EMPIRICAL EVALUATION OF INNOVATIONS IN INTERVAL BRANCH AND BOUND ALGORITHMS FOR NONLINEAR SYSTEMS

R. BAKER KEARFOTT*

Abstract. Interval branch and bound algorithms for finding all roots use a combination of a computational existence / uniqueness procedure and a tesselation process (generalized bisection). Such algorithms identify, *with mathematical rigor*, a set of boxes that contains unique roots and a second set within which all remaining roots must lie. Though each root is contained in a box in one of the sets, the second set may have several boxes in clusters near a single root. Thus, the output is of higher quality if there are relatively more boxes in the first set. In contrast to previously implemented similar techniques, a box expansion technique in this paper, based on using an approximate root finder, ϵ-inflation and exact set complementation, decreases the size of the second set, increases the size of the first set, and never loses roots.

In addition to the expansion technique, use of second-order extensions to eliminate small boxes that do not contain roots, and interval slopes versus interval derivative matrices are studied. These items are evaluated empirically on a significant test problem set, within a Fortran 90 environment designed for such purposes. The results are compared to previous results, and show that careful incorporation of the techniques yields both quantitatively and qualitatively superior computer codes.

Key words. nonlinear algebraic systems, branch-and-bound methods, interval computations, ϵ-inflation, slope matrices, Fortran 90, second order extensions

AMS subject classifications. 65-04, 65-05, 65G10, 65H20

1. Introduction, Background, and Motivation.

Interval arithmetic has proven itself useful in many contexts, for automatically supplying rigorous bounds on solutions or for verification of solutions that have been computed by approximate methods. Since it can supply rigorous bounds on ranges of functions, interval arithmetic has been particularly successful in branch and bound algorithms for global optimization and in finding all solutions to nonlinear systems of equations $F(X) = 0$, $F : \mathbb{R}^n \rightarrow \mathbb{R}^n$ within a given region. However, naive application easily leads to unsatisfactory results, and experts agree that interval arithmetic should appear only where necessary, and then, only appropriately.

In branch and bound methods for *rigorously* finding *all* roots of nonlinear algebraic systems of equations, details of both how the interval arithmetic enters and how the search process (branch and bound) is carried out have an extreme impact on the success of an implementation. The purpose of this paper is to investigate some of these details.

Branch and bound methods for nonlinear systems are roughly based on the following algorithmic steps.

ALGORITHM 1. (Skeletal branch and bound algorithm)

INPUT: initial bounds $[a_i, b_i]$, $1 \le i \le n$, defining the region $\mathbf{X}_0$, an internal representation for the function $F : \mathbf{X}_0 \subset \mathbb{R}^n \rightarrow \mathbb{R}^n$, and the tolerance ϵ_{d}.

OUTPUT: a list $\mathcal{R}$ of boxes such that each box $\mathbf{X} \in \mathcal{R}$ has been rigorously verified to contain a unique root of F, and a list $\mathcal{U}$ of boxes of maximum side lengths on the order of $\sqrt{\epsilon_{\mathrm{d}}}$, such that all remaining roots of F in $\mathbf{X}_0$ must lie in boxes in $\mathcal{U}$.

* Department of Mathematics, University of Southwestern Louisiana, U.S.L. Box 4-1010, Lafayette, LA 70504-1010 USA. email: rbk@usl.edu This work was supported in part by National Science Foundation grant CCR-9203730.

- *Place* $\mathbf{X}_0$ *into the stack* $\mathcal{S}$ *of regions to be considered.*
- DO WHILE $\mathcal{S}$ *is non-empty.*
  1. *Pop an item from* $\mathcal{S}$ *and let it become the current region* $\mathbf{X}$.
  2. *Use an iterative scheme (e.g. some form of interval Newton method) to replace* $\mathbf{X}$ *by a smaller box* $\tilde{\mathbf{X}}$ *such that all roots of* F *in* $\mathbf{X}$ *must lie in* $\tilde{\mathbf{X}}$.
  3. *Use an* inclusion test *to determine if* $\mathbf{X}$ *contains a unique root.*
  4. IF $\mathbf{X}$ *is proven to contain a unique root* THEN *Store* $\mathbf{X}$ *on a list* $\mathcal{R}$.
  5. IF $\mathbf{X}$ *is proven to contain no roots, then cycle to step 1.*
  6. IF *the diameter of* $\mathbf{X}$ *has been made smaller than* ϵ_{d} *in step 2, but the tests in steps 3 and 5 were inconclusive,* THEN *store* $\mathbf{X}$ *in a list* $\mathcal{U}$ *of boxes that may contain roots.*
  7. IF *there was no progress in steps 2, 3, 5, or 6* THEN *bisect* $\mathbf{X}$ *along a coordinate direction* i*, placing the resulting two regions on* $\mathcal{S}$.

  END DO

Algorithms with similar basic steps have appeared in [25], [5] and the references therein, and elsewhere. An abstract version of such an algorithm along with an analysis appears in [8].

Commonly the regions $\mathbf{X}$ are taken to be *boxes,* i.e. interval vectors or, geometrically, rectangular parallelepipeds. Along these lines, implementations generally involve some form of interval Newton method for steps 2, 3 and 5. The most common such interval Newton methods appear to be the Krawczyk method (from [20] and explained in [24] and [26]) and the interval Gauss–Seidel method (such as in [9] and [17]), although preconditioned interval Gaussian elimination could also be used (see [29]). Theory and practice indicate that the interval Gauss–Seidel method is somewhat better than Krawczyk's method for most purposes; see [29]. Here we use the interval Gauss–Seidel method.

Regardless of whether Krawczyk or interval Gauss–Seidel iteration is used, effectiveness depends on (i) the *preconditioner* employed in the corresponding linear system, and (ii) the *variation bound matrix* (interval Jacobi matrix, more general Lipschitz matrix or slope matrix) used in the interval Newton equation. The inverse midpoint preconditioner, with theoretical properties summarized in [29], is commonly used, while we have found preconditioners satisfying various types of optimality conditions can give better performance when solving nonlinear systems problems in small to moderate dimensions; see [11] and [16].

To date, we have preferred interval Jacobi matrices to slopes, since it is simple to incorporate them in a computational uniqueness test, such as in [24] or in [29, Theorem 5.1.7]. Interval slopes provide tighter bounds and faster convergence in step 2, but only allow existence (and not uniqueness) to be shown when used in a straightforward manner. However, Rump has recently pointed out (in [32]) that slopes can be effective in a two-step process in which we first verify existence in as small a region as possible, then verify uniqueness in as large a region as possible. This *slope-verification* procedure shows much promise.

There are also various ways of choosing the coordinate direction i in step 7 of Algorithm 1. In this work, we use the *maximal smear* scheme proposed in [17]: We choose i such that $s_i = \min_{j \in \mathcal{C}} s_j$, where

$$s_j = \max_{1 \le k \le n} \{ |\mathbf{S}_{k,j}(F, X_a, \mathbf{W})| (\overline{w}_j - \underline{w}_j) \}.$$

This takes the function variation into account and works better than bisecting the coordinate direction of (unscaled) maximum width.

The exclusion test in step 5 of Algorithm 1 combines both an interval Newton method and an interval function evaluation. In the interval Newton method, if the image $\mathbf{N}(\mathbf{X})$ of a region $\mathbf{X}$ is disjoint from the region $\mathbf{X}$, then there cannot be any solutions of F within $\mathbf{X}$. In the function evaluation, we obtain a region $\mathbf{F}(\mathbf{X})$ that contains the range of F over the region $\mathbf{X}$; F has no roots in $\mathbf{X}$ if $\mathbf{F}(\mathbf{X})$ does not contain the zero vector. In [15], theoretical and empirical analysis indicates that the *order* of the interval extension (reviewed in §4 below) of an objective function in global optimization greatly affects the ability of the algorithm to exclude regions. We examine whether this is true for nonlinear systems in §4.

Finally, due to overestimation in the variation bound matrix and in the interval extension of the function, sometimes combined with poor conditioning, steps 2 and 7 of Algorithm 1 may result in many boxes that can neither be verified as root-containing nor be rejected as not containing roots. As in the global optimization case analyzed in [15], such boxes may consist of clusters about roots. The algorithm's output is more difficult to comprehend when such clusters occur: a list of boxes each of which has been verified to contain a unique root is more meaningful than a large list of small regions such that all of the roots must lie within boxes in the list. Furthermore, a proliferation of small regions can cause a fatal increase in the computational effort of the algorithm.

In [8], we proposed a general scheme to eliminate boxes in a cluster, and proved, under technical assumptions, that such a scheme resulted in a list of boxes, each of which contained a unique root, and such that all roots were contained in boxes in the list. This scheme, based roughly on replacing each small box entering step 6 by a larger one, then deleting all boxes in the list that intersect the larger box, was implemented heuristically in the Fortran-77 program INTBIS of [17]. The scheme was not totally successful at eliminating clusters. Furthermore, since the process was based on theoretical assumptions from [8] that were not computationally verified, it also could possibly eliminate a box containing a root, in favor of a neighboring box[1].

In [10], we described a form of *trisection*, in which a small box was cut exactly from a larger box, to remove points where the function was nearly zero and the Jacobi matrix was ill-conditioned. The algorithms in [10] also used the fact that the classical Newton method would often converge to roots where the Jacobi matrix is singular, even though interval verification processes fail there; the removed box portions were constructed around approximate solutions so found. Later, in [18] and [33], reminiscent of the trisection process, we developed a *geometrical complementation algorithm*, in which we could form a new list of boxes from an old list, such that the union of the elements of the new list was the complement of a given box in the old list. In [18] we used the process to eliminate from the search space portions of a curve that had already been found.

In fact, the list complementation process from [18] can be combined with the box expansion idea from [17] to allow us to rigorously eliminate clusters, without discarding roots. Furthermore, use of a classical root-finder (such as a "globalized" quasi-Newton method, as in [27]) need not be confined to the search for roots where the Jacobi matrix is ill-conditioned or singular. Assuming that it is easier to find an approximate solution and then verify it using interval arithmetic, rather than to find a verified solution from the beginning with interval arithmetic, we could: (i) Try wherever possible to find approximate roots; (ii) construct as large a box as possible,

[1] INTBIS provided an option to disable this.

about each such approximate root, in which we can verify that there is a unique solution; (iii) take the geometrical complement of each such box within the stack $\mathcal{S}$ and list $\mathcal{U}$ of Algorithm 1. Such a procedure may result in reduced clustering and lower execution time. On the other hand, the complementation process replaces a single box in the stack or list by up to $2n$ new boxes (cf. §5 below); though these boxes may be processed more easily, the larger number of boxes could be costly. Also, running the approximate solver takes CPU time. Thus, experimentation is needed.

The idea of first obtaining an approximate solution, then verifying it, is ubiquitous throughout interval computations. In fact, it can be considered one of the two paradigms in algorithms with automatic result verification: In the first, existence is assumed, and bounds on the solution are refined, while existence is verified *a posteriori* (after computation of an approximate solution) in the second. Verification of approximate solutions began with Krawczyk [20] and Moore [24], and continued with the introduction of fixed point theory by Rump [31]. In global search algorithms for nonlinear systems, besides in [10], it has been used in the univariate global optimization algorithm proposed by Caprani and Madsen in [3] and in the multivariate global optimization algorithm in [7], and is discussed in [29, p. 211].

The goals of the present study are to: (i) examine the practical worth of the approximate root-finding / complementation process; (ii) examine the practicality of second-order extensions versus first-order extensions; (iii) determine the usefulness of interval slopes and Rump's slope-based uniqueness test; (iv) compare the implementation of these innovations (in a new environment) to our previous implementations in [17] and [11]. We first define our notation, then explain slope-based computational uniqueness tests, second-order interval extensions, and the tesselation process. We then give our overall algorithm pattern in §6, while experimental results appear in §7. We summarize in §9.

We assume prior familiarity with interval analysis, and will not repeat elementary details. Introductions to the field are [25] or [1], while a treatise on the theory of interval methods for nonlinear systems is [29]. A well-prepared up-to-date general review of advanced aspects of the subject is [6].

2. Notation. Throughout, we will use boldface to denote intervals, lower case to denote scalar quantities, and upper case to denote vectors and matrices. We will use underscores to denote lower bounds of intervals and overscores to denote upper bounds of intervals. For components of vectors, we will use corresponding lower case letters. For example, we may have:

$$\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n]^T,$$

where $\mathbf{x}_i = [\underline{x}_i, \overline{x}_i]$. The notation $\check{x}$ will denote the midpoint of the interval $\mathbf{x}$. The *magnitude* of an interval is defined as $|\mathbf{x}| = \max\{|\underline{x}|, |\overline{x}|\}$.

The width of an interval $\mathbf{x}$ will be denoted by $w(\mathbf{x}) = \overline{x} - \underline{x}$, and the width of an interval vector $\mathbf{X}$, denoted $w(\mathbf{X})$, will be defined componentwise. We will use $w(\mathbf{X})$ in the context of $\|w(\mathbf{X})\| = \|w(\mathbf{X})\|_\infty$.

The symbol $\mathbf{F}_1(\mathbf{X})$ will denote a natural (first-order) interval extension of F over $\mathbf{X}$, where $F : \mathbb{R}^n \to \mathbb{R}^n$, while $\mathbf{F}_2$ will denote the second-order extension described in §4 of this paper. $(\mathbf{F}_1(\mathbf{X}))_i$ will denote the i-th component of $\mathbf{F}_1(\mathbf{X})$. The exact range of F over $\mathbf{X}$ will be denoted[2] by $\mathbf{F}_u(\mathbf{X})$.

Whenever $\|\cdot\|$ is used, it will mean $\|\cdot\|_\infty$.

[2] suggesting "united extension," a term first used by Moore

We will use calligraphic letters such as $\mathcal{S}$, $\mathcal{U}$ and $\mathcal{R}$ to denote stacks and lists of boxes.

Brackets $[\cdot]$ will be used to delimit intervals, while parentheses $(\cdot)$ will delimit matrices and vectors.

3. Slope-Based Uniqueness and Epsilon Inflation. Depending on goals, it is possible to craft various branch and bound root-finding algorithms. For example, to obtain merely a list of boxes, the union of which must contain all roots and the total measure of which is less than the measure of the initial region, a computational uniqueness test is not necessary. However, to isolate all roots or determine the precise number of roots, such a computational uniqueness test is crucial. Furthermore, accepting as large a box as possible in which uniqueness can be proven leads to a faster, more practical algorithm. Thus, the work here is based on trying to prove uniqueness wherever possible.

Until recently, computational uniqueness tests were based on *Lipschitz* matrices, defined in [29, p. 174]. For example, any componentwise interval extension of the Jacobi matrix[3] of F over $\mathbf{X}$ is a Lipschitz matrix for F over $\mathbf{X}$. Theory such as [29, Theorem 5.1.7] indicates that interval Newton methods based on the linearized system

$$\mathbf{A}(\tilde{\mathbf{X}} - \check{X}) = -F(\check{X}) \tag{1}$$

prove uniqueness when the computed $\tilde{\mathbf{X}}$ is a subset of $\mathbf{X}$, provided $\mathbf{A}$ is a Lipschitz set for F over $\mathbf{X}$.

On the other hand, the entries of *slope matrices* generally have smaller widths than those of corresponding Lipschitz matrices, and thus are more likely to lead to $\tilde{\mathbf{X}} \subset \mathbf{X}$ when bounding solutions of Equation 1. In [32, §3], [21] and other works, we see

DEFINITION 1. *Let* $F : \mathbf{D} \subseteq \mathbf{R}^n \to \mathbf{R}^n$ *be a continuous function and let* $\mathbf{X} \subseteq \mathbf{D}$ *and* $\mathbf{X}_c \subseteq \mathbf{D}$. *An interval matrix* $\mathbf{S}(F,\mathbf{X},\mathbf{X}_c)$ *with* $\mathbf{X}_c \subseteq \mathbf{X}$ *is called a* slope matrix *for* F *over* $\mathbf{X}$ *at* $\mathbf{X}_c$ *if* $\forall X \in \mathbf{X}$, $\forall X_c \in \mathbf{X}_c$, $\exists S \in \mathbf{S}$ *such that* $F(X) - F(X_c) = S(X - X_c)$.

However, if $\mathbf{S}(F, \mathbf{X}, \check{X})$ is substituted for the Lipschitz matrix $\mathbf{A}$ in equation (1), then $\tilde{\mathbf{X}} \subset \mathbf{X}$ only implies that there *exists* a solution of $F(X) = 0$ within $\mathbf{X}$, and not that this solution is unique; see [29, Corollary 5.4.3] and [32, Fig. 2.1]. Nonetheless, Rump has recently proposed a scheme that can prove uniqueness, using only slopes, and can possibly be more effective than verification based on interval Jacobi matrices. We identify it as:

ALGORITHM 2. (Algorithm 2.1 in [32])

INPUT: an initial guess for an approximate root of F

OUTPUT: an approximate root X_a, a box $\mathbf{X}_a \ni X_a$ in which there must exist a root, and a box $\mathbf{W} \supseteq \mathbf{X}_a$ such that F has a unique root in $\mathbf{W}$

Use a two-stage process, involving only slopes, to verify existence in a small box containing X_a *and uniqueness in a larger box.*

End Algorithm 2

Algorithm 2 is practical because slopes can be implemented in a process similar to automatic differentiation, so the user need only program the function itself. The review [32] contains tips on how to do this efficiently and with small output intervals; we have implemented such slope computations as a subroutine in our Fortran 90 system of [13].

[3] e.g. that natural interval extension obtained by a given form of automatic differentiation and evaluation with interval arithmetic

Obtaining $\mathbf{X}_a$ and $\mathbf{W}$ involves a process termed *ϵ-inflation*, originated by Rump in [31] and further described by Mayer, e.g. in [22]. This process works by constructing a small box *centered* at X_a, then expanding it until existence (or uniqueness) can be verified. To find just one root, this process potentially is much less costly than beginning with a large box and tesselating it; we investigate this possibility experimentally in §7.

The ϵ-inflation algorithm in the experiments of §7 is based on Algorithm 2. In this algorithm, in [32], an iterative process is applied to X_a, once found, to reduce its size. This process is not explicit in the following algorithm, although it is an option in the code; see §7 below.

ALGORITHM 3. (Find an approximate root, and verify uniqueness within as large a box as possible about that root.)

INPUT: the initial box (overall bounds) $\mathbf{X}_0$, the current box $\mathbf{X}_c \subseteq \mathbf{X}_0$*, and the smallest box size* ϵ_{d} *produced by bisection in the overall branch and bound algorithm.*

OUTPUT: one of the following: (1) an approximate root $X_a \in \mathbf{X}_c$ and a box $\mathbf{W}$ containing X_a, $\mathbf{W} \subseteq \mathbf{X}_0$, such that F has a unique root in $\mathbf{W}$*; (2) an approximate root* $X_a \in \mathbf{X}_c$ *and a box* $\mathbf{W}$ *containing* X_a *such that existence of a root within* $\mathbf{W}$ *has been proven; (3) an approximate root* X_a *(according to the approximate solver), but no box* $\mathbf{W}$*; (4) failure to compute an approximate root.*

1. *Depending on the widths of the coordinates of* $\mathbf{X}_c$ *and* ϵ_{d}*, either find* X_a *with the MINPACK1 routine HYBRD1 or take* X_a *to be the center of* $\mathbf{X}_c$*.*
2. IF $X_a \notin \mathbf{X}_c$ *or no solution is found* THEN **EXIT**.
3. *Construct a small box* $\mathbf{X}_a$*,* $X_a \in \mathbf{X}_a$*, in which existence can be proven as in Algorithm 2.*
4. (Verify uniqueness within as large a box as possible) IF *existence was verified in step 3* THEN*, expanding the widths of* $\mathbf{X}_a$ *coordinate by coordinate, construct as large a box* $\mathbf{W} \supseteq \mathbf{X}_a$ *as possible in which uniqueness can be proven.*

End Algorithm 3

4. Second-order Interval Extensions. Natural interval extensions $\mathbf{F}_1$ of F, that is, extensions obtained by directly evaluating the expressions[4] for F using interval arithmetic, are *first order* in the sense that

$$\|w(\mathbf{F}_1(\mathbf{X}))\| - \|w(\mathbf{F}_u(\mathbf{X}))\| = \mathcal{O}(\|w(\mathbf{X})\|). \tag{2}$$

On the other hand, *second order* extensions obey

$$\|w(\mathbf{F}_2(\mathbf{X}))\| - \|w(\mathbf{F}_u(\mathbf{X}))\| = \mathcal{O}(\|w(\mathbf{X})\|)^2. \tag{3}$$

For example, with the mean value theorem and slopes, the interval extension $\mathbf{F}_2$ defined by

$$\mathbf{F}_2(\mathbf{X}) = F(\check{X}) + \mathbf{S}(F, \mathbf{X}, \check{X})(\mathbf{X} - \check{X}) \tag{4}$$

is second order under natural conditions (cf. [29, §2.3, Corollary 2.3.4])[5].

Condition 3 is better than Condition 2 for determining $0 \notin F(\mathbf{X})$ for small $w(\mathbf{X})$, but $\mathbf{F}_2$ requires more work to evaluate than $\mathbf{F}_1$. The following algorithm combines the two.

[4] including, possibly, loops and subroutines

[5] Various interval extensions, of first and second order, are detailed in [30].

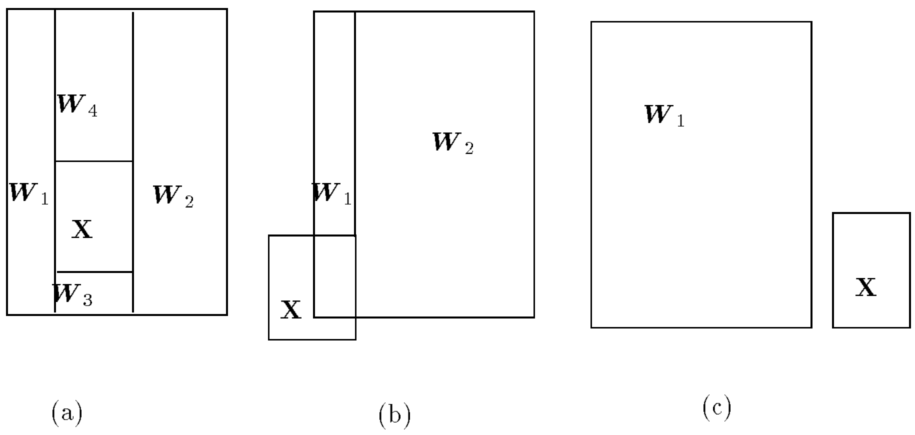

FIG. 1. *Complementation of a box* $\mathbf{X}$ *in a box* $\mathbf{W}$

ALGORITHM 4. (Try a first-order, then second-order function evaluation.)

INPUT: An internal representation for F and the current box $\mathbf{X}$

OUTPUT: either "unknown" or "no root in $\mathbf{X}$"

1. *Compute the natural first-order extension* $\mathbf{F}_1(\mathbf{X}_c)$ *to* F*;* IF $0 \not\in \mathbf{F}_1(\mathbf{X}_c)$ THEN RETURN "no root in $\mathbf{X}$."
2. *Compute (a more costly) second-order extension* $\mathbf{F}_2(\mathbf{X}_c)$ *to* F*, based on the mean value-extension with a slope matrix*[6]*;* IF $0 \not\in \mathbf{F}_2(\mathbf{X}_c)$ THEN RETURN "no root in $\mathbf{X}$."
3. *If neither step 1 nor step 2 verified* $0 \not\in \mathbf{F}_u(\mathbf{X}_c)$*, then* RETURN "unknown."

End Algorithm 4

Actually, Step 2 determines that there is no root in $\mathbf{X}$ precisely when the image of the Jacobi method does not intersect $\mathbf{X}$. Thus, Step 2 is probably best implemented by performing a step of the Gauss–Seidel method. Although the Gauss–Seidel method is used as an overall iteration procedure in the algorithm, Step 2 is not superfluous, since it involves recomputation of a slope matrix and checking at points in the branch and bound process where it would not otherwise be done, i.e. in steps 2 and 4(a) of Algorithm 5 and Step 3(c)i of Algorithm 7. In these places, it is conceptually appropriate to think of it as a second-order function evaluation.

5. The Complementation Process. The algorithms that generate the lists of boxes consist of generalized bisection and of taking the complement of a box in a list of boxes. Generalized bisection is step 7 of Algorithm 1 in the introduction, while the algorithms for taking the complement of a box in a box and for generating a list of boxes whose union is the complement of a box in the union of the boxes in an original list first appeared in [33] and [18], but had not been tried in general nonlinear systems codes. Details are available from the author. The important observation here is that a list of at most $2n$ boxes can be produced by complementing a box $\mathbf{X}$ in a box $\mathbf{W}$ with a simple $\mathcal{O}(n)$ computation. Figure 1 illustrates the process.

6. Overall Algorithm Pattern. The overall algorithm is based on the ϵ-inflation of Algorithm 3, on the basic generalized bisection structure in [8] and [17], and on taking the complement of the boxes produced by Algorithm 3 in the lists of

---

[6] or do a step of the Jacobi method: See the remark below this algorithm.

boxes produced in the generalized bisection structure. A crucial question is where in the overall algorithm to find and verify approximate roots (i.e. where to apply Algorithm 2). We can attempt to find another approximate root at the beginning and whenever a box is cut through bisection or complementation, or we can do so merely when small boxes that have not been verified are produced.

We use a heuristic, based on a parameter α between 0 and 1, to decide when to attempt approximate root finding: $\alpha = 0$ implies that Algorithm 3 is always attempted, while $\alpha = 1$ implies Algorithm 3 is never attempted, except for starting points within boxes of diameters less than the domain tolerance. Specifically, approximate root-finding and verification are attempted within a box with relative diameter greater than $\epsilon_{\rm d}$ whenever $\min\{|\underline{F}_i|, |\overline{F}_i|\}/\max\{|\underline{F}_i|, |\overline{F}_i|\} < \alpha$ for each component $\mathbf{F}_i = [\underline{F}_i, \overline{F}_i]$ of $\mathbf{F}$ is larger than α. The presumption is that when zero is centered in the interval estimate for the range, it is more likely that the actual range, without overestimation, contains zero; cf. step 1 of Algorithm 7.

The following algorithm embodies these considerations.

ALGORITHM 5. (Overall Tesselation / Complementation process)

INPUT: the initial box $\mathbf{X}_0$, a symbolic representation for the function F, the heuristic parameter α, the maximum allowable number of boxes M to be processed, and a domain tolerance $\epsilon_{\rm d}$

OUTPUT: *If the search was successful:* a list $\mathcal{R}$ such that each box $\mathbf{X} \in \mathcal{R}$ has been verified to contain a unique root, and a list $\mathcal{U}$, each of whose boxes has relative side lengths on the order of $\sqrt{\epsilon_{\rm d}}$, such that all roots of F in $\mathbf{X}_0$ not in boxes in $\mathcal{R}$ are in boxes in $\mathcal{U}$. *If the search did not complete with M boxes processed:* a list $\mathcal{R}$ as above, a list $\mathcal{U}$ of boxes with diameters on the order of $\sqrt{\epsilon_{\rm d}}$ that may contain roots, and a stack $\mathcal{S}$ of boxes in the set complement of the union of the boxes in $\mathcal{R}$ and $\mathcal{U}$ that have not been fully analyzed.

- *Place $\mathbf{X}_0$ onto the stack $\mathcal{S}$.*
- **Overall box processing loop:** DO $k = 1$ to M WHILE $\mathcal{S} \neq \emptyset$.
  1. *Remove the first box to have been placed in $\mathcal{S}$ to obtain the current box $\mathbf{X}_c$.*
  2. (Check function and find as many approximate roots as possible first.) DO WHILE *this step results in a change in $\mathcal{S}$, $\mathcal{R}$ or $\mathcal{U}$:*
     (a) *Perform Algorithm 4;* IF $0 \notin \mathbf{F}_u(\mathbf{X}_c)$ *is verified* THEN CYCLE **overall box processing loop**.
     (b) (Find approximate roots in current box) IF $\omega(\mathbf{X}_c) > \epsilon_{\rm d}$ THEN *Find and verify an approximate root within $\mathbf{X}_c$ using Algorithm 3; modify the stack $\mathcal{S}$, the list $\mathcal{R}$ of root-containing boxes and the list $\mathcal{U}$ of small boxes of unknown status by taking the complement of the verified boxes $\mathbf{W}$ in these lists, using Algorithm 7.*
     (c) *If $\mathcal{S}$ has been altered, then remove the first box placed in $\mathcal{S}$, to obtain the current box $\mathbf{X}_c$.*

     END DO
  3. *Reduce the widths of the current box with Gauss–Seidel iteration and bisection (Algorithm 6.)*
  4. (Boxes exiting Algorithm 6 either are proven to have no roots, or are small and of "unknown" status.) IF *the status of $\mathbf{X}_c$ is still "unknown,"* THEN
     (a) *Check the function values: perform Algorithm 4;* IF $0 \notin \mathbf{F}_u(\mathbf{X}_c)$ *is verified* THEN CYCLE **overall box processing loop**.
     (b) (Unconditionally attempt to find a root in the small box.)
        i. *Find and verify approximate roots within $\mathbf{X}_c$, and modify the stack $\mathcal{S}$, the list $\mathcal{R}$ of root-containing boxes and the list $\mathcal{U}$ of small boxes of unknown status, using Algorithm 7. However, use $\tilde{\alpha} = 0$, rather than the heuristic parameter α in step 1 of Algorithm 7, to make sure an attempt is made.*

ii. IF $\mathcal{S}$ *is empty upon return from Algorithm 7,* THEN EXIT **overall box processing loop**

(c) (Attempt to avoid clusters at singular or ill-conditioned (near) roots by artificial expansion.) IF *step 4(b)i did not result any change in* $\mathbf{X}_c$, $\mathcal{R}$ *or* $\mathcal{U}$

THEN

i. For $i = 1$ *to* n *replace* $\mathbf{x}_{c,i}$ *by* $[\underline{x_{c,i}} - \sigma_i, \overline{x_{c,i}} + \sigma_i]$ *where* $\sigma_i = 1/\sqrt{\epsilon_d}\max\{|\mathbf{x}_{c,i}|\epsilon_d, \sqrt{\epsilon_m}\}$, *where* ϵ_m *is the machine epsilon.*

ii. Take the complement of $\mathbf{X}_c$ *in* $\mathcal{S}$ *and in* $\mathcal{U}$.

iii. Insert $\mathbf{X}_c$ *into* $\mathcal{U}$.

ELSE *Push* $\mathbf{X}_c$ *onto* $\mathcal{S}$.

END IF (Cluster avoidance)

END DO **overall box processing loop**

- IF k *has exceeded* M *in the overall processing loop*
  THEN *print* $\mathcal{S}$, $\mathcal{R}$, *and* $\mathcal{U}$
  ELSE *return* $\mathcal{R}$, $\mathcal{U}$ *and performance statistics.*

End Algorithm 5

The expansion factor in step 4(c)i of Algorithm 5 is crucial. Our motivation for it is that, speaking roughly, roots corresponding to singular Jacobi matrices can only be computed with accuracy proportional to the square root of the machine precision. Additionally, this expansion factor works well in practice (tending to cause each root to be isolated in a single box), whereas smaller or larger ones do not.

Step 3 (to reduce the size of a box in overall Algorithm 5 and to reject or verify boxes as in [17]), appears below as Algorithm 6. Details of the Gauss–Seidel sweep, a variant of the algorithm explained in e.g. [11], are omitted, but are available from the author upon request. Algorithm 6 calls for bisection if Gauss–Seidel is unsuccessful, and applies the approximate root-finding procedure (Algorithm 2) when a new box is produced from bisection.

ALGORITHM 6. (Process the current box $\mathbf{X}_c$ in Algorithm 5.)

INPUT: the current box $\mathbf{X}_c$, the internal symbolic representation for F, and the current stack $\mathcal{S}$

OUTPUT: (1) a new or altered box $\mathbf{X}_c$; (2) the status "unknown" or "has no root" associated with $\mathbf{X}_c$, such that, if the status of $\mathbf{X}_c$ is "unknown", then the maximum relative width of a coordinate of $\mathbf{X}_c$ is on the order of ϵ_d; (3) a (possibly) altered stack $\mathcal{S}$

DO WHILE $\omega(\mathbf{X}_c) > \epsilon_d$, *where* $\omega(\mathbf{X}_c)$ *is the relative diameter of* $\mathbf{X}_c$ *and* ϵ_d *is the domain tolerance.*

1. *Compute the slope matrix* $\mathbf{S}(F, \check{X}_c, \mathbf{X}_c)$ *of* F *centered at* $\check{X}_c$ *and over the box* $\mathbf{X}_c$, *where* $\check{X}_c$ *is the midpoint of* $\mathbf{X}_c$.
2. *Compute* $F(\check{X}_c)$ *using interval arithmetic to bound roundoff errors.*
3. *Perform a Gauss–Seidel sweep, beginning with* $\mathbf{X}_c$.
4. IF *the Gauss–Seidel sweep proved that* $\mathbf{X}_c$ *could not contain any roots* THEN EXIT.
5. IF *the Gauss–Seidel sweep did not result in a change in* $\mathbf{X}_c$, $\mathcal{S}$, *or* $\mathcal{U}$, THEN
   (a) Bisect $\mathbf{X}_c$, *modifying* $\mathbf{X}_c$ *and* $\mathcal{S}$.
   (b) Use the natural first-order extension to check if the range of F *over the new current box* $\mathbf{X}_c$ *returned from bisection contains zero; if not, then mark* $\mathbf{X}_c$ *as not root-containing and* EXIT.
   (c) (Find approximate roots in new current box)
   - *Find and verify approximate roots within* $\mathbf{X}_c$, *and modify the stack* $\mathcal{S}$, *the list* $\mathcal{R}$ *of root-containing boxes and the list* $\mathcal{U}$ *of small boxes of unknown status, using Algorithm 7.*

END IF (bisection process)

END DO

End Algorithm 6

Finally, the heuristic for when to invoke Algorithm 3, as well as management of the lists when taking complements, appears in:

ALGORITHM 7. (Find and verify approximate roots, then take their complements in the stack.)

INPUT: the heuristic parameter α, the lists $\mathcal{R}$ and $\mathcal{U}$, the stack $\mathcal{S}$, and the interval value $\mathbf{F}$ of F over $\mathbf{X}_c$ (either $\mathbf{F}_1(\mathbf{X}_c)$ or $\mathbf{F}_2(\mathbf{X}_c)$)

OUTPUT: (1) a (possibly) new $\mathbf{X}_c$, (possibly) altered lists $\mathcal{R}$ and $\mathcal{U}$, and a (possibly) altered stack $\mathcal{S}$; (2) an indicator stating whether any of $\mathbf{X}_c$, the lists, or the stack was altered

1. IF $\min_{i=1,\ldots,n}\left\{\min\left\{|\underline{F}_i|,|\overline{F}_i|\right\}/|(\mathbf{F})_i|\right\}<\alpha$, THEN EXIT *with no action.*
2. *Use Algorithm 3 to find an approximate root, to construct a box* $\mathbf{X}_r$ *about this approximate root, and to obtain information about existence and uniqueness* $\mathbf{X}_r$.
3. IF *an approximate root* X_a *could be found in step 2,* THEN
   (a) IF *no box* $\mathbf{X}_r$ *could be found in which uniqueness could be verified,* THEN *analogously to step 4(c) of Algorithm 5, form a box* $\mathbf{X}_r$ *by expansion about* X_a:
      - *For* $i=1$ *to* n *set* $\mathbf{X}_{ri}=[X_a-\sigma_i, X_a+\sigma_i]$,
        where $\sigma_i=1/\sqrt{\epsilon_{\mathrm{d}}}\max\left\{|X_a|\epsilon_{\mathrm{d}},\sqrt{\epsilon_m}\right\}$,
        and where ϵ_m *is the machine epsilon.*

   (b) *Push* $\mathbf{X}_c$ *onto* $\mathcal{S}$.

   (c) (This step is to guard against termination of the approximate solver at non-roots or at approximate roots that are farther away from the actual root than the stated tolerance.) IF $\mathbf{X}_r$ *was formed in step 3(a),* THEN

      i. *Apply Algorithm 4 to the smaller box* $\mathbf{X}_s$ *whose coordinates are defined by*

$$\mathbf{x}_{s,i}=\begin{array}{ll}[& X_a-\max\left\{|X_a|\epsilon_{\mathrm{d}},\sqrt{\epsilon_m}\right\}, \\ & X_a+\max\left\{|X_a|\epsilon_{\mathrm{d}},\sqrt{\epsilon_m}\right\}]\end{array}$$

      ii. IF $0\notin\mathbf{F}_u(\mathbf{X}_c)$ THEN EXIT *with failure to find an approximate root.*

   (d) *Take the complement of* $\mathbf{X}_r$ *in* $\mathcal{S}$ *and in* $\mathcal{U}$.

   (e) IF $\mathbf{X}_r$ *has been verified to contain a unique root* THEN *insert* $\mathbf{X}_r$ *into* $\mathcal{R}$, ELSE *insert* $\mathbf{X}_r$ *into* $\mathcal{U}$.

   (f) (Note that $\mathcal{S}$ could be empty from step 3(d).) IF $\mathcal{S}$ *is non-empty* THEN *pop the first item from* $\mathcal{S}$ *into* $\mathbf{X}_c$.

   END IF

End Algorithm 7

To summarize, Figure 2 gives a calling diagram for the algorithms described above, showing how each is incorporated into the overall root-finding code. Typeset details or the Fortran 90 code are available from the author.

7. Experimental Results. The experiments have the following goals:

- Determine the optimal value of the parameter α in step 1 of Algorithm 7, to determine when (or where) in the overall search algorithm (Algorithm 5) we should use a local root-finder to find approximate roots.
- Compare algorithm performance when using slopes in the Gauss–Seidel method (Algorithm 6) to that when using interval Jacobi matrices.
- Compare algorithm performance when not using second order extensions to when second order extensions are used.
- Examine the ability of the algorithm to produce a list of boxes that are verified to contain roots, and to reject boxes that do not contain roots.

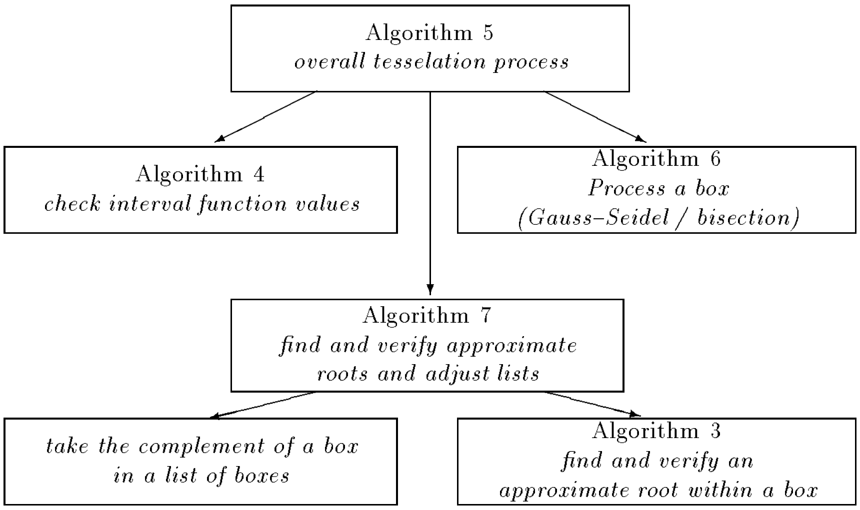

FIG. 2. *Overall algorithm structure*

- Examine the effect of the domain tolerance ϵ_{d} on the algorithm.
- Compare the algorithm to the algorithms and experimental results in [9] and [11], using those performance measures in common to these previous sets of experiments.
- Present performance results on specific problems, to impart an idea of the practicality of the algorithm.

7.1. The Problem Set. The problems consisted of

1. a subset of the more challenging problems from [9] and [11],
2. an eighteenth-degree polynomial arising in modeling an isothermal flash in chemical engineering ("Gritton's second problem"), upon which INTBIS ([17]) failed,
3. a set of four problems arising in the modeling of nonlinear electrical circuits.

Since the problems from [9] and [11] are described [9], details are not given here. The problems selected are 1, 2, 3, 4, 10, 11, 12, 15, 16, and 17; they will be designated "TOMS1," "TOMS2," etc. in the tables below. The remaining problems are briefly described; full definitions are available from the author.

Gritton-2 "Gritton's second problem," an eighteenth degree polynomial arising from a chemical engineering problem[7], upon which INTBIS failed. The problem has eighteen roots in the initial interval $[-12, 8]$, with the smallest root at $x \approx -11.09$ and the largest root at $x \approx 6.958$. Although this is a one-dimensional problem, the root at $x \approx 1.381$ is difficult for interval branch and bound methods to isolate because of a combination of the geometry of the graph and interval dependencies. In particular, the function decreases rapidly from $x = -12$, then is relatively flat in the interval $[-11, 8]$; it is extremely flat in the interval $[1, 2]$; graphs are available from the author. Step 4

[7] from G. V. Balaji and J. D. Seader, private communication

of Algorithm 5, particularly step 4(c), as well as use of second order extensions are important for this problem. The function is labelled GRIT2 in the tables below.

Mladenov-3 This moderately difficult problem, whose components consist of exponential and linear terms, arose from nonlinear electrical circuit analysis. It has nine roots within the initial box $[-2,1]\times[-5,1]\times[-2,1]\times[-5,1]\subset\mathbb{R}^4$. V. Mladenov (private communication and [23]) developed special algorithms for handling the special structure in the nonlinearities in this and the following two problems. The function is labelled MLAD3.

Mladenov-4 This relatively easy circuit-analysis problem consists of exponential and linear terms, and has two roots in the initial box $[-2,2]\times[-2,2]\times[-2,2]\times[-2,2]$. The function is labelled MLAD4.

Mladenov-5 This system of four generic second degree polynomials in four variables has two roots within the original box $[-2,2]^4$. The function is labelled MLAD5.

Mladenov-5b This problem has been derived from Mladenov-5 by appending equations corresponding to intermediate quantities generated during evaluation of the components of MLAD5, as explained in [12], to make it a fourteen-dimensional problem; details are available from the author. The idea is to make explicit information about the dependencies among the intermediate quantities available to the Gauss–Seidel preconditioner computation. Such expanded formulations reduce the total number of boxes processed by Algorithm 5, but increase total execution time is unless the system structure is carefully used. The function is labelled MLAD5B.

Due to subdistributivity, the quality of the interval extension depends on the way in which the expressions are written. We have rewritten using Horner's scheme wherever possible, although this does not always result in optimal bounds on ranges. Our actual Fortran-90 subroutines are available upon request.

7.2. The Implementation Environment. The algorithm was implemented within the research and prototyping environment described in [13]. This environment has an interval data type that uses the routines in INTLIB ([14]), as well as dynamically allocated linked lists of boxes. The problems are input as expressions, loops and subroutines in Fortran syntax, and an internal representation is then generated. This single internal representation is then interpreted by various generic subroutines, for interval function values, floating point function values, slope matrices, interval Jacobi matrices, etc. Thus, some performance is sacrificed in favor of extreme ease of programming new functions.

The INTLIB package was used with only one small low-level modification: As suggested by Knüppel in his BIAS subroutine package [19], we replaced the simulated directed rounding subroutine with an assembler language routine that allowed true directed roundings on our machine, and we restructured the subroutines for the four elementary operations to take advantage of it. This approximately doubled our execution speed, besides giving somewhat narrower intervals.

7.3. On Timings. The experiments were run on a Sun Sparc 20 Model 51 with a floating point accelerator. All code was compiled with the NAG f90 compiler version 2.1. All CPU times are written in terms of Standard Time Units (STU's), one thousand evaluations of the Shekel function as defined in [4, pp. 12–14]. The "f77" library function DSECND was used to obtain CPU times.

The STU is defined by programming the Shekel function in double precision as

Task	CPU sec.	No. STU's
1000 eval. of Shekel function (STU)	.056	1
1000 eval. of Shekel, using internal rep.	.450	8.0
1000 interval eval. of Shekel, using internal rep.	.986	17.6

TABLE 1

Comparison of standard time units (STU's) with floating point and interval evaluations, within our programming environment

α	NSL	NPFUN	NFUN	NV	NNV	NR	NBOX	TIME	TAP	TPT
0	12485	66758	16332	66	6	0	3290	3218	620	467
0.1	12298	50652	16246	66	6	0	3290	3038	441	377
0.2	12068	39792	16155	66	6	0	3291	2812	343	126
0.3	18541	27803	16063	66	6	0	3292	2731	263	126
0.4	18558	19152	15962	66	6	0	3296	2628	173	18
0.5	18558	13112	15866	66	6	0	3296	2587	130	54
0.6	18578	8259	15765	66	6	0	3301	2548	90	54
0.7	18578	5895	15673	66	6	0	3708	4559	61	54
0.8	18578	4376	15571	66	6	0	3708	4511	43	36
0.9	18578	2941	15411	66	6	0	3708	4568	28	0
1.1	18697	0	15165	66	6	0	3742	4613	0	0

TABLE 2

Comparison of performance versus heuristic parameter α

a subroutine. Since an STU is on the order of the granularity limit (0.01 second) of DSECND, we averaged 100 calculations of an STU. This entire averaging process was run several times, with and without contending loads on the machine. The resulting values were all within ten percent of each other.

As in [13], an internal representation of the function was created beforehand, then interpreted at run time, to obtain function and derivative values for all data types. There is a CPU penalty for this process, even for evaluation of floating point types. Table 1 shows the relationship between the STU, floating point function evaluations, and interval function evaluations within our programming environment.

There is also a sizable overhead associated with dynamic allocation and deallocation of boxes in lists, that is not present in previous implementations. However, we have not attempted to quantify this here.

Another performance measure reported in the experimental results is the total number of boxes the code processed. In the subset of experiments represented in Table 2, the correlation coefficient between total number of boxes and CPU time was approximately 0.97.

7.4. Results.

7.4.1. The Heuristic Parameter α. The rows of Table 2 give performance measures for different values of α in Algorithm 5 and Algorithm 7. The row $\alpha = 0$ corresponds to attempting approximate root-finding wherever possible, while rows corresponding to $\alpha = 1.1$ correspond to only attempting approximate root-finding in small boxes that have exited the Gauss–Seidel / box-processing loop without verification of either existence or non-existence (step 4(b) of Algorithm 5). The entries are simple sums of measures for all of the test problems excluding MLAD5B. We did not include the latter in such sums because its running time was comparable to that of all of the other problems combined (cf. Table 4).

The experiments in Table 2 were done without the refinement step for $\mathbf{X}_a$ of Algorithm 2. Also, in all cases, ϵ_{d} was taken to be 10^{-6}.

The columns of Table 2 have the following meanings.

NSL is the total number of slope matrix evaluations.

Method	NSL	NPFUN	NFUN	NV	NNV	NR	NBOX	TIME	TAP	TPT
Usual	12485	66758	16332	66	6	0	3290	3218	620	467
$\epsilon_d = 10^{-8}$	18967	13977	16450	71	1	0	3410	2649	129	108
Interval Jac.	11856	17846	45062	65	8	1	8160	2844	164	180
Refinement	18627	13080	16079	68	4	0	3367	2610	124	72

TABLE 3

Comparison of algorithm variations, with $\alpha = .5$

NPFUN is the total number of non-interval ("point") function evaluations.

NFUN is the total number of interval function (i.e. natural first-order extension) evaluations. This column ideally equals the total number of roots.

NV is the total number of boxes in which a unique root was verified.

NNV is the total number of boxes that were small but could not be verified to contain unique roots.

NR is the number of "redundantly listed" boxes, a measure of undesirable multiple small boxes near roots that cannot be rejected. This column equals the total number of roots minus the two previous columns, and ideally equals zero. A negative number would indicate either an improperly functioning algorithm (since not all roots were found), or two or more roots that are in the same box in $\mathcal{U}$.

NBOX is the total number of boxes processed in step 1 of Algorithm 5.

TIME is the total execution time given in STU's, as explained in §7.3.

TAP is the total amount of time in STU's spent in the approximate solver (in this case MINPACK-1).

TPT is the total amount of time in STU's spent computing non-interval function values (in the approximate solver).

In Table 2, a gradual decrease of total execution time relative to α is observed until $\alpha = .4$, there is a relatively insensitive region between $\alpha = .4$ and $\alpha = .6$, then there is a rapid increase to a plateau, until the maximum meaningful value of α is reached.

The verification power appears independent of α, and that the code is very successful at isolating roots. In fact, examination of the details reveals that, for each α, each root was uniquely enclosed in a box. Of the six roots for which uniqueness was not verified, one was the root of Powell's singular function (at which the Jacobi matrix has a null space of dimension two), one was for TOMS10, a combustion chemistry problem with an unusual scaling at its root, and the remaining were four roots of GRIT2, approximately 1.480, 1.594, 1.752, and 1.967, in the interval of relative flatness. Examining the results for $\alpha = .5$, existence was verified for the boxes containing the roots 1.480, 1.752, and 1.967, but not for 1.594, or for the boxes containing the roots of TOMS3 and TOMS10.

We expect optimal α to be somewhat different, with a lower optimal value of α, if the floating point function and derivative evaluations proceed from compiled programs, rather than by interpreting a symbolic representation. However, the total amount of time spent in the approximate solver is not large in relation to the total time, so the dependence on α seems unimportant.

7.4.2. Variants of the Algorithm. Table 3 reports performance measures similar to those in Table 2, for algorithm variants. Based on Table 2, we chose $\alpha = .5$, near maximum insensitivity to α, for all experiments represented in Table 3. The variants are as follows:

Usual denotes the method in Table 2 with $\alpha = .5$, i.e. the last entry in Table 2, for comparison.

$\epsilon_{\mathrm{d}} = 10^{-8}$ denotes the usual method as above, but with the domain tolerance (minimum box size) set to 10^{-8}, rather than 10^{-6}.

Interval Jac. denotes the method with an interval Jacobi matrix replacing an interval slope matrix in step 1 of Algorithm 6. With this modification, the images under the Gauss–Seidel method may be wider, but uniqueness can be verified with the Gauss–Seidel method itself. Note that, in this case, the number of interval Jacobi matrix evaluations is not reported.

Refinement denotes the method that *includes* the refinement step for the box $\mathbf{X}_a$ in which existence has been proven, as mentioned in §3. Since we may obtain sharper bounds on the root with this refinement, the slope matrices in the uniqueness verification step may be narrower, and it may be possible to verify uniqueness in a larger box.

REMARK 1. *We also attempted to run the test set without the second order interval extensions, i.e. without step 2 of Algorithm 4. However, problem GRIT2 did not complete in 171 CPU minutes (184200 STU's), while there did not seem to be a large advantage for the other problems. We thus concluded that the second order extension was an essential part of the algorithm.*

None of the variants differ significantly in CPU time from our primary scheme, although we suspect slopes to become much more efficient than interval Jacobi matrices when we implement them more efficiently; note that use of interval Jacobi matrices resulted in more than twice as many boxes.

We observe insensitivity of the work on the tolerance[8], but the smaller tolerance allows more roots to be verified. Tuning of the relationships among the tolerances and box expansion factors, for particular F, will probably allow verification of all roots, in most cases. The results in table 3 for $\epsilon_{\mathrm{d}} = 10^{-8}$ represent ideal verification and isolation behavior, since the root of TOMS3 (Powell's singular function) is at a singularity of the Jacobi matrix, and cannot be enclosed in a box in which it is verified to be unique.

Refining the small boxes in which existence has been proven does not appear to have much effect, except that there is weak evidence that it facilitates uniqueness verification.

7.4.3. Performance on the Individual Problems. Table 4 gives cost measures by problem for $\alpha = .5$.

7.4.4. Comparison with INTBIS. Table 5 compares performance measures of the usual algorithm, with $\alpha = .5$ to those of INTBIS [17]. To obtain the data, INTBIS was run on the Sparc system under the same conditions as the other experiments. Since INTBIS uses an inverse midpoint preconditioner instead of the linear programming preconditioner of [11], certain problems take much more time with INTBIS.

On the other problems, comparison of total execution times with INTBIS reveals the overhead in interpretive evaluation of functions and slopes, as well as the overhead in dynamic allocation and deallocation used in the system of [13]. We also see that the verification power and cluster rejection power of the algorithm presented here are superior to that of INTBIS. Furthermore, we emphasize that INTBIS could not solve GRIT2 at all.

[8] In earlier experiments, without the expansion of [10], step 4(c)i of Algorithm 5, the work actually decreased as we decreased the tolerance between 10^{-6} and 10^{-9}.

Function	NFUN	NSL	NBOX	NV	NNV	NR	TIME	TAP	TPT
TOMS1	29	59	6	3	0	0	2.0	0.5	0.0
TOMS2	42	42	11	1	0	0	1.4	0.2	0.0
TOMS3	35	17	16	0	1	0	1.4	0.5	0.0
TOMS4	88	123	24	2	0	0	14.5	1.6	0.0
TOMS10	223	157	59	0	1	0	16.9	0.4	0.0
TOMS11	389	473	110	16	0	0	197.3	6.6	0.0
TOMS12	618	517	188	12	0	0	45.8	2.9	0.0
TOMS15	6	18	1	1	0	0	0.7	0.0	0.0
TOMS16	5	19	1	1	0	0	1.6	0.2	0.0
TOMS17	318	203	94	1	0	0	59.2	5.4	0.0
GRIT2	9098	6318	1802	14	4	0	213.6	24.6	0.0
MLAD3	894	618	265	9	0	0	1427.3	37.3	35.9
MLAD4	138	126	31	2	0	0	72.0	4.5	18.0
MLAD5	2540	1750	688	2	0	0	532.7	45.2	0.0
MLAD5B	1443	1015	407	2	0	0	2080.6	45.2	35.9
Totals	15866	11455	3703	66	6	0	4667.1	175.2	89.77

TABLE 4
Performance measures by problem

Function	TIME	NV	NNV	NR	NBOX	NFUN	NSL	package
TOMS1	1.6	3	0	0	6	29	59	new
	0.4	1	2	0	8	43	21	intbis
TOMS2	1.3	1	0	0	11	42	42	new
	1.4	0	1	0	41	119	51	intbis
TOMS3	1.6	0	1	0	16	35	17	new
	60.7	0	7	6	1350	3705	1475	intbis
TOMS4	10.4	2	0	0	24	84	91	new
	498.0	1	1	0	6632	14992	6417	intbis
TOMS10	18.3	0	1	0	62	232	163	new
	5.4	1	9	0	85	251	111	intbis
TOMS11	189.8	16	0	0	110	389	473	new
	44.0	0	16	0	229	749	374	intbis
TOMS12	48.1	12	0	0	188	618	517	new
	22.4	4	10	3	355	1281	575	intbis
TOMS15	0.5	1	0	0	1	6	18	new
	0.2	1	0	0	1	4	2	intbis
TOMS16	1.6	1	0	0	1	5	19	new
	0.0	1	0	0	1	4	2	intbis
TOMS17	71.6	1	0	0	94	318	203	new
	7.5	1	0	0	76	189	80	intbis
Totals	344.9	36	2	0	419	1440	1399	new
	632.5	9	37	8	8702	21148	9028	intbis
Ratios	0.98	4.00	0.05	0.00	0.05	0.07	0.16	

TABLE 5
Performance measure comparisons with INTBIS

We only compared INTBIS on those test problems representing polynomial systems, since these are the only systems easily implementable in INTBIS. The problem GRIT2 was not included, since INTBIS failed on this problem. (Severe clustering occurred, and the fixed-size storage for the box list filled.)

The column labels in Table 5 are as in the other tables. Rows are grouped in pairs; the first in a pair represents performance of the present algorithm, while

Function	NBOX	NBOX/P	NSL	NJAC/P	NFUN	NFUN/P
TOMS1	6	19	59	35	29	73
TOMS2	11	29	42	47	42	104
TOMS3	16	368	17	423	35	1064
TOMS4	24	33	91	58	84	123
TOMS10	62	247	163	247	232	601
TOMS11	110	367	473	525	389	1210
TOMS12	188	255	517	475	618	1050
TOMS15	1	1	18	2	6	4
TOMS16	1	1	19	2	5	4
TOMS17	94	23	203	38	318	83
Totals	513	1343	1602	1852	1758	4316
Ratios	0.38		0.87		0.41	

TABLE 6
Comparison with the preconditioner code of [11] (columns /P)

the second in the pair represents performance of INTBIS. In of column "NSL," the number reported for the present code is the number of slope matrix evaluations, while the number reported for INTBIS is the number of Jacobi matrix evaluations. The last row, labelled "Ratios," gives the ratio of the performance measure of the present algorithm to the corresponding performance measure of INTBIS.

The results for INTBIS are skewed by TOMS3 and TOMS4, Powell's singular function and Brown's almost linear function. The poor performance of INTBIS on these problems is due to use of the inverse midpoint preconditioner, as opposed to the linear programming preconditioner of [11]. On the other problems, the new algorithm is more effective at verifying uniqueness, and generally requires that a smaller number of boxes be processed. However, the overall running times appear to reflect additional overhead in the new algorithm; examine the results for TOMS1, TOMS2, and TOMS12.

The only problem for which the new code processed a larger number of boxes than INTBIS was TOMS17. The interval Newton method works best without a preconditioner on this strange problem, and the complementation process in the new code also may have produced more boxes than simple bisection.

Table 4 indicates that the extra time spent in the approximate solver is insignificant.

7.4.5. Comparison with a Preconditioner Testing Code. Table 6 compares performance measures of the new algorithm, with $\alpha = .5$, to those of the research code in [11]. The code for the experiments in [11] was ad-hoc and never published, and has not been installed on our present computing equipment. For this reason, we compare only those performance measures listed in [11].

The columns of Table 6 are labelled as in the other tables, except that NBOX/P means the number of boxes processed by the code of [11], NJAC/P is the number of Jacobi matrix evaluations from [11], and NFUN/P is the number of interval function evaluations from [11].

We see some variation from problem to problem[9], but, overall, the new algorithm appears to be superior. Note that the new code processed only about a third

[9] In particular, TOMS17 seems to have been solved more efficiently with the code from [11].

as many boxes and required about two fifths as many interval function evaluations. The number of slope matrix computations was nearer to the number of Jacobi matrix evaluations in the preconditioner code; this was probably due to the additional evaluations in the expansion process in Algorithm 3 and to the slope matrices required for the second-order evaluations in Algorithm 4.

Although not reflected in Table 6, the verification and root isolation power of the new code are superior to that of [11].

8. On Convergence, Rigor and Efficiency. The overall algorithm (Algorithm 5) should always complete, in theory, with $\mathcal{S}$ empty, provided M is sufficiently large. In each iteration of the overall box processing loop, at least one coordinate width of the box $\mathbf{X}_c$ is decreased by an amount that is bounded below; the actual minimal amount of decrease is determined by the bisection process and by a tolerance defining "change" in step 5. Furthermore, the "maximum smear" coordinate selection for generalized bisection assures that each coordinate of a box is chosen an infinite number of times, if the algorithm runs indefinitely, provided

$$\frac{\max_{1\le j\le n}\{\max_{1\le i\le n}|\mathbf{S}_{i,j}|\}}{\min_{1\le j\le n}\{\max_{1\le i\le n}|\mathbf{S}_{i,j}|\}}$$

is bounded below. Finally, processing of a box is terminated when its relative coordinate widths[10] fall below ϵ_{d}. These facts should form the elements of a convergence theory, to be formalized in another work. However, it will probably be difficult to obtain realistic upper bounds on the amount of work required by Algorithm 5, for many problems.

The following is a consequence of established underlying theory of interval computations.

THEOREM 8.1. *Suppose that interval arithmetic is correctly implemented on a computer with correct circuitry, and suppose that Algorithm 5 is correctly implemented. Then, if Algorithm 5 completes with $\mathcal{S}$ empty, it constitutes a mathematical proof that all roots of F within $\mathbf{X}_0$ are in boxes in $\mathcal{R}$ and $\mathcal{U}$, and each box in $\mathcal{R}$ contains a unique root of F.*

Theorem 8.1 motivates study of such interval branch and bound algorithms. However, such algorithms can also be competitive, efficiency wise, with less rigorous alternatives, on some problems; see the discussion in [9].

A less rigorous, "probability one" alternative for polynomial systems is continuation methods, described in [2]. Good theory and practice have been developed for such algorithms by Morgan et al, eg. [28]. These algorithms, though particular to polynomial systems and not rigorous in the sense of Theorem 8.1, are useful in many applications.

9. Summary. We have implemented a rigorous algorithm to compute all roots within a box in $\mathbb{R}^n$, within our new Fortran-90 environment for research and prototyping. This algorithm incorporates recent research results and a technique for removing root-containing boxes from the search region. This removal technique is more rigorous than that of [8] and [17].

Experimental results show the effect of individual innovations and the effect of a heuristic parameter in the algorithm. These results also reveal differences between the

[10] A possible improvement to the algorithm may be to define the j-th relative coordinate width by $w_j = s_j = \max_{1\le i\le n}\{|\mathbf{S}_{i,j}(F, X_a, \mathbf{X})|(\overline{x}_j - \underline{x}_j)\}$, consistent with maximum smear.

new algorithm, a previously published code, and a previous research code. Conclusions include the following:

- The new overall algorithm is more effective, and solves problems for which previous algorithms fail.
- The new complementation / deletion process is better than previous schemes at isolating boxes in which roots are unique.
- An approximate root-solver with ϵ-inflation can decrease overall running time.
- Second-order extensions based on interval slopes are crucial for some problems.
- Use of slopes, rather than interval Jacobi matrices, in the interval Newton size-reduction iteration reduces the total number of boxes considered, and may reduce the total amount of work significantly if slopes can be implemented more efficiently.
- Slopes are effective at proving both existence and uniqueness.
- The new algorithm seems relatively insensitive to the tolerance, within a certain range.
- The total amount of work in the algorithm appears to depend nearly linearly on the number of boxes to be processed.

In addition to the above, preliminary experiments have led us to the following conclusions, not evident in the presented results:

- An effective algorithm involves several acceleration techniques that interact. Assessment of a particular technique would change depending on what other techniques are present and when they are applied.
- The relationship among the tolerances used to declare a box to be small, to stop the approximate root solver, and to expand a box about a region in which neither uniqueness or non-uniqueness could be proved is important[11].

Of course, none of the above considerations would affect the *rigor* of the algorithm, only its *efficiency*.

10. Acknowledgements. The author wishes to thank both referees, as well as the editor Margaret Wright. The first had several useful overall suggestions giving perspective and guiding in improvement of the exposition. The second referee exhibited a deep understanding of the material, gave a careful and laborious report, and made many stimulating suggestions that improved the paper; I thank him for pointing out the equivalence of Step 2 of Algorithm 4 to a step of the Jacobi method. Discussions with Dr. Wright resulted in an improved, more compact exposition.

REFERENCES

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

[2] E. Allgower and K. Georg, *Numerical Continuation Methods: An Introduction*, Springer-Verlag, New York, 1990.

[3] O. Caprani, B. Godthaab, and K. Madsen, *Use of a real-valued local minimum in parallel interval global optimization*, Interval Computations, 1993 (1993), pp. 71–82.

[4] L. C. W. Dixon and G. P. Szegö, *The global optimization problem: An introduction*, in Towards Global Optimization 2, L. C. W. Dixon and G. P. Szegö, eds., Amsterdam, Netherlands, 1978, North-Holland, pp. 1–15.

[5] E. R. Hansen, *Global Optimization Using Interval Analysis*, Marcel Dekker, Inc., New York, 1992.

[11] Also,the tolerance chosen for the minimum box size should be such that none of the tolerances is smaller than the precision of the computations.

[6] J. Herzberger, ed., *Topics in Validated Computations*, Studies in Computational Mathematics, Amsterdam, 1994, Elsevier Science Publishers.
[7] C. Jansson and Knüppel, O., *A global minimization method: The multi-dimensional case*, Tech. Report 92.1, Informathinstechnik, Technische Uni. Hamburg–Harburg, 1992.
[8] R. B. Kearfott, *Abstract generalized bisection and a cost bound*, Math. Comp., 49 (1987), pp. 187–202.
[9] ———, *Some tests of generalized bisection*, ACM Trans. Math. Software, 13 (1987), pp. 197–220.
[10] ———, *Interval newton / generalized bisection when there are singularities near roots*, Annals of Operations Research, 25 (1990), pp. 181–196.
[11] ———, *Preconditioners for the interval gauss–seidel method*, SIAM J. Numer. Anal., 27 (1990), pp. 804–822.
[12] ———, *Decomposition of arithmetic expressions to improve the behavior of interval iteration for nonlinear systems*, Computing, 47 (1991), pp. 169–191.
[13] ———, *A Fortran 90 environment for research and prototyping of enclosure algorithms for nonlinear equations and global optimization*, ACM Trans. Math. Software, 21 (1995), pp. 63–78.
[14] R. B. Kearfott, M. Dawande, K.-S. Du, and C.-Y. Hu, *Algorithm 737: INTLIB: A portable FORTRAN 77 interval standard function library*, ACM Trans. Math. Software, 20 (1994), pp. 447–459.
[15] R. B. Kearfott and K. Du, *The cluster problem in multivariate global optimization*, Journal of Global Optimization, 5 (1994), pp. 253–265.
[16] R. B. Kearfott, C. Hu, and M. I. Novoa, *A review of preconditioners for the Interval Gauss–Seidel method*, Interval Computations, 1 (1991), pp. 59–85.
[17] R. B. Kearfott and M. Novoa, *INTBIS, a portable interval newton/bisection package (Algorithm 681)*, ACM Trans. Math. Software, 16 (1990), pp. 152–157.
[18] R. B. Kearfott and Z. Xing, *Rigorous computation of surface patch intersection curves*, 1993.
[19] O. Knüppel, *PROFIL/BIAS — A fast interval library*, Computing, 53 (1994), pp. 277–287.
[20] R. Krawczyk, *Newton-Algorithmen zur Bestimmung von Nullstellen mit Fehlershranken*, Computing, 4 (1969), pp. 187–201.
[21] R. Krawczyk and A. Neumaier, *Interval slopes for rational functions and associated centered forms*, SIAM J. Numer. Anal., 22 (1985), pp. 604–616.
[22] G. Mayer, *Epsilon–inflation in verification algorithms*, 1993.
[23] V. Mladenov, *An improved interval method for solving nonlinear systems of monotone functions*, in Mathematical Modelling and Scientific Computing, S. M. Markov, ed., Sofia, 1993, DATECS Publishing, pp. 23–26.
[24] R. E. Moore, *A test for existence of solutions to nonlinear systems*, SIAM J. Numer. Anal., 14 (1977), pp. 611–615.
[25] ———, *Methods and Applications of Interval Analysis*, SIAM, Philadelphia, 1979.
[26] R. E. Moore and S. T. Jones, *Safe starting regions for iterative methods*, SIAM J. Numer. Anal., 14 (1977), pp. 1051–1065.
[27] J. J. Moré, B. S. Garbow, and K. E. Hillstrom, *User guide for MINPACK-1*, Tech. Report ANL-80-74, Argonne National Laboratories, 1980.
[28] A. J. Morgan, A. P. Sommese, *Computing all solutions to polynomial systems using homotopy continuation*, Appl. Math. Comput., 24 (1987), pp. 115–138.
[29] A. Neumaier, *Interval Methods for Systems of Equations*, Cambridge University Press, Cambridge, England, 1990.
[30] H. Ratschek and J. Rokne, *Computer Methods for the Range of Functions*, Horwood, Chichester, England, 1984.
[31] S. M. Rump, *Kleine Fehlerschranken bei Matrixproblemen*, PhD thesis, Universität Karlsruhe, 1980.
[32] ———, *Verification methods for dense and sparse systems of equations*, in Topics in Validated Computations, J. Herzberger, ed., Amsterdam, 1994, Elsevier Science Publishers.
[33] Z. Xing, *Rigorous Step Control for Continuation*, PhD thesis, University of Southwestern Louisiana, 1993.