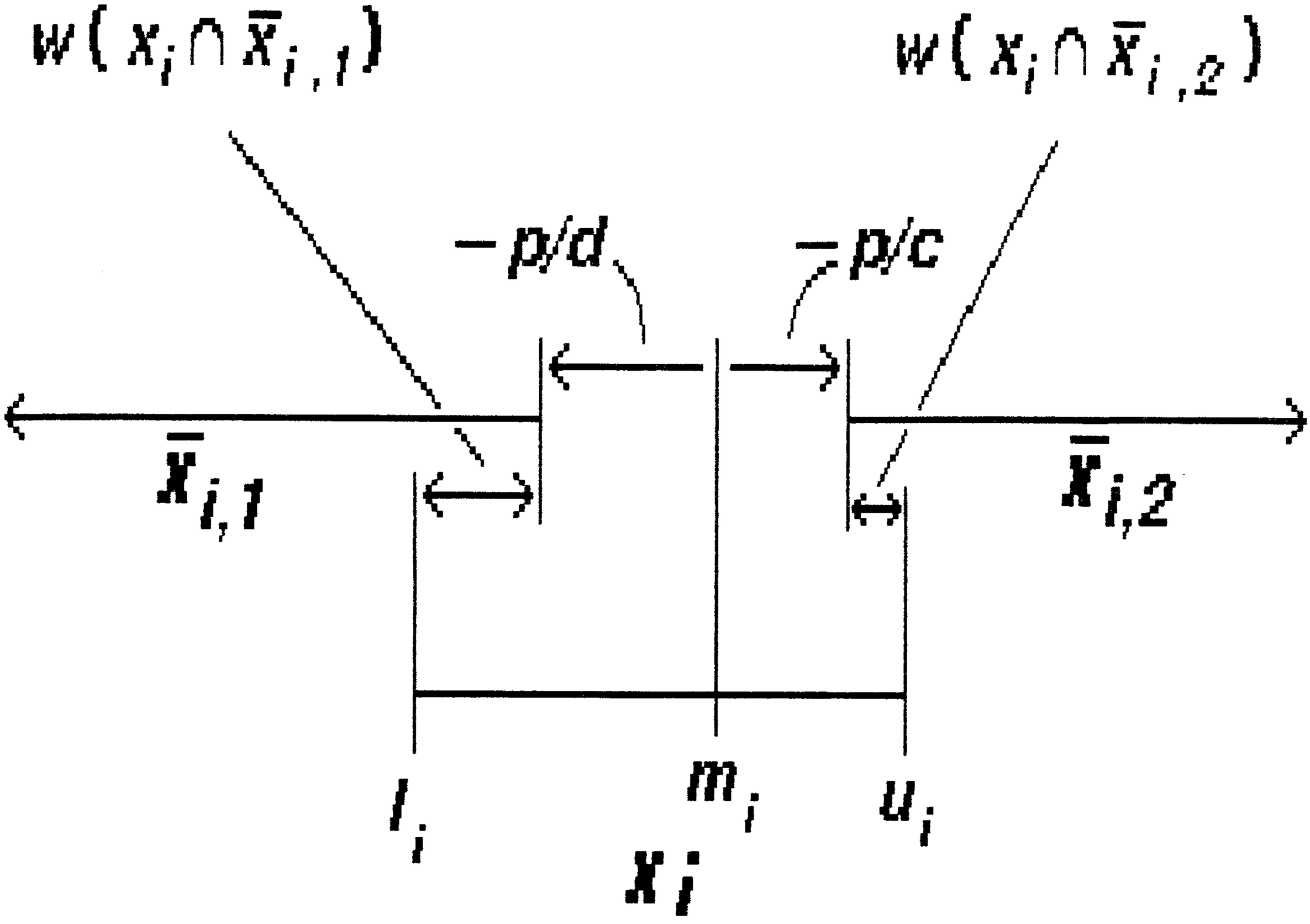

Figure 2. Illustration of two semi–infinite intervals obtained when [p,q] > 0 and [c,d] contains zero.

oot may lie
here.

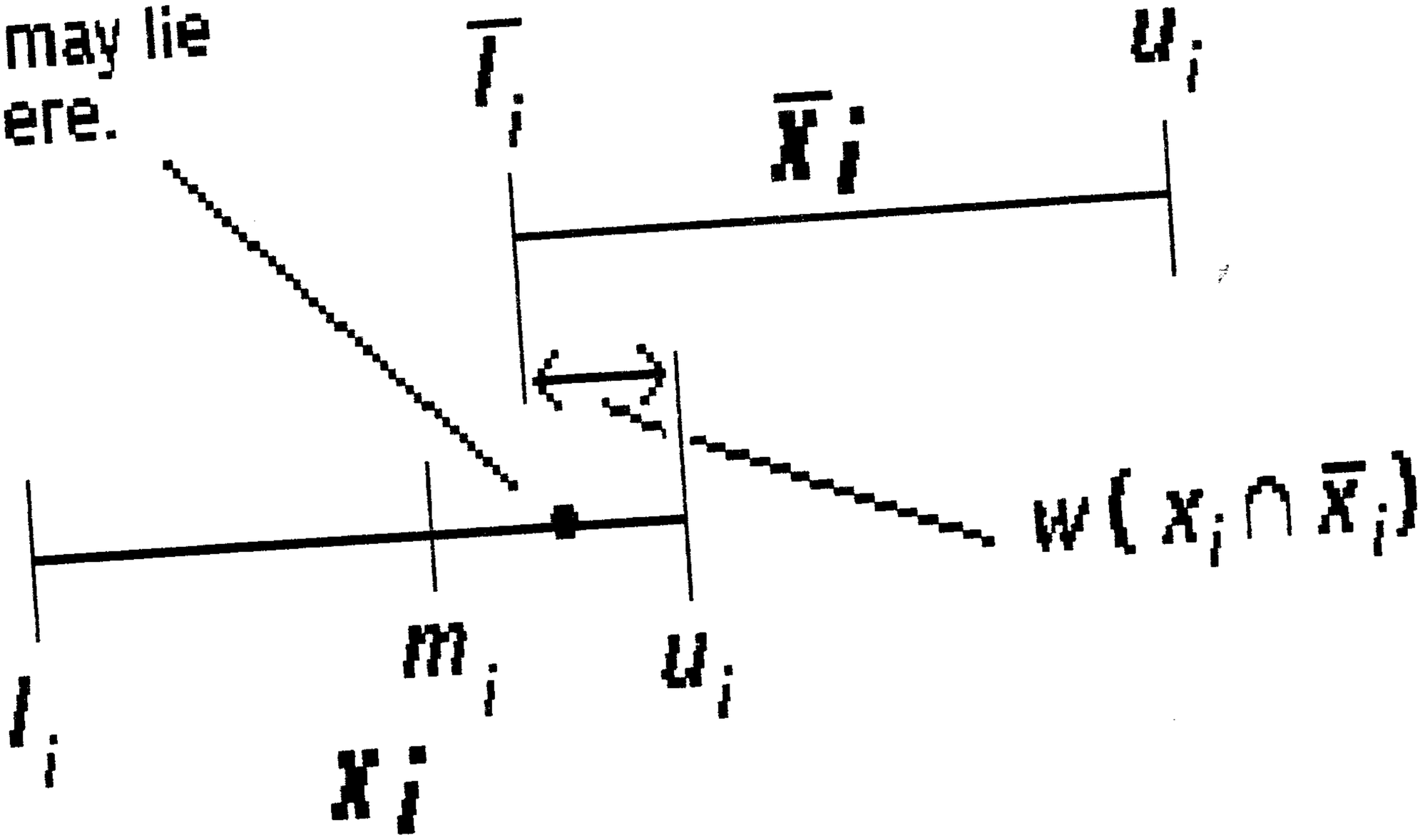

Figure 3a. In this case, Strategy 1 (maximiz—ing $\overline{I}_i$) seems best.

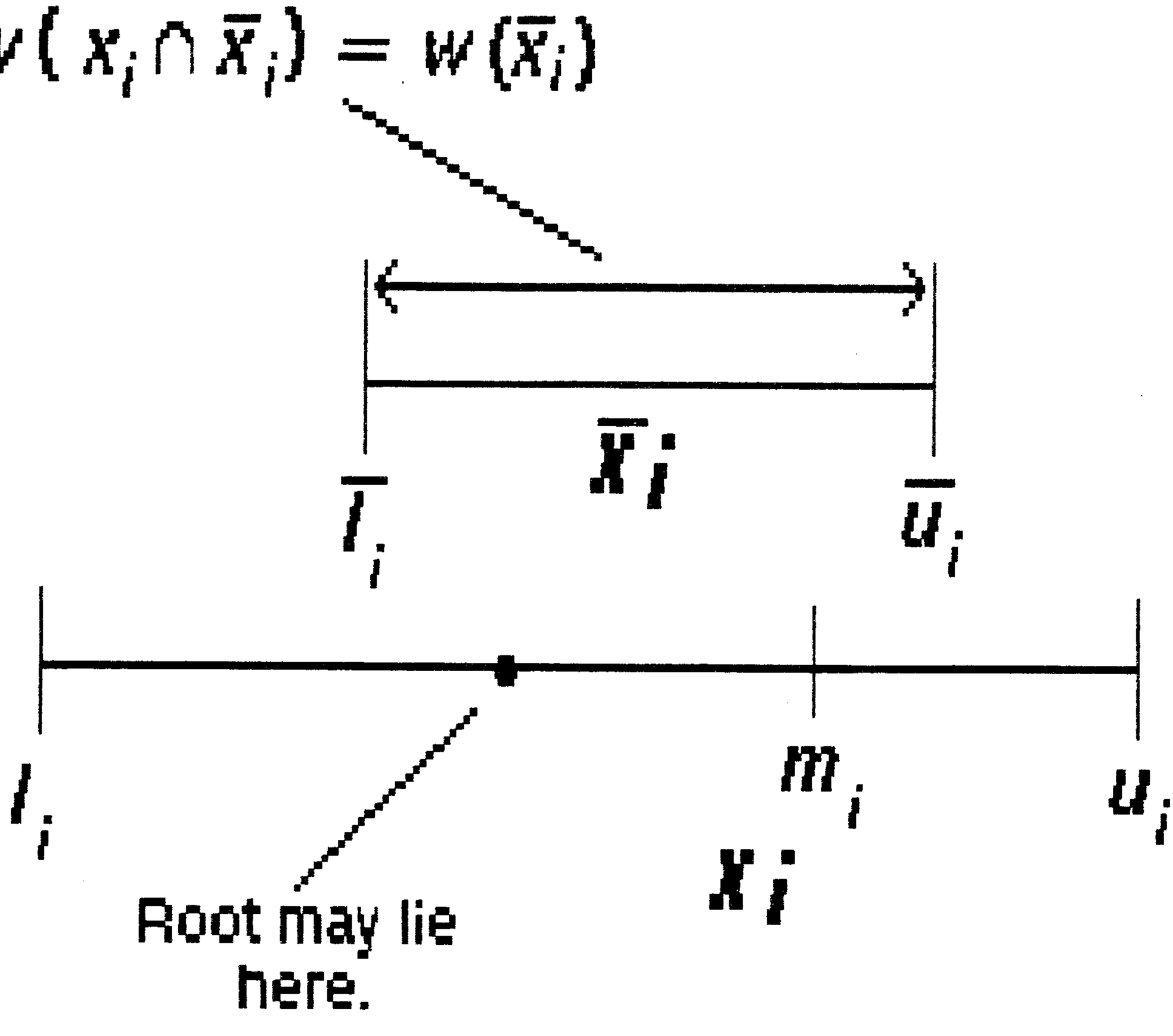

Figure 3c. In this case, Strategy 3 (minimizing $w(\bar{x}_i)$) seems most appropriate.

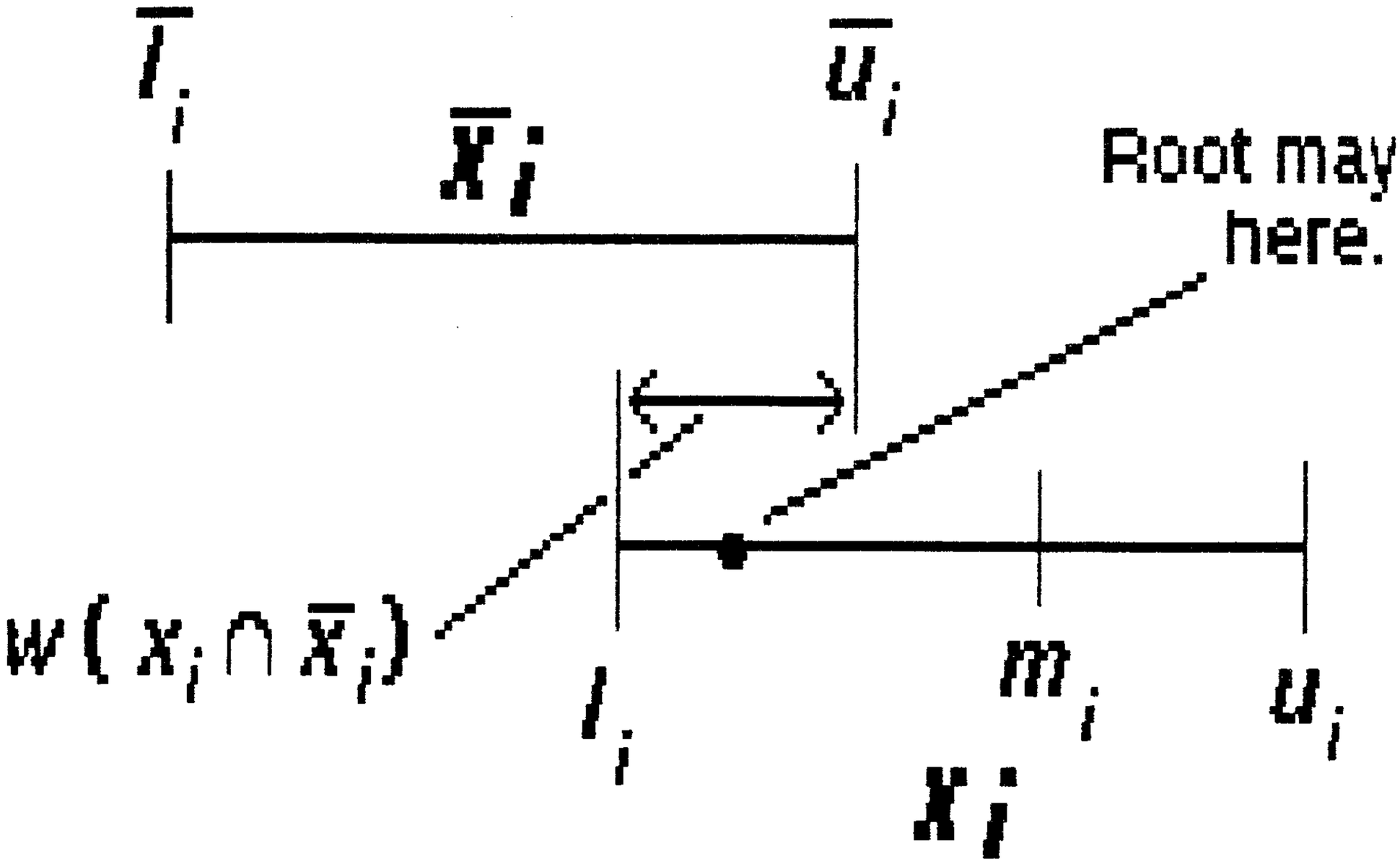

Figure 3b. In this case, Strategy 2 (minimizing $\overline{u}_i$) seems most appropriate.