Some Tests of Generalized Bisection

R. BAKER KEARFOTT
The University of Southwestern Louisiana

This paper addresses the task of reliably finding approximations to all solutions to a system of nonlinear equations within a region defined by bounds on each of the individual coordinates. Various forms of generalized bisection were proposed some time ago for this task. This paper systematically compares such generalized bisection algorithms to themselves, to continuation methods, and to hybrid steepest descent/quasi-Newton methods. A specific algorithm containing novel "expansion" and "exclusion" steps is fully described, and the effectiveness of these steps is evaluated. A test problem consisting of a small, high-degree polynomial system that is appropriate for generalized bisection, but very difficult for continuation methods, is presented. This problem forms part of a set of 17 test problems from published literature on the methods being compared; this test set is fully described here.

Categories and Subject Descriptors: G.1.5 [**Numerical Analysis**]: Roots of Nonlinear Equations—*systems of equations*

General Terms: Algorithms, Performance

Additional Key Words and Phrases: Generalized bisection, global constrained optimization, homotopy method, interval arithmetic, MINPACK, quasi-Newton method

1. MOTIVATION, PURPOSE, AND SCOPE

We consider the following general problem:

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

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

where bounds a_i and b_i are known such that:

$$a_i \leq x_i \leq b_i \qquad \text{for} \quad 1 \leq i \leq n.$$

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

Here, we assume n is of moderate size (so that sparsity is not a consideration). Relevant properties of methods for solving (1.1) include

(i) computational efficiency,
(ii) mathematical reliability,

Author's address: Department of Mathematics, P.O. Box 41010, The University of Southwestern Louisiana, Lafayette, LA 70504-1010.

(iii) computational reliability,
(iv) efficiency of implementation, and
(v) range of application.

Computational efficiency deals with the total number of computer operations in order to solve (1.1), and generally includes both evaluation of the residuals on the left-hand side of (1.1) and intrinsic overhead in the method. By mathematical reliability, we mean the existence of theorems which state that, under simplifying assumptions and in exact arithmetic, the method will solve (1.1). When we say a method is computationally reliable, we mean the method is stable when implemented in floating-point systems and heuristics introduced in the implementation, but not in the theory, are effective. A method is efficient to implement when it is simple from a computer programmer's point of view, and when its performance is not sensitive to details of the implementation or to "tuning." Range of application deals with the types of problems for which the method works (e.g., just a single equation in a single variable, just polynomial equations, etc.).

Clearly, there are conflict and interplay between these properties, so that just one method is probably not "best" for all problems. However, a good algorithm should be strong in several of the areas.

Some alternate techniques for solving (1.1) include: (i) random search; (ii) an exhaustive grid search on the box $[a_1, b_1] \times [a_2, b_2] \times \cdots \times [a_n, b_n]$; (iii) Newton's method with repeated random starts; (iv) ad hoc methods, such as the Jenkins–Traub method for finding all roots of a single polynomial; (v) homotopy continuation methods; and (vi) pure interval Newton methods. Random search is neither mathematically nor computationally reliable. Grid search can be made mathematically and computationally reliable, although it is computationally very inefficient. Newton's method with repeated random starts, or "globalized" quasi-Newton methods, are sometimes used as exploratory tools. However, such methods are not mathematically reliable for solving (1.1) in general when there is more than one solution. Specialized methods can be reliable, efficient, and easily implemented, but with a limited range of application.

Homotopy continuation methods can solve variants of (1.1) effectively in many instances (cf., [20]). For example, if the variables are complex, all solutions (not just those in a bounded region) are desired, and the system consists of polynomial equations, then such methods are recommended. Such methods are also recommended for specific systems where significant a priori analysis can be performed. Nonetheless, if just real roots within a particular region are required, then homotopy continuation methods can be relatively expensive. Furthermore, state-of-the-art implementations involve both numerous heuristics and mathematical sophistication, and some early methods are computationally unreliable.

The method presented in this paper makes use of an interval Newton method (though the basic root inclusion test is general and could possibly also be implemented with alternate techniques). However, pure interval Newton methods (without a bisection component) may have difficulty with (1.1) when there is more than one solution or when the Jacobian matrix $F'(X)$ is singular for numerous $X \in B$.

In this paper, we intend to show that generalized bisection is a competitive method for solving (1.1) in terms of all five of the desirable attributes. Generalized bisection is an exhaustive search for roots, and necessarily is computationally and mathematically reliable (cf., [11]). However, the search proceeds by adaptively and nonuniformly subdividing the domain of the variables into regions that are (ideally) just small enough such that Newton's method starting in any of the subregions will converge to a unique root within that subregion. In that sense, bisection is optimal.

The method in this paper employs an interval form of Newton's method [13] and is thus similar to algorithms proposed by Moore et al. [15], Hansen (see [6, 7, 8, 9], etc.), Alefeld et al. [2], etc., although we do not emphasize the interval arithmetic aspects of the method. Here, we test the method on 17 problems that include artificial and real-world examples from the literature on quasi-Newton methods, homotopy continuation methods, other bisection methods, etc. This test set has been chosen to illustrate the strengths and weaknesses of our generalized bisection method vis a vis other methods. We thus hope to convince persons wishing to solve (1.1) that generalized bisection has its place in their store of tools.

In Section 2 of this paper, we briefly describe the mathematical framework of the method. Some particulars of the implementation are presented in Section 3. In Section 4, we give the test set. The numerical results are shown in Section 5. In Section 6, we give conclusions and directions for further study.

2. THE BASIC METHOD

A careful analysis of the basic ideas underlying generalized bisection is given in [11]; here we describe these ideas in the context of our specific algorithm.

Our generalized bisection algorithm consists of

(i) a geometrical bisection process,
(ii) a root inclusion test, and
(iii) a binary search algorithm.

In the results presented here, the geometrical bisection process is similar to that given in [6], [15], etc. To each box

$$\mathbf{B} = [a_1, b_1] \times [a_2, b_2] \times \cdots \times [a_n, b_n] \tag{2.1}$$

we find k such that

$$k = \max_{1 \le j \le n} (b_j - a_j), \tag{2.2}$$

we set

$$m_k = (a_k + b_k)/2, \tag{2.3}$$

then we form two boxes $\mathbf{B}_1$ and $\mathbf{B}_2$ such that

$$\begin{aligned} \mathbf{B}_1 &= [a_1, b_1] \times [a_2, b_2] \times \cdots \times [m_k, b_k] \times \cdots \times [a_n, b_n] \\ \mathbf{B}_2 &= [a_1, b_1] \times [a_2, b_2] \times \cdots \times [a_k, m_k] \times \cdots \times [a_n, b_n]. \end{aligned} \tag{2.4}$$

That is, we cut the box $\mathbf{B}$ into two boxes $\mathbf{B}_1$ and $\mathbf{B}_2$ by cutting in the coordinate direction in which $\mathbf{B}$ is longest. (Note that the coordinate index k may be chosen by "pivot" strategies other than (2.2); however, for well-scaled problems, (2.2) works reasonably well and also has the advantage of simplicity.) Important aspects of the geometrical bisection process are (i) as it is iterated on $\mathbf{B}_1$ and $\mathbf{B}_2$, the diameters of the resulting boxes tend to zero at a known rate; and (ii) the boxes do not get arbitrarily "thin."

Our root inclusion test is a function $\mathbf{T}_F$ which associates to each box $\mathbf{B}$ the values "true," "false," and "unknown." Defined formally in [11], it has the following properties:

(i) $\mathbf{T}_F(\mathbf{B})$ = "true" implies that there is a unique solution of the system (1.1) within $\mathbf{B}$, and Newton's method or one of its variants will converge to that solution from any starting point in $\mathbf{B}$.

(ii) $\mathbf{T}_F(\mathbf{B})$ = "false" implies that there are no solutions of the system (1.1) within $\mathbf{B}$.

(iii) If the diameter of $\mathbf{B}$ is sufficiently small, if any solutions of (1.1) lie sufficiently far, relative to the size of $\mathbf{B}$, from the boundary of $\mathbf{B}$; and if the Jacobian matrix of the residual function F as in (1.1) is nonsingular at any solutions of (1.1), then $\mathbf{T}_F(\mathbf{B})$ = "true" or $\mathbf{T}_F(\mathbf{B})$ = "false."

Here, the root inclusion test is similar to that in [13]. In particular, our root inclusion test is based partly on computing intervals $[c_i, d_i]$ such that

$$c_i \leq f_i(x_1, x_2, \ldots, x_n) \leq d_i, \qquad \text{for} \quad 1 \leq i \leq n, \tag{2.5}$$

where

$$a_j \leq x_j \leq b_j, \quad 1 \leq j \leq n.$$

Thus, if, for any i, $c_i > 0$ or $d_i < 0$, then f_i cannot vanish in the box $\mathbf{B}$, and we set $\mathbf{T}_F(\mathbf{B})$ = "false." To compute $\mathbf{T}_F(\mathbf{B})$ = "true" when there is a solution of (1.1) within $\mathbf{B}$, we compute similar bounds on the partial derivatives of the f_i. Using these bounds, we obtain a box $\overline{\mathbf{B}}$ such that all images of points in $\mathbf{B}$ under a single application of the chord method (with a specified iteration matrix independent of the point) are contained in $\overline{\mathbf{B}}$. (The chord method is an iteration of the form $X_{k+1} \leftarrow X_k - YF(X_k)$, where Y is held fixed relative to X_k, and is only an approximate inverse to $F'(X_k)$; see [23], p. 181.) Thus, if $\overline{\mathbf{B}} \subset \mathbf{B}$, the Schauder fixed point theorem and the fact that stationary points of the chord method correspond to solutions of (1.1) show that there is a solution of (1.1) in $\mathbf{B}$. Furthermore, if the norm of a certain matrix is less than one, then this solution is unique, and the chord method will converge to it. Details are given in Section 3 and in [13].

The bisection process and the root inclusion test define a binary search algorithm naturally: The siblings of $\mathbf{B}$ are $\mathbf{B}_1$ and $\mathbf{B}_2$, while the internal nodes of the binary tree correspond to subboxes of $\mathbf{B}$ that are labeled "unknown." Here, we use a depth-first search. Our algorithm also differs from similar bisection algorithms in that we sometimes "expand" the box when it appears that solutions occur too near its boundary to conclude that they are inside or outside. A full

description and rigorous justification of this process in a general setting are given in [11], while the essentials of the algorithm are repeated below.

3. IMPLEMENTATION DETAILS

The code is based upon Algorithm 2.6 in [11]; for clarity we present a particular version here. Below, each $\mathbf{B}^L$ produced from $\mathbf{B}$ at level L in the bisection process is assumed to have the same form as $\mathbf{B}$ in (2.1), i.e.,

$$\mathbf{B}^L = [a_1^L, b_1^L] \times [a_2^L, b_2^L] \times \cdots \times [a_n^L, b_n^L]. \tag{3.1}$$

Furthermore, the diameter of $\mathbf{B}^L$ is defined by

$$d(\mathbf{B}^L) = \max_{1 \le j \le n} | b_j^L - a_j^L |. \tag{3.2}$$

In the algorithm, a number ϵ must be input. In the theory in [11], ϵ is such that $\mathbf{T}_F(\mathbf{B}^L)$ = "true" if: (i) $d(\mathbf{B}^L) < \epsilon$ and solutions of (1.1) lie sufficiently far from the boundary of $\mathbf{B}^L$; and (ii) the distance between solutions of (1.1) is at least $\epsilon/2$. In practice, ϵ may be chosen heuristically to be the smallest allowable box dimension. If this is done, there is no danger of entirely missing roots, but we run the risk that two roots that are closer than ϵ to each other may be approximated by the same small box.

Additionally, a number ϵ_F is input such that, if

$$f_i(x_1, x_2, \ldots, x_n) | < \epsilon_F$$

for every $(x_1, x_2, \ldots, x_n)$ such that (3.3)

$$a_j^L \le x_j \le b_j^L,$$

then the box $\mathbf{B}^L$ is accepted as though $\mathbf{T}_F(\mathbf{B}^L)$ were "true." The reason ϵ_F is needed is to stop the refinement process near solutions of (1.1) at which the Jacobian matrix is singular. In theory, the root inclusion test based on an interval Newton method that is implemented here must fail at such solutions. (See [11] for conditions under which a general root inclusion test will work.) Also, in theory, use of ϵ_F to stop the algorithm could give points at which the residuals f_i are small but which do not actually correspond to roots. In the main experiments in Section 5, we chose $\epsilon_F = 10^{-10}$ and $\epsilon = 10^{-5}$; in all cases, all of the roots were properly isolated. Only in the case of Powell's singular function (see problem 3 in Section 4) did the algorithm give boxes in which the residuals were small but in which there were not actually roots. Those five boxes were clustered around the actual quadruple root, at which the Jacobian matrix was null.

Algorithm 3.1 (*Generalized bisection based on boxes*)

1. (Initialization phase)
   (a) Set: $L \leftarrow 1$.
   (b) Set: $\mathbf{B}^1 \leftarrow \mathbf{B}$.
   (c) Input the numbers ϵ and ϵ_F as described above.
2. (Subdivision phase)
   (a) Form $\mathbf{B}_1^L$ and $\mathbf{B}_2^L$ as in (2.4).
   (b) $\mathbf{B}^{L+1} \leftarrow \mathbf{B}_1^L$.
   (c) $L \leftarrow L + 1$.

3. (Test phase and storage of roots)
   (a) Compute $d(\mathbf{B}^L)$.
   (b) If $d(\mathbf{B}^L) < \epsilon/4$ and $\mathbf{B}^L$ has nonnull intersection with a box $\mathbf{B}'$ in list L such that $d(\mathbf{B}^L) + d(\mathbf{B}') < \epsilon/2$ then go to Step 5. Otherwise, continue to Step 3 (c).
   (c) Compute $\mathbf{T}_F(\mathbf{B}^L)$.
   (d) If $\mathbf{T}_F(\mathbf{B}^L)$ = 'unknown' and $d(\mathbf{B}^L) \geq \epsilon/16$, then return to Step 2.
   (e) If $\mathbf{T}_F(\mathbf{B}^L)$ = 'false', then go to Step 5.
   (f) If $\mathbf{T}_F(\mathbf{B}^L)$ = 'true' or $|f_i(x_1, x_2, \ldots, x_n)| < \epsilon_F$ for each i and $(x_1, x_2, \ldots, x_n)$ as in (3.3), then:
      (i) If $\mathbf{B}^L$ has null intersection with every box $\mathbf{B}' \in \mathbf{L}$ with $d(\mathbf{B}') < \epsilon/4$, then store $\mathbf{B}^L$ in list $\mathbf{L}$.
      (ii) Go to Step 5.
4. (Adjustment step for roots on a boundary: in this case $\mathbf{T}_F(\mathbf{B}^L)$ = 'unknown' and $d(\mathbf{B}^L) < \epsilon/16$).
   (a) Replace $\mathbf{B}^L$ by the box $\mathbf{B}_b^L$ with same midpoints as $\mathbf{B}^L$ but four times as large:

$$\mathbf{B}_b^L = [\tilde{a}_1^L, \bar{b}_1^L] \times [\tilde{a}_2^L, \bar{b}_2^L] \times \cdots \times [\tilde{a}_n^L, \bar{b}_n^L]$$

   where

$$\tilde{a}_j^L = m_j^L + 4\,(a_j^L - m_j^L),$$
$$\bar{b}_j^L = m_j^L + 4\,(b_j^L - m_j^L),$$

   where

$$m_j^L = (a_j^L + b_j^L)/2.$$

   (b) Delete from $\mathbf{L}$ all $\mathbf{B}' \in \mathbf{L}$ for which $\mathbf{B}' \cap \mathbf{B}_b^L$ is nonempty and $d(\mathbf{B}') < \epsilon/4$.
   (c) Store $\mathbf{B}_b^L$ in list $\mathbf{L}$.
5. (Backtrack to less subdivided boxes)
   (a) If $L = 1$, then exit with the list $\mathbf{L}$.
   (b) If $\mathbf{B}^L$ was $\mathbf{B}_1^{L-1}$, then:
      (i) Set: $\mathbf{B}^L \leftarrow \mathbf{B}_2^{L-1}$.
      (ii) Go to Step 3.
   (c) $L \leftarrow L - 1$.
   (d) Return to Step 5(a).

We conclude this section with some details concerning the computation of $\mathbf{T}_F$.

Various methods of obtaining the bounds c_i and d_i in (2.5) are possible. For example, if bounds on the partial derivatives of the f_i are known, then the mean value theorem may be used. We have, however, elected to think in terms of interval arithmetic extensions to the function F since we find it conceptually natural, especially when the f_i are polynomials, and since most ways of computing the c_i and d_i can be viewed in terms of interval extensions of the f_i.

Thorough introductions to interval mathematics are given in [14] and in [1]. For the uninitiated, we summarize the salient ideas here. Interval arithmetic consists of the set of intervals $[a, b]$ where a and b are real numbers, and extensions of the four elementary operations "+", "−", "*", and "/" such that

$$[a_1, b_1] \mathbf{op} [a_2, b_2] = \{x_1 \mathbf{op} x_2 : a_1 \leq x_1 \leq b_1 \text{ and } a_2 \leq x_2 \leq b_2\}, \qquad (3.4)$$

where **op** is "+", "−", "*", or "/".

For example, $[a_1, b_1] - [a_2, b_2] = [a_1 - b_2, b_1 - a_2]$, while the other operations are also given by fairly simple rules. Thus, if the f_i are polynomials, these rules can be used to compute the bounds c_i and d_i. (We note, however, that the distributive law does not hold for interval arithmetic, and that the c_i and d_i are

generally sharper the more highly factored the polynomial is; see, e.g., [14, Sect. 2.3]). More generally, an interval extension to a function f of n variables over a box $\mathbf{B}$ as in (2.1) is a function $\overline{f}$ such that $\overline{f}(\mathbf{B})$ is an interval and

$$\{f(x_1, x_2, \ldots, x_n) : a_j \leq x_j \leq b_j \text{ for all } j\} \subset \overline{f}(\mathbf{B}) \tag{3.5}$$

In order to determine $\mathbf{T}_F(\mathbf{B})$ = "true," we use the Krawczyk test as explained in [13]. In particular, if $\mathbf{B}$ is as in (2.1) and $m_j = (a_j + b_j)/2$, then define

$$\begin{aligned} &y = (m_1, m_2, \ldots, m_n) \quad \text{and} \\ &Y \text{ an approximation to } [F'(y)]^{-1} \end{aligned} \tag{3.6}$$

where $F'(y)$ is the Jacobian matrix of the residual function F at y. We then define the box $\overline{\mathbf{B}}$ by

$$\overline{\mathbf{B}} = \mathbf{K}(\mathbf{B}) = y - YF(y) + (I - YF'(\mathbf{B}))(\mathbf{B} - y). \tag{3.7}$$

In (3.7), I is the identity matrix, and the last term of the right member is evaluated using standard rules of matrix-vector addition and multiplication, but with interval arithmetic. (In such calculations, a real number y_j is taken to be a degenerate interval $[y_j, y_j]$). Also note that the term $y - YF(y)$ is simply the image under Newton's method of the point y. In fact, a mean value theorem argument in [13] shows that $\overline{\mathbf{B}}$ contains all images of $\mathbf{B}$ under the chord method with iteration matrix Y. Also, it is shown in [13] that if $\overline{\mathbf{B}} \subset \mathbf{B}$ and $\| I - YF'(\mathbf{B}) \| < 1$, then there is a unique solution of (1.1) within $\mathbf{B}$ and the chord method based on the iteration matrix Y will converge to that solution from any starting point within $\mathbf{B}$. Here, the norm of the interval matrix $I - YF'(\mathbf{B})$ is computed as the infinity norm, where each interval entry $[a, b]$ is replaced by $\max\{|a|, |b|\}$. Thus,

$$\begin{aligned} &\mathbf{T}_F(\mathbf{B}) = \text{'true'} && \text{if } \overline{\mathbf{B}} \subset \mathbf{B} \text{ and } \| I - YF'(\mathbf{B}) \| < 1, \\ &\mathbf{T}_F(\mathbf{B}) = \text{'false'} && \text{if } 0 < c_i \text{ or } 0 > d_i \text{ for some } i \text{ as in} \\ &&& \quad (2.5) \text{ or if } \overline{\mathbf{B}} \cap \mathbf{B} \text{ is empty.} \\ &\mathbf{T}_F(\mathbf{B}) = \text{'unknown'} && \text{otherwise.} \end{aligned} \tag{3.8}$$

In the actual computations, we iterate (3.7), replacing $\mathbf{B}$ by $\overline{\mathbf{B}} \cap \mathbf{B}$ until the diameter of $\mathbf{B}$ is less than $\epsilon/4$, provided the relative ratio D of $\mathrm{d}(\mathbf{B})$ and $\mathrm{d}(\overline{\mathbf{B}} \cap \mathbf{B})$ obeys

$$D \leq 1 - (\tfrac{1}{2})^{1/n}. \tag{3.9}$$

(The rationale for this procedure is that if (3.9) does not hold, then, on average, the value of the solution is refined more efficiently with bisection than with iteration of (3.7).) The tolerance $\epsilon/4$ ensures that every box in list $\mathbf{L}$ will have diameter at most $\epsilon/4$, so that the diameters need not be checked in Steps 3(b), 3(f), 3(i), and 4(b). We counted function and Jacobian evaluations in this iteration within the overall total. A more sophisticated version of the code would use the standard floating-point Newton method in lieu of iteration of (3.7) (once the first line of (3.8) is satisfied, which ensures convergence), for a savings in execution time.

We mention here that more sophisticated versions of the right-hand side of (3.7) have been proven to also lead to root inclusion tests (see [9] and [16]). Also,

Neumaier develops a general theory of when interval Newton methods based on interval inverses of matrices lead to root inclusion tests in [22]. Incorporation of these methods should lead to somewhat more efficient codes than that reported on here; they will be included in future versions.

To actually do the interval arithmetic, we designed subroutines that implement the rules implied by (3.4) in floating-point arithmetic. We then programmed the functions comprising F and F' and the expression in (3.7) manually by calling these subroutines. This scheme differs from true interval floating-point arithmetic because directed roundings are not used to obtain rigorous bounds on the round-off errors in the floating-point computations. However, in this nonlinear equation context, the intervals are usually large compared to the round-off errors, and we did not encounter difficulties due to round-off error. Also, directed rounding is difficult to access from standard FORTRAN 77, and general packages that include a software implementation of directed rounding execute more slowly than our scheme.

We note that alternate approaches to interval floating-point arithmetic include: (i) use of a precompiler that supports the interval data type (cf., [3]) in conjunction with a subroutine package (cf., [25]); and (ii) use of a compiler, such as CDC's M77 compiler or Pascal-SC, that supports an interval data type. We believe the first alternative would result in execution times similar to those reported here (i.e., about 200 times slower than floating-point arithmetic; see [3] and [25]), whereas the second alternative would result in execution that is an order-of-magnitude faster (with the interval arithmetic about five times slower than normal floating-point arithmetic). Either of these alternatives would make the code easier to use.

A code based on programming the residuals and bounds on their derivatives in standard floating-point arithmetic in conjunction with direct application of the Mean Value Theorem would also be easier to use. However, that may result in bounds in (2.5) that are not as sharp, and hence in larger L in Algorithm 3.1 and in slower execution times.

4. THE TEST SET

The 17 test problems were chosen from diverse published reports of computational tests. These included tests of quasi-Newton methods [18]; a Garcia-Zangwill continuation method [19], a simplicial bisection method [5]; a "simple" box bisection method [21]; a problem arising from robot kinematics [12]; a problem arising from a technique for analyzing bifurcation points [10]; and a problem used to test an interval Newton method that is somewhat more sophisticated than (3.7), [9]. The methods in these tests are fundamentally different from one another, the computing machinery used varied, and measures of efficiency differed from report to report. However, all of the problems can be considered variants of (1.1). Our goal in the choice of the test set was to enable us to begin to sort out the strong and weak points of generalized bisection in relation to these other methods. We now give the test problems by category. We do not give full descriptions of the problems, some of which have appeared various times in published experiments, but we cite references where appropriate.

4.1 Problems Used to Test a Homotopy Method ([19])

1. *Cubic-Parabola*:

$$4x_1^3 - 3x_1 - x_2 = 0$$
$$x_1^2 - x_2 = 0$$

Initial Box: $[-2, 2] \times [-2, 2]$

Remarks: This problem has three solutions in the box. It poses a problem for the noninterval version of Newton's method, but the Jacobian is well-conditioned at the solutions.

2. *A counterexample to a method of Branin*:

$$4(x_1 + x_2) = 0$$
$$4(x_1 + x_2) + (x_1 - x_2)[(x_1 - 2)^2 + x_2^2 - 1] = 0$$

Initial box: $[-2, 2] \times [-2, 2]$

Remarks: This problem has one solution in the box, and does not cause particular trouble for generalized bisection.

3. *Powell's singular function*:

$$x_1 + 10x_2 = 0$$
$$\sqrt{5}(x_3 - x_4) = 0$$
$$(x_2 - 2x_3)^2 = 0$$
$$\sqrt{10}(x_1 - x_4)^2 = 0$$

Initial box: $[-2, 2]^4$

Remarks: This problem has a single solution at (0, 0, 0, 0). However, the Jacobian matrix is null at this solution, and it poses a severe test of most methods.

4. *Brown's almost linear system*:

$$f_i = x_i + \sum_{j=1}^{n} x_j - (n + 1) = 0,$$
$$f_n = \left[\prod_{j=1}^{n} x_j\right] - 1 = 0$$

Initial box: $n = 5$ and box is $[-2, 2]^5$.

Remarks: There are two roots within the box. The Jacobian matrix is ill-conditioned at these roots. This function is found both in [19] and in [21].

4.2 Problems Used to Test a "Simple" Box-Bisection Method ([20])

5–8. *A family of linear systems*:

$$f_1 = x_1 - .5 = 0$$
$$f_2 = -ax_1 + x_2 + .5(a - 1) = 0$$

where

$$a = 1/\tan(\theta_p) \text{ and } \theta_5 = 0°1', \theta_6 = 1°, \theta_7 = 10°, \text{ and } \theta_8 = 30°.$$

Initial box: $[0, 1] \times [0, 1]$

Remarks: This system represents two lines crossing at angle θ_p. In the bisection method in [21], small θ_p posed more of a problem due to "clustering."

9. *Circle-circle intersection*:

$$f_1 = (x_1 - x_1^0)^2 + (x_2 - x_2^0)^2 - r^2 = 0$$
$$f_2 = (x_1 - x_1^{00})^2 + (x_2 - .5)^2 - (.5 - x_1^{00})^2 = 0$$

where $x_1^0 = 100$, $\theta = 1'$, $x_1^{00} = -100$,

with $x_2^0 = .5 + (x_1^0 - .5)\tan(\theta)$, *and*
$r^2 = (.5 - x_1^0)^2(1 + \tan^2(\theta))$.

Initial box: $[0, 1] \times [0, 1]$.

Remarks: These equations represent two intersecting circles with centers at $x_1^0$and x_1^{00}. The linearization of the equations at one of the solution points is system no. 5 above. The values of x_1^0, x_1^{00}, and θ given here caused the most problems for the box bisection method in [21].

10. *Combustion chemistry problem* ([12]):

$$f_1 = \alpha_1 x_2 x_4 + \alpha_2 x_2 + \alpha_3 x_1 x_4 + \alpha_4 x_1 + \alpha_5 x_4 = 0$$
$$f_2 = \beta_1 x_2 x_4 + \beta_2 x_1 x_3 + \beta_3 x_1 x_4 + \beta_4 x_3 x_4 + \beta_5 x_3 + \beta_6 x_4 + \beta_7 = 0$$
$$f_3 = x_1^2 - x_2 = 0$$
$$f_4 = x_4^2 - x_3 = 0$$

where

$$\alpha_1 = -1.697 \cdot 10^7,\ \alpha_2 = 2.177 \cdot 10^7,\ \alpha_3 = 0.55,$$
$$\alpha_4 = 0.45,\ \alpha_5 = -1.0,$$

and

$$\beta_1 = 1.585 \cdot 10^{14},\ \beta_2 = 4.126 \cdot 10^7,$$
$$\beta_3 = -8.285 \cdot 10^6,\ \beta_4 = 2.284 \cdot 10^7,$$
$$\beta_5 = -1.918 \cdot 10^7,\ \beta_6 = 48.4, \text{ and } \beta_7 = -27.73$$

Initial box: $[0, 1] \times [0, 1] \times [0, 1] \times [0, 1]$

Remarks: This real-world problem represents hydrocarbon combustion with excess fuel. It was originally posed as a system of two cubics; we could solve it directly in that form. However, we have chosen the system of quadratics since this form was used in [21]. The problem has been successfully solved via continuation methods. It has a unique solution within the nonnegative unit box, but has other, nonphysical solutions in larger domains.

11. *A robot kinematics problem* ([24]):

$$
\begin{aligned}
f_1 &= \alpha_1 x_1 x_3 + \alpha_2 x_2 x_3 \\
&\quad + \alpha_3 x_1 + \alpha_4 x_2 + \alpha_5 x_4 + \alpha_6 x_7 + \alpha_7 = 0 \\
f_2 &= \alpha_8 x_1 x_3 + \alpha_9 x_2 x_3 \\
&\quad + \alpha_{10} x_1 + \alpha_{11} x_2 + \alpha_{12} x_4 + \alpha_{13} = 0 \\
f_3 &= \alpha_{14} x_6 x_8 + \alpha_{15} x_1 + \alpha_{16} x_2 = 0 \\
f_4 &= \alpha_{17} x_1 + \alpha_{18} x_2 + \alpha_{19} = 0 \\
f_5 &= x_1^2 + x_2^2 - 1 = 0 \\
f_6 &= x_3^2 + x_4^2 - 1 = 0 \\
f_7 &= x_5^2 + x_6^2 - 1 = 0 \\
f_8 &= x_7^2 + x_8^2 - 1 = 0
\end{aligned}
$$

where

$$
\begin{aligned}
&\alpha_1 = 4.731 \cdot 10^{-3}, \quad \alpha_2 = -.3578, \quad \alpha_3 = -.1238, \\
&\alpha_4 = -1.637 \cdot 10^{-3}, \quad \alpha_5 = -.9338, \quad \alpha_6 = 1.0, \\
&\alpha_7 = -.3571, \quad \alpha_8 = .2238, \quad \alpha_9 = .7623, \\
&\alpha_{10} = .2638, \quad \alpha_{11} = -.7745 \cdot 10^{-1}, \quad \alpha_{12} = -.6734, \\
&\alpha_{13} = -.6022, \quad \alpha_{14} = 1.0, \quad \alpha_{15} = .3578, \\
&\alpha_{16} = 4.731 \cdot 10^{-3}, \quad \alpha_{17} = -.7623, \quad \alpha_{18} = .2238, \\
&\alpha_{19} = .3461
\end{aligned}
$$

Initial box: $[-1, 1]^8$.

Remarks: This problem has been solved successfully via continuation methods. It has 16 solutions within the given box.

4.3 A Test Problem for a Numerical Bifurcation Technique ([10])

12. *A high-degree polynomial system*:

$$
\begin{aligned}
f_1 &= 5x_1^9 - 6x_1^5 x_2^2 + x_1 x_2^4 + 2x_1 x_3 = 0 \\
f_2 &= -2x_1^6 x_2 + 2x_1^2 x_2^3 + 2x_2 x_3 = 0 \\
f_3 &= x_1^2 + x_2^2 - .265625 = 0
\end{aligned}
$$

Initial box: $[-.6, 6] \times [-.6, .6] \times [-5, 5]$

Remarks: This problem has 12 real solutions, which are all within the given box. However, it has 126 complex solutions, counting multiplicities, and a solution of very high multiplicity at infinity. Thus, it causes trouble for most homotopy continuation methods. The system corresponds to (2.3) in [10], with $v_1 = (1, 0)^T$, $v_2 = (0, .5)^T$, $a_1 = x_1 = x$, $a_2 = x_2 = 2\ (\lambda - .5)$, and $\Lambda = x_3$.

4.4 Problems Used to Test a Simplicial Method of Bisection ([5])

13. *The 3-dimensional identity*:

$$f_i = x_i = 0, \quad 1 \le i \le 3$$

Initial box: $[-.25, .25] \times [-.25, .25] \times [-.25, .25]$

Remarks: This is a very easy problem for our method. However, we include it for comparison with the method in [5]. The initial box is chosen to be the same scale as the initial simplex in [5], but has 6 times the volume.

14. *Two intersecting parabolas*:

$$f_1 = x_1^2 - 4x_2 = 0$$
$$f_2 = x_2^2 - 2x_1 + 4x_2 = 0$$

Initial box: $[-4, 4] \times [-4, 4]$

Remarks: This system has two solutions within the given box. The topological index at the solution $x_1 = 0$, $x_2 = 0$, is -1, whereas the topological index at the solution $x_1 \simeq 1.695$, $x_2 \simeq .7186$ is $+1$. Thus the topological degree with respect to any region containing both roots must be 0; this fact would cause topological degree-based bisection methods to fail to find either root.

15. *Rosenbrock's function*:

$$f_1 = 1 - x_1 = 0$$
$$f_2 = 10(x_2 - x_1^2) = 0$$

Initial box: $[-4, 4] \times [-4, 4]$

Remarks: This problem also appears in [18] and in previous literature.

16. *A variable-dimension system of quadratics*:

$$f_i = (x_i - 0.1)^2 + x_{i+1} - 0.1 = 0 \quad \text{for} \quad 1 \le i \le n - 1;$$
$$f_n = (x_n - 0.1)^2 + x_1 - 0.1 = 0.$$

Initial box: $[-.2, .2]^4$ $(n = 4)$

Remarks: This simple system of quadratics can be used to test the effects of increasing the dimension on bisection methods. Since its Jacobian matrix is sparse, it can also be used to debug techniques for handling structured problems. Its Jacobian matrix is well-conditioned at the two real roots of $x_1 = .1$, $1 \le i \le n$ and $x_i = -.9$, $1 \le i \le n$. The topological index of the first root is -1, while the topological index of the second root is $+1$. Note that it is easily reducible to a single polynomial equation of degree 2^n.

4.5 A Problem Used to Test an Improved Interval Newton Method ([9])

17. *Broyden's banded function*:

$$f_i = x_1(2 + 5x_i^2) + 1 - \sum_{j \in J_i} x_j(1 + x_j), \quad 1 \le i \le n,$$

where

$$J_i = \{j: j \ne i, \max(1, i - 5) \le j \le \min(n, i + 1)\}$$

Initial box: $[-1, 1]^5$ $(n = 5)$.

Remarks: This problem also appears in [18] and elsewhere, and is slightly more difficult than problem 16 above.

5. NUMERICAL RESULTS

We have organized the numerical results to do the following:

(A) Compare our generalized bisection method with published numerical results on homotopy continuation methods.

(B) Compare our generalized bisection method with the "simple" bisection method in [21].

(C) Compare our generalized bisection method with published results on the simplicial method of bisection in [5].

(D) Compare the interval Newton portion (i.e., (3.7)) of our generalized bisection method to the improved interval Newton method in [8].

(E) Compare our generalized bisection method to MINPACK, which is an established software package for the hybrid (quasi-Newton and steepest descent with trust regions or "dogleg") methods.

(F) Present the behavior of our method on a high-degree polynomial system.

(G) Compare our generalized bisection method with the expansion and deletion steps (Steps 3(b), 3(f) substep (i), and 4(a)–4(b) of Algorithm 3.1) to the generalized bisection method obtained by omitting this step (and simply storing small boxes in the list **L**).

Homotopy continuation is similar to Algorithm 3.1 in the sense that it seeks to find all solutions (but differs in that bounds are not placed on the desired portion of the solution set). In comparison (A), we assume that only the real solutions are desired. In the case where all complex solutions are needed, and not just solutions within a specified box, the continuation methods may do better, since the paths that converge to nonreal solutions then do not represent overhead.

The box bisection method in [21] is qualitatively like Algorithm 3.1, but with a root inclusion test $\mathbf{T}_F$ that never takes on the value "true." With neither that part of the inclusion test nor expansion and exclusion steps, expensive "clustering" occurs. We illustrate with comparison (B). Also, two important real-world problems appear in [21] and thus in our comparisons.

The method in [5] was devised with a similar philosophy as Algorithm 3.1. It has the advantage of requiring only low-accuracy function evaluations (and no Jacobian matrix evaluations), and the disadvantage of relying more heavily on heuristics. Using the systems of equations in [5], we weigh the relative merits of that method and ours in comparison (C).

Considerable work has been published on interval Newton methods which are improvements to (3.7) in the sense that the component intervals of the image box $\overline{\mathbf{B}}$ have smaller widths. Such a method is reported on in [9]. Thus our comparison (D) will help to gauge the importance of such improvements, and will indicate how much our method can be improved by using them instead of (3.7). (We used (3.7) since it is easier to program and is efficient with minimal software or hardware support for interval arithmetic.)

Modules from the MINPACK package are perhaps the most widely available high-quality software for solving nonlinear systems of equations. The advantages of the general mathematical approach embodied in MINPACK are applicability

to large, sparse systems and economy for appropriate problems. However, the approach is often not well-suited to finding all solutions, and it works best when the function has only one solution and $|F|^2$ has only one local minimum. Our comparison (E) is meant to show that generalized bisection can possibly be used in a general way, performing reasonably well on problems that, after initial exploration, can be solved by other methods.

Our algorithm differs from similar bisection methods in the box expansion step (Step 4 of Algorithm 3.1), and in the deletion steps in Steps 3(b) and 3(f) substep (i). The purpose of Step 4 is to eliminate ambiguity in the root-inclusion test in the case that solutions happen to occur near the boundary of one of the boxes. This could lead to listing many boxes corresponding to the same solution or to large numbers of boxes with very small diameters; the problem would increase with dimension. Comparison (G) helps us to determine the effect of the expansion and rejection steps on the algorithm's efficiency and on its ability to reject unwanted boxes.

In all cases, in Algorithm 3.1 we chose the domain tolerance ϵ to be 10^{-5} and the range tolerance ϵ_F to be 10^{-10}. This allows the root-inclusion test to work rigorously in cases where the problem is near a problem with a quadratic singularity.

We ran Algorithm 3.1 for all problems on a Zenith 150 IBM PC compatible with a 4.77 mhz clock and an 8087 numeric coprocessor; the code was compiled with IBM Professional Fortran.

5.1 Comparison with a Continuation Method

In order to compare with continuation methods, we use the published results in [19], and tabulate results with problems 1, 2, 3, and 4. We can only obtain an idea of the relative behavior of these techniques since (1) numerous problem types have been omitted. (2) improvements have been made to the continuation method in [19] (cf., [20]), and (3) since improvements can be made in our method of bisection.

Because bisection and continuation are qualitatively so different, some idea of the relative overhead of each in solving these problems would be desirable. However, all of our experiments were on a Zenith 150 (IBM PC compatible) with a numeric coprocessor (8087) chip, with code compiled by the IBM Professional Fortran compiler, whereas the results in [19] were obtained from an IBM 3033 with code compiled with the Fortran "H" compiler. To get a *very* rough idea of the total number of instructions executed, we used the LINPACK benchmark timings of these machines and compilers reported in [4]; based on this benchmark, the IBM PC executes floating-point operations 150 times slower than the IBM 3033. Thus, in Table 5.1, the actual CPU times in hours:min.:sec. for the PC appear in the column "CPU Equivalent," whereas the times reported in [19] multiplied by 150 appear there. Also, for the PC results, the time before and after the bisection process during which I/O occurred was not counted.

We also note redundant or spurious roots in Table 5.1. For problems 1 and 2, there were complex roots that did not correspond to real roots. These were found by the continuation method.

Table 5.1. A very rough comparison of CPU times for bisection versus continuation, on an IBM PC compatible.*

Problem no.	n	Redundant/spurious roots		Equivalent CPU Times	
		Bisection	Continuation	Bisection	Continuation
1	2	0	3	0:00:05	0:01:24
2	2	0	1	0:00:05	0:00:57
3	4	4	0	0:07:41	0:02:05
4	5	0	0	1:08:00	0:05:27

* The times reported for continuation were not actual times on an IBM PC (see the text).

Table 5.2. Comparison of the numbers of function evaluations and Jacobian evaluations for bisection and a continuation method.*

Problem no.	NFCALL		NJCALL		Equivalent NFCALL	
	Bisection	Continuation	Bisection	Continuation	Bisection	Continuation
1	80	—	66	888	1,060	2,664
2	62	—	53	636	840	1,908
3	2,114	—	1,597	676	42,510	3,380
4	10,108	—	8,013	1,240	250,865	7,440

* For the continuation method, the actual NFCALL and NJCALL were abstracted from the tables in [19]. The "equivalent NFCALL" values are computed to hopefully represent a more accurate measure of the total work (see the text).

In problem 3, Algorithm 3.1 returned a "cluster" of 5 roots around the root (0, 0, 0, 0), at which the Jacobian matrix is null; the theory in [11] excludes such clusters only when the roots are not singular. The clustering at singular roots should be less severe when the range tolerance ϵ_F is set smaller relative to the domain tolerance ϵ.

In addition to CPU time, we report the number of evaluations of the residual function F and the number of evaluations of the Jacobian matrix of F for both bisection and the continuation method. Since the function and Jacobian evaluations for bisection were interval evaluations, whereas continuation uses simple floating-point arithmetic, the two methods are also difficult to compare with these quantities. With an efficient software implementation, interval arithmetic runs five times slower than floating-point arithmetic. (If interval arithmetic is implemented in hardware, it runs at roughly the same speed as floating-point arithmetic.) Thus, to get a (conservative) "equivalent" number of evaluations, we multiply the number of interval evaluations in the bisection method by 5. In the case of both bisection and continuation, we multiply the number of Jacobian evaluations by n and add to the number of function evaluations to translate Jacobian evaluations to rough numbers of function evaluations for the types of problems solved here. (We make no attempt to keep track of the linear algebra and other overhead.) These results appear in Table 5.2.

In Tables 5.1 and 5.2, we note that bisection seems very reasonable for problems 1 and 2, but had more trouble than the continuation method on problems 3 and 4. This is due to the singularity and ill-conditioning, respectively, at the roots being found.

5.2 Comparison with a "Simple" Bisection Method

In this section, we compare the behavior of Algorithm 3.1 to the "simple" bisection algorithm in [21], utilizing problems 5 through 11. Problems 5 through 8 are linear systems with varying degrees of a type of ill-conditioning, and were used to test the code tried in [21] and to illustrate a difficulty inherent in the "simple" approach. Since the root-inclusion test in Algorithm 3.1 is partially based on an interval Newton method, Algorithm 3.1 solved all of these systems in one step; the method in [21] took widely varying numbers of steps depending on the conditioning parameter θ. Problem 9 appears in [21] with an aim similar to that in problems 5 through 8, and exhibits an ill-conditioning similar to that in problem 5.

The method in [21] is similar to Algorithm 3.1 in that it tests boxes for the possibility of including a root, and subdivides boxes that may contain further roots. Its crucial difference is that the inclusion test never signals "true," but only "false" or "unknown," and all boxes with label "unknown" are subdivided further. (Additionally, the search is breadth-first instead of depth-first, and the subdivision is by halving all coordinate directions simultaneously to yield 2^n subboxes for each initial box.) As is explained in [21], this leads to a "clustering" of numerous boxes around the actual solution. (Algorithm 3.1 should behave similarly if $\mathbf{T}_F$ were altered to never signal "true.")

Because this is the information that appears in [21], and because of the nature of the methods, we compare the methods based on the total number of boxes tested. We also report the total number of boxes in the final cluster for the method in [21]. (In all cases, Algorithm 3.1 gave a unique box for each root in the specified region). For problems 5 through 8, we compute the total number of boxes in "simple" bisection from a formula given in [21], in order to achieve a diameter reduction of $2^{-16} \simeq 1.5 \times 10^{-5}$. (In all cases, Algorithm 3.1 required a diameter reduction of 10^{-5}.) In problems 10 and 11, the total number of boxes tested is not available for the "simple" method, but some additional details are given in [21]. In all of the problems, the initial boxes were identical for both the "simple" method and for Algorithm 3.1: They were $[0, 1]^n$ for problems 5 through 10 and $[-1, 1]^8$ for problem 11. The results appear in Table 5.3.

5.3 Comparison with a Published Simplicial Bisection Method

Computational experiments for a special simplicial bisection method are reported in [5]. The overall idea is similar to that in [11] (and the algorithm here) in the sense that polygonal regions in $\mathbf{R}^n$ are successively "bisected" and a test is used on the regions to determine whether or not roots exist. However, in [5] the regions are simplices, and bisection is by bisecting the longest edge. Also, the analog of the root inclusion test for the initial simplex in the algorithm in [5] is based on computing the Brouwer degree of the simplex; this is a generalization of the sign change test in the classical bisection method, and hence will not always indicate "true" when there are roots in the simplex. Also, in [5], the Brouwer degree is not computed for each simplex, but a heuristic based on a linearization of F is employed where possible. Moreover, the method for computation of the Brouwer degree there contains a heuristic parameter.

Table 5.3. Comparison of the total number of boxes tested and the number of boxes in the final "cluster" with a "simple" bisection method that has an inclusion test that never signals "true."

Problem no.	n	Stopping tol.		Final no. boxes in cluster		Total no. boxes tested	
		3.1	other	3.1	other	3.1	other
5	2	10^{-5}	2^{-16}	1	6,900	1	35,814
6	2	10^{-5}	2^{-16}	1	118	1	1,318
7	2	10^{-5}	2^{-16}	1	14	1	384
8	2	10^{-5}	2^{-16}	1	6	1	84
9	2	10^{-5}	2^{-15}	2	6,896	11	398,597
10	4	10^{-5}	2^{-18}	1	38	373	unknown
11	8	10^{-5}	2^{-15}	16	22,968	485	unknown

Computational results on problems 13, 14, 15, and 16 appear in [5]. In that algorithm, only low accuracy function evaluations (and no Jacobian evaluations) are required. For a rough comparison, we report function evaluations, Jacobian evaluations, and "equivalent function evaluations" for Algorithm 3.1: As in Table 5.2, the equivalent function evaluations are defined to be $5 \times$ (NFCALL + $n \times$ NJCALL). (The factor of 5 is to roughly take account of the difference between floating-point and interval arithmetic).

In [5], a domain-stopping tolerance and a range-stopping tolerance are used. In all cases, the range-stopping tolerance was set to 10^{-8}, while the domain-stopping tolerance in the runs we transcribe here was either 10^{-5} or 10^{-6}. In Algorithm 3.1, the domain-stopping tolerance ϵ was set to 10^{-5}, while the range-stopping tolerance ϵ_F was set to 10^{-10}. We suspect that the algorithm in [5] will do relatively better for larger tolerances, since the convergence is linear, while iteration of (3.7) can lead to faster convergence. Also, singularity at the root itself should affect the method in [5] relatively less.

In addition to the stopping tolerance, the volume of the initial box in Algorithm 3.1 and the volume of the initial simplex in the simplicial bisection algorithm of [5] affect the total amount of work. This makes the algorithms difficult to compare, since the unit box in n-space contains $n!$ times the volume of the unit n-simplex. (Also, in our opinion, it is easier to formulate an initial box for an application than to formulate an initial simplex.) For our comparisons, we took boxes containing the simplices in [5], with roughly the same diameters, and we did not try to take account of the differences in containing volumes. The initial boxes and corresponding initial simplices appear in Table 5.4.

The results appear in Table 5.5. In problem 14, there are two roots in the initial box and the initial simplex, and the Brouwer degree over the entire simplex is 0, causing the method in [5] to fail; when the simplex $\mathbf{S} = \langle (\frac{1}{3}, \frac{1}{3}), (-\frac{1}{3}, 0), (0, -\frac{1}{3}) \rangle$, which only contains one root, was tried, that method found the unique root in $\mathbf{S}$ in 45 function evaluations.

5.4 Comparison with an Improved Interval Newton Method

Hansen et al. have been working on interval Newton methods (of which the iteration implicit in (3.7) is an example) for some time. In particular, in [8] they publish a more sophisticated such method that should be competitive from the

Table 5.4. Comparison of Algorithm 3.1 with a simplicial bisection method.*

Problem no.	n	Domain tol. 3.1	Domain tol. other	NFCALL 3.1	NFCALL other	NJCALL 3.1	NJCALL other	Equivalent NFCALL 3.1	Equivalent NFCALL other
13	3	10^{-5}	10^{-6}	1	106	1	—	20	106
14	2	10^{-5}	10^{-5}	49	fail	45	—	695	fail
15	2	10^{-5}	10^{-5}	2	70	2	—	30	70
16	4	10^{-5}	10^{-5}	4	132	4	—	100	132

* See the text for an explanation of "equivalent NFCALL." (NFCALL and NJCALL are the number of function and number of Jacobian matrix evaluations, respectively.) Also see Table 5.5 for information related to the tolerances.

Table 5.5. Initial boxes for Algorithm 3.1 and initial simplices for the simplicial algorithm.

Problem no.	Initial Box (Algorithm 3.1)	Initial Simplex (Simplicial Algorithm)
13	$[-.25, .25]^3$	$\langle(-.25, 0, 0), (0, -.25, 0), (0, 0, -.25)\rangle$
14	$[-4, 4]^2$	$\langle(3.\overline{3}, 3.\overline{3}), (-3.\overline{3}, 0), (0, -3.\overline{3})\rangle$
15	$[-4, 4]^2$	$\langle(3.\overline{3}, 3.\overline{3}), (-3.\overline{3}, 0), (0, -3.\overline{3})\rangle$
16	$[-.2, .2]^4$	$\langle(.2, .2, .2, .2), (-.2, 0, 0, 0),$ $(0, -.2, 0, 0), (0, 0, -.2, 0), (0, 0, 0, -.2)\rangle$

point of view of numbers of function and numbers of Jacobian evaluations; in [16], Moore and Qi show that this method both leads to a root inclusion test and is more efficient than the test based on (3.7) and (3.8), which is presently implemented in our codes. We compare Algorithm 3.1 with results presented there for problem 17.

Under certain conditions, the algorithm in [8] performs "real" inner iterations, which consist of iterations of the chord method with floating-point (and not interval) arithmetic. Thus, in addition to numbers of function and numbers of Jacobian evaluations, we also report the numbers of "real" function evaluations.

In both [8] and in our experiments, the initial box was $[-1, 1]^5$. In [8], the box diameter for stopping (i.e., the domain tolerance) was set to 10^{-8}, whereas we used our standard tolerance of 10^{-5}. The results appear in Table 5.6.

Table 5.6 indicates that (3.7) can be improved via the techniques in [8]. (In particular, there is possibly a gain in speed of about 3, which could be taken into account in interpreting the other comparisons here.) Careful comparison on more problems will prove useful.

5.5 Comparison with Quasi-Newton Software

Here, we compare Algorithm 3.1 to the routine HYBRJ1 from MINPACK (cf., [17]) on problems 3, 4, 15, and 17, which are problems 2, 8, 1, and 14, respectively, in the standard test set distributed with MINPACK (cf., [18]).

The standard test driver for MINPACK supplies starting points and tolerances (see [18]). For each of the problems appearing here, there were three starting points; we report the results for each point separately. The relative domain-error tolerance is computed in the driver to be the square root of the machine epsilon, about 10^{-8}, while the components of the three starting vectors for each problem

Table 5.6. Comparison with an algorithm using a more sophisticated interval Newton method.*

	Algorithm 3.1	Hansen/Greenberg algorithm
NFCALL	216	88
NJCALL	149	46
NRFCALL	—	47

* Results are for problem 17. NFCALL is the number of interval function evaluations, NJCALL is the number of interval Jacobian matrix evaluations, and NRFCALL is the number of function evaluations using noninterval floating-point arithmetic.

Table 5.7. Comparison of Algorithm 3.1 with the routine HYBRJ1 in MINPACK.*

Problem	NFCALL				NJCALL				Time in min:sec			
no.	3.1	#1	#2	#3	3.1	#1	#2	#3	3.1	#1	#2	#3
3	2,114	123	116	122	1,597	7	6	6	7:41	0:15	0:14	0:14
4	10,108	13	14	16	8,013	2	2	2	68:00	0:02	0:02	0:02
15	2	16	7	7	2	3	1	1	0:.1	0:.8	0:.3	0:.3
17	216	18	23	33	149	1	2	2	1:26	0:.3	0:.4	0:.5

* The heading #i denotes the ith starting point in the standard test driver for HYBRJ1, for i = 1, 2, 3.

are on the order of 1, 10, and 100. For Algorithm 3.1, we used our standard tolerance of 10^{-5}, and we used the initial boxes listed with the problems in Section 4. Since the components of the roots are on the order of 1 and since the algorithms converge superlinearly near the roots, the tolerances are roughly comparable. The initial boxes are roughly comparable to the first HYBRJ1 starting points; additional testing with larger boxes will be required to rigorously compare with the other two points.

Problem 4 is exceptional because there are two solutions in the initial box input to Algorithm 3.1. For all three initial guesses, HYBRJ1 converged to (1, 1, 1, 1, 1), whereas Algorithm 3.1 found both solutions.

We compiled MINPACK with IBM Professional Fortran and ran the standard test driver on the Zenith 150. Since this was the identical treatment as in Algorithm 3.1, elapsed times are meaningful. (The elapsed times are the times spent in HYBRJ1, where no I/O is performed, and in the non-I/O portions of Algorithm 3.1.) We additionally report numbers of function and Jacobian matrix evaluations; in the case of Algorithm 3.1, these were interval evaluations, whereas these were evaluations in standard double-precison arithmetic in the MINPACK results. The results appear in Table 5.7.

Here, as with the other comparisons, we emphasize that our goal is to demonstrate range of applicability, and not to prove superior efficiency of one method over another.

The exceptionally poor showing of Algorithm 3.1 for problems 3 and 4 is due to its inability to efficiently handle singularity or near-singularity. (Note, however, that problem 4 had two solutions, one of which HYBRJ1 didn't find.) The good showing of Algorithm 3.1 on problem 15 is due to the fact that one component is linear and one component is quadratic; this leads to small intervals in (3.7).

Table 5.8. Performance of Algorithm 3.1 on Problem 12 (a high-degree polynomial system).*

Domain error tolerance:	10^{-5}
Range error tolerance:	10^{-10}
Number of roots in box:	12
Number of function evaluations:	1,339
Number of Jacobian matrix evaluations:	1,019
Number of roots redundantly listed:	NONE
CPU time on a Zenith 150:	3 min. 48 sec.

* See the text for information about continuation method performance on this problem.

5.6 Performance on a High-Degree Polynomial System

Problem 12 in the form presented is difficult for continuation methods, due to the large number of paths and the large multiplicity of the solution at infinity ([10]; also A. P. Morgan (personal conversation)). We present the results for Algorithm 3.1 in Table 5.8.

To get an idea of how a continuation method may behave on this problem, we ran a sophisticated code (embodying improvements over that in [19]) for solving polynomial systems of equations; we used the standard tolerances supplied with the code. This method is fully documented in [20]. The code successfully found the 12 real solutions while it traversed the 126 paths. However, due to excessive numbers of steps, it terminated prematurely on many of the paths leading to the singular solution at infinity (in projective space). The total number of corrector iterations for all paths was 109,053; this should be roughly the number of function and Jacobian evaluations. The code was run on an IBM 3090-200 after having been compiled with VS Fortran, and took approximately 201 seconds. The LINPACK benchmark ran 1,054 times as fast on the 3090-200 as on the IBM PC; based on this, it appears that Algorithm 3.1 found the 12 real roots about 900 times as fast as the algorithm from [20]. However, we must note that: (i) the LINPACK benchmark may not accurately represent the speed difference for this problem; (ii) the algorithm from [20] really found 126 roots, counting multiplicities (so that, per root, continuation was just as fast); (iii) we evaluated the function for the continuation method with a general routine that made the function easier to input than for Algorithm 3.1, but which may have multiplied the execution time by a significant factor; and (iv) problem 12 can be reduced to make the effort expended by the continuation method less. We let the reader draw conclusions.

5.7 Results With and Without the Expansion Step

Among box bisection methods, Algorithm 3.1 is unique due to Step 3(b), Step 3(f) substep (i), and Step 4(a)–4(b). The main purpose of these steps is to avoid redundant listing of roots in a rigorous fashion so that no roots are missed. They will also reduce the total size of the search tree (and thus reduce execution time). The latter effect is limited, however; if these steps are bypassed, the boxes will continue to be bisected until they are at most a factor of 4 smaller than otherwise. Here we compare Algorithm 3.1 with and without these steps.

In Tables 5.9 and 5.10, we list results for all 17 problems in the test set. Table 5.9 deals with the computational work performed by the variants of the

Table 5.9. Comparison of the computational effort of Algorithm 3.1 with and without the expansion and rejection steps ("w" denotes "with the step" and "w/o" denotes "without the step").

	CPU		NF		NJ		NT		NL	
#	w	w/o	w	w/o	w	w/o	w	w/o	w	w/o
1	00:06	00:06	80	80	66	66	47	47	24	24
2	00:05	00:05	62	62	53	53	39	39	20	20
3	08:01	09:21	2,114	2,133	1,597	1,613	1,180	1,199	600	600
4	68:12	68:12	10,108	10,108	8,013	8,013	7,571	7,571	3,786	3,786
5	00:00	00:00*	1	1	1	1	1	1	1	1
6	00:00	00:00*	1	1	1	1	1	1	1	1
7	00:00	00:00*	1	1	1	1	1	1	1	1
8	00:00	00:00*	1	1	1	1	1	1	1	1
9	00:03	00:03	32	32	31	31	11	11	6	6
10	02:34	02:34	601	601	480	480	373	373	187	187
11	18:41	18:40	989	989	830	830	485	485	243	243
12	03:53	14:31	1,339	4,655	1,019	4,365	943	887	472	444
13	0:.16	0:.16	1	1	1	1	1	1	1	1
14	00:03	00:03	49	49	45	45	21	21	11	11
15	0:.16	0:.17	2	2	2	2	1	1	1	1
16	00:01	00:01	4	4	4	4	1	1	1	1
17	01:26	01:26	216	216	149	149	139	139	70	70

* The times were .05 sec., .11 sec., or .06 sec. The relative values are thought to include some effect other than computational time since the number of operations for all of these problems should have been identical.

Table 5.10. Comparison of Algorithm 3.1 with and without the expansion and rejection steps: The number of redundantly listed roots.*

	N4				NINLST	
#	w	w/o	NROOTS	NREJ	w	w/o
1	0	0	3	6	3	9
2	0	0	1	3	1	4
3	0	0	1	24	5	26
4	0	0	2	31	2	33
5	0	0	1	0	1	1
6	0	0	1	0	1	1
7	0	0	1	0	1	1
8	0	0	1	0	1	1
9	0	0	2	1	2	3
10	1	1	1	0	1	1
11	0	0	16	0	16	16
12	0	—	12	12	12	—**
13	0	0	1	0	1	1
14	0	0	2	3	2	5
15	0	0	1	0	1	1
16	0	0	1	0	1	1
17	0	0	1	0	1	1

* Here, "w" means "with the rejection/expansion steps" and "w/o" means "without the rejection/expansion steps."

** The algorithm failed when the expansion and deletion steps were omitted; see the footnote in Table 5.11.

Table 5.11. Comparison of Algorithm 3.1 with and without the expansion and rejection steps as in Table 5.10, except with a domain tolerance ϵ set to 10^{-1} instead of 10^{-5}.

	N4				NINLST		CPU	
#	w	w/o	NROOTS	NREJ	w	w/o	w	w/o
1	0	0	3	6	3	9	00:04	00:04
2	0	0	1	3	4	4	00:04	00:04
3	0	0	1	24	2	18	02:43	02:46
4	0	0	2	31	4	35	67:21	67:21
5	0	0	1	0	1	1	00:00	00:00
6	0	0	1	0	1	1	00:00	00:00
7	0	0	1	0	1	1	00:00	00:00
8	0	0	1	0	1	1	00:00	00:00
9	0	0	2	2	3	5	00:02	00:02
10	0	0	1	3	4	7	01:48	01:48
11	0	0	16	0	16	16	17:30	17:30
12	0	—	12	20	12	—	03:32	—*
13	0	0	1	0	1	1	00:00	00:00
14	0	0	2	3	2	5	00:03	00:03
15	0	0	1	0	1	1	00:00	00:00
16	0	0	1	0	1	1	00:00	00:00
17	0	0	1	0	1	1	01:24	01:24

* The algorithm failed due to unflagged arithmetic exceptions in the function. When run on a machine with different arithmetic, there were 32 boxes in the list, 480 leaves, 959 boxes tested, 2,823 function calls, and 2,511 Jacobian calls when the exclusion and deletion steps were omitted. On this same machine, with the exclusion and deletion steps, there were 12 boxes in the list, 476 leaves, 951 boxes tested, 1,263 function calls, 951 Jacobian calls, and 20 boxes rejected.

algorithm; we list CPU time in min:sec, number of function calls (NF) and number of Jacobian calls (NJ), number of boxes tested (NT), and number of leaves in the search tree (NL) (i.e., the number of times Step 5 was entered) there.

Table 5.10 deals with the effectiveness of rejecting redundantly listed roots; there we list number of times Step 4 was entered (N4), number of boxes that were rejected in Steps 3(b), 3(f) substep (i), and 4 (NREJ), number of actual roots (NROOTS), and number of boxes in the final list (NINLST).

Entering Step 4 indicates failure of the root-inclusion test to signal "true" or "false" for a sufficiently large diameter box. This could be due to a root near the boundary, too large a domain tolerance ϵ, an inefficient inclusion test $\mathbf{T}_F$, or a singular root. To test the effect of the tolerance, we reran the test set with $\epsilon = 10^{-1}$. The summary results appear in Table 5.11.

In a final test, we took $\epsilon = 10^{-1}$, $\epsilon_F = 10^{-10}$, and did not iterate (3.7). Since the total number of operations for the entire problem set was prohibitive, we ran this set on a mainframe computer. We observed that: (1) handling of singularities is relatively better; and (2) the exclusion and deletion steps come more into play. The results are summarized in Table 5.12.

6. CONCLUSIONS AND FURTHER STUDY

The results in Section 5 indicate that bisection is a reasonable method for solving many types of problems, but is particularly suited for (1.1). Whether to use bisection or a continuation method depends on the particulars of the problem.

Table 5.12. Comparison of Algorithm 3.1 with and without the expansion and rejection steps, as in Table 5.12, except that formula (3.7) is never iterated in the root inclusion test.*

	N4				NINLST		NF		NJ	
#	w	w/o	NROOTS	NREJ	w	w/o	w	w/o	w	w/o
1	3	12	3	10	3	12	321	391	174	215
2	1	4	1	3	1	4	224	247	126	145
3	1	28	1	16	1	28	1,657	2,027	837	1,041
4**	1	27	2	22	1	27	100,000	100,000	60,132	60,377
5	0	0	1	0	1	1	1	1	1	1
6	0	0	1	0	1	1	1	1	1	1
7	0	0	1	0	1	1	1	1	1	1
8	0	0	1	0	1	1	1	1	1	1
9	4	18	2	12	2	18	109	163	65	99
10***	284	51	1	1,209	27	>51	15,393	1,086	8,713	599
11***	14	51	16	159	18	>51	48,762	18,789	29,341	11,523
12	8	20	12	58	12	24	2,217	3,623	1,243	1,959
13	0	0	1	4	1	1	623	639	316	325
14	0	0	1	0	1	1	1	1	1	1
15	2	7	1	6	2	7	237	261	142	156
16	1	4	1	4	1	4	107	123	68	78
17	0	0	1	0	0	0	9	9	5	5

* Here, NF is the number of function calls and NJ is the number of calls to the Jacobian matrix routine.

** For this function, the maximum number of allowed function evaluations (100,000) was exceeded both with and without the exclusion and rejection steps.

*** For this function, the maximum number of allowed boxes in the list (50) was exceeded early in the computation when exclusion and rejection steps were not done.

We have noted problems with bisection on singular systems. It will be useful to develop variants of the inclusion test to handle this eventuality.

Further study of the expansion step, with and without iteration of (3.7) (and improvements thereof) would be interesting. Likewise, further study of the effects of increasing and decreasing the tolerances could also reveal how to increase the overall efficiency of the algorithm.

Finally, accessible end-user software should be developed. Unless interval arithmetic implementations become more widely available, this software should be designed around ordinary floating-point arithmetic. It should include easy means of inputting the function, Jacobian matrix, and bounds on the second derivatives, and should include automatic scaling so that the tolerances ϵ and ϵ_F are easy to interpret.

ACKNOWLEDGMENTS

I wish to thank Alexander Morgan for the stimulating conversations that led to the formulation of this paper. I also wish to thank the referee for pointing out [22], and so on.

REFERENCES

1. ALEFELD, G., AND HERZBERGER, J. *Introduction to Interval Computations.* Academic Press, New York, 1983.

2. ALEFELD, G., AND PLATZÖDER, L. A quadratically convergent Krawczyk-like algorithm. *SIAM J. Numer. Anal. 20*, 1 (Feb. 1983), 210–219.
3. CRARY, F. The AUGMENT precompiler. Mathematics Research Center Rep. 1470, Univ. of Wisconsin, Madison, 1976.
4. DONGARRA, J. J. Performance of various computers using standard linear equations software in a Fortran environment. Mathematics and Computer Science Memo 23, Argonne National Lab., Argonne, Ill., 1985.
5. EIGER, A., SIKORSKI, K., AND STENGER, F. A method of bisections for solving n nonlinear equations. *ACM Trans. Math. Softw.* 10, 4 (Dec. 1984), 367–377.
6. HANSEN, E. R. Interval forms of Newton's method. *Computing 20* (1978), 153–163.
7. HANSEN, E. R. A globally convergent interval method for computing and bounding real roots. *BIT 18*, 4 (1978), 415–424.
8. HANSEN, E. R., AND GREENBERG, R. I. An interval Newton method. *Appl. Math. Comput. 12* (1983), 89–98.
9. HANSEN, E. R., AND SENGUPTA, S. Bounding solutions of systems of equations using interval arithmetic. *BIT 21* (1981), 203–211.
10. KEARFOTT, R. B. On a general technique for finding directions proceeding from bifurcation points. In *Numerical Methods for Bifurcation Problems*. T. Küpper, H. D. Mittelmann, and H. Weber, Eds, International Series of Numerical Mathematics 70, Birkhäuser, Boston, 1984, 210–218.
11. KEARFOTT, R. B. Abstract generalized bisection and a cost bound. *Math. Comput. 49*, 179 (July 1987), 187–202.
12. MEINTJES, K., AND MORGAN, A. P. A methodology for solving chemical equilibrium problems. Rep. GMR-4971, General Motors Research Labs., Warren, Mich., 1985. To be published in *Appl. Math. Comput.*
13. MOORE, R. E. A test for existence of solutions to nonlinear systems. *SIAM J. Numer. Anal. 14*, 4 (Sept. 1977), 611–615.
14. MOORE, R. E. *Methods and Applications of Interval Analysis*. SIAM, Philadelphia, Pa., 1979.
15. MOORE, R. E., AND JONES, S. T. Safe starting regions for iterative methods. *SIAM J. Numer. Anal. 14*, 6 (Dec. 1977), 1051–1065.
16. MOORE, R. E., AND QI, L. A successive interval test for nonlinear systems. *SIAM J. Numer. Anal. 19*, 4 (Aug. 1982), 845–850.
17. MORÉ, J. J., GARBOW, B. S., AND HILLSTROM, K. E. User Guide for MINPACK-1. Rep. ANL-80-74, Argonne National Labs., Argonne, Ill., 1980.
18. MORÉ, J. J., GARBOW, B. S., AND HILLSTROM, K. E. Testing unconstrained optimization software. *ACM Trans. Math. Softw. 7*, 1 (Mar. 1981), 17–41.
19. MORGAN, A. P. A method for computing all solutions to systems of polynomial equations. *ACM Trans. Math. Softw. 9*, 1 (Mar. 1983) 1–17.
20. MORGAN, A. P. *Solving Polynomial Systems Using Continuation for Engineering and Scientific Problems*, Prentice-Hall, Englewood Cliffs, N.J., 1987.
21. MORGAN, A., AND SHAPIRO, V. Box-bisection for solving second-degree systems and the problem of clustering. *ACM Trans. Math. Softw. 13*, 2 (June 1987), 152–167.
22. NEUMAIER, A. Interval iteration for zeros of systems of equations, *BIT 25*, 1 (1985), 256–273.
23. ORTEGA, J. M., AND RHEINBOLDT, W. C. *Iterative Solution of Nonlinear Equations in Several Variables*. Academic Press, New York, 1970.
24. TSAI, L. W., AND MORGAN, A. P. Solving the kinematics of the most general six- and five-degree-of-freedom manipulators by continuation methods. Rep. GMR-4631, General Motors Research Labs., Warren, Mich., 1984. (To be published in the *ASME J. Mechanisms, Transmissions, and Automation in Design.*
25. YOHE, J. M. Software for interval arithmetic: a reasonably portable package. *ACM Trans. Math. Softw. 5*, 1 (Mar. 1979), 50–63.

Received February 1987; accepted May 1987