æ

Decomposition of Arithmetic Expressions to Improve the Behavior of Interval Iteration for Nonlinear Systems

Verbesserung der Intervalliteration für nichtlineare Gleichungssysteme durch Zerlegung arithmetischer Ausdrücke

R. Baker Kearfott

Department of Mathematics
University of Southwestern Louisiana

Abstract. Interval iteration can be used, in conjunction with other techniques, for rigorously bounding all solutions to a nonlinear system of equations within a given region, or for verifying approximate solutions. However, because of overestimation which occurs when the interval Jacobian matrix is accumulated and applied, straightforward linearization of the original nonlinear system sometimes leads to nonconvergent iteration.

In this paper, we examine interval iterations based on an expanded system obtained from the intermediate quantities in the original system. In this system, there is no overestimation in entries of the interval Jacobi matrix, and nonlinearities can be taken into account to obtain sharp bounds. We present an example in detail, algorithms, and detailed experimental results obtained from applying our algorithms to the example.

Intervalliterationen Können in Verbindung mit anderen Verfahren verwendet werden, um alle Lösungen eines nichlinearen Gleichungssystems in einem gegebenen Gebiet mit Sicherheit abzuschätzen, und auch um Approximationen der Lösungen solcher Systeme zu verifizieren. Die Abschätzungen in den Verfahren sind jedoch manchmal nicht hinreichend genau, da Überschätzungen in der Berechnung und in dem Gebrauch der Invervall-Jacobi Matrix auftreten.

In der vorliegenden Arbeit werden Intervalliterationen auf einem erweiterten Gleichungssystem behandelt. In diesem System gibt es keine Überschätzungen der Einzelkomponenten der Intervall-Jacobi Matrix, und für die Nichtlinearitäten können Abschätzungen angegeben werden. Anhand eines Beispiels wird die Wirkungsweise der behandelten Algorithmen demonstriert.

AMS Subject Classifications: Primary: 65H10; Secondary: 65G10
keywords: nonlinear algebraic systems, interval arithmetic, automatic differentiation

1. Introduction, Notation, and Goals

Our general goal is to solve the following problem.

> Find, with certainty, approximations to all solutions of the nonlinear system
>
> $$F(X) = \begin{pmatrix} f_1(x_1, x_2, \ldots, x_n) \\ \vdots \\ f_n(x_1, x_2, \ldots, x_n) \end{pmatrix} = 0,$$
>
> where bounds $\underline{x}_i$ and $\overline{x}_i$ are known such that
>
> $$\underline{x}_i \le x_i \le \overline{x}_i \text{ for } 1 \le i \le n.$$

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

In contrast to other techniques, which typically converge rapidly to an approximation to a particular solution, interval iteration in conjunction with generalized bisection can be used to solve (1.1) with mathematical rigor, or to verify solutions which have been found by some other method. In these methods, we first transform $F(X) = 0$ to the linear interval system

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

where $\mathbf{F}'(\mathbf{X}_k)$ is a suitable interval extension of the Jacobian matrix over the box $\mathbf{X}_k$ (with $\mathbf{X}_0 = \mathbf{B}$), and where $X_k \in \mathbf{X}_k$ represents a predictor or initial guess point. If we then use some technique (such as Gaussian elimination with interval arithmetic, the interval Gauss–Seidel method, the Krawczyk method, etc.; cf. [**16**, ch. 4]), to formally "solve" (1.2), the mean value theorem and properties of interval arithmetic imply that the resulting box $\tilde{\mathbf{X}}_k$ will contain all solutions to $F(X) = 0$ in $\mathbf{X}_k$. We may then iterate $\mathbf{X}_{k+1}$ with the formula

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

to obtain tighter bounds on all possible roots.

Good introductions to interval arithmetic are [**1**] or [**15**]. A thorough treatement of use of interval arithmetic in solving nonlinear systems of equations is [**17**].

If the coordinate intervals of $\mathbf{X}_{k+1}$ are not smaller than those of $\mathbf{X}_k$, then we may bisect one of these intervals to form two new boxes;

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

we then continue the iteration (1.3) with one of these boxes, and put the other one on a stack for later consideration. (This procedure was suggested by Moore and Jones in [**14**], in [**20**], and perhaps even earlier.) For a general treatement of this procedure in the abstract, see [**8**]; for a more sophisticated version, see [**9**]. As explained there and elsewhere, the following fact (from [**15**, p. 263]) allows such a composite generalized bisection algorithm to compute all solutions to (1.1(b)) *with mathematical certainty*. We have,

(1.4) for many methods of solving (1.2), if $\tilde{\mathbf{X}}_k \subset \mathbf{X}_k$, then the system of equations in (1.1) has a unique solution in $\mathbf{X}_k$. Conversely, if $\tilde{\mathbf{X}}_k \cap \mathbf{X}_k = \emptyset$ then there are no solutions of the system in (1.1) in $\mathbf{X}_k$. (cf. [**16**], theorems 5.1.7, 5.1.8, etc. for specifics.)

For efficiency, repeated bisections are to be avoided. Thus, we wish to arrange the interval iteration (1.3) so that the image of $\mathbf{X}_{k+1}$ under the solution process is as small as we can make it with a reasonable amount of effort. This is the general task this paper addresses.

We write $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n)$ for $\mathbf{X}_k$ and we write $\mathbf{f}'_{i,j}$ for the interval in the i-th row and j-th column of $\mathbf{F}' = \mathbf{F}'(\mathbf{X})$. Similarly, we write[2] $F(X_k) = F = (f_1, f_2, \ldots, f_n)^T$, and $X_k = (x_1, x_2, \ldots, x_n)^T$, so that (1.2) becomes

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

As explained in [**17**], the *preconditioned interval Gauss–Seidel* method is superior to many ways of iteratively obtaining sharper solution bounds $\tilde{\mathbf{X}}_k$ (cf. [**17**], 4.3.5 and 4.3.6 on p. 138). Assuming that the system has already been preconditioned, the method may be stated as

ALGORITHM 1.1. *(Interval Gauss-Seidel) Do the following for $i = 1$ to n.*

1. Compute

$$\tilde{\mathbf{x}}_i = x_i - \left[f_i + \sum_{\substack{j=1 \\ j\neq i}}^{n} \mathbf{f}'_{i,j}(\mathbf{x}_j - x_j) \right] \Big/ \mathbf{f}'_{i,i} \tag{1.6}$$

[2] Though the components of F are, in theory, point values and not intervals, they must be evaluated in interval arithmetic with directed roundings or else roundoff error may cause Algorithm 1.1 to miss a root.

using interval arithmetic.

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

To avoid bisections in generalized bisection algorithms or to enable Algorithm 1.1 to be applied on its own in as many situations as possible, we attempt to arrange the computations so that $\tilde{\mathbf{x}}_i$ in (1.6) has small width. In the past, we have studied two approaches to this. The first (in common use in one form or another) is to *precondition* (1.5), i.e., to multiply both sides of (1.5) on the left by some matrix Y so that the resulting widths of the intervals $\tilde{\mathbf{x}}_i$ in the interval Gauss–Seidel method are smaller. (See [**10**] and [**19**].) The second approach is to formally solve for one or more variables (not necessarily $\mathbf{x}_i$) in the i-th containment of (1.5), to use in iteration as in (1.6). (In particular, we may solve for each variable in each equation.) See [**6**] or [**7**] for presentations of this. This second approach does not force convergence in as many cases as more general preconditioners Y, but it is less costly per iteration, and is often effective.

Aside from properties intrinsic to the system of equations $F(X) = 0$, various phenomena will prevent the bounds $\tilde{\mathbf{x}}_i$ from being smaller than the original intervals $\mathbf{x}_i$. Some of these phenomena are related to the method of solution of the linear system (1.2) (provided we explicitly form that system). We refer the reader to the treatise [**17**] and to our recent work [**6**], [**7**], [**10**], [**11**], and [**19**] for efficient techniques to obtain reasonably sharp $\tilde{\mathbf{X}}_k$, given $\mathbf{F}'(\mathbf{X}_k)$[3].

Other phenomena preventing the bounds $\tilde{\mathbf{x}}_i$ from being small are related to the fact that component intervals from the expression

$$F(X_k) + \mathbf{F}'(\mathbf{X}_k)(\mathbf{X}_k - X_k) \tag{1.7}$$

which is implicit in (1.2) are only *bounds* on the range of the component functions of F over $\mathbf{X}_k$, and are not sharp. The following are salient reasons for this.

(i) The entries of the interval Jacobain matrix $\mathbf{F}'(\mathbf{X}_k)$ may not be sharp bounds on the ranges of the corresponding entries of the scalar Jacobian matrix.

[3] These techniques are empirically of a fairly low computational complexity, and give optimal results in a certain sense; see [**10**], [**11**], and [**19**]; computation of the actual solution set, see results of Rohn, summarized in [**17**], ch. 6.

(ii) (1.7) is only a linear model of a nonlinear function.

(iii) Even if the individual entries of $\mathbf{F}'(\mathbf{X}_k)$ are sharp bounds, the actual matrix $\mathbf{F}'(\mathbf{X}_k)$ may not sharply include the set of operators $\{F'(X) \mid |X \in \mathbf{X}_k\}$.

Item (i) is due in a large part to the fact that interval arithmetic is only *sub*-distributive. We remind readers of the problem with the following example. If

$$f(x) = x(x-1)(x-2),$$

then the range of f on $[-3, 3]$ is $[-60, 6]$ whereas the straightforward interval evaluation gives

$$\mathbf{f}([-3,3]) = [-3,3][-4,2][-5,1] = [-12,12][-5,1] = [-60,60].$$

Likewise, the range of the derivative

$$f'(x) = 3x^2 - 6x + 2$$

is $[-1, 47]$ but its interval value is

$$[0,27] - [-18,18] + [2,2] = [-16,47].$$

The problem is that the three factors x, $(x-1)$, and $(x-2)$ do not vary independently, but the interval arithmetic implicitly assumes that they do. (This phenomenon is commonly referred to as *interval dependency.*) Let us, however define

$$v_1 = x,$$

then set

$$\begin{aligned} v_2 - (v_1 - 1) &= 0, \\ v_3 - (v_1 - 2) &= 0, \\ v_4 - (v_1 v_2) &= 0, \\ \text{and} \qquad v_3 v_4 &= 0; \end{aligned}$$

then entries of the interval Jacobian matrix for the resulting system of four equations in v_1, v_2, v_3, and v_4 are *equal* to the corresponding ranges. Furthermore, if v_2, v_3, and v_4 are initially set via forward substitution, then the explicit form of the 4 by 4 system can be utilized, whereas the pre-accumulated derivative value $[-16, 47]$ has lost this information. (Note that the last equation in this system corresponds to the original equation, and thus does not contain any additional intermediate variables.)

The above process does not eliminate interval dependencies in the overall system, but transforms the system to one in which preconditioning is more effective.

As an example of (ii), take the very simple equation $x_2 - x_1^2 = 0$. If we are doing interval iteration, then the equation corresponding to (1.6) for x_1 in terms of x_2 could be

$$\tilde{\mathbf{x}}_1 = x_1 - \left[(x_2 - x_1^2) + (\mathbf{x}_2 - x_2)\right]/(-2\mathbf{x}_1).$$

If we take current interval value for $\mathbf{x}_1$ to be $[-2, 2]$ and the current interval value of $\mathbf{x}_2$ to be $[0, 1]$, and if we take the corresponding scalar values x_1 and x_2 to equal the midpoints $x_1 = 0$ and $x_2 = .5$, the above equation becomes

$$\tilde{\mathbf{x}}_1 = 0 - [.5 + [-.5, .5]]/[-4, 4],$$

so that $\tilde{\mathbf{x}}_1$ is the entire real line. However, we may explicitly compute the (multivalued) inverse of the elementary function $f(x_1) = x_1^2$ to obtain $x_1 \in [-1, 0] = -\sqrt{\mathbf{x}_2}$ or $x_1 \in [0, 1] = \sqrt{\mathbf{x}_2}$. When this is done, we obtain the better bounds $[-1, 1]$ for the interval value of x_1. Note that this can be thought of as a *nonlinear* interval Gauss–Seidel step, and is *not* the same as an interval Gauss–Seidel step without preconditioning.

Item (iii) above is due partially to the fact that the matrices

$$\{F'(X) \mid X \in \mathbf{X}_k\}$$

do not fill the box $\mathbf{F}'(\mathbf{X}_k) \in \mathbb{R}^{n\times n}$. To get sharper bounds, we may use *interval slopes in lieu* of $\mathbf{F}'$; this is described in [**13**], [**18**], and [**21**]. Though, for simplicity of exposition and implementation, we do not make use of these techniques in this paper, they may be employed in conjunction with the techniques we do present.

In this paper, we consider a derived, or expanded system based on the *code list*, in which only one elementary operation or only one elementary function evaluation occurs in each equation. Such a code list is generated naturally by a compiler as a result of parsing an arithmetic expression, and has been used as a tool in various contexts. For example, in [**4**] Griewank bases an automatic differentiation technique for efficient computation of steps for the classical Newton method on it. In [**2**], Böhm develops a method based on the code list which transforms the problem of optimally accurate evaluation of arithmetic expressions to optimally accurate solution of a linear system of equations[4]. Similar uses of the code list were undoubtedly advocated even earlier by Rall and others.

In our context (that of problem (1.1), considering the intermediate quantities in the code list to be variables accomplishes the following two things:

- We explicitly spell out the interval dependencies in the system, so that each of the new equations is an exact range; though dependencies seemingly are reintroduced during the algorithmic process, the explicit form of the larger system is more effective in conjunction with our optimizing preconditioners.
- We use explicit inverses of the elementary functions in a separate iteration process. This is roughly analogous to a Gauss–Seidel step without preconditioner on the larger system, but leads to sharper results, as we do not first do a linearization for interval inclusions, but compute exact ranges of nonlinear functions.

Additionally, the complexity theory (concerning the number of operations required to evaluate the Jacobian matrix) in [**4**] and techniques for automating the process in [**4**] (related to automatic differentiation) is applicable in our context.

The goal of this paper is to present examples and algorithms. In §2, we introduce a simple example problem which requires extensive generalized bisection to solve if it is not decomposed, but which requires only a few iterations if the expanded system is utilized appropriately. In §3, we present our algorithm. In §4, we present experimental results in detail for our example problem, as well a results on several problems when we imbed the algorithm in a generalized biection code. In §5, a variant of our algorithm is pointed out to be at least as good as the similarly preconditioned interval Gauss–Seidel method applied to the original system. A discussion in the context of the Hansen/Greenberg algorithm appears in §6. Conclusions and avenues for further work appear in §7.

2. An Example

Here, we present a detailed example to illustrate how decomposition of the arithmetic expressions can be useful. As above, we note that the resulting equations do not eliminate interval dependencies in the entire, coupled system, but allow computation of exact bounds on intermediate results of the evaluations.

[4]Unfortunately, Böhm's techniques are not directly applicable here, since in our context, the original quantities, in addition to the intermediate ones, must be thought of as variables.

EXAMPLE 2.1. *Define $F(X) : \mathbb{R}^2 \to \mathbb{R}^2$ by*

$$\begin{aligned} f_1(x_1, x_2) &= x_1^3 + x_1^2 x_2 + x_2^2 + 1 \\ f_2(x_1, x_2) &= x_1^3 - 3x_1^2 x_2 + x_2^2 + 1, \end{aligned}$$

set the initial box to

$$\mathbf{X} = \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \end{pmatrix} = \begin{pmatrix} [-2, 0] \\ [-1, 1] \end{pmatrix},$$

and initial guess point

$$X = \begin{pmatrix} -1 \\ 0 \end{pmatrix}.$$

We note that, within the box, this system of equations has a unique solution at the initial guess point $(-1, 0)^T$. However, the interval Jacobian matrix is

$$\mathbf{F}'(\mathbf{X}) = \begin{pmatrix} 3\mathbf{x}_1^2 + 2\mathbf{x}_1\mathbf{x}_2 & \mathbf{x}_1^2 + 2\mathbf{x}_2 \\ 3\mathbf{x}_1^2 - 6\mathbf{x}_1\mathbf{x}_2 & -3\mathbf{x}_1^2 + 2\mathbf{x}_2 \end{pmatrix} = \begin{pmatrix} [-4, 16] & [-2, 6] \\ [-12, 24] & [-14, 2] \end{pmatrix}.$$

Therefore, since $\mathbf{F}'(\mathbf{X})$ contains the zero matrix, $Y\mathbf{F}'(\mathbf{X})$ will contain the zero matrix for any preconditioner Y. Thus, the denominator in (1.6) will contain zero, regardless of how the function is preconditioned, and each component of $\tilde{\mathbf{X}}_k$ will be infinite if it is obtained via the interval Gauss–Seidel method; if, in addition, $F(X_k)$ is small, then

$$\tilde{\mathbf{X}}_k = \begin{pmatrix} \mathbb{R} \\ \mathbb{R} \end{pmatrix}.$$

However, we may use the technique in [**4**] to rewrite the system in terms of its component operations as follows.

EXAMPLE 2.1(B). *(Example 2.1 rewritten in terms of its component operations) Set*

$$\begin{aligned} v_1 &= x_1 & v_2 &= x_2 \\ v_3 &= v_1^2 & v_4 &= v_2^2 \\ v_5 &= v_3 v_2 & v_6 &= v_1^3 \\ v_7 &= v_6 + v_4 + 1 \\ v_7 + v_5 &= 0 \\ v_7 - 3v_5 &= 0, \end{aligned} \tag{2.1}$$

so that we have the expanded nonlinear system of equations

$$(2.2) \qquad F_E(V) = \begin{pmatrix} v_3 - v_1^2 \\ v_4 - v_2^2 \\ v_5 - v_3 v_2 \\ v_6 - v_1^3 \\ v_7 - (v_6 + v_4 + 1) \\ v_7 + v_5 \\ v_7 - 3v_5 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}.$$

Use forward substitution with (2.1), *starting with* $\mathbf{v}_1 = [-2, 0]$ *and* $\mathbf{v}_2 = [-1, 1]$ *to obtain the initial box*

$$(2.3) \qquad \mathbf{V} = \begin{pmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \\ \mathbf{v}_3 \\ \mathbf{v}_4 \\ \mathbf{v}_5 \\ \mathbf{v}_6 \\ \mathbf{v}_7 \end{pmatrix} = \begin{pmatrix} [-2, 0] \\ [-1, 1] \\ [0, 4] \\ [0, 1] \\ [-4, 4] \\ [-8, 0] \\ [-7, 2] \end{pmatrix}$$

and initial guess point

$$(2.3b) \qquad V = \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \\ v_7 \end{pmatrix} = \begin{pmatrix} -1 \\ 0 \\ 2 \\ 0.5 \\ 0 \\ -4 \\ -2.5 \end{pmatrix}.$$

Since it is now possible to (formally) solve for any variable in any equation in which it occurs, we may now use (2.2) directly to compute tighter bounds in a way not possible in the original system. For example, we may solve for $\mathbf{v}_5$ in equation 6 to obtain

$$\tilde{\mathbf{v}}_5 = -\mathbf{v}_7 = [-2, 7],$$

and we may replace $\mathbf{v}_5$ by $\mathbf{v}_5 \cap \tilde{\mathbf{v}}_5 = [-2, 4]$. We may then solve for $\mathbf{v}_5$ in equation 7 to obtain

$$\tilde{\mathbf{v}}_5 = \mathbf{v}_7/3 = [-\frac{7}{3}, \frac{2}{3}],$$

and we may replace $\mathbf{v}_5$ by $\mathbf{v}_5 \cap \tilde{\mathbf{v}}_5 = [-2, 2/3]$. Finally, we may solve for $\mathbf{v}_7$ in equation 6 to obtain

$$\tilde{\mathbf{v}}_7 = -\mathbf{v}_5 = [-\frac{2}{3}, 2],$$

and we may replace $\mathbf{v}_7$ by $\mathbf{v}_7 \cap \tilde{\mathbf{v}}_7 = [-2/3, 2]$. (Note: The above three computations were equivalent to Gauss–Seidel steps. However, this is in general not the case. For example, it may have been advantageous to solve for $\mathbf{v}_1$ in the first equation, to obtain

$$\tilde{\mathbf{v}}_1 = -\sqrt{\mathbf{v}_3} \cup \sqrt{\mathbf{v}_3},$$

where the right member must represent a smaller set than that obtained from the corresponding Gauss–Seidel method, which would be

$$\tilde{\mathbf{v}}_1 = v_1 - \left[(v_3 - v_1^2) + (\mathbf{v}_3 - v_3)\right] / (-2\mathbf{v}_1).$$

The Jacobi matrix of F_E is

$$\begin{pmatrix} -2v_1 & & & 1 & & & & \\ & -2v_2 & & & 1 & & & \\ & -v_3 & -v_2 & & & 1 & & \\ -3v_2^2 & & & & & & 1 & \\ & & & & -1 & & -1 & 1 \\ & & & & & 1 & & 1 \\ & & & & & -3 & & 1 \end{pmatrix},$$

where blank entries are zero. The initial interval value of this matrix is

$$\mathbf{F}'_E(\mathbf{V}) = \begin{pmatrix} [0,4] & & & 1 & & & & \\ & [-2,2] & & & 1 & & & \\ & [-4,0] & [-1,1] & & & 1 & & \\ [-3,0] & & & & & & 1 & \\ & & & & -1 & & -1 & 1 \\ & & & & & 1 & & 1 \\ & & & & & -3 & & 1 \end{pmatrix}.$$

Even though the formal determinant $\det \mathbf{F}'_E(\mathbf{V})$ contains zero, many of the entries do not, and optimizing preconditioners as in [**9**] can still be effective. For example, if, after making the above substitutions, we use the width-optimizing preconditioner row

$$Y_5 = (0, 0, 0, 0, 0, -.25, .25),$$

we obtain

$$Y_5\mathbf{F}'_E(\mathbf{V}) = (0,0,0,0,-1,0,0).$$

If we use $v_5 = m(\mathbf{v}_5) = (-2+2/3)/2 = -4/3$ and $v_7 = m(\mathbf{v}_7) = (-2/3+2)/2 = 4/3$, we obtain the point value

$$F_E(V) = (1, 1/2, -4/3, -3, 23/6, 0, 16/3)^T,$$

and

$$Y_5F_E(V) = (0,0,0,0,4/3,0,0)^T.$$

Using these preconditioned values in place of $\mathbf{f}'_{5,*}$ and f_5 in (1.6), we obtain

$$\tilde{\mathbf{x}}_5 = -4/3 - [4/3]\,/([-1,-1]) = [0,0],$$

a substantial improvement over the previous value of $[-2/3, 2]$.

In §4, we indicate how continuation of this process results in propagation of such width improvements in this example back to the original variables $\mathbf{v}_1$ and $\mathbf{v}_2$. Such improvements are not possible, even with optimizing preconditioners, when the original system is used; solution of the original system would require a more lengthy generalized bisection process.

3. Algorithms

In this section, we specify two algorithms for performing interval iterations on expanded systems, corresponding to the two processes illustrated in Example 2.1(b). We view these two algorithms as a *single* composite algorithm, with the output of the first algorithm serving as input to the second.

In the first algorithm, we obtain exact ranges based on solving (nonlinearly) for each variable in each equation; as mentioned, this can be done because each equation in the expanded system $F_E(V) = 0$ contains only a single operation or a single elementary function. In the second algorithm, we employ the optimizing preconditioners from [**10**] and [**19**] to perform a preconditioned interval Gauss–Seidel iteration. Though the two algorithms are to a large extent analogous, the second algorithm is necessary because, even though the nonlinear solution technique is superior to non-preconditioned Gauss–Seidel iteration, more than one equation must often simultaneously be considered to obtain width reductions.

Algorithm 3.1. *(Solution for each variable in each equation, using exact ranges on the inverses of elementary operations and functions)*

0. (**Input**) *Let N be the dimension of the original system (as in Example 2.1) and let N_E be the dimension of the expanded system (as in Example 2.1(b)).*

   a) Input initial interval values $\mathbf{v}_j$, $1 \le j \le N$.

   b) Via forward substitution using equations analogous to the third through seventh equation of (2.1), *compute $\mathbf{v}_{N+1}$ to $\mathbf{v}_{N_E}$.*

   c) Input a convergence tolerance ϵ.

1. (**Initial Scan on all variables**)

For $i = 1$ to N_E **DO**:

  For each equation index j such that $\mathbf{v}_i$ occurs in the j-th equation **DO**:

   a) Compute bounds $\tilde{\mathbf{v}}_i$ on the i-th variable in the j-th equation by using the analytic solution for it.

   b) If $\tilde{\mathbf{v}}_i \cap \mathbf{v}_i = \emptyset$, then return, indicating that there are no roots of the function within the box given by $(\mathbf{v}_1, \ldots, \mathbf{v}_N)$.

   c) If $\tilde{\mathbf{v}}_i \cap \mathbf{v}_i \ne \mathbf{v}_i$, then

    (i) push i onto a stack $\mathcal{V}$ of variables which have been changed.

    (ii) Replace $\mathbf{v}_i$ by $\tilde{\mathbf{v}}_i \cap \mathbf{v}_i$.

  END DO

END DO

2. (**Return if diameters are small enough**) *If the widths $w(\mathbf{v}_k)$ are each less than ϵ for $k = 1, \cdots, N$, then return indicating convergence to tight solution bounds*[5]

3. (**Adjustment of variables coupled to changed variables**)

DO WHILE *the stack $\mathcal{V}$ is nonempty:*

   a) Pop an index i from the stack $\mathcal{V}$.

   b) For each equation index j such that the j-th equation depends on the i-th variable **DO**:

    For each variable index $l \ne i$ such that $\mathbf{v}_l$ appears in the j-th equation **DO**:

     (i) Compute bounds $\tilde{\mathbf{v}}_l$ on the l-th variable in the j-th equation by using the analytic solution for it.

     (ii) If $\tilde{\mathbf{v}}_l \cap \mathbf{v}_l = \emptyset$, then return, indicating that there are no roots of the function within the box given by $\mathbf{v}_k$, $k = 1, \cdots, N$.

[5] Another possibility here is that not enough digits are carried to determine that there are no solutions.

(iii) If $\tilde{\mathbf{v}}_l \cap \mathbf{v}_l \neq \mathbf{v}_l$, then

(α) push l onto the stack $\mathcal{V}$ of variables which have been changed, provided l is not already on the stack.

(β) Replace $\mathbf{v}_l$ by $\tilde{\mathbf{v}}_l \cap \mathbf{v}_l$.

END DO

END DO

END DO

4. Return.

The actual bound computation in Algorithm 3.1 occurs in step 1a) and step 3b)(i). As an example of this, suppose the function is as in Example 2.1(b), but suppose we have $\mathbf{v}_4 = [0, 4]$, and $\mathbf{v}_2 = [-9, 9]$, and suppose we enter step 3b)(i) of Algorithm 3.1 with $i = 4$, $j = 2$, and $l = 2$. Then we would have

$$\tilde{\mathbf{v}}_2 = \pm\sqrt{\mathbf{v}_4} = [-2, 0] \cup [0, 2] = [-2, 2],$$

so

$$\tilde{\mathbf{v}}_2 \cap \mathbf{v}_2 = [-2, 2] \cap [-9, 9] = [-2, 2] \neq \mathbf{v}_2,$$

and the step would result in tighter bounds on $\mathbf{v}_2$.

We note that, had the current value of $\mathbf{v}_4$ been, say, $[1, 4]$, then

$$\tilde{\mathbf{v}}_2 = [-2, -1] \cup [1, 2],$$

so $\tilde{\mathbf{v}}_2 \cap \mathbf{v}_2$ would have been a union of two disjoint intervals; this case can be handled within the generalized bisection framework by pushing one of the boxes $\mathbf{V}$ so obtained onto a stack for later consideration, just as if it had been produced via generalized bisection. This would probably result in a reasonable algorithm, but one which is somewhat more complicated. In our initial investigations, we have replaced such sets of two disjoint intervals by their convex hull which, though not representing sharp bounds, results in a somewhat simpler algorithm. (We could handle division by intervals containing zero similarly, but we similarly do not allow this case in these initial experiments.)

Step 1 of Algorithm 3.1 is similar to application of the *all row* preconditioner described in [**7**] and [**11**], except that we use sharp functional inverses instead of solving for the variable in a linearization.

The next algorithm is similar to Algorithm 3.1, except that we use the preconditioned interval Gauss–Seidel method to compute bounds $\tilde{\mathbf{x}}_i$ for the i-th variable, instead of solving for it in each equation which contains it. The preconditioner is functionally identical to that in [**10**], but is computed with the improvements expounded in [**19**]. Execution of this algorithm is usually more costly per loop iteration than that of Algorithm 3.1, but is reasonable in many cases in which tight bounds on a variable cannot be obtained by just using a single equation.

ALGORITHM 3.2. *(Using the linear programming preconditioner)*

0. (**Input**) *Input as in step 0 of Algorithm 3.1, or else use the output* $\mathbf{V}$ *of algorithm 3.1.*

1. (**Initial Scan on all variables**)
For $i = 1$ *to* N_E **DO**:
- *a) Compute a new interval* $\tilde{\mathbf{v}}_i$ *for the* i*-th variable by using the preconditioned interval Gauss–Seidel method with LP preconditioner as in* [**10**] *and* [**19**].
- *b) If* $\tilde{\mathbf{v}}_i \cap \mathbf{v}_i = \emptyset$, *then return, indicating that there are no roots of the function within the box given by* $\mathbf{v}_k$, $k = 1, \cdots, N$.
- *c) If* $\tilde{\mathbf{v}}_i \cap \mathbf{v}_i \neq \mathbf{v}_i$, *then*
  - *(i) push* i *onto a stack* $\mathcal{V}$ *of variables which have been changed.*
  - *(ii) Replace* $\mathbf{v}_i$ *by* $\tilde{\mathbf{v}}_i \cap \mathbf{v}_i$.

END DO

2. (**Return if widths are small**) *If the widths* $w(\mathbf{v}_k)$ *are each less than* ϵ *for* $k = 1, \cdots, N$, *then return indicating convergence to tight solution bounds.*

3. (**Adjustment of variables coupled to changed variables**)
DO WHILE *the stack* $\mathcal{V}$ *is nonempty:*
- *a) Pop an index* i *from the stack* $\mathcal{V}$.
- *b)* (**Adjustment with explicit functional inverses**) *For each equation index* j *such that the* j*-th equation depends on the* i*-th variable* **DO**:

  For each variable index $l \neq i$ *such that* $\mathbf{v}_l$ *appears in the* j*-th equation* **DO**:
  - *(i) Compute bounds* $\tilde{\mathbf{v}}_l$ *on the* l*-th variable in the* j*-th equation by using the analytic solution for it.*
  - *(ii) If* $\tilde{\mathbf{v}}_l \cap \mathbf{v}_l = \emptyset$, *then return, indicating that there are no roots of the function within the box given by* $(\mathbf{v}_1, \cdots, \mathbf{v}_N)$.
  - *(iii) If* $\tilde{\mathbf{v}}_l \cap \mathbf{v}_l \neq \mathbf{v}_l$, *then*
    - *(α) push* l *onto a stack* $\mathcal{V}$ *of variables which have been changed, provided* l *is not already on the stack.*
    - *(β) Replace* $\mathbf{v}_l$ *by* $\tilde{\mathbf{v}}_l \cap \mathbf{v}_l$.

  END DO

END DO

c) *(***Adjustment of variables coupled to changed variables via the LP preconditioner; only do the adjustment on those variables which were not changed in step 3.***)* *For each index $l \neq i$ which just occurred in step 3b)(i) whose variable width $w(\mathbf{v}_l)$ is greater than ϵ and whose width was not decreased in step 3b)(iii)(β),* **DO***:*

(i) *Compute a new interval $\tilde{\mathbf{v}}_l$ for the l-th variable by using the preconditioned interval Gauss–Seidel method with LP preconditioner as in [***10***] and [***19***].*

(ii) *If $\tilde{\mathbf{v}}_l \cap \mathbf{v}_l = \emptyset$, then return, indicating that there are no roots of the function within the box given by $(\mathbf{v}_1, \cdots, \mathbf{v}_N)$.*

(iii) *If $\tilde{\mathbf{v}}_l \cap \mathbf{v}_l \neq \mathbf{v}_l$, then*

(α) *push l onto a stack $\mathcal{V}$ of variables which have been changed, provided l is not already on the stack.*

(β) *Replace $\mathbf{v}_l$ by $\tilde{\mathbf{v}}_l \cap \mathbf{v}_l$.*

d) *(***Return if convergence has occurred***)* *If the widths $w(\mathbf{v}_k)$ are each less than ϵ for $k = 1, \cdots, N$, then return indicating convergence to tight solution bounds.*

END DO

END DO

4. Return.

As mentioned, it is usually relatively inexpensive to use functional inverses in single equations, compared to solving the linear programming problems, applying the resulting preconditioner row, and then performing the interval Gauss–Seidel method (or, equivalently, the interval Jacobi method, since we are just examining a single row), as in step 1a) and 3c)(i) of Algorithm 3.2. Furthermore, due to nonlinearities, there are cases when the explicit functional inverses could give tighter bounds than the interval Gauss–Seidel method. This is why we include step 3b) before step 3c) in Algorithm 3.2. Similarly, it is reasonable to first execute Algorithm 3.1, then immediately execute Algorithm 3.2 in the overall root-finding process.

The Jacobi matrix of the expanded system is *always* very sparse; we highlight this in

LEMMA 3.3. *Suppose that evaluation of the components f_i in the original nonlnear system is decomposed in such a way that only a single unary or binary operation is involved in each equation of the resulting expanded system. Then the derived system has at most three nonzero entries in each row of its Jacobi matrix.*

To see the validity of lemma 3.3, observe that an equation in the resulting derived system depends only on the variables in each operand

and on the variable defined to be the result of the operation. See also e. g. [**4**] for more theory and operations counts.

In some cases, we may wish to allow certain rows of the Jacobi matrix to be more dense, such as when we consider a dot product to be a single operation.

Sparse form is necessary for the method to be practical on many problems. We have found it useful to index the Jacobi matrix both on its rows and on its columns. Furthermore, sparse linear programming solvers are applicable for computing the preconditioners.

4. Experimental Results

We have implemented Algorithm 3.1 and Algorithm 3.2 in Fortran 77, and have run them on an IBM PC-compatible after compiling with the Microsoft Fortran compiler. We linked with the interval arithmetic package associated with INTBIS ([**12**]). We first ran Algorithm 3.1 on Example 2.1(b). We then used the box returned by Algorithm 3.1 as input to Algorithm 3.2. We obtained the following results.

In step 1 of Algorithm 3.1:

1. Solving for variable $i = 5$ in equation number $j = 6$, we obtained
$$\tilde{\mathbf{v}}_5 \approx [-2, 4].$$
2. Solving for variable $i = 5$ in equation number $j = 7$, we obtained
$$\tilde{\mathbf{v}}_5 \approx \left[-2, \frac{2}{3}\right].$$
3. Solving for variable $i = 7$ in equation number $j = 6$, we obtained
$$\tilde{\mathbf{v}}_7 \approx \left[-\frac{2}{3}, 2\right].$$

No bounds were tightened in Step 3 of Algorithm 3.1. Thus we obtained the following **upon return from Algorithm 3.1:**

$$\mathbf{V} \approx \begin{pmatrix} [-2,0] \\ [-1,1] \\ [0,4] \\ [0,1] \\ [-2,2/3] \\ [-8,0] \\ [-2/3,2] \end{pmatrix} \tag{4.1}$$

Consistent with our view of Algorithm 3.1 and Algorithm 3.2 as a single composite algorithm, we used the bounds (4.1) upon entry to Algorithm 3.2; we used $\epsilon \approx 10^{-5}$, with the following results.

In step 1 of Algorithm 3.2:

1. Performing a preconditioned Gauss–Seidel step for variable $i = 5$, we obtained
$$\tilde{\mathbf{v}}_5 \approx [-1.8 \times 10^{-15}, 1.6 \times 10^{-15}].$$
2. Performing a preconditioned Gauss–Seidel step for variable $i = 6$, we obtained
$$\tilde{\mathbf{v}}_6 \approx [-2, -1].$$
3. Performing a preconditioned Gauss–Seidel step for variable $i = 7$, we obtained
$$\tilde{\mathbf{v}}_7 \approx [-1.2 \times 10^{-15}, 1.3 \times 10^{-15}].$$

In step 3b) of Algorithm 3.2:

4. From changed variable $i = 6$, using equation $j = 4$, variable $l = 1$ was tightened to
$$\tilde{\mathbf{v}}_1 \approx [-1.26, -1].$$
5. From changed variable $i = 1$, using equation $j = 1$, variable $l = 3$ was tightened to
$$\tilde{\mathbf{v}}_3 \approx [1, 1.587].$$
6. From changed variable $i = 3$, using equation $j = 3$, variable $l = 2$ was tightened to
$$\tilde{\mathbf{v}}_2 \approx [-1.8 \times 10^{-15}, 1.6 \times 10^{-15}].$$

In step 3c) of Algorithm 3.2:

7. Performing a preconditioned Gauss–Seidel step for variable $i = 1$, we obtained
$$\tilde{\mathbf{v}}_1 \approx [-1.037, -1].$$

In step 3b) of Algorithm 3.2:

8. From changed variable $i = 1$, using equation $j = 1$, variable $l = 3$ was tightened to
$$\tilde{\mathbf{v}}_3 \approx [1, 1.075].$$
9. From changed variable $i = 1$, using equation $j = 4$, variable $l = 6$ was tightened to
$$\tilde{\mathbf{v}}_6 \approx [-1.115, -1].$$

In step 3c) of Algorithm 3.2:

10. Performing a preconditioned Gauss–Seidel step for variable $i = 3$, we obtained
$$\tilde{\mathbf{v}}_3 \approx [1, 1.002].$$

In step 3b) of Algorithm 3.2:

11. From changed variable $i = 6$, using equation $j = 5$, variable $l = 4$ was tightened to
$$\tilde{\mathbf{v}}_4 \approx [0, .1151].$$

In step 3c) of Algorithm 3.2:

12. Performing a preconditioned Gauss–Seidel step for variable $i = 1$, we obtained
$$\tilde{\mathbf{v}}_1 \approx [-1.001, -1].$$

In step 3b) of Algorithm 3.2:

13. From changed variable $i = 1$, using equation $j = 1$, variable $l = 3$ was tightened to
$$\tilde{\mathbf{v}}_3 \approx [1, 1.002].$$

13. From changed variable $i = 1$, using equation $j = 4$, variable $l = 6$ was tightened to
$$\tilde{\mathbf{v}}_6 \approx [-1.003, -1].$$

In step 3c) of Algorithm 3.2:

14. Performing a preconditioned Gauss–Seidel step for variable $i = 3$, we obtained
$$\tilde{\mathbf{v}}_3 \approx [1, 1].$$

In step 3b) of Algorithm 3.2:

15. From changed variable $i = 6$, using equation $j = 5$, variable $l = 4$ was tightened to
$$\tilde{\mathbf{v}}_4 \approx [0, 2.93 \times 10^{-3}].$$

In step 3c) of Algorithm 3.2:

16. Performing a preconditioned Gauss–Seidel step for variable $i = 1$, we obtained
$$\tilde{\mathbf{v}}_1 \approx [-1, -1].$$

It is informative to examine some of the statistics concerning execution of the preconditioned Gauss–Seidel steps in Algorithm 3.2. In some instances step 3c)(i) cannot be completed because the appropriate preconditioner does not exist. However, in the example, step 3c)(i) was completed a total of 16 times; of these, we observe that 5 resulted in successful tightening of the bounds.

Since the Jacobi matrices resulting from the expanded system are in general extremely sparse, it is of interest to note how many entries of the preconditioner are nonzero. In particular, if the Jacobi matrix approximates a point matrix (i.e. if the bounds in the entries are tight), then the preconditioner vector for the i-th variable may approximate the i-th row of the inverse of this point matrix. Results by Duff et al ([**3**]) indicate that we may expect the inverse of an arbitrary sparse matrix to be dense. However, we observed in the example that most preconditioner rows had only one or two nonzero entries, as indicated in the following table

Table 1. Number of Nonzero entries in Preconditioner Row
Results from Example 2.1(b)

No. nonzeros	1	2	3	4	5	6	7
No. preconditioners	8	2	4	0	0	2	0

This observation can save execution time, since preconditioners with just one non-zero entry correspond to a Gauss–Seidel step which is an imperfect, linearized version of a step which was already performed exactly in Step 3b) of the algorithm; that is, it can be proven that none of the preconditioners with only one non-zero entry would result in tightening of bounds. Thus, in the example, only 8 Gauss–Seidel steps need have been completed, and 5 of them would have successfully tightened variable bounds. (We also note that, with improvements in [**19**], our preconditioner computations usually represent only a small portion of the work in the overall Gauss–Seidel step, especially when the resulting optimal preconditioner has only a few nonzero entries.)

We implemented the function and inverse evaluations in steps 1a) and 3b) of Algorithm 3.1 and Algorithm 3.2 by table lookup by equation and variable, so that no redundant function evaluations were done. However, we explicitly programmed the function and inverse function evaluations using assembly language-like calls to our interval arithmetic package; we similarly programmed the function and Jacobi matrix required for executing the Gauss–Seidel steps.

In another test, we embedded the Algorithm 3.1–3.2 combination into an update of our generalized bisection code INTBIS of [**12**]; specifically, we replaced the routine for the interval Gauss–Seidel iteration by Algorithm 3.1–3.2; we then compared the results to those obtained with identical input, but with the original system of equations and a hybrid optimally preconditioned scheme (Algorithm 5.1 in [**11**]). We tried the following functions and initial boxes.

1. The example from §2, but with an initial box of

$$\mathbf{X}_0 = \begin{pmatrix} [-200, 200] \\ [-200, 200] \end{pmatrix}$$

This problem has only one solution in the box.

This example is simple enough to program by hand, but has a substantial amount of interval dependency. Thus, the derived system should do well in a generalized bisection algorithm relative to the original system.

2. The system (previously used as a test problem in generalized bisection and elsewhere)

$$\begin{aligned} f_1(x_1,x_2) &= 4x_1^3 - 3x_1 - x_2 = 0 \\ f_2(x_1,x_2) &= x_1^2 - x_2 \qquad\quad\;\, = 0 \end{aligned}$$

with initial box:

$$\mathbf{X}_0 = \begin{pmatrix} [-2,2] \\ [-2,2] \end{pmatrix}$$

This problem has three solutions in the box.

This system does not exhibit such severe interval dependencies as the previous example, so the derived system should not do so much better than the original system.

We obtained the following actual results when we ran the algorithm on an IBM 3090.

Our Previous Example

Method	# Boxes	CPU	Arith. Ops.
Usual	119	.21	5628
Oper. Decomp.	1	.06	394

The Cubic–Parabola Problem

Method	# Boxes	CPU	Arith. Ops.
Usual	21	.040	736
Oper. Decomp.	7	.077	561

Note: The number of arithmetic operations above was computed formally from accumulated numbers of certain subroutine calls, and is mainly of theoretical importance. CPU time is seconds on the IBM 3090.

More extensive experimentation is dependent upon our automating creation of the expanded system (as in Example 2.1(b)) from the original system (as in Example 2.1).

5. On a Theory

Interval methods for bounding solution sets of nonlinear systems of equations as in (1.1) can have one or more of the following three goals.

1. Computationally determine that a solution *exists* within the initial box $\mathbf{X}_0$.
2. Computationally determine that a solution within the initial box $\mathbf{X}_0$ is *unique*.
3. Determine new bounds $\mathbf{X}_1$ such that $\mathbf{X}_1$ is as small as possible subject to the condition that $X \in \mathbf{X}_0$ and $F(X) = 0$ implies $X \in \mathbf{X}_1$.

Algorithm 3.1–3.2 is based on heuristic considerations for approximately achieving goal 3, and the numerical experiments above indicate that it may do this in various contexts. In fact it is not difficult to show that a variant of Algorithm 3.2 alone will do at least as well as the optimally preconditioned interval Gauss–Seidel method applied to the original system alone. To do this, we first recall the width-optimal preconditioner of formula (3.5) in [**11**] (also stated more generally as formula (6) in [**19**]).

Definition 5.1. *Let F be as in* (1.1), *and assume we are to replace F by Y_iF in the computation* (1.6), *for some row vector Y_i. Also let $\mathbf{B} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n)^T$ and let X be the midpoint vector of $\mathbf{B}$. Then the* **width optimal** *LP preconditioner Y_i^C is defined to be*

$$Y_i^C = (y_1' - y_1'', y_2' - y_2'', \ldots, y_n' - y_n''),$$

where the y_j' and y_j'' are defined by

$$\text{(5.1a)} \quad \textit{minimize } W(y_1', \ldots, y_n', y_1'', \ldots, y_n'', u_1, \ldots, u_{i-1}, u_{i+1}, \ldots, u_n) = \sum_{\substack{j=1 \\ j\neq i}}^{n} \left(-\sum_{l=1}^{n} y_l' \underline{f}_{l,j}' + \sum_{l=1}^{n} y_l'' \overline{f}_{l,j}' + u_j \right) w(\mathbf{x}_j)$$

subject to

(5.1b)

$$u_j \geq \sum_{l=1}^{n} (y_l' - y_l'') \left(\underline{f}_{l,j}' + \overline{f}_{l,j}' \right), \quad 1 \leq j \leq n, \;\; j \neq i,$$

(5.1c)

$$1 = \sum_{l=1}^{n} y_l' \underline{f}_{l,i}' - \sum_{l=1}^{n} y_l'' \overline{f}_{l,i}',$$

and

$$\text{(5.1d)} \qquad y_l', y_l'', u_j \geq 0 \qquad \textit{for } 1 \leq j \leq n,\, j \neq i,\, l \leq 1 \leq n.$$

In [**19**], conditions are stated under which this preconditioner minimizes the width $w(\tilde{\mathbf{x}}_i)$ in (1.6) over all preconditioners Y; this preconditioner has also been shown superior in practice, as illustrated in [**11**].

To apply (5.1) theoretically in the context of the expanded system, we observe that the outer sum in (5.1a) is a sum over the rows of the interval Jacobi matrix, excluding the i-th row, and the conditions in (5.1b) also correspond to these rows. Because of this, the linear programming problem (5.1) may be reformulated for rectangular systems with more equations than unknowns. In particular, suppose that there are n equations $f_i(x_1, \ldots, x_n) = 0$, $1 \le i \le n$ in the original system and N equations $f_{i,E}(x_1, \ldots, x_n, v_{n+1}, \ldots, v_N) = 0$, $1 \le i \le N$. Then we may form the $(n+N) \times N$ system

$$(5.2)\qquad \begin{pmatrix} f_1(x_1, \ldots, x_n) \\ \vdots \\ f_n(x_1, \ldots, x_n) \\ f_{1,E}(x_1, \ldots, x_n, v_{n+1}, \ldots, v_N) \\ \vdots \\ f_{N,E}(x_1, \ldots, x_n, v_{n+1}, \ldots, v_N) \end{pmatrix} = \begin{pmatrix} \phi_1(v_1, \ldots, v_N) \\ \phi_{n+N}(v_1, \ldots, v_N) \end{pmatrix} = 0,$$

and corresponding linear programming problem

(5.3a) minimize

$$W(y_1', \ldots, y_{n+N}', y_1'', \ldots, y_{n+N}'', u_1, \ldots, u_{i-1}, u_{i+1}, \ldots, u_n) = \sum_{\substack{j=1 \\ j \ne i}}^{n} \left(-\sum_{l=1}^{n+N} y_l' \underline{\phi}_{l,j}' + \sum_{l=1}^{n+N} y_l'' \overline{\phi}_{l,j}' + u_j \right) w(\mathbf{x}_j)$$

subject to

(5.3b)

$$u_j \ge \sum_{l=1}^{n+N} (y_l' - y_l'') \left(\underline{\phi}_{l,j}' + \overline{\phi}_{l,j}' \right), \quad 1 \le j \le n+N, \;\; j \ne i,$$

(5.3c)

$$1 = \sum_{l=1}^{n+N} y_l' \underline{f}_{l,i}' - \sum_{l=1}^{n+N} y_l'' \overline{f}_{l,i}',$$

and

$$(5.3\text{d})\qquad y_l', y_l'', u_j \ge 0 \qquad \text{for } 1 \le j \le n,\ j \ne i,\ l \le 1 \le n+N.$$

We have

THEOREM 5.1. *Suppose* $F = (f_1, f_2, \ldots, f_n)^T$ *and* $F_E = (f_{1,E}, \ldots, f_{N,E})^T$ *are as above. Suppose further that, for some* i, $1 \le i \le n$, *the conditions in* [**19**] *are met so that*

$$W(y_1', \ldots, y_n', y_1'', \ldots, y_n'', u_1, \ldots, u_{i-1}, u_{i+1}, \ldots, u_n) = w(\tilde{\mathbf{x}}_i)$$

in (5.1), *and that this value is minimal over all preconditioner rows. Also suppose that* $\mathbf{v}_{n+1}$ *through* $\mathbf{v}_N$ *have been assigned through the forward substitution process explained in* §2. *Then*

(i) *If the* i*-th component of* $\mathbf{B}$ *is replaced by* $\tilde{\mathbf{x}}_{i,E} \cap \mathbf{x}_i$, *where* $\tilde{\mathbf{x}}_{i,E} = \mathbf{v}_i$ *is computed via the preconditioner defined by* (5.3), *then any solutions of* $F = 0$ *in the old* $\mathbf{B}$ *are in the new* $\mathbf{B}$.

(ii) *If* $\tilde{\mathbf{x}}_{i,E}$ *is computed from the system* (5.2) *preconditioned with a solution of* (5.3), *and* $\tilde{\mathbf{x}}_i$ *is computed from the original system preconditioned with a solution of* (5.1), *then* $w(\tilde{\mathbf{x}}_{i,E}) \le w(\tilde{\mathbf{x}}_i)$.

SKETCH OF PROOF: Assertion (i) follows from an argument similar to the proof of Theorem 1.4 of [**10**].

To prove assertion (ii), let $Y_i = (y_1, y_2, \ldots, y_n)$ be the preconditioner obtained from (5.1), and define $\tilde{Y}_{i,E} \in \mathbb{R}^{n+N}$ by

$$\begin{aligned}\tilde{Y}_{i,E} &= (\tilde{y}_1, \tilde{y}_2, \ldots, \tilde{y}_{n+N})\\ &= (y_1, y_2, \ldots, y_n, 0, 0, \ldots, 0).\end{aligned}$$

Then, setting $y_l' = \tilde{y}_l^+$ and $y_l'' = \tilde{y}_l^-$, we will see that $\tilde{Y}_{i,E}$ corresponds to a feasible point of (5.3), and the corresponding Gauss–Seidel result is $\tilde{\mathbf{x}}_i$, and has the property

$$W(\tilde{y}_1', \ldots, \tilde{y}_{n+N}', \tilde{y}_1'', \ldots, \tilde{y}_{n+N}'', \tilde{u}_1, \ldots, \tilde{u}_{i-1}, \tilde{u}_{i+1}, \ldots, \tilde{u}_{n+N}) = w(\tilde{\mathbf{x}}_i).$$

Thus the minimum W in (5.3) is at most $w(\tilde{\mathbf{x}}_i)$. Using techniques from [**19**], we may then show that this minimum W corresponds to a $Y_{i,E}$ such that $w(\tilde{\mathbf{x}}_{i,E})$ is also at most $w(\tilde{\mathbf{x}}_i)$. ∎

Our experience to date indicates that Theorem 5.1 can probably be strengthened. In particular, it should not be necessary to include the original equations, as in

CONJECTURE 5.2. *Let* $F_E = (f_{1,E}, \ldots, f_{N,E})^T$ *be as above, and let* $1 \le i \le n$. *Define* $\tilde{\mathbf{x}}_{i,\hat{E}}$ *to be the image of the* $i-th$ *coordinate under the preconditioned interval Gauss–Seidel method, when the preconditioner is computed via* (5.1), *but where* F_E *replaces* F. *Then* $w(\tilde{\mathbf{x}}_{i,\hat{E}}) \le w(\tilde{\mathbf{x}}_i)$, *where* $\tilde{\mathbf{x}}_i$ *is computed from the preconditioned interval Gauss–Seidel method applied to the original* F *and with preconditioner as in* (5.1).

We also state

THEOREM 5.3. *Suppose coordinates $\tilde{\mathbf{X}} = (\tilde{x}_1, \ldots, \tilde{x}_n)^T$ have been produced from application of Algorithm 3.1–3.2, starting with $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n)^T$, where the intermediate variables $\mathbf{v}_j$, $n+1 \le j \le N_E$ have been produced with the forward substitution process. Then any roots of F in $\mathbf{X}$ must also be in $\tilde{\mathbf{X}}$.*

SKETCH OF PROOF: The components of $\mathbf{X}$ are changed only in steps 1c)(ii) and 3b)(iii)(β) of Algorithm 3.1, as well as steps 1c), 3b)(iii), and 3c)(iii) of Algorithm 3.2. The proof can proceed formally by induction on the number of times one of these four steps is executed. For the case of steps 1c) and 3b), we use fundamental properties of interval arithmetic, and the fact that the root must satisfy the relationships among the intermediate quantities in the equations. For the case of step 3c), we use the fact that the image under the preconditioned interval Gauss–Seidel method must contain the roots in the original box; this fact is well-known. ∎

Interval existence tests are often based upon verification of some contraction mapping principle or, more weakly, on the Brouwer fixed point theorem. It may be possible to show that such a principle is valid for the practical algorithm 3.1–3.2, provided $\tilde{\mathbf{v}}_i \subset \mathbf{v}_i$ whenever an $\tilde{\mathbf{v}}_i$ is computed, but such a condition is probably more likely to occur if the pure preconditioned Gauss–Seidel algorithm (just step 3c) of Algorithm 3.2) is considered, as we cannot expect all of the individual nonlinear solution steps to be contractions.

Similarly, interval uniqueness tests are often based upon convergence of a fixed-point iteration, and is related to regularity of the Jacobi matrix over the box. If just preconditioned Gauss–Seidel is considered, then the known existence and uniqueness theorems for that method can be applied to the expanded system. However, relating the preconditioned expanded system to the preconditioned original system and using the fact that the individual entries of the Jacobi matrix for the expanded system are exact ranges (to within rounding out), we may be able to show that existence and uniqueness tests based on the expanded system are sharper.

6. COMPARISON WITH THE HANSEN–GREENBERG ALGORITHM

We present a practical discussion of the differences between the implementation just discussed and the interval root-finding algorithm, explained in [**5**] and in [**20**]. That particular root-finding algorithm, incorporating years of research, involves a combination of interval Gauss–Seidel steps, real iteration to obtain point estimates of roots, preconditioned interval Gaussian elimination, and possible generalized bisection.

The decomposition technique highlighted in this paper can be thought of as an additional tool to be included in such an overall algorithm. Indeed, the implementation in §5 is similar to the Hansen–Greenberg algorithm, but uses only the midpoint vector as a point estimate, and does only Gauss–Seidel steps (and not interval Gaussian elimination). On the other hand, the Hansen–Greenberg algorithm as defined in [**5**] uses only the inverse midpoint matrix as a preconditioner, and its Gauss–Seidel step may thus not be as effective. Also, use of point estimates is more important for interval Gaussian elimination, where small right hand sides matter, than for interval Gauss–Seidel iteration.

Our implementation in §4 does not include extended interval arithmetic (and splittings), as does the Hansen–Greenberg algorithm, although we have used this technique effectively in some of our algorithms ([**7**] and [**11**]). We omitted it in the experiments in §4 for simplicity (to reduce programming burden and to study the decomposition technique in a relatively simple environment), and also because extended interval arithmetic's usefulness depends strongly on where and how it is implemented.

Interval Gaussian elimination as in the Hansen–Greenberg algorithm would fail in cases in which we envision using the decomposition technique (such as Example 2.1(a)), but interval Gaussian elimination applied to the decomposed system may succeed. However, elimination of the intermediate variables in a particular order is equivalent to evaluation of the original Jacobi matrix! (See [**4**].) On the other hand, the decomposition technique may be of less value in cases where interval Gaussian elimination succeeds.

The underlying algorithm in which we embedded our new technique is as in [**10**]; experimental results were reported for the Hansen–Greenberg method in [**5**] on a problem reproduced as problem 17 in [**10**]. The Hansen Greenberg method required 46 Jacobi matrix evaluations and 88 interval function evaluations, while our underlying method (without the decomposition of this paper) required 38 Jacobi matrix evaluations and 83 interval function evaluations. Since tolerances and other factors were slightly different, these results can be considered roughly comparable; similarly, the actual overall speeds are difficult to assess independently from the machine, programming language, and implementation.

7. Summary, Applications, and Future Work

With examples and algorithmic descriptions, we have presented techniques for interval iterations based upon use of the expanded system derived from the code list. These techniques can sharpen bounds for initial data for which the preconditioned interval Gauss–Seidel method applied to the original system can not. Furthermore, in the algorithms, computation-intensive steps (preconditioned interval Gauss–Seidel) are often replaced by simpler steps (explicit function evaluation) which, in certain cases, may even lead to faster convergence to a point.

One very promising application is in robust geometric computation. We have come across various small systems of cubics in that context for which phenomena (i) and (ii) below (1.7) in §1 seem to play a large rôle. Thus, solutions to these systems could possibly be rigorously found and verified much more efficiently with Algorithm 3.1 and Algorithm 3.2 than with traditional interval Newton methods.

The following items point to possible future work on these methods.

1. Complete the theory along the lines of §5 above.
2. Develop an automatic parser to create the expanded system from the original equations, to enable easier experimentation and application. Possibly, this would be done in a language allowing operator overloading, such as C++ or Fortran 90.
3. Implement the process within an overall generalized bisection code such as INTBIS.
4. Implement the stacking operation, as mentioned in §2 below Algorithm 3.1, when the interval inverse of one of the elementary functions consists of a union of two disjoint intervals.
5. Integrate sparse linear programming solvers into the code.
6. Do more extensive experimentation, especially on applications-oriented examples.

Acknowledgement. I wish to thank the referee for the suggestions for clarification of the ideas in this paper, for help with the German abstract, and for patient reading of two revisions.

References

1. Alefeld, Götz, and Herzberger, Jürgen: Introduction to Interval Computations, New York, etc.: Academic Press 1983.
2. Böhm, H.: Evaluation of Arithmetic Expressions with Maximum Accuracy, in *A New Approach to Scientific Computation*, Academic Press, New York, 1983.
3. Duff, I. S., Erisman, A. M., Gear, C. W., and Reid, J. K.: Sparsity structure and Gaussian elimination. ACM SIGNUM Newsletter 23, 2–8 (1988)
4. Griewank, A.: Direct Calculation of Newton Steps without Accumulating Jacobians. Preprint, Mathematics and Computer Science Division, Argonne National Laboratories.

5. Hansen, E. R., and Greenberg, R. I.: An Interval Newton Method. Applied Mathematics and Computation 12, 89–98 (1983)
6. Hu, C.-Y., and Kearfott, R. B.: A Width Characterization and Row Selection Strategy for the Interval Gauss-Seidel Method. Preprint.
7. Hu, C.-Y.: Preconditioners for Interval Newton Methods: Ph. D. Dissertation, Univ. of Southwestern Louisiana 1990.
8. Kearfott, R. B.: Abstract Generalized Bisection and a Cost Bound. Math. Comp. 49, 187–202 (1987)
9. Kearfott, R. B.: Interval Newton / Generalized Bisection When There are Singularities near Roots. Annals of Operations Research 25, 181–196 (1990)
10. Kearfott, R. B.: Preconditioners for the Interval Gauss-Seidel method: SIAM J. Numer. Anal. 27, 804–822 (1990)
11. R. Baker Kearfott, Chenyi Hu, and Manuel Novoa III: A Review of Preconditioners for the Interval Gauss–Seidel Method, preprint.
12. Kearfott, R. B., and Novoa, M.: INTBIS, A Portable Interval Newton/Bisection Package ACM. Trans. Math. Software 16, 152–157 (1990)
13. Krawczyk, R., and Neumaier, A.: Interval Slopes for Rational Functions and Associated Centered Forms. SIAM J. Numer. Anal. 22, 604–616 (1985)
14. Moore, R. E., and Jones, S. T.: Safe Starting Regions for Iterative Methods. SIAM J. Numer. Anal. 14, 1051–1065 (1977)
15. Moore, Ramon E.: Methods and Applications of Interval Analysis, Philadelphia: SIAM 1979.
16. Neumaier, A.: Interval Iteration for Zeros of Systems of Equations. BIT 25, 256–273 (1985)
17. Neumaier, A.: Interval Methods for Systems of Equations, Cambridge: Cambridge University Press 1990.
18. Neumaier, A., and Z. Shen: The Krawczyk Operator and Kantorovich's Theorem. Mathematical Analysis and Applications (to appear)
19. Novoa, M.: Linear Programming Preconditioners for the Interval Gauss–Seidel Method and their Implementation in Generalized Bisection. Ph.D. Dissertation, Univ. of Southwestern Louisiana 1990.
20. Ratschek, H., and Rokne, J.: New Computer Methods for Global Optimization, New York: Wiley 1988
21. Z. Shen and Wolfe, M. A.: A Note on the Comparison of the Kantorovich and Moore Theorems: Math. Analysis: Theory, Methods, and Applications (to appear)

R. Baker Kearfott, Associate professor, Department of Mathematics, University of

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