Test Results for an Interval Branch and Bound Algorithm for Equality-Constrained Optimization*

R. BAKER KEARFOTT rbk@usl.edu
Department of Mathematics
University of Southwestern Louisiana
U.S.L. Box 4-1010, Lafayette, Louisiana 70504-1010

Editor:

Abstract. Various techniques have been proposed for incorporating constraints in interval branch and bound algorithms for global optimization. However, few reports of practical experience with these techniques have appeared to date. Such experimental results appear here. The underlying implementation includes use of an approximate optimizer combined with a careful tesselation process and rigorous verification of feasibility. The experiments include comparison of methods of handling bound constraints and comparison of two methods for normalizing Lagrange multipliers. Selected test problems from the Floudas / Pardalos monograph are used, as well as selected unconstrained test problems appearing in reports of interval branch and bound methods for unconstrained global optimization.

Keywords: constrained global optimization, verified computations, interval computations, bound constraints, experimental results

1. Introduction

We consider the constrained global optimization problem

$$\begin{array}{rrcl} \text{minimize} & \phi(X) & & \\ \text{subject to} & c_i(X) & = & 0, \quad i=1,\ldots,m \\ & a_{i_j} \le & x_{i_j} & \le b_{i_j}, \quad j=1,\ldots,q, \end{array} \tag{1}$$

where $X=(x_1,\ldots,x_n)^T$, and where one of each pair a_{i_j} and b_{i_j} may be infinite. A general constrained optimization problem, including inequality constraints $g(X)\le 0$ is put into this form by introducing slack variables s, replacing by $s+g(X)=0$, and appending the bound constraint $0\le s<\infty$. Here, *solving* problem 1 will mean finding *all* minimizers and *verifying* bounds on the local minimum.

In contrast to the actual bound constraints in problem 1, we also have a *search region*

$$\mathbf{X}_0 = \left[[\underline{x}_{1,0},\overline{x}_{1,0}],\ldots[\underline{x}_{n,0},\overline{x}_{n,0}]\right].$$

* This work was supported in part by National Science Foundation grant CCR-9203730.

Below, coordinate bounds $\underline{x}_{i,0}$ or $\overline{x}_{i,0}$ of the overall search region corresponding to finite bound constraints a_i or b_i equal the bound constraint; however, bound constraints and limits of the search region are not to be confused.

The search region is also termed an "interval vector" or a "box."

Interval branch and bound methods for global optimization began with work of R. L. Moore and E. R. Hansen in the 1960's. Results such as [28] have shown the promise of such algorithms. The monographs [24] and [8] give complete introductions and numerous suggestions for techniques to incorporate. Recently, a number of researchers have developed computer codes. These codes are reviewed in [19].

Various suggestions for handling both inequality and equality constraints, as well as citations to the original literature such as [9], appear in [24] and [8]. However, except for perhaps [29], few reports of experiences with interval branch and bound methods for constrained optimization have appeared.

In [19], the main constraint-handling techniques of [24] and [8], etc., as well as certain relevant techniques in common with unconstrained optimization codes, were put into perspective, and several new techniques were proposed. Salient among these was transformation of inequality constraints into a combination of equality constraints and bound constraints, as in [3], combined with a process of handling bound constraints with reduced gradients and the "peeling" process that first appeared in [17].

It has recently become apparent that branch and bound methods for nonlinear equations and for nonlinear optimization benefit from use of a floating-point code to obtain approximate optimizers. This has been explored for nonlinear equations, first in the context of handling singular roots in [13], then more generally and thoroughly in [18]. There, a tesselation process was given for explicitly verifying uniqueness and removing regions around roots that had been quickly found by a local, floating point algorithm. The analogous technique for unconstrained optimization, using a floating-point local optimizer, has perhaps been most studied by Caprani, Godthaab and Madsen [1], although it was also employed in [11] and for bound-constrained optimization in [17]; see [19].

An important use of a local approximate optimum in an interval branch and bound code is to obtain a rigorous upper bound on global minima. In particular, if $\tilde{X}$ is an approximate optimum to the unconstrained problem: *minimize* $\phi(X)$, and $[\underline{\phi}, \overline{\phi}]$ is obtained from evaluating $\phi(\tilde{X})$ using outwardly rounded interval arithmetic, then $\overline{\phi}$ is a rigorous upper bound on the global minimum; $\overline{\phi}$ can be used to eliminate subregions (usually sub-boxes, i.e. interval vectors) of $\mathbf{X}_0$ from further search.

Computation of a rigorous upper bound is less straightforward for equality-constrained problems, since an interval evaluation of ϕ at $\tilde{X}$ can be used only provided it has been rigorously verified that $\tilde{X}$ is feasible. This was noted in [8], §12.3–12.5. In [20], techniques of [8] and other proposed techniques for verifying feasibility were examined computationally on a number of test problems. Also in [20], it was observed that, typically, the number of coordinates not corresponding to active bound constraints was less than the number of equality constraints; since verification that a point satisfies all equality constraints (with an interval Newton method) requires a square system of equations (over a region with non-empty interior), verification cannot proceed directly. In [20], several perturbation techniques to address this difficulty were proposed and tested, leading to a workable algorithm for obtaining rigorous upper bounds $\overline{\phi}$.

Interval Newton methods are important in branch and bound algorithms for constrained optimization. An interval Newton method is an operator $\mathbf{N}(\hat{\mathbf{X}}, F)$, operating on a box $\hat{\mathbf{X}} \in \mathbb{R}^p$, for some p, and a function $F : \mathbb{R}^p \rightarrow \mathbb{R}^p$ defined over $\hat{\mathbf{X}}$, such that $\mathbf{N}(\hat{\mathbf{X}}, F)$ is an interval vector containing the solution set to

$$\mathbf{A}(X - \check{X}) = -F(\check{X}) \tag{2}$$

for some[1] $\check{X} \in \hat{\mathbf{X}}$, and where $\mathbf{A}$ is either a Lipschitz matrix [23], p. 174 or a slope matrix [23], p. 202. (The set of Lipschitz matrices is contained in the set of slope matrices.) For example, an interval extension to the Jacobi matrix is a Lipschitz matrix, while the preconditioned interval Gauss–Seidel method, interval Gaussian elimination, or the Krawczyk method can bound the solution set to Equation (2). Interval Newton methods have the following properties:

- All roots of F in $\hat{\mathbf{X}}$ are also in $\mathbf{N}(\hat{\mathbf{X}}, F)$.
- If $\hat{\mathbf{X}} \cap \mathbf{N}(\hat{\mathbf{X}}, F) = \emptyset$, then there are no solutions of $F(X) = 0$ in $\hat{\mathbf{X}}$.
- Depending on how $\mathbf{A}$ is computed, how the solution set to Equation (2) is bounded, and subject to certain non-singularity conditions and an initial $\hat{\mathbf{X}}$ with sufficiently small widths, iteration of $\hat{\mathbf{X}} \leftarrow \mathbf{N}(\hat{\mathbf{X}}, F)$ results in quadratic convergence of the widths of the coordinate intervals of $\hat{\mathbf{X}}$ to zero.
- Depending on whether $\mathbf{A}$ Lipschitz matrix or just a slope matrix[2], $\mathbf{N}(\hat{\mathbf{X}}, F)$ is contained in the interior of $\hat{\mathbf{X}}$, we may conclude either (1) $F(X) = 0$ has a unique solution within $\hat{\mathbf{X}}$, or (2) $F(X) = 0$ has at least one solution within $\hat{\mathbf{X}}$.

Proofs of these facts are known, and appear in specific form e.g. in [23].
For example, an interval Newton method is used in [20] to verify that there is at least one feasible point $\hat{X} \in \hat{\mathbf{X}}$. In [9], interval Newton methods applied to the Fritz–John equations are discussed; the Fritz–John equations can be used in iterations $\hat{\mathbf{X}} \leftarrow \mathbf{N}(\hat{\mathbf{X}}, F)$ to reduce the size of subregions of $\mathbf{X}_0$ without bisection or other tesselation. See [19] for a perspective on interval Newton methods in constrained optimization, and see [24] or [8] for introductions.

We have developed a constrained global optimization code employing the feasibility verification process described in [20], the boundary tesselation process of [17] and [19] and other techniques of [19], [8] and [24], and using the Fortran 90 prototyping environment described in [22]. The purpose of this paper is to present test results for this code. Various alternatives are compared, and the code is compared to similar codes for unconstrained or bound-constrained optimization. The overall algorithm appears in §3, while alternative techniques and issues appear in §4. Our test set appears in §5.1, and the actual results appear in §6.

It is noted here that *exact* feasibility of equality constraints is verified with the projection and perturbation process explained in [20]. An alternative is to verify that the equality constraints are *approximately* satisfied, thus solving a perturbed problem. Such a procedure is advocated in [29] and by Ratschek and Rokne in [10], pp. 803–804. The approach in [20], and underlying the computations in this paper, is similar to that on [10], pp. 805–806 and [8], §12.3–§12.4: boxes are found within which it can be verified that there exists at least one feasible point. The existence and uniqueness verification process is explained in [20]; an overall branch and bound algorithm is studied in this paper.

2. Notation and Background

Throughout, boldface will be used to denote intervals, lower case to denote scalar quantities, and upper case to denote vectors and matrices. Underscores will denote lower bounds of intervals and overscores will denote upper bounds of intervals. For components of vectors, corresponding lower case letters will be used. For example, we may have:

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

where $\mathbf{x}_i = [\underline{x}_i, \overline{x}_i]$.

Calligraphic symbols such as $\mathcal{L}$, $\mathcal{U}$ and $\mathcal{C}$ will denote lists of boxes.

The notation $\check{x}$ will denote a representative point[3] of the interval $\mathbf{x}$, while $\check{X}$ will denote a corresponding representative vector in the interval vector or "box" $\mathbf{X}$. The 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 $w(\mathbf{x}) = \overline{x} - \underline{x}$.

The interval hull of two interval vectors $\mathbf{X}$ and $\tilde{\mathbf{X}}$, that is, the smallest interval vector that contains both $\mathbf{X}$ and $\tilde{\mathbf{X}}$, will be denoted by $\mathbf{X}\underline{\cup}\tilde{\mathbf{X}}$. If $\mathcal{L}$ is a list of boxes and $\mathbf{X}$ is a box, then $\mathcal{L} \setminus \mathbf{X}$ will denote a list the union of whose elements is the complement of $\mathbf{X}$ in the union of elements of $\mathcal{L}$. Algorithms 7 and 8, and Figure 2 in [18] describe how to create $\mathcal{L} \setminus \mathbf{X}$.

The symbols $\phi(\mathbf{X}) = [\underline{\phi}(\mathbf{X}), \overline{\phi}(\mathbf{X})]$ will denote an interval extension of ϕ over $\mathbf{X}$. Consistent with the above, $C(X) = (c_1(X), \ldots c_m(X))^T = 0$, $C: \mathbb{R}^n \to \mathbb{R}^m$, will denote the set of equality constraints, $\mathbf{C}(\mathbf{X})$ will denote the set of interval residuals of the equality constraints over $\mathbf{X}$, and $\nabla\mathbf{C}$ will denote a componentwise interval extension of the Jacobi matrix of the constraints C.

Brackets $[\cdot]$ will be used to delimit intervals, matrices and vectors. Occasionally, parentheses $(\cdot)$ will denote points in $\mathbb{R}^p$.

This paper contains minimal background, since the reader may consult the recent review [19], containing many of the underlying ideas, or [10], pp. 751–829. The somewhat earlier books [8] and [24] contain thorough introduction and explanation of interval branch and bound algorithms in global optimization.

3. The Overall Algorithm

The code follows this general pattern.

Algorithm 1 (Overall branch and bound pattern)

INPUT: (1) the initial box $\mathbf{X}_0$, (2) an internal symbolic representation for ϕ, (3) a domain tolerance ϵ_{d}, and (4) the maximum number of boxes to be processed M

OUTPUT: (1) list $\mathcal{C}$ of boxes containing approximate optimizers, each of which has been verified to contain at least one feasible point and a list $\mathcal{U}$ of boxes with approximate optimizers and approximate feasible points, for which feasibility has not been verified, such that all global optima are contained in the union of the boxes in $\mathcal{C}$ and $\mathcal{U}$ and such that the largest relative coordinate width in any box in $\mathcal{C}$ or $\mathcal{U}$ is ϵ_{d}; (2) a rigorous upper bound $\overline{\phi}$ on the global optimum, as well as a rigorous lower bound $\underline{\phi}$

1. *Attempt to compute an approximate optimum using a floating-point optimizer.*

   IF *an approximate optimum $\hat{X}$ could be computed and a box $\hat{\mathbf{X}}$ could be constructed around it in which a feasible point is proven to exist,* THEN

   (A) $\overline{\phi} \leftarrow \overline{\phi}(\hat{\mathbf{X}})$

   (B) Construct a box $\mathbf{X}_a$ about $\hat{X}$ according to Algorithm 2.

   (C) Insert $\mathbf{X}_a$ in $\mathcal{U}$.

   END IF

2. ("peeling" the boundary; this step varies according to configuration, as explained in §4)

   IF *the algorithm works with reduced gradients and subspaces*

   THEN

   Generate boundary boxes $\hat{\mathbf{X}}$ of $\mathbf{X}_0$ corresponding to bound constraints, placing those for which $\underline{\phi}(\hat{\mathbf{X}}) < \overline{\phi}$ in the list $\mathcal{L}$ in order of increasing $\underline{\phi}(\hat{\mathbf{X}})$; also place $\mathbf{X}_0$, representing the interior, in $\mathcal{L}$.

   ELSE

   Place $\mathbf{X}_0$ into $\mathcal{L}$.

   END IF

3. IF $\overline{\phi}$ *was found in step 1 or step 2* THEN *form the set-complement of $\mathbf{X}_a$ in $\mathcal{L}$.*

4. Overall loop DO *for* NBOX $= 1$ *to* M

   (A) Remove the first box from $\mathcal{L}$ and place it in $\mathbf{X}$.

   (B) (Check scaled diameter) IF $\max_{1 \le i \le n} w(\mathbf{x}_i)/\max\{|\mathbf{x}_i|, 1\} \le \epsilon_{\mathrm{d}}$ THEN

i. *Execute Algorithm 3.*

ii. CYCLE Overall loop

(C) *Perform an interval Newton method to eliminate* $\mathbf{X}$ *or reduce its size.*

(D) IF *step 4C proved that there could be no optimum in* $\mathbf{X}$ THEN CYCLE Overall loop.

(E) (Check scaled diameter) IF $\max_{1\le i\le n} w(\mathbf{x}_i)/\max\{|\mathbf{x}_i|,1\} \le \epsilon_{\mathrm{d}}$ THEN

i. *Execute Algorithm 3.*

ii. CYCLE Overall loop

(F) (Bisection)

i. *Compute* $i = \arg\max_{1\le j\le n} |\frac{\partial\phi}{\partial x_j}(\mathbf{X})| w(\mathbf{x}_j)$.

ii. *Form two new boxes* $\mathbf{X}^{(1)}$ *and* $\mathbf{X}^{(2)}$ *from* $\mathbf{X}$ *by setting*

$$\begin{aligned}
\underline{x}_i^{(1)} &\leftarrow (\underline{x}_i + \overline{x}_i)/2 \\
\overline{x}_i^{(1)} &\leftarrow \overline{x}_i \\
\underline{x}_i^{(2)} &\leftarrow \underline{x}_i \\
\overline{x}_i^{(2)} &\leftarrow (\underline{x}_i + \overline{x}_i)/2,
\end{aligned}$$

and $\mathbf{x}_j^{(1)} = \mathbf{x}_j$, $\mathbf{x}_j^{(2)} = \mathbf{x}_j$ *for* $1 \le j \le n$, $j \ne i$.

iii. *Place the two new boxes in* $\mathcal{L}$ *in order of increasing* $\underline{\phi}$, *provided* $\underline{\phi} \le \overline{\phi}$.

END DO Overall loop

The approximate optimizer in step 1 and elsewhere is the routine DAUGLG from the Lancelot package [2].

Step 2 of Algorithm 1 is explained in detail as Algorithm 4 of [19]. For example, suppose $\mathbf{X}_0 = [[-1,1],[-1,1]]$ and $x_1 = -1$ and $x_2 = 1$ were bound constraints. Then, upon exit from step 2, $\mathcal{L}$ will contain the boxes $\hat{\mathbf{X}}_1 = [[-1,1],[-1,1]]$, $\hat{\mathbf{X}}_2 = [[-1,-1],[-1,1]]$, $\hat{\mathbf{X}}_3 = [[-1,1],[1,1]]$ and $\hat{\mathbf{X}}_4 = [[-1,-1],[1,1]]$, not necessarily in that order, assuming that none of the boxes was eliminated due to $\underline{\phi}(\hat{\mathbf{X}}_i) > \overline{\phi}$. During step 2, an attempt is made to compute a value of $\overline{\phi}$, provided it has not already been obtained in step 1.

The complementation process in step 3 is Algorithm 8 of [18].

The interval Newton method in step 4C consists of interval Gaussian elimination, interval Gauss–Seidel iteration, or a combination, as explained in §4. Also, as in §4, the system of equations could be in the whole space or in a subspace, and could vary according to a certain normalization equation. Finally, checks for feasibility and checks on the function value are made by computing interval values $C(\mathbf{X})$ and $\phi(\mathbf{X})$ during iteration of the actual interval Newton method.

The bisection variable selection in step 4(F) i is the *maximal smear* introduced for nonlinear systems[4] in [15], and also recommended as Rule C in [4] and in [25]. Essentially, the direction i is, approximately, the direction over which there is the most change in the function ϕ over the box $\mathbf{X}$. Results of careful experiments in [4] indicate that this direction usually, but not always, results in a smaller number of total boxes, in a simplified branch and bound algorithm that does not involve interval Newton methods. The technique in step 4(F) i is used exclusively in the experiments in §6 below. However, alternate techniques may result in success in some situations where failure was reported here.

Algorithm 2 (Construct a well-sized box about approximate optima.)

INPUT: An approximate optimum X_a, an expansion factor F, and the domain tolerance ϵ_{d}

OUTPUT: A box $\mathbf{X}_a$ containing X_a

DO *for* $i = 1$ *to* n

1. $\sigma \leftarrow \mathrm{F}\max\{|X_{a,i}|, 1\}\sqrt{\epsilon_{\mathrm{d}}}$
2. $\mathbf{x}_{a,i} \leftarrow [X_{a,i} - \sigma, X_{a,i} + \sigma] \cap \mathbf{x}_{0,i}$

END DO

In the experiments reported in §6, F = 10. This factor makes it likely that $\mathbf{X}_a$ contains appropriate portions of the boundary of $\mathbf{X}_0$ when X_a corresponds to a solution, with active bound constraints, that has been perturbed off the bounds with techniques in [20]. The factor $\sqrt{\epsilon_{\mathrm{d}}}$ assures that Algorithm 1 does not generate and store large clusters of boxes centered at minimizers: Boxes $\mathbf{X}$ with largest relative side width ϵ_{d} are easily rejected by $\underline{\phi}(\mathbf{X}) > \overline{\phi}$ when they lie at least $\sqrt{\epsilon_{\mathrm{d}}}$ away from the point at which $\overline{\phi}$ is attained.

Algorithm 3, called in steps 4(B) i and 4(E) i of Algorithm 1, involves constructing sufficiently large boxes around tiny regions (approximate optimizers) that could not be eliminated in step 4 of Algorithm 1. Once constructed, such small boxes are removed from the search region by taking the set complement of the box in each of the lists $\mathcal{U}$, $\mathcal{C}$ and $\mathcal{L}$, identifying each list $\mathcal{U}$, $\mathcal{C}$ and $\mathcal{L}$ with the union of the boxes in it. Such complementation operations appear in steps 4(C) ii, 4(C) iv, 4(C) iA, 4(C) iC, 4(C) i, 4(C) iii, and 4(C) v. The actual process can be done relatively efficiently, and is described in detail in Algorithm 7, Algorithm 8 and Figure 2 of [18]; also see §2 here for an explanation of the notation $\mathcal{L} \leftarrow \mathcal{L} \setminus \mathbf{X}_a$.

Algorithm 3 (Handle small boxes with approximate optimizers)

INPUT: (from Algorithm 1): the small, approximately-optimizing box $\mathbf{X}$, $\overline{\phi}$ (if previously computed), $\mathcal{U}$, $\mathcal{C}$ and $\mathcal{L}$

OUTPUT: possibly altered $\mathcal{U}$, $\mathcal{C}$ and $\mathcal{L}$ and $\overline{\phi}$

1. *Set X_a to the center of $\mathbf{X}$.*
2. *Apply Algorithm 2 to X_a to obtain $\mathbf{X}_a$.*

3. *Starting with X_a as an initial guess, attempt to compute an approximate optimum.*

4. IF *an approximate optimum (within $\mathbf{X}_a$ or not) could be found in step 3, and feasibility verified within $\mathbf{X}_f$,* THEN

   (A) $\overline{\phi} \leftarrow \min\{\overline{\phi}, \overline{\phi}(\mathbf{X}_f)\}$.

   (B) Remove boxes $\mathbf{X}$ *for which* $\underline{\phi}(\mathbf{X}) > \overline{\phi}$ *from* $\mathcal{U}$, $\mathcal{C}$ *and* $\mathcal{L}$.

   (C) IF $\mathbf{X}_f \cap \mathbf{X}_a \neq \emptyset$ THEN (Insert the interval hull of $\mathbf{X}_a$ and $\mathbf{X}_f$.)

      i. $\mathbf{X}_a \leftarrow \mathbf{X}_a \underline{\cup} \mathbf{X}_f$.

      ii. $\mathcal{U} \leftarrow \mathcal{U} \setminus \mathbf{X}_a$.

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

      iv. $\mathcal{L} \leftarrow \mathcal{L} \setminus \mathbf{X}_a$ *and* $\mathcal{C} \leftarrow \mathcal{C} \setminus \mathbf{X}_a$.

   ELSE (Insert both $\mathbf{X}_a$ and $\mathbf{X}_f$.)

      i. IF $\underline{\phi}(\mathbf{X}_a) \leq \overline{\phi}$ THEN

         A. $\mathcal{C} \leftarrow \mathcal{C} \setminus \mathbf{X}_a$.

         B. Insert $\mathbf{X}_a$ *into* $\mathcal{C}$.

         C. $\mathcal{L} \leftarrow \mathcal{L} \setminus \mathbf{X}_a$ *and* $\mathcal{U} \leftarrow \mathcal{U} \setminus \mathbf{X}_a$.

         ELSE

            $\mathcal{L} \leftarrow \mathcal{L} \setminus \mathbf{X}_a$, $\mathcal{U} \leftarrow \mathcal{U} \setminus \mathbf{X}_a$, *and* $\mathcal{C} \leftarrow$ *completedlist* $\setminus \mathbf{X}_a$.

         END IF

      ii. Apply Algorithm 2 to the midpoint of $\mathbf{X}_f$ *to obtain a new* $\mathbf{X}_a$

      iii. $\mathcal{U} \leftarrow \mathcal{U} \setminus \mathbf{X}_a$.

      iv. Insert $\mathbf{X}_a$ *into* $\mathcal{U}$.

      v. $\mathcal{L} \leftarrow \mathcal{L} \setminus \mathbf{X}_a$ *and* $\mathcal{C} \leftarrow \mathcal{C} \setminus \mathbf{X}_a$.

   END IF

   ELSE (an approximate optimum could not be found at all)

   (A) $\mathcal{C} \leftarrow \mathcal{C} \setminus \mathbf{X}_a$.

   (B) Insert $\mathbf{X}_a$ *into* $\mathcal{C}$.

   (C) $\mathcal{L} \leftarrow \mathcal{L} \setminus \mathbf{X}_a$ *and* $\mathcal{U} \leftarrow \mathcal{U} \setminus \mathbf{X}_a$.

   END IF

The details of Algorithm 3 are important to the efficiency of the overall code. Fortran 90 source corresponding to the experiments in this paper is available from the author.

4. Issues

There is significant documented computational experience in interval branch and bound techniques for unconstrained optimization. Thus, many of the remaining questions deal with how constraints are handled. In [9] (and reviewed in [8]), Hansen and Walster presented a general Fritz–John system for interval Newton iteration when there are both equality and inequality constraints. The system is of the form $F = 0$, where

$$F(W) = \begin{bmatrix} u_0 \nabla \phi(X) + \sum_{i=1}^{m} v_i \nabla c_i(X) \\ c_1(X) \\ \vdots \\ c_m(X) \\ u_1 g_1(x) \\ \vdots \\ u_q g_q(x) \\ N(U,V) \end{bmatrix} = 0, \qquad (3)$$

The variables are $X = (x_1, \ldots, x_n)$, $V = (v_1, \ldots, v_m)$, and $U = (u_0, u_1, \ldots, u_q)$, for a total of $n+m+q+1$ variables, written $W = (X, U, V)$. The v_i represent Lagrange multipliers for the equality constraints $c_i = 0$, while the u_i represent Lagrange multipliers for the objective function (u_0) and for the q inequality constraints. The last equation $N(U, V) = 0$ is a normalization condition.

Bound constraints can be included as inequality constraints; e.g. $x_i \leq b$ can be written as $g(X) = b - x_i \leq 0$, or $x_i \geq a$ can be written as $a - x_i \leq 0$. However, this increases the size of the system 3. Because of this, we have advocated in [19] and [20] not including *any* inequality constraints in Equation (3), but converting all inequality constraints to bound constraints with slack variables, then applying Equation (3) in subspaces, using reduced gradients corresponding to active constraints. As mentioned in [20], this method not only reduces the dimension, but also avoids singularities: The system 3, with bound constraints included, is singular, in theory and practice, at points where moderate numbers of bound constraints are active.

On the other hand, the algorithm variant that works in the reduced space involves executing step 2 of Algorithm 1. As explained in [19], §2.3, it is possible that 3^p boxes are stored, where p is the number of bound constraints[5]. Furthermore, even though the version of Equation (3) can be expected to have singular Jacobi matrix at solutions, use of preconditioners of the forms in [14] and [16] allows the interval Gauss–Seidel method to reduce the widths of some of the coordinates of boxes $\mathbf{X}$ even if the Jacobi matrix of F has singularities in $\mathbf{X}$.

We have tested the following variants of Algorithm 1:

Variant LF in which a large version of the Fritz–John equations is used (including bound constraints, but in which inequality constraints are rewritten in terms of bound constraints), and the boundaries are not "peeled."

Variant SF in which bound constraints are not included in the Fritz–John equations, but boxes corresponding to active bound constraints are generated in step 2 of Algorithm 1.

In addition to how the bound constraints are included, the form of the normalization equation $N(U, V) = 0$ is at issue. Hansen and Walster [9] suggest a linear interval condition

$$N_1(U, V) = \left\{ \sum_{i=0}^{q} u_i + \sum_{i=1}^{m} v_i [1, 1+\epsilon] \right\} - 1 = 0, \tag{4}$$

where ϵ is on the order of the computational precision, or a quadratic condition

$$N_2(U, V) = \left\{ \sum_{i=0}^{q} u_i + \sum_{i=1}^{m} v_i^2 \right\} - 1 = 0. \tag{5}$$

If N_1 is used, then initial bounds on U and V are not needed, but can, in principle, be computed with preconditioned interval Gaussian elimination. In practice, we have found this option of very limited use in conjunction with Variant LF, since the interval Jacobi matrix is then often singular. If variant N_2 is used, then $u_i \in [0, 1]$, $0 \le i \le q$ and $v_j \in [-1, 1]$, $1 \le j \le m$, but interval values of u_i and v_j must explicitly enter into the entries of the derivative matrix for the system 3.

Thus, we have tested four possible algorithms: choosing Variant LF or Variant SF and choosing normalization N_1 or normalization N_2. In the corresponding interval Newton methods, various combinations of preconditioners are used, and interval Gaussian elimination is followed by interval Gauss–Seidel iteration for N_1. At various points, interval slope matrices computed with the technique of [27], combined with Hansen's technique of [7] and [8], §6.4 (combined as explained in [19], §3.3). We do not take advantage of splittings as in [26]. For brevity, we do not present all details of these interval Newton methods, but the Fortran 90 source code is available from the author.

A final set of issues impacts both constrained and unconstrained optimization. These issues deal with the size of the box constructed by Algorithm 2, when computation of an approximate optimizer is attempted, or when Algorithm 2 is applied in Algorithm 1 and Algorithm 3. Such issues surface in the works [1], [11], [13], [17] and possibly elsewhere. Our experience indicates that these details are crucial for the practicality of interval branch and bound for some problems. We believe the expansion size in Algorithm 2 to be good from the point of view of computational effort. Note, however, that we expand *each* approximate-optimum-containing box by the same amount, without attempting to determine a box size in which uniqueness of critical points can be verified. See the discussion in [18].

5. The Test Problems and Testing Environment

5.1. The Test Problems

We have chosen the standard problems from [5], as well as more challenging problems from [6], and salient problems used to test other recent interval branch and bound codes for global optimization. We have included unconstrained problems among these, for comparison of our general techniques (including those in Algorithm 2 and Algorithm 3) with other current optimization codes. Our test set is taken from the following four sources.

- The three problems from [5], pp. 12–14 with parameter values as in [12].
- The selected constrained optimization problems from [6] that appear in the feasibility verification study [20].
- The constrained optimization problems used to test the methods in [29].
- A selection of the unconstrained optimization problems from [12].
- An unconstrained test problem from [4]. (See Formula 6 and Table 1 below.)

We identify the test problems from [5] as follows.

shekel is the Shekel function with $n = 4$ and five data points[6], also Example 36 in [12]. We interpret the search region limits as bound constraints, so $m = 0$ and there are eight bound constraints.

hartmn is the Hartman function[7] with $n = 3$ and four data points, also Example 37 in [12]. Interpreting the search region limits as bound constraints, there are six bound constraints.

branin is Branin's function (RCOS), also Example 34 in [12]. We interpret the search region limits as bound constraints, so $n = 2$, $m = 0$, and there are four bound constraints.

The test problems from [6] were chosen to give a variety of realistic problems. Since they are described in [6] and [20], we do not present them here. We identify them with the same labels as in [20]: **fpnlp3**, **fpqp3**, **fpnlp6**, **fppb1**, and **fphe1**. Similarly, the problems from [29] appear in [20], and are labeled **gould**, **bracken**, and **wolfe3**.

The remaining test problems from [12] are as follows.

levya is Example 9 from [12], Levy No. 11. In it, $n = 8$, $m = 0$, and we interpret the constraints in [12] as bound constraints, for a total of 16 bound constraints.

levyb is Example 15 from [12], Levy No. 18. In it, $n = 7$, $m = 0$ and there are 14 bound constraints.

Table 1. The f's for problem **csendes**

i	1	2	3	4	5	6
$f_{*,1}$	5.0	3.0	2.0	1.5	1.2	1.1
$f_{*,2}$	-5.0	-2.0	-1.0	-0.5	-0.2	-0.1

schwfa is Example 16 from [12], Schwefel No. 1.2. In it, $n = 4$, $m = 0$ and there are 8 bound constraints.

kowlk is Example 22 from [12], the Kowalik Problem. In it, $n = 4$, $m = 0$ and there are 8 bound constraints.

griew is Example 31 from [12], the Griewank Function. In it, $n = 2$, $m = 0$ and there are 4 bound constraints.

The problem from [4], labeled **csendes**, is a pulmonary impedance model. Verification of the global optimum was first obtained with the codes described in [4]. The function is:

$$\phi(X) = \sum_{i=1}^{6} \left\{ \left[f_{i,1} - \left(x_1 + \frac{x_2}{\omega_i^3} \right) \right]^2 + \left[f_{i,2} - \left(\omega_i x_4 - \frac{x_5}{\omega_i^3} \right) \right]^2 \right\}, \tag{6}$$

where $\omega_i = \pi i/20$, $1 \le i \le 6$ and the f's are as in table 1. The initial box (as in [4]) is $\big([0,2],[0,1],[1.1,3.0],[0,1],[0,1]\big)$, and we interpret none of the bounds to be bound constraints. The single optimum is at approximately

$$(0.6063, 0.5568, 1.132, 0.7502, 0.6219).$$

If there are questions about details, the Fortran 90 source and data files are available from the author.

5.2. The Implementation Environment

The experiments were run on a Sparc 20 model 51 with a floating point accelerator, as with the experiments in [18]. The code was implemented in the system described in [22], with interval arithmetic of [21], modified to do true directed roundings on the particular hardware used. All times are given in terms of Standard Time Units (STU's), defined in [5], pp. 12–14. On the system used, an STU is approximately 0.056 CPU seconds. Also, there is an overhead factor of approximately 8 in floating point computations when functions are evaluated interpretively using the internal representation of [22], rather than being programmed as subroutines; the interval arithmetic using the internal representation is a factor of 17.6 times slower than floating point arithmetic programmed as subroutines; see [18]. There is an additional time penalty due to the fact that a list of all intermediate variables, including those for ϕ, $\nabla\phi$, C and ∇C is evaluated each time one of these quantities is needed, and not all quantities are needed at each point.

To concentrate on the issues at hand (global search and verification), we did not attempt to tune the approximate optimizer (Lancelot). However, to reduce the amount of work on the initial call to this optimizer, we provided the option of inputting a good initial guess. Initial guesses were used whenever it would reduce the amount of time spent in the initial call to the approximate optimizer. This did not affect the number of boxes processed or the execution times excluding the time for the approximate optimizer.

In all cases, the domain tolerance ϵ_{d} was set to 10^{-6}.

6. Experimental Results

Preliminary experiments seemed to indicate that the fastest overall method was variant SF of §4 combined with the quadratic normalization. However, when we attempted to run the entire problem set with that scheme, we experienced difficulties with fpqp3, fphe1, and fppb1. The problem fpqp3, a quadratic programming problem with many bound constraints, generated too large a list $\mathcal{L}$ to fit into electronic memory (32 megabytes), while problems fppb1 and fphe1 had boxes remaining in $\mathcal{L}$ after $M = 20000$ boxes had been processed. For this reason, we temporarily excluded these problems from the test set, but ran them separately to determine if the linear normalization or variant LF could solve them.

It immediately became apparent that, for most problems, variant SF was far more efficient. Most CPU time for variant LF seemed to be spent in the LP solver[8], and there were many failures to compute preconditioners. This is consistent with multiple singularities in the Fritz–John matrix. It may be possible to make our LP solver more robust[9]; however, we were unable at present to run the entire test set for variant LF.

Problem wolfe3 could not complete after considering 20000 boxes, with variant SF and linear normalization, so it is not included in the totals at the bottom. Results for the test problems, excluding fpqp3, fphe1 fppb1, and wolfe3, for variant SF and both quadratic and linear normalizations, appear in table 2. The columns of this table are as follows.

NBOX is the total number of boxes processed (in the loop of step 4 of Algorithm 1).

NBIS is the total number of bisections (in step 4F of Algorithm 1).

T_{tot} is the total execution time, in STU's, excluding the initial call to the approximate optimizer, for the algorithm.

T_{iN} is the total execution time spent in the interval Newton routine (including function, gradient and matrix evaluations, step 4C of Algorithm 1).

T_{peel} is the total execution time spent in the boundary separation process of [19], §2.3 (step 2 of Algorithm 1).

Table 2. Results for the algorithm with the "peeling" process

Prob.	normalization	NBOX	NBIS	T_{tot}	T_{iN}	T_{peel}	NV	NNV
shekel	quadratic	77	34	49.9	42.5	0.5	1	0
shekel	linear	77	34	47.9	41.5	0.4	1	0
hartmn	quadratic	100	46	60.5	51.2	0.7	1	0
hartmn	linear	100	46	59.4	50.1	0.5	1	0
branin	quadratic	15	8	3.8	2.9	0.2	3	0
branin	linear	15	8	4.1	3.4	0.2	3	0
fpnlp3	quadratic	63	1	151.9	140.6	5.7	1	0
fpnlp3	linear	7903	3921	13251.0	11862.0	5.9	1	0
fpnlp6	quadratic	1043	510	479.0	454.2	0.4	1	0
fpnlp6	linear	4849	2413	3181.1	3065.1	0.2	1	0
gould	quadratic	19	1	6.3	5.6	0.4	1	0
gould	linear	8753	4368	1411.5	1247.2	0.2	1	0
brackn	quadratic	4	0	0.9	0.7	0.0	1	0
brackn	linear	106	51	35.2	33.6	0.0	1	0
levya	quadratic	1	0	3.1	0.0	1.6	1	0
levya	linear	1	0	3.2	0.0	1.6	1	0
levyb	quadratic	1	0	2.5	0.0	1.3	1	0
levyb	linear	1	0	2.3	0.0	1.1	1	0
schwfa	quadratic	14	0	2.0	1.4	0.2	1	0
schwfa	linear	14	0	1.8	1.4	0.2	1	0
kowlk	quadratic	7856	3920	24067.8	18798.9	1.1	1	0
kowlk	linear	7856	3920	21981.3	17752.1	0.9	1	0
griew	quadratic	1	0	0.4	0.0	0.2	1	0
griew	linear	1	0	0.2	0.0	0.2	1	0
csendes	quadratic	137	63	390.7	351.7	1.1	1	0
csendes	linear	137	63	392.8	355.5	0.9	1	0
wolfe3	quadratic	13	7	44.5	11.7	31.8	1	0
Total:		9331	4583	25218.6	19849.7	13.3		
		29813	14824	40371.9	34411.8	12.2		
Ratio:		0.31	0.31	0.62	0.58	1.09		

Table 3. Comparisons with an algorithm by Jansson and Knüppel

Problem	J/K	Here	J/K	Here	STU rat.	NFJ rat.
	STU	STU	NFJ	NFJ		
shekel	2.00	49.91	7	41	0.04	0.17
hartmn	6.90	60.50	12	94	0.11	0.13
branin	2.30	3.77	18	25	0.61	0.72
levya	3.20	3.05	5	0	1.05	∞
levyb	5.90	2.51	11	0	2.35	∞
schwfa	317.20	1.97	9	13	160.62	0.69
griew	2.40	0.36	18	0	6.68	∞
Totals:	339.9	122.08	80	173	2.78	0.46

NV is the final number of boxes at which feasibility was verified, in the list $\mathcal{U}$.

NNV is the final number of boxes for which feasibility could not be verified, in the list $\mathcal{C}$.

It is immediately apparent from the table that the algorithm is very effective at isolating global minima within a single box and verifying that a feasible point exists in the box. We attribute that to the tolerance used in the local optimizer $(O(\epsilon_{\mathrm{d}})^2))$ and the size of the constructed box in Algorithm 2 $(O(\epsilon_{\mathrm{d}})^{\cdot}5)$.

It is also clear that the quadratic normalization is superior to the linear normalization, with this implementation. One would expect this to be especially so for variant LF, since the interval Fritz-John matrix cannot be expected to be regular, and interval Gaussian elimination, used to obtain initial estimates for the Lagrange multipliers, is then problematical.

The results for csendes compare very favorably with those of [4]. We believe this to be due to our use of an approximate optimizer, combined with Algorithm 2 and our choice of tolerances. Also, failure to complete kowlk is reported in [12], whereas our algorithm completes. This may be due to our allowing more execution time.

Comparisons for the other problems in [12] appear in table 3. There, NFJ represents the number of evaluations of the Fritz–John matrix (equal to a slope evaluation, with Hansen's modification, for the Hessian matrix for these unconstrained problems). This evaluation is the most expensive part of our code, since there are still various inefficiencies in our evaluation of slopes. The column labeled "STU rat." gives the ratio of standard time units for the results in [12] to the results here. The column "NFJ rat." gives a similar ratio for the number of second derivative matrix evaluations. Table 3 does not unequivocally favor one algorithm over the other. We attribute many of the lower STU values for the code from [12] partially to their very efficient implementation, relative to ours, and partially to algorithm differences. We attribute cases where our algorithm does better to the relation among our tolerances and to the box size in Algorithm 2, and possibly other items.

Table 4. The four algorithm variants on the Shekel function (shekel)

Algorithm	T_{tot}	T_{IN}	T_{la}	NBOX	N_{cl}	N_{pre}	N_{fail}	N_{sm}	N_{rej}
SF, quad.	49.9	42.5	0.9	77	8	164	0	8	0
SF, linear	47.9	41.5	0.0	77	8	165	0	0	0
LF, quad.	28157.1	28135.4	24316.8	233	8	50347	45659	7333	0
LF, linear	807.5	787.2	676.1	233	8	1420	194	0	0

Due to lengthy execution times, we were not able to complete runs for variant LF for the entire test set. However, table 4 gives results for shekel for all four possible algorithms. In this table, T_{tot}, T_{IN}, and NBOX are as in Table 2, while:

T_{la} is the total amount of time doing linear algebra, including computation of preconditioners for interval Gauss–Seidel steps.

N_{cl} is the number of boxes in the list after possibly "peeling" the boundary and taking the complement around the first approximate optimum (i.e. after step 3 of Algorithm 1).

N_{pre} is the total number of attempts to compute a preconditioner row for a Gauss–Seidel step.

N_{fail} is the number of times the LP solver failed to compute a preconditioner.

N_{sm} is the number of Gauss–Seidel steps that succeeded in making a coordinate width smaller.

N_{rej} is the number of times a Gauss–Seidel step succeeded in rejecting a box due to an empty intersection.

It is evident from table 4 that, overall, the Gauss–Seidel iteration is more effective on variant SF. Intuitively, this is not surprising, since the Fritz–John matrix cannot be regular in variant LF, in many cases[10] However, it is presently unknown why the LP solver failed in so many cases: A C-LP preconditioner should exist under fairly general conditions, but it may not be unique. Further investigation may be useful.

It is possible that variant LF could be made to work better than variant SF for problems, such as wolfe that have many bound constraints but few active ones at the optimizers. Also, alternate techniques or improvements are in order for problems such as fpqp3, fphe1 or fppb1. Table 5, though somewhat negative, represents the experiments we have done by the writing of this report. It can be taken as a challenge for further innovation.

Tables of full information, including additional performance measures and debugging information not appearing here, are available from the author.

Table 5. Failures on some difficult problems

Problem	SF, quadratic	SF, linear	LF, quadratic	LF, linear
fpqp3	Out of memory (32mb)	Out of memory (32mb)	untried	Did not finish in 11 CPU hours
fphe1	Out of memory (32mb)	untried	system shut-down while running (about 1 CPU hour)	untried
fppb1	5652 boxes left after 20000 boxes processed	untried	untried	untried

7. Summary and Possible Improvements

We have tested a general constrained global optimization algorithm implemented within the environment of [22]. A linear and a quadratic normalization of the Lagrange multipliers, as well as two methods of handling bound constraints, SF and LF, were compared. For most problems, it was found that the quadratic normalization and variant SF (generating boundary boxes and working in a reduced space, rather than including bound constraints in the Fritz–John equations) were faster.

Variant LF could possibly be made faster with a better LP solver for preconditioners. Also, A more efficient way of computing slope matrices would benefit the algorithm greatly. Finally, the system of [22] was meant for research and testing; significant efficiency (with respect to CPU time) could be gained by programming and compiling the various routines for evaluation of the function, constraints, gradients, and second derivative matrices separately. This may make solution of some of the more challenging problems practical.

A possible improvement would be to allow the interval Newton method to return two boxes ("splitting" as in [26]). This complicating factor was not included here, and the contraction preconditioners used generally do not produce linear interval systems that result in splits. However, possible splits were observed frequently with the test set used and algorithm variant SF, when the dimension of the reduced space was 1 (and hence no preconditioner was required).

An additional gradient test can be implemented to reject boxes within which there can be no critical points subject to the equality constraints. Appearing in [10], p. 18 and p. 21, this test can be restated in the present context as

$$0 \in \boldsymbol{\nabla}\boldsymbol{\phi}(\mathbf{X}) \circ \boldsymbol{\nabla}\mathbf{C}(\mathbf{X}),$$

where $\boldsymbol{\nabla}\boldsymbol{\phi}(\mathbf{X})$ is an interval extension to $\nabla\phi$, viewed as a row vector, $\boldsymbol{\nabla}\mathbf{C}(\mathbf{X})$ is a similar interval extension to ∇C, viewed as an $n \times m$ matrix, "$\circ$" is matrix multiplication, and "0" is interpreted as a row vector in $\mathbb{R}^m$.

Finally, alternate strategies to the maximal smear scheme of step 4(F) i of Algorithm 1 may result in success on some of the problems for which failure was reported here. This will be the subject of a separate investigation.

Acknowledgments

I wish to acknowledge professors Floudas and Pardalos for their encouragement. I also wish to acknowledge Shiying Ning for programming and testing several of the test functions. Finally, I wish to thank the referee for a number of suggestions that make the paper clearer, as well as pointing out the test in [10], p. 18.

Notes

1. $\check{X}$ is often chosen to be the midpoint of $\hat{\mathbf{X}}$.
2. although see [27], §2.3 and [18], §4
3. often, the midpoint
4. but applied to the analogy of $\nabla\phi$, rather than ϕ itself, as here
5. although good evaluations of $\underline{\phi}(\mathbf{X})$ can eliminate most such boxes before they are formed, in many cases
6. This is the number m in [5], p. 12.
7. Note the error in [5], p. 13: the upper limit of the inner sum there should be n, not m.
8. A dense LP solver, a variant of the simplex method written specially by Manuel Novoa to compute LP preconditioners efficiently
9. and it is unclear how the inverse midpoint preconditioner will work for large-width interval matrices that are not regular
10. The successes for the quadratic normalization and variant SF were probably due to reduction of the Lagrange multiplier bounds $\mathbf{V}$.

References

1. Caprani, O., Godthaab, B., and Madsen, K., *Use of a Real-Valued Local Minimum in Parallel Interval Global Optimization*, Interval Computations **1993** (2), pp. 71–82, 1993.
2. Conn, A. R., Gould, N. and Toint, Ph.L., *LANCELOT: A Fortran Package for Large-Scale Nonlinear Optimization*, Springer-Verlag, New York, 1992.
3. Conn, A. R., Gould, N., and Toint, Ph. L., *A Note on Exploiting Structure when using Slack Variables*, Math. Prog. **67** (1), pp. 89–99, 1994.
4. Csendes, T. and Ratz, D., *Subdivision Direction Selection in Interval Methods for Global Optimization*, preprint, 1994.
5. Dixon, L. C. W. and Szegö, G. P., *The Global Optimization Problem: An Introduction*, in Towards Global Optimization 2, ed. Dixon, L. C. W. and Szegö, G. P., pp. 1–15, North-Holland, Amsterdam, Netherlands, 1978.
6. Floudas, C. A. and Pardalos, P. M., *A Collection of Test Problems for Constrained Global Optimization Algorithms*, Springer-Verlag, New York, 1990.
7. Hansen, Eldon, *Interval Forms of Newton's Method*, Computing **20**, pp. 153–163, 1978.
8. Hansen, E. R., *Global Optimization Using Interval Analysis*, Marcel Dekker, Inc., New York, 1992.

9. Hansen, E. R. and Walster, G. W., *Bounds for Lagrange Multipliers and Optimal Points*, Comput. Math. Appl. **25** (10), pp. 59, 1993.
10. Horst, R. and Pardalos, M., eds., *Handbook of Global Optimization*, Kluwer, Dordrecht, Netherlands, 1995.
11. Jansson, C. and Knüppel, O., *A Global Minimization Method: The Multi-Dimensional Case*, preprint, 1992.
12. Jansson, C. and Knüppel, O., *Numerical Results for a Self-Validating Global Optimization Method*, technical report no. 94.1, 1994.
13. Kearfott, R. B., *Interval Newton / Generalized Bisection When There are Singularities near Roots*, Annals of Operations Research **25**, pp. 181–196, 1990.
14. Kearfott, R. B., *Preconditioners for the Interval Gauss–Seidel Method*, SIAM J. Numer. Anal. **27** (3), pp. 804–822, 1990.
15. Kearfott, R. B., and Novoa, M., *INTBIS, A Portable Interval Newton/Bisection Package (Algorithm 681)*, ACM Trans. Math. Software **16** (2), pp. 152–157, 1990.
16. Kearfott, R. B., Hu, C. Y., Novoa, M. III, *A Review of Preconditioners for the Interval Gauss–Seidel Method*, Interval Computations **1** (1), pp. 59–85, 1991.
17. Kearfott, R. B., *An Interval Branch and Bound Algorithm for Bound Constrained Optimization Problems*, Journal of Global Optimization **2**, pp. 259–280, 1992.
18. Kearfott, R. B., *Empirical Evaluation of Innovations in Interval Branch and Bound Algorithms for Nonlinear Algebraic Systems*, accepted for publication in SIAM J. Sci. Comput..
19. Kearfott, R. B., *A Review of Techniques in the Verified Solution of Constrained Global Optimization Problems*, preprint, 1994.
20. Kearfott, R. B., *On Verifying Feasibility in Equality Constrained Optimization Problems*, preprint, 1994.
21. Kearfott, R. B., Dawande, M., Du K.-S. and Hu, C.-Y., *Algorithm 737: INTLIB: A Portable FORTRAN 77 Interval Standard Function Library*, ACM Trans. Math. Software **20** (4), pp. 447–459, 1994.
22. Kearfott, R. B., *A Fortran 90 Environment for Research and Prototyping of Enclosure Algorithms for Constrained and Unconstrained Nonlinear Equations*, ACM Trans. Math. Software **21** (1), pp. 63–78, 1995.
23. Neumaier, A., *Interval Methods for Systems of Equations*, Cambridge University Press, Cambridge, England, 1990.
24. Ratschek, H., and Rokne, J., *New Computer Methods for Global Optimization*, Wiley, New York, 1988.
25. Ratz, D., *Automatische Ergebnisverifikation bei globalen Optimierungsproblemen*, Ph.D. dissertation, Universität Karlsruhe, 1992.
26. Ratz, D., *Box-Splitting Strategies for the Interval Gauss–Seidel Step in a Global Optimization Method*, Computing **53**, pp. 337–354, 1994.
27. Rump, S. M., *Verification Methods for Dense and Sparse Systems of Equations*, in Topics in Validated Computations, ed. J. Herzberger, Elsevier Science Publishers, Amsterdam, 1994.
28. Walster, G. W., Hansen, E. R. and Sengupta, S., *Test Results for a Global Optimization Algorithm*, in Numerical Optimization 1984, ed. P. T. Boggs, R. H. Byrd, and R. B. Schnabel, pp. 272–287, SIAM, Philadelphia, 1985.
29. Wolfe, M. A., *An Interval Algorithm for Constrained Global Optimization*, J. Comput. Appl. Math. **50**, pp. 605–612, 1994.