On Verifying Feasibility in Equality Constrained Optimization Problems

R. Baker Kearfott*
University of Southwestern Louisiana

December 18, 1996

Abstract

Techniques for verifying feasibility of equality constraints are presented. The underlying verification procedures are similar to a proposed algorithm of Hansen, but various possibilities, as well as additional procedures for handling bound constraints, are investigated. The overall scheme differs from some algorithms in that it *rigorously* verifies *exact* (rather than approximate) feasibility. The scheme starts with an approximate feasible point, then constructs a box (i.e. a set of tolerances) about this point within which it is rigorously verified that a feasible point exists. Alternate ways of proceeding are compared, and numerical results on a set of test problems appear.

Key words. constrained global optimization, verified computations, interval computations, bound constraints.

AMS subject classifications. 65K05, 90C26

1 Introduction and Motivation

The context of our present study is the global optimization problem

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

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

where $X = (x_1, \ldots, x_n)^T$. A general constrained optimization problem, including inequality constraints $g(X) \leq 0$ can be put into this form by introducing slack variables $s + g(X) = 0$ along with the bound constraint $0 \leq s \leq \infty$. See [14] for a practical discussion of this. See [19] and [6] for introductions to rigorously verified global optimization algorithms.

The conditions $a_{i_j} \leq x_{i_j} \leq b_{i_j}, \quad j = 1, \ldots, q$ represent actual bound constraints intrinsic to the problem formulation. In rigorous branch and bound algorithms, an*overall search region* $\mathbf{X}_0$ is generally defined through similar bounds $a_i \leq x_i \leq b_i$, $1 \leq i \leq n$; only those bounds corresponding to the index set $\{i_j\}_{j=1}^{q}$ should be treated as actual bound constraints.

In automatic verification techniques (normally using interval computations), rigorous verification of an approximate answer is usually much faster (and often more practical) than solution of the global problem with branch and bound techniques. In turn, a rigorously verified local solution can be incorporated into a branch and bound algorithm to effectively obtain global solutions; see, for example [3]. Thus, the verification process is an important step in both the verification and global search contexts.

In constrained global optimization, an upper bound $\overline{\phi}$ to the global minimum is invaluable in eliminating regions over which the range of ϕ lies above $\overline{\phi}$. See [7] or [3] for recent effective algorithms, as well as [19] or [6] for earlier examples. A rigorous upper bound $\overline{\phi}$ can be obtained by evaluating ϕ over a small region $\hat{\mathbf{X}}$ containing an approximate minimizer, using interval arithmetic to bound roundoff error. However, in the case of equality-constrained problems a rigorous upper bound can be obtained only if it is certain that $\hat{\mathbf{X}}$ contains a feasible point.

Hansen proposes a technique in [6, §12.3 ff.] for determining whether a particular box[1] $\mathbf{X}$ contains a feasible point. The underlying technique advocated in this paper is similar to that one. However, we recommend using it to verify *small* boxes around an approximate feasible point that has been found by a conventional floating-point method. Furthermore, in addition to Hansen's recommendation for the choice of coordinates to be held fixed, we also investigate two alternative schemes: choosing the null space by directly analyzing the null space of the constraint gradient matrix and by working directly with the corresponding rectangular system, without holding coordinates fixed.

Experiments with various practical bound-constrained problems has in-

[1] Throughout, we will speak of a *box* $\mathbf{X}$ to mean a rectangular parallelepiped; cf. §2 below.

dicated that optima often occur on lower-dimensional boundaries, with many active bound constraints. In contrast, for verification of feasibility of equality constraints, the dimension of the space in which the optimum occurs should be greater than or equal to the number of such constraints. In fact, verification succeeds most often when the approximate optimum is very accurately near a feasible point, and when the approximation is at the center of the box (coordinate bounds) used for verification. For this reason, approximate optima obtained from conventional floating point optimizers must first be perturbed away from bound constraints, then adjusted. Two schemes for this, one employing a heuristic and a more expensive one that is less likely to fail, are presented.

The next section gives our notational conventions, while §3 contains three algorithms[2] for verifying that a given box $\hat{\mathbf{X}}$ contains a feasible point, as well as an ϵ-inflation algorithm. Recommendations for tolerances for accuracy of the approximate point and for the epsilon-inflation algorithm also appear. Our techniques for perturbing approximate optima off bound constraints appear in §4. We describe the problems used in the experiments in §5, while implementation details appear in §6. The actual experiments are reported in §7. Some thoughts concerning handling constraints that are dependent appear in §8, while overall conclusions can be found in §9. The paper concludes with an acknowledgement.

2 Notation

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

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 $|\mathbf{x}| = \max\{|\underline{x}|, |\overline{x}|\}$.

[2] one due to Hansen

[3] often, the midpoint

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

The symbol $\phi(\mathbf{X})$ 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, braces $\{\cdot\}$ will denote vectors and parentheses $(\cdot)$ will denote points in $\mathbb{R}^p$.

We will use $\mathrm{int}(\mathbf{X})$ to denote the topological interior of a box $\mathbf{X}$.

Interval arithmetic will not be reviewed here. The reader may consult any of the numerous introductions, such as those in [1], [17], [18] or [19].

3 The Primary Algorithms

In the methods in [6, §12.3 ff.] and here, the computations rigorously prove $C(X) = 0$ at at least one point within a given box. The methods are based on applying an interval Newton method to the system $c_i(x) = 0$, $i = 1, \ldots, m$ with $n - m$ of the components of the box $\mathbf{X}$ held fixed.

The first technique proposed here involves computing the null space of the matrix of constraint gradients at a representative point, then setting those variables corresponding to coordinate directions most nearly lying in the null space to constants. This heuristic should maximize the chance that the linear space in which the interval Newton method is applied contains a feasible point. This idea is illustrated with $m = 1$ and $n = 2$ in figure 1. In summary, our method is as follows:

Algorithm 1 (Prove feasibility)

1. *Input an approximation $\check{X}$ to a feasible point, obtained through a conventional algorithm such as that of [4].*

2. *Compute a basis for the null space to $\nabla\mathbf{C}(\check{X})$, and store it in the columns of a matrix V .*

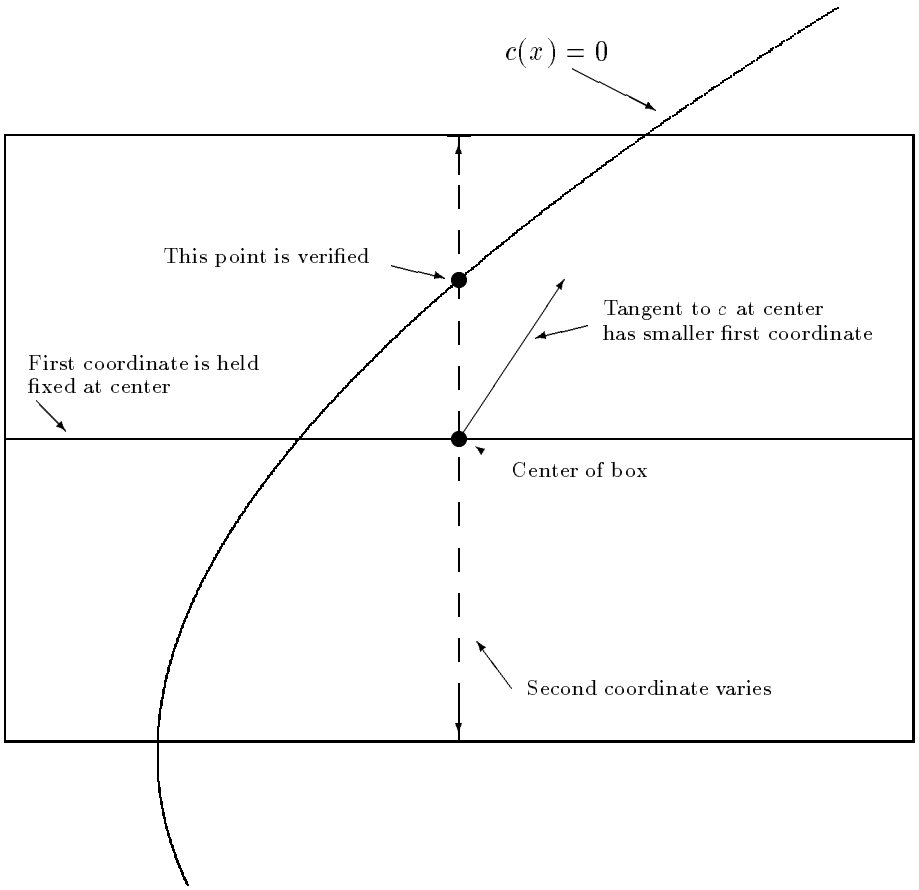

Figure 1: Verifying a feasible point with Algorithm 1

3. *Let* $V = \{v_{i,j}\}$, $1 \leq i \leq n$, $1 \leq j \leq n-m$ *with* $m \leq n$. *Then choose coordinates* $\{p_k\}_{k=1}^{m}$ *such that*

$$\sum_{i=1}^{n-m} |v_{p_k,i}| = \min_j \sum_{i=1}^{n-m} |v_{j,i}|$$

.

4. *Evaluate* $C(\check{X})$ *and* $\nabla \mathbf{C}(\hat{\mathbf{X}})$ *where* $\hat{\mathbf{X}}$ *has coordinates* $\hat{\mathbf{x}}_i = \check{x}_i$ *if* $i \neq p_k$ *for any* k *and* $\hat{\mathbf{x}}_i = [\check{x}_i - \epsilon, \check{x}_i + \epsilon]$ *for* $i = p_k$ *for some* k *and for an appropriately chosen* ϵ. *(See below.)*

5. *Let* Y *be the inverse of the* $m \times m$ *matrix consisting of columns* p_1 *through* p_m *of the midpoint matrix of* $\nabla \mathbf{C}(\hat{\mathbf{X}})$.

6. *Apply an interval Newton method corresponding to* $Y \nabla \mathbf{C}(\hat{\mathbf{X}})(X - \check{X}) = -YC(\check{X})$, *to obtain an image* $\hat{\mathbf{X}}_{new}$.

7. *If the result* $\hat{\mathbf{X}}_{new} \subseteq \text{int}\hat{\mathbf{X}}$, *then the point* $\check{X}$ *is feasible; Find* $\phi(\check{X})$ *and take the upper bound as a rigorous upper bound for a minimum.*

End Algorithm 1

We briefly describe our particular implementation of the interval Newton method in §6. Here, we can consider the interval Newton method to be an operator $\mathbf{N}(\hat{\mathbf{X}}, F)$ operating on a box $\hat{\mathbf{X}} \in \mathbb{R}^m$ and a function $F : \mathbb{R}^m \to \mathbb{R}^m$ defined over $\hat{\mathbf{X}}$, such that if the set $\mathbf{N}(\hat{\mathbf{X}}, F)$ is contained in the interior of $\hat{\mathbf{X}}$, then $F(X) = 0$ has a unique solution within $\hat{\mathbf{X}}$.

As seen in figure 1, the main idea is to start with a good approximation $\check{X}$ to a feasible point, and to envision a box to be constructed in $\mathbb{R}^n$ (the full parameter space[4]) around $\check{X}$, through which the feasible set, locally an $m-n$-manifold, passes. A hyperplane is then chosen in such a way that the chances that it intersects this manifold within the box are maximized.

The importance of having a good approximation to a feasible point centered in the box is illustrated in figure 2. It is also important to choose a sufficiently small box to avoid overestimation of the constraint gradient ranges by the interval evaluations used to compute $\mathbf{N}(\hat{\mathbf{X}}, F)$. However, a sufficiently large box, relative to the tolerance with which the center approximates the feasible point, is also necessary: otherwise, the picture will be more like figure 2 than figure 1. In our implementation and experiments

[4]or the full *reduced* parameter space, as explained in §4

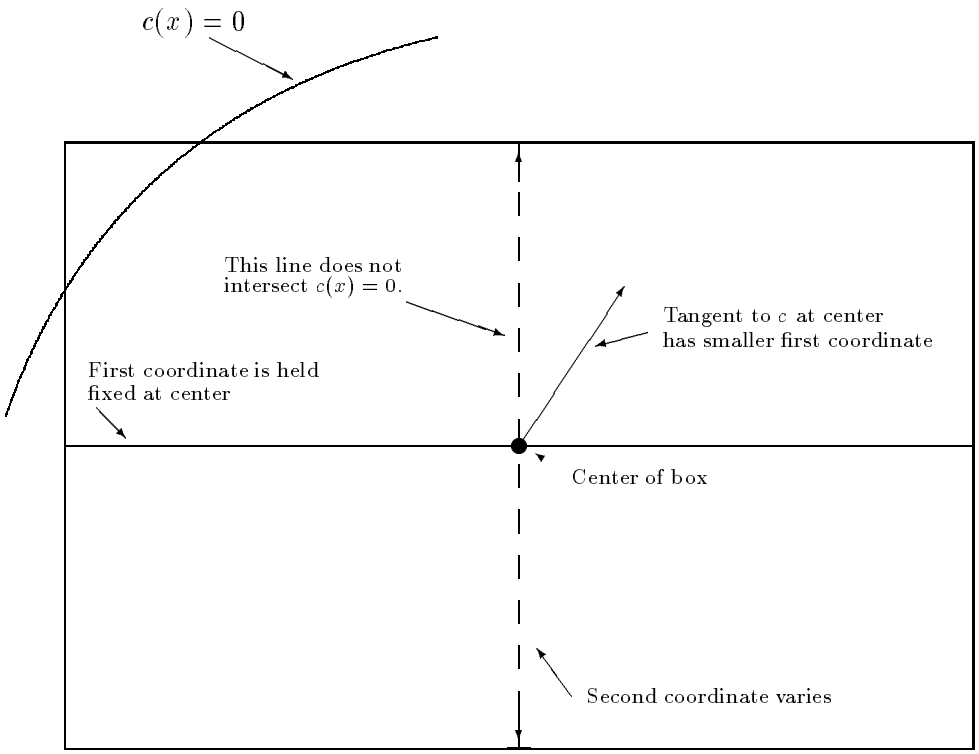

Figure 2: A feasible point should be centered in the box.

we input a domain tolerance $\epsilon_{\rm d}$. A conventional optimizer is then used to compute an approximately feasible approximate local optimum $\check{X}$ to the (heuristic) tolerance $\epsilon_{\rm d}{}^{1.5}$. A box $\hat{\mathbf{X}}$ is then constructed about $\check{X}$ such that the i-th coordinate bounds are $[\check{x}_i - \sigma, \check{x}_i + \sigma]$, where $\sigma = \max\{|\check{x}_i|, 1\}(\epsilon_{\rm d}/2)$.

Additionally, Algorithm 1 can be executed iteratively: we adjust the boxes $\hat{\mathbf{X}}$ in step 4 with an ϵ-inflation procedure [5] from iteration to iteration, to make it likely that the interval Newton method can verify the root. In fact, a modification of step 3 of Algorithm 3 in [13], as follows, can be used. However, our experience indicates that, if the tolerances for $\check{X}$ and the coordinate widths of $\hat{\mathbf{X}}$ are chosen as indicated above, feasibility is usually proven without inflation, if it is proven at all; see table 2.

Algorithm 2 (ϵ-inflation: Repeat Algorithm 1 while adjusting $\hat{\mathbf{X}}$)

1. *Input* $\check{X}$. *Also input* $\mathbf{X}_0$, *the original search region*[6] *Finally input a domain tolerance* $\epsilon_{\rm d}$.

2. *Construct* $\hat{\mathbf{X}}$ *centered* $\check{X}$ *such that the i-th coordinate is* $[\check{x}_i - \epsilon_{\rm d}\sigma, \check{x}_i + \epsilon_{\rm d}\sigma]$, *where* $\sigma = \max\{|\check{x}_i|, 100\epsilon_m\}$, *where* ϵ_m *is the machine epsilon.*

3. (Adjust $\hat{\mathbf{X}}$ until either existence can be proven or the box gets too big.) DO

   (a) Execute Algorithm 1.

   (b) IF *existence was verified in step 3a,*

   THEN

   EXIT *this algorithm with feasibility verified.*

   ELSE

   i. (Expand a coordinate of $\hat{\mathbf{X}}$.) *Expand one or more coordinates of* $\hat{\mathbf{X}}$, *provided the resulting box still lies within* $\mathbf{X}_0$. *(See equation 3 in § 7 below.)*

   ii. IF *a new box could not be generated in step 3(b)i,* THEN EXIT *this algorithm with failure to verify feasibility.*

   END IF

   END DO

[5] ϵ-inflation first appeared in [20].

[6] The original search region is used in step 3 to limit the size of the box $\hat{\mathbf{X}}$.

End Algorithm 2

Hansen's technique of [6, §12.3-12.4], as we have implemented it, differs from Algorithm 1 only in steps 2 and 3. In particular, it is

Algorithm 3 (Hansen's technique for choosing the coordinates)

Proceed as in Algorithm 1, except replace steps 2 and 3 by

2′ (Do Gaussian elimination with full pivoting)

(a) Compute the midpoint matrix $A \in \mathbb{R}^{m \times n}$ *of* $\nabla \mathbf{C}(\hat{\mathbf{X}})$.

(b) Perform Gaussian elimination with complete row and column pivoting on the rectangular matrix A.

3′ Choose the original indices of the columns of A that have been permuted into the last $n - m$ *columns during the elimination process to be the indices of those variables to be held fixed (i.e. to be replaced by points) in the interval Newton method. Equivalently, choose* $\{p_k\}_{k=1}^{m}$ *to be the original indices of the first* m *columns of the reduced matrix.*

In both Algorithm 1 and Algorithm 3, $n - m$ of the coordinates are to be points. A big advantage of this is that possible overestimation due to interval dependency in those coordinates is avoided. Also, if these coordinates are not fixed, the interval Newton would need to verify that, for *each* value that such parameters can take on within their respective intervals, there is a unique solution to the m-dimensional system corresponding to the m coordinates p_k considered to be variables. This geometrically limits the thin-ness of the images of these coordinate intervals under the interval Newton method, as is illustrated in figure 3. computations. The situation is analogous to that in in [15] (in which $n - m = 1$), where we verified the existence of the *entire* manifold $C(X) = 0$ within the box.

Nonetheless, Algorithm 1, and possibly Algorithm 3, may choose coordinates p_k for which feasibility cannot be verified, whereas a different choice of coordinates will allow verification. In such instances, one may work directly with the rectangular interval system, as in [15]. In particular, the *width optimal preconditioners* of [8] and [9] may be used[7]. The interval Newton

[7] Width-optimality minimizes the width of the image under the interval Newton method, and does not necessarily maximize the likelihood that the image is contained in the original coordinate bounds. Preconditioners satisfying the latter condition can be implemented, but they are somewhat more expensive to compute, and the two conditions are similar when the widths of the original coordinate bounds are small.

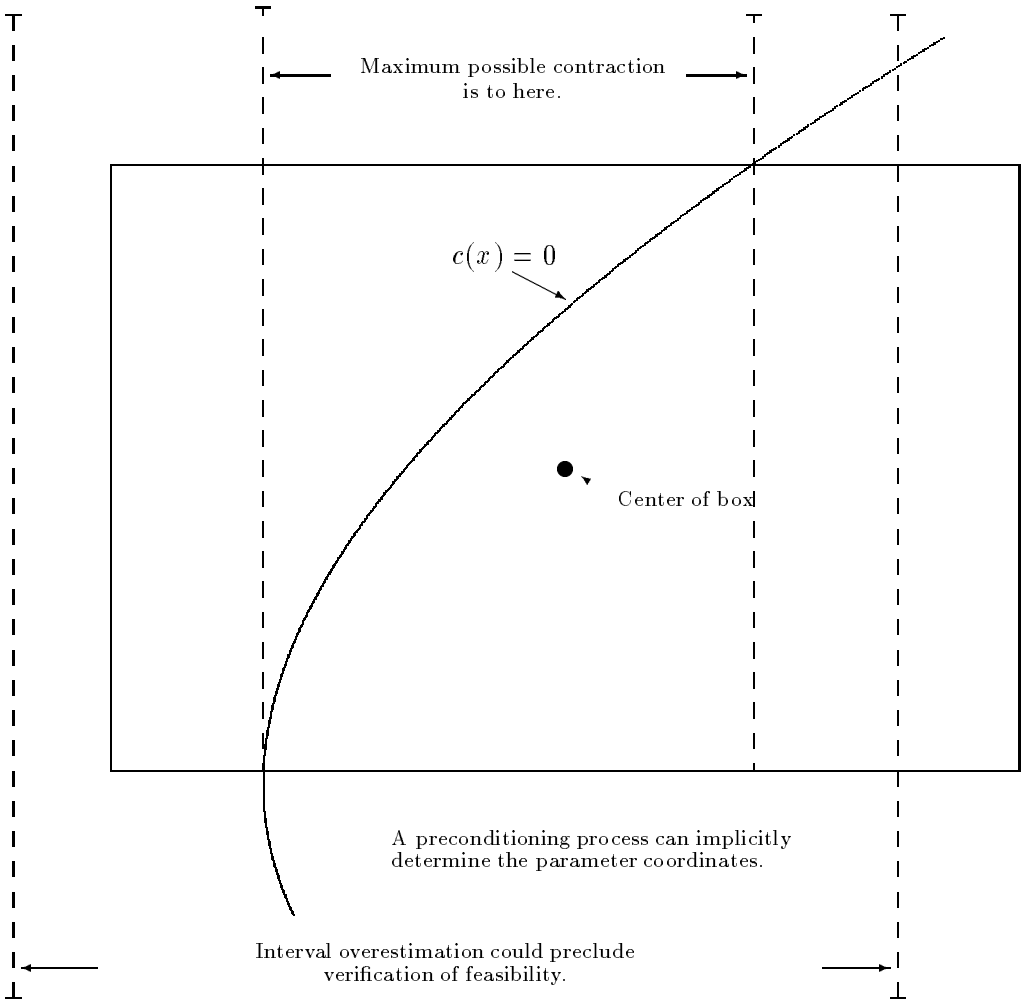

Figure 3: Verification with no coordinates held fixed

method is applied to *each* of the n coordinates, and feasibility is verified if at least m of the n images are contained in the corresponding original coordinate bounds. This is summarized in

Algorithm 4 (Verify feasibility using LP preconditioners on the original system)

1. *Input the box* $\hat{\mathbf{X}}$ *and approximate feasible point as in Algorithm 1 or Algorithm 3.*
2. *Compute the* $m \times n$ *interval matrix* $\mathbf{A} = \nabla \mathbf{C}(\hat{\mathbf{X}})$.
3. $\iota \leftarrow 0$.
4. DO *for* $i = 1$ *to* n.
   (a) *Compute the optimal LP preconditioner* $Y_i \in \mathbb{R}^m$ *corresponding to the* i*-th coordinate of the system* $\nabla \mathbf{C}(\hat{\mathbf{X}})(X - \check{X}) = -C(\check{X})$.
   (b) *Do a Gauss-Seidel step (i.e. formally solve for the* $i-$*th coordinate of* X*) on the preconditioned row* $Y_i \nabla \mathbf{C}(\hat{\mathbf{X}})(X - \check{X}) = -Y_i C(\check{X})$, *obtaining new coordinate bounds* $\tilde{\mathbf{x}}_i$.
      - IF $\tilde{\mathbf{x}}_i \cap \hat{\mathbf{x}}_i = \emptyset$ THEN EXIT *without verification.*
      - IF $\tilde{\mathbf{x}}_i \mathrm{int}(\hat{\mathbf{x}})$ THEN $\iota \leftarrow \iota + 1$

   END DO
5. IF $\iota \geq m$ THEN *mark feasibility as verified.*

End Algorithm 4

The actual Fortran 90 code for these procedures, only several pages, is available from the author upon request.

4 On Bound Constraints

In bound-constrained problems, it is often the rule, rather than the exception, that bound constraints are active at the optimizers. This is also true of inequality-constrained problems, which for simplicity and other reasons, we convert to equality- and bound-constrained problems; cf. [14]. An extreme case of this is linear programming, in which a maximal number of bound constraints must be active.

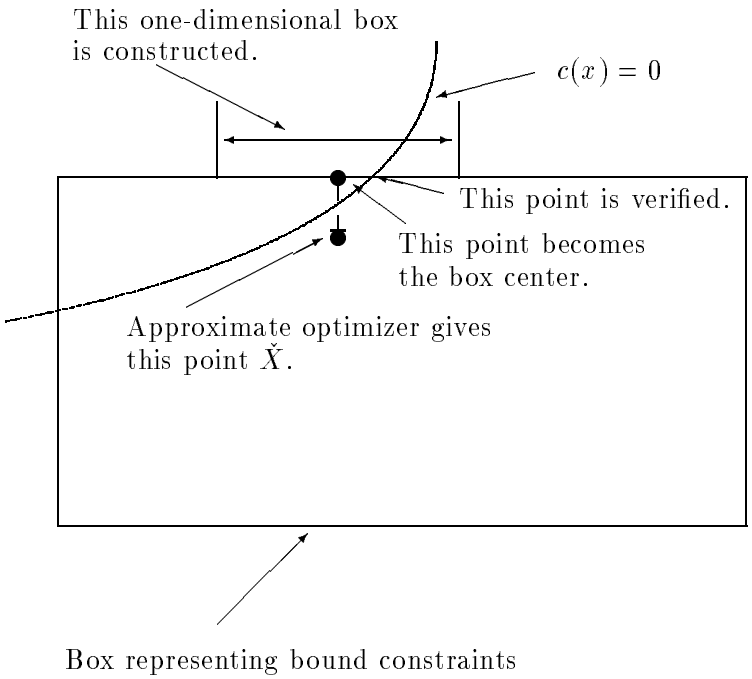

Figure 4: Projecting onto coordinate bounds

4.1 Projecting onto bound constraints

Good conventional floating-point general optimizers, such as that of [4], will typically provide an approximate optimizer $\check{X}$ that is close to one or more bound constraint surfaces, as is illustrated in figure 4. We can then *project* onto the exact active set of those bound constraints, and construct the box as in figure 1 within the n_r-dimensional subspace defined by the active constraints. Algorithm 1 then proceeds within this subspace. It is preferable to work within this subspace, provided its dimension is at least m. This is because the box corresponding to $\hat{\mathbf{X}}$ of Algorithm 1 can then be centered about the approximate feasible point, rather than having coordinate bounds that extend into the feasible region on one side and just a small distance to the boundary on the other. Interval Newton methods can more easily verify existence and uniqueness when the approximate feasible point is near the center, provided the approximate feasible point $\check{X}$ is near enough to an actual feasible point. In fact, the criterion $\mathbf{N}(\hat{\mathbf{X}}, F) \subset \mathrm{int}(\hat{\mathbf{X}})$ generally cannot be satisfied when the root to be verified lies on the boundary of $\hat{\mathbf{X}}$.

This process can be considered a way of pre-selecting coordinate indices $\{i_k\}_{k=1}^{r}$, $r \leq q$, whose coordinates are to be held fixed in Algorithm 1, Algorithm 3 or Algorithm 4. In the case of Algorithms 1 or 3, the remaining $n - m - r$ coordinate indices to be held fixed are then chosen in step 3 or $3'$, while verification proceeds in the subspace. In the case of Algorithm 4, n_r replaces n. (Equivalently, the coordinates to be varied are chosen from the complement $\{i\}_{i=1}^{n} \setminus \{i_k\}_{k=1}^{r}$.)

4.2 Moving off bound constraints – moving all coordinates

Despite advantages of working in as small a subspace as possible, a typical occurrence in our test real-world problems is that the number m of equality constraints is *more* than the dimension n_r of the space of variables corresponding to inactive bound constraints. In those cases, to rigorously verify feasibility, we should center a box *away* from some of the bound constraints. That is, to allow enough degrees of freedom for verification, we must perturb the point $\check{X}$ away from some of the bounds. The bounds to be made non-active can be chosen according to the same criteria used to choose the variables to be held fixed (as in step 3 of Algorithm 1), except that V is formed over the *entire space*, rather than the n_r-dimensional reduced space, and we choose coordinates to vary in an order opposite to the order in which we would choose coordinates to be fixed. Alternately, we could choose the

coordinates to be moved off their bounds according to the column pivoting criterion of Algorithm 3. In particular, applying Gaussian elimination with full column pivoting to the full $m \times n$ constraint gradient matrix, we simply choose the coordinates to be perturbed to correspond to those columns that have been moved into the first m positions of the column pivoting process. Specifically, the variables encountered first in scanning from left to right may be moved off their constraints.

When a coordinate of the approximate feasible point is moved off its bound, the other coordinates should be adjusted in the direction of the null space of the constraints, so that the perturbed point will remain near the manifold of feasible points. Because of this, it is not in general sufficient to perturb only m of the coordinates. Because of this, when $m < n$, to obtain enough degrees of freedom to adjust the point tangent to the null space we choose an additional variable to be perturbed, but to be held fixed for purposes of the interval Newton method. Let $\nabla C(\check{X}) = A \in \mathbb{R}^{m \times n}$, let the columns of $W_2 \in \mathbb{R}^{m \times m}$ be set to the columns of A corresponding to the m variables to be perturbed. Furthermore, $W_1 \in \mathbb{R}^{(m) \times 1}$ be that column of A corresponding to the variable to be perturbed but be fixed[8]. We then obtain the following system:

$$W_2 \Delta = -W_1 \delta_0, \tag{2}$$

where the entries of Δ represent increments in corresponding coordinates of $\check{X}$, and δ_0 corresponds to the fixed increment of the variable to be adjusted but held fixed in the interval Newton method. The system 2 can be uniquely solved for the increment vector Δ. If this increment vector does not point into the original box, then verification fails. Otherwise, the perturbation is made, the appropriate coordinates are marked as not corresponding to active bound constraints, and the box is constructed as before.

Care should be taken in the choice of δ_0. In particular, we have chosen δ_0 to be on the order of the square root of the domain tolerance ϵ_{d}, whereas the initial box in step 2 of Algorithm 2 is on the order of ϵ_{d}. Thus, although we have perturbed $\check{X}$ in the direction of the null space of ∇C, it may be necessary to correct it again. This can be done by calling the local optimizer again, using the perturbed point as a starting point while fixing the coordinate corresponding to W_1.

[8] This variable is chosen to be that variable from among the ones corresponding to active bound constraints, such that the sum of step 3 of Algorithm 1 is minimized, or from the last $n - m$ columns of the matrix obtained by full column pivoting.

4.3 Moving off bound constraints – one coordinate at a time

Preliminary experience has indicated that neither method of choosing the coordinates to be perturbed described above works universally. In particular, if the coordinates to be chosen are taken by analysis of the null space matrix V, then W_2 may be singular. Even if W_2 is non-singular, and has been chosen from the column pivot information in Gaussian elimination, some of the coordinates may be perturbed outside of their bound constraints. In such instances, we can apply the following algorithm.

Algorithm 5 (Move coordinates off their bound constraints one by one)

1. *Input* $\check{X}$ *and the domain tolerance* ϵ_{d}.

2. (Make sure coordinates not on bound constraints cannot be perturbed onto them, by redefining the bound constraints for the correction step.) IF $\check{x}_i$ *does not correspond to a bound constraint,* THEN *set* $\tilde{a}_i = a_i + \sigma$, $\tilde{b}_i = b_i - \sigma$, *where* $[a_i, b_i]$ *was the original coordinate range and* $\sigma = \max\{|\check{x}_i|, 1\}\epsilon_{\mathrm{d}}$.

3. *Choose*[9] $n - n_r$ *coordinates* $\{\iota_j\}_{j=1}^{n-n_r}$ *as candidates for perturbation.*

4. DO $j = 1$ *to* $n - n_r$.

   IF $\check{x}_{\iota_j}$ *is on its bound coordinate,* THEN

   (a) *Replace* $\check{x}_{\iota_j}$ *by* $\check{x}_{\iota_j} + \delta$, *where* $\delta = \max\{|\check{x}_{\iota_j}|, 1\}\sqrt{\epsilon_{\mathrm{d}}}$ *if* $\check{x}_{\iota_j}$ *was on its lower bound and* $\delta = -\max\{|\check{x}_{\iota_j}|, 1\}\sqrt{\epsilon_{\mathrm{d}}}$ *if* $\check{x}_{\iota_j}$ *was on its upper bound.*

   (b) *If* $\check{x}_{\iota_j}$ *was originally on its lower bound, redefine the lower bound for the correction step to be the perturbed value of* $\check{x}_{\iota_j}$. *Similarly redefine the upper bound if* $\check{x}_{\iota_j}$ *was originally on its upper bound.*

   (c) (Correction Step) *Run the local bound-constrained optimizer, using the temporary bound constraints defined in steps 2 and 4b to obtain a new* $\check{X}$.

   (d) (Reassess active bound constraints) *Recompute which bound constraints are active, possibly projecting onto bound constraints as in figure 4; obtain a new reduced dimension* n_r.

   (e) IF $n_r \geq m$ THEN EXIT *with success.*

[9] using one of the two schemes mentioned above

END IF

END DO

5. IF $n_r < m$ THEN EXIT *with failure.*

End Algorithm 5

4.4 On an alternative general procedure

Instead of trying to identify active bound constraints and adjust the number of degrees of freedom, it is possible to include the bound constraints in the Fritz–John conditions. We then verify feasibility by applying an interval Newton method to this Fritz–John system in the full-dimensional space. However, in preliminary experiments we have seen several cases (problems fpqp3, fppb1 and fphe1 in §5 below) in which feasibility can be verified by the methods advocated here, but the Fritz–John system is singular. These are precisely those problems in which perturbation is required. This can be seen by examining the block structure of the Fritz John system for interval evaluations over boxes that contain both lower and upper bounds.

Thus, in cases when feasibility can be verified with the Fritz–John system, the techniques expounded here are more efficient, while the perturbation techniques expounded here still work in the other cases.

5 The Test Set

Most of the problems in the test set are taken from [5]. We selected these to be non-trivial problems with a variety of constraint types, as well as differing numbers of variables and constraints. We also tried the problems from [23]. Although the latter are relatively simple, [23] contains one of the few published experimental results for general interval constrained optimization algorithms. Also, inclusion of these problems allows contrasting the relative ease of *verifying*, as done in this paper, with global search algorithms such as that of [23].

Each problem is identified with a mmemonic, given below.

Basic attributes of the test problems appear in table 1. In each problem, each non-trivial inequality constraint in the original formulation was replaced by an equality constraint and a bound constraint on a slack variable. The numbers of variables in the table reflect these added slack variables; the

Table 1: Summary attributes of the test problems

Problem name	# vars.	# equality constr.	# bound constr.	type of constraints
fpnlp3	6	3	8	linear
fpqp3	23	9	32	linear
fplnp6	4	2	6	degree 4
fppb1	9	6	13	bilinear
fphe1	16	13	28	bilinear
gould	4	2	4	quadratic
bracken	3	2	1	quadratic
wolfe3	3	2	2	quadratic

bound constraints reflect original bound constraints, natural lower bounds of zero, and lower bounds on the added slack variables.

We now present information on those feasible points analyzed in the experiments. We list the bound constraints for each problem, along with the coordinates of the approximate optimum for which feasibility was verified. The bound constraints are listed as interval vectors, with "$\cdot$" representing no constraint. The set of active bound constraints at the approximate optimum is similarly represented as a vector; components with "A" correspond to active bound constraints and components with "$\cdot$" correspond to inactive bound constraints.

fpnlp3 is the third nonlinear programming test problem, [5, p. 28]. The bound constraints are

$$([0,3],[0,\cdot],[0,\cdot],[0,1],[0,\cdot],[0,\cdot])$$

while the approximate solution of interest is

$$(\frac{4}{3},4,0,0,\frac{8}{3},0),$$

so the active bound constraints at the approximate solution are

$$(\cdot,\cdot,A,A,\cdot,A).$$

The dimension n_r of the reduced space is thus 3, equal to the number of constraints, so no perturbation of the approximate solution is required to verify feasibility.

fpqp3 is the third quadratic programming test problem, [5, p. 8]. The bound constraints are

$$([0,1],[0,1],[0,1],[0,1],[0,1],[0,1],[0,1],[0,1],[0,1],[0,1],[0,\cdot],[0,\cdot],[0,\cdot],[0,1],[0,\cdot],[0,\cdot],[0,\cdot],[0,\cdot],[0,\cdot],[0,\cdot],[0,\cdot],[0,\cdot],[0,\cdot])$$

while the approximate solution of interest is

$$(1,1,1,1,1,1,1,1,1,3,3,3,1,0,0,0,5,5,0,6,6),$$

so the active bound constraints at the approximate solution are

$$(A,A,A,A,A,A,A,A,A,\cdot,\cdot,\cdot,A,A,A,A,\cdot,\cdot,\cdot,A,\cdot,\cdot).$$

The dimension n_r of the reduced space is thus 8, while the number of equality constraints is 9. Thus, perturbation of the approximate solution is required for verification of feasibility.

fpnlp6 is the sixth nonlinear programming test problem, [5, p. 30]. The bound constraints are

$$[0,3],[0,4],[0,\cdot],[0,\cdot]$$

while the approximate solution of interest is

$$(2.3295,3.1783,0,0),$$

so the active bound constraints at the approximate solution are

$$(\cdot,\cdot,A,A).$$

The dimension n_r of the reduced space is thus 2, equal to the number of constraints, so no perturbation is required.

fppb1 is the first pooling-blending test problem, [5, p. 59]. The bound constraints are

$$([0,100],[0,200],[0,\cdot],[0,\cdot],[0,\cdot],[0,\cdot],[0,\cdot],[0,\cdot],[0,\cdot],[0,\cdot],[0,\cdot])$$

while the approximate solution of interest is

$$(0,200,0,100,0,100,0,100,1,0,0),$$

so the active bound constraints at the approximate solution are

$$(A, A, A, \cdot, A, \cdot, A, \cdot, \cdot, A, A).$$

The dimension n_r of the reduced space is thus 4, while the number of equality constraints is 6. Thus, perturbation of the approximate solution is required for verification of feasibility.

fphe1 is the first heat exchanger network test problem, [5, pp. 63–66]. The bound constraints are

$$([0, 10], [0, 10], [2.941, 10], [3.158, 10], [0, 10], [0, 10], [150, 240], [150, 190], [250, 490], [210, 340], [0, 10], [0, 10], [10, \cdot], [10, \cdot], [10, \cdot], [10, \cdot])$$

while the approximate solution of interest is

$$(0, 10, 10, 10, 10, 0, 210, 150, 310, 210, 10, 0, 190, 140, 40, 50),$$

so the active bound constraints at the approximate solution are

$$(A, A, A, A, A, A, \cdot, A, \cdot, A, A, A, \cdot, \cdot, \cdot, \cdot).$$

The dimension n_r of the reduced space is thus 6, while the number of equality constraints is 13. Thus, perturbation of the approximate solution is required for verification of feasibility.

gould is the first test problem in [23]. The bound constraints are

$$([13, \cdot], [0, \cdot], [0, \cdot], [0, \cdot])$$

while the approximate solution of interest is

$$(14.095, .842960788, 0, 0),$$

so the active bound constraints at the approximate solution are

$$(\cdot, \cdot, A, A).$$

The dimension n_r of the reduced space is thus 2, equal to the number of constraints, so no perturbation is required.

bracken is the second test problem in [23]. The bound constraints are

$$([\cdot, \cdot], [\cdot, \cdot], [0, \cdot]),$$

while the approximate solution of interest is

$$(0.822875653899075, 0.911437827385507, 0),$$

so the active bound constraints at the approximate solution are

$$(\cdot, \cdot, A).$$

The dimension n_r of the reduced space is thus 2, equal to the number of constraints, so no perturbation is required.

wolfe3 is the third test problem in [23]. The bound constraints are

$$([0, \cdot], [0, \cdot], [\cdot, \cdot])$$

while the approximate solution of interest is

$$(1.2247448866122757, 1.2247448567842063, 1.7320508069462872).$$

Thus, no bound constraints are active, $n_r = n = 3 > m = 2$, so no perturbation is required for feasibility verification.

We feel these problems are representative, particularly from the point of view of varying numbers of active bound constraints and parameter space dimensions. We are aware of more extensive test sets, such as the huge collection described in [2]. The latter problems are in SIF format[10]. We will either develop a SIF converter or otherwise investigate these problems[11] in the future.

In particular, it would be of interest to try the techniques on problems with highly nonlinear, transcendental constraints. However, most of the problems in [5] involve only mildly nonlinear constraints, or can be easily put into such a form. Appearance of transcendental functions *per se* should not affect verification, but complicated expressions conceivably could[12].

[10] a generalization of MPS format, explained in [4]

[11] Our system, described in [11], also requires minimal programming work: All that is necessary is programming the objective and constraints in a natural Fortran syntax.

[12] although such expressions pose less of an interval dependency problem in the verification context, with small boxes, than in the global branch and bound context

6 Some Implementation Details

The algorithms in §3 were programmed in the Fortran 90 environment developed for that purpose and described in [11]. Similarly, the functions described in §5 were programmed using the same Fortran 90 system, and an internal symbolic representation of the objective function, constraints and gradient of the objective function was generated prior to execution of the numerical tests. In the actual tests, generic routines then interpreted this single internal representation to obtain both floating point and interval values and derivative matrices.

The Lancelot package routine "DAUGLG", described in [4], was used to obtain the approximate feasible points $\check{X}$. We included the objective functions when we called this routine, since a primary use of verified feasible points in global optimization algorithms is to obtain good upper bounds on the global minimum. For convenience, a generic interface to DAUGLG was used. An unfortunate side effect of this was that no structure information (linearity information, special separability structure) that DAUGLG could utilize was passed to DAUGLG. Exact gradients and Hessian matrices were passed to DAUGLG. We needed to use a starting penalty parameter in DAUGLG much smaller than the default ($\sqrt{\epsilon_m} \approx 1.49 \times 10^{-8}$ instead of .01; cf.[4]), since in several problems DAUGLG did not exhibit convergence, even when the initial guess was extremely close to a solution.

The interval Newton method was the interval Gauss–Seidel method preconditioned with the inverse midpoint matrix. Interval derivative matrices were used. Though not optimal, these choices are common. Special preconditioning matrices as in [8] or [9] would be more effective, but should not give results substantially different from the inverse midpoint preconditioner when the boxes $\hat{\mathbf{X}}$ have small widths; furthermore, such preconditioners were originally designed to reduce the widths of boxes, not to verify existence; a slightly different formulation (to be published elsewhere) is advisable for verification.

Finally, it is possible to use slope matrices, as described in [22], *together* with an idea of Hansen, as in [6, §6.6], for sharper bounds, and hence a sharper verification test. However, our present implementation of slopes, following [22], has wide intervals due to roundoff error when the boxes are small. Also, there is theoretically little difference when the widths of $\mathbf{X}$ are small, so we merely used interval derivative matrices.

7 Experimental Results

7.1 Environment and parameters

The NAG Fortran 90 compiler, version 2 was used on a Sparc 20. Execution times were measured using the routine "DSECND" (as would be used with the f77 libraries).

In all cases, ϵ_{d} was set to 10^{-5}.

A good initial guess was handed to DAUGLG, which then corrected it. Afterwords, the procedure of §4.1 was used to determine the active bound constraints. The procedure in §4.2 was then tried[13], provided the number of inactive bound constraints was less than the number of equality constraints. If this procedure did not succeed, then Algorithm 5 was tried; the latter succeeded in all cases in producing an approximate feasible point with a sufficient number of coordinates off their bound constraints.

After the perturbed feasible point was produced, each of Algorithm 1, Algorithm 4 and and Algorithm 3 was tried. Success of verification, along with CPU times and other performance information, was recorded.

7.2 Output and conclusions

Table 2 lists the success of each of the three algorithms, as well as the number of times the box was expanded in Algorithm 2 before feasibility of a point within the box was either verified, or until there was failure, with "1" indicating no expansions. Failure to prove feasibility was due either to failure to compute an inverse midpoint preconditioner or due to the overlapping of the region expanded about the approximate solution with the boundary of the original region. Simultaneous expansion of each coordinate was done according to:

$$[\text{new width}] = [\text{old width}](1.1)^{n_r k}, \tag{3}$$

where k is the number of times step 3a of Algorithm 2 had already been executed.

From table 2, it is clear that, with our choice of tolerances and expansion factors, it is not advantageous to expand the box in Algorithm 2: feasibility was never verified after an expansion. In fact, closer examination of the experimental output shows that failure to prove feasibility was not due to

[13] using the null space V, rather than the Gaussian elimination technique. We found no significant differences in success rates in preliminary experiments.

Table 2: Verification success of the three schemes

	null space		LP		elimination	
problem	verified	ninfl	verified	ninfl	verified	ninfl
fpnlp3	yes	1	yes	1	yes	1
fpqp3	–	1	–	4	yes	1
fpnlp6	yes	1	yes	1	yes	1
fppb1	yes	1	–	10	yes	1
fphe1	–	4	yes	1	yes	1
gould	yes	1	yes	1	yes	1
bracken	yes	1	yes	1	yes	1
wolfe3	yes	1	–	7	yes	1

roundoff problems (where increasing the box size would help), but due to intrinsic properties of the algorithm and problem geometry.

A second conclusion from table 2 is that choosing the coordinates to be held fixed by Gaussian elimination (advocated by Hansen) is the most reliable method.

In table 3, we give numbers of interval evaluations of the constraints and constraint gradients, for each of the three methods and each of the problems. This can be used for comparison with other results and tasks, such as the global search algorithm in [23].

In table 4, we list CPU times in seconds for the three verification algorithms.

Tables 2, 3 and 4 indicate that the Hansen variant is both more reliable and less costly, although most of the difference in cost is attributable to the fact that the Hansen variant is more reliable, and failure to prove feasibility cost more[14].

An astounding aspect of these experiments is that it took far more effort to perturb the feasible point (by the procedures in §4.1 and Algorithm 5) than to verify the feasible point, once perturbed. This is indicated in table 5, where CPU times are given for the perturbation along the null space and for Algorithm 5. Perturbation in the direction of the null space of $\nabla \mathbf{C}$ was tried (and necessary) only if there were not already a sufficient number of bound constraints; similarly Algorithm 5 was attempted only if perturbation in the

[14] because of repeated, but unnecessary, expansion steps

Table 3: Interval constraint and constraint gradient evaluations

	null space		LP		elimination	
problem	C	$\nabla\mathbf{C}$	C	$\nabla\mathbf{C}$	C	$\nabla\mathbf{C}$
fpnlp3	9	9	6	9	9	9
fpqp3	27	396	72	792	27	396
fpnlp6	6	4	4	4	6	4
fppb1	18	84	120	420	18	84
fphe1	156	832	26	208	39	416
gould	6	4	4	4	6	4
bracken	6	4	4	4	6	4
wolfe3	6	12	28	42	6	12
Totals	234	1345	264	1483	117	929

Table 4: CPU times for the verification steps

problem	null space	LP	elimination
fpnlp3	0.02	0.01	0.01
fpqp3	0.05	0.43	0.05
fpnlp6	0.01	0.01	0.01
fppb1	0.03	0.26	0.03
fphe1	0.33	0.20	0.11
gould	0.01	0.01	0.00
bracken	0.01	0.01	0.01
wolfe3	0.02	0.07	0.01
Totals:	0.48	1.00	0.23

Table 5: CPU times for perturbation off the bounds

	in null space		Algorithm 5		
problem	necssary?	CPU	necessary?	CPU	NCALLS
fpnlp3	no	–	no	–	–
fpqp3	yes	0.04	yes	4.99	13
fpnlp6	no	–	no	–	–
fppb1	yes	0.19	no	–	–
fphe1	yes	0.04	yes	292.8	3
gould	no	–	no	–	–
bracken	no	–	no	–	–
wolfe3	no	–	no	–	–

direction of the null space failed. In addition to CPU times, table 5 contains the number NCALLS of calls to the floating point optimizer AUGLG for Algorithm 5; this is equal to the number of single coordinates perturbed. (The null space perturbation, if successful, required one such call to AUGLG, and none otherwise.)

Most of the CPU time in these perturbation steps was spent in the constrained optimizer AUGLG. Some of this was due to excessive iteration because of tight tolerances and roundoff error. Also, we demanded *optimization* from AUGLG, not just location of an approximate feasible point. This is consistent with the idea is that verified feasibility will be used to get a rigorous but good upper bound on a global optimum. The objective function in "fphe1" is highly nonlinear, which could have contributed to the difficulty apparent in the table. The CPU times can undoubtedly be improved by tuning with respect to stopping tolerances, initial penalty parameter, method of linear equation solution, etc. In these experiments, we did not do so beyond what was necessary to get the algorithm to terminate.

8 On Linear Dependence in the Constraints

It is not possible to verify feasibility via the algorithms in §3 if the c_i are linearly dependent at the feasible point. However, approximate feasibility can be verified in the sense explained in [19, §5.4] and used in [23]. In

particular, given $\epsilon > 0$, it is possible to verify that

$$\exists X_\epsilon \in \mathbf{X} \quad \text{such that} \quad \|C(X)\| \leq \epsilon. \tag{4}$$

In [23], the condition $\|C(X)\| \leq \epsilon$ is used in a global search of $\mathbf{X}$ to bound feasible sets. In contrast, for verifying feasible points, the object of the study here, the following procedure can be used.

Algorithm 6 (Approximate feasibility for singular problems)

1. *Input an approximation $\check{X}$ to a feasible point, obtained through a conventional algorithm such as that of [4].*
2. *Compute the dimension of the null space to $\nabla \mathbf{C}(\check{X})$.*
3. IF *the (numerical) dimension of the null space to $\nabla \mathbf{C}(\check{X})$ is equal to m*

   THEN *continue with steps 3 through 7 of Algorithm 1.*

   ELSE

   (a) *Check that $\|C(\check{X})\| \leq \epsilon$ with interval arithmetic.*

   (b) IF $\|C(\check{X})\| \leq \epsilon$

   THEN

   i. *Mark that the constraints have only been verified approximately.*

   ii. *Find $\phi(\check{X})$ and take the upper bound as a rigorous upper bound for a minimum of the relaxed problem (that is, the problem obtained from the original problem by replacing $C(X) = 0$ by $C(X) \in [-\epsilon, \epsilon]^m$).*

   END IF

   END IF

End Algorithm 6

We do not recommend applying Algorithm 6 in a general context. In particular, it is theoretically possible that, even though $C(\check{X}) \in [-\epsilon, \epsilon]^m$, $\check{X}$ is infeasible and the upper bound $\overline{\phi}$ obtained from interval evaluation of $\phi(\check{X})$ is smaller than the global optimum actually attained at a feasible point. In this case, a branch and bound algorithm that did not take account throughout of the fact that problem 1 has been replaced by the relaxed

problem with the equality constraints replaced by conditions 4 could reject actual global optimizers in favor of the infeasible point. Furthermore, it may be possible to verify feasibility at other points in the region and thus obtain rigorous but useful $\overline{\phi}$.

Nonetheless, rigorous branch and bound algorithms can be devised *a priori* completely around the relaxed problem. This approach would make sense if it were difficult to verify feasibility with the techniques in §3; because a good $\overline{\phi}$ would then not be available, the overall branch and bound algorithm would then became impractically expensive unless the relaxed problem were solved instead. Furthermore, the relaxed problem may have as much significance in the original application as problem 1.

There are additional criteria that can be used in the case of singular constraints. For example, using techniques as in Algorithm 1, it should be possible to verify that all combinations of p of the constraints are simultaneously feasible in a box around $\check{X}$, as in figure 5 for $p = 1$. That figure can be interpreted as a cross-section in $\mathbb{R}^3$, where the constraints $c_1 = 0$ and $c_2 = 0$ represent two surfaces that intersect tangentially[15].

Finally, perhaps true linear dependence in the constraints should be detected algebraically before numerical solution is attempted.

9 Summary

We have proposed various techniques for proving feasibility of a point in a neighborhood of an approximate solution of an optimization problem posed in terms of equality constraints and bound constraints. We have tested these techniques on a small but significant set of test problems. The techniques appear to be reliable and inexpensive, relative to the local floating point optimizers used in conjunction with them. They should prove valuable in global branch and bound algorithms and other applications.

Although verifying feasibility has been discussed in the literature, to our knowledge, there has been little previous thorough development and empirical evaluation of techniques. A notable exception is [16]. There, as large a box as possible was constructed within which inequalities of the form $g(x) < 0$ can be rigorously verified; the algorithm was applied to a significant engineering design problem (of composite laminates).

[15] The usual situation for two constraints in $\mathbb{R}^2$ would be that there are no degrees of freedom left.

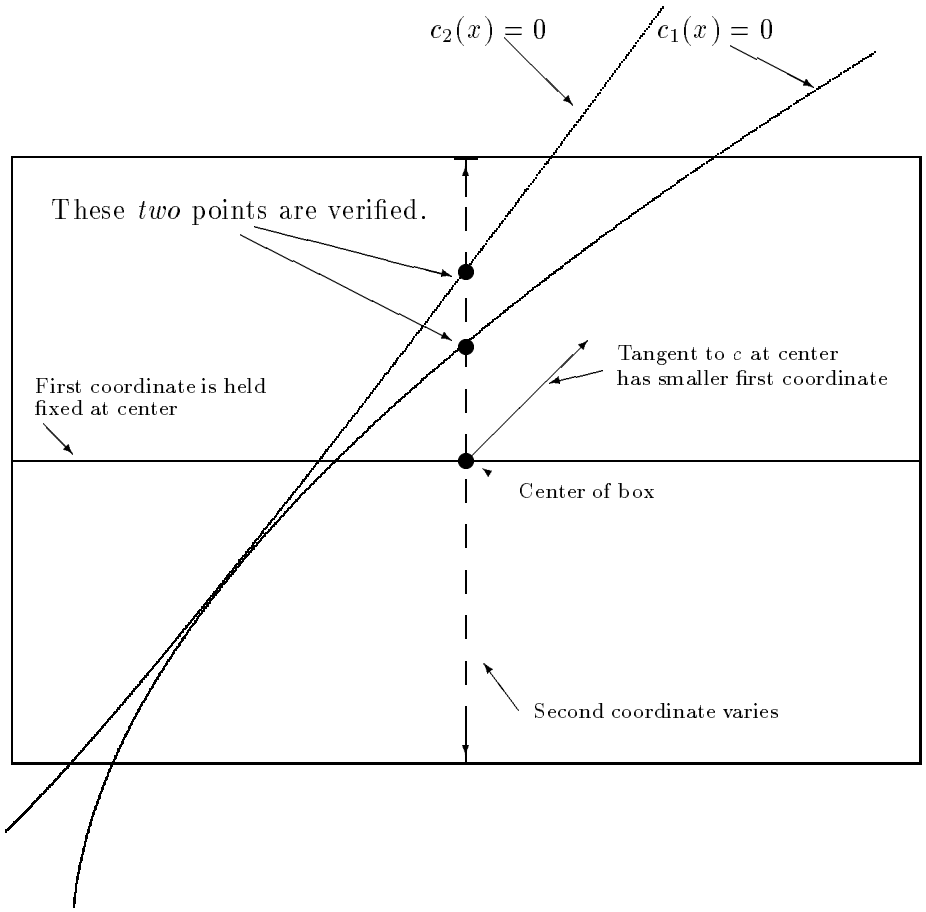

Figure 5: Partial verification when the constraint gradients are linearly dependent

10 Acknowledgement

I wish to acknowledge Shiying Ning for programming and checking the more complicated test problems used here.

References

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

[2] Bongartz, I., Conn, A. R., Gould, N., and Toint, Ph.L., *CUTE: Constrained and Unconstrained Testing Environment*, preprint, 1993.

[3] 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.

[4] Conn, A. R., Gould, N. and Toint, Ph.L., *LANCELOT: A Fortran Package for Large-Scale Nonlinear Optimization*, Springer-Verlag, New York, 1992.

[5] Floudas, C. A. and Pardalos, P. M., *A Collection of Test Problems for Constrained Global Optimization Algorithms*, Springer-Verlag, New York, 1990.

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

[7] Jansson, C. and Knüppel, O., *A Global Minimization Method: The Multi-Dimensional Case*, preprint, 1992.

[8] Kearfott, R. B., *Preconditioners for the Interval Gauss–Seidel Method*, SIAM J. Numer. Anal. **27** (3), pp. 804–822, 1990.

[9] 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.

[10] Kearfott, R. B., Dawande, M., Du K.-S. and Hu, C.-Y., *INTLIB: A Portable FORTRAN 77 Interval Standard Function Library*, accepted for publication in ACM Trans. Math. Software.

[11] Kearfott, R. B., *A Fortran 90 Environment for Research and Prototyping of Enclosure Algorithms for Constrained and Unconstrained Nonlinear Equations*, accepted for publication in ACM Trans. Math. Software.

[12] Kearfott, R. B., *Empirical Evaluation of Innovations in Interval Branch and Bound Algorithms for Nonlinear Algebraic Systems*, preprint, 1994.

[13] Kearfott, R. B., *Empirical Evaluation of Innovations in Interval Branch and Bound Algorithms for Nonlinear Algebraic Systems*, preprint, 1994.

[14] Kearfott, R. B., *Techniques in the Verified Solution of Constrained Global Optimization Problems*, preprint, 1994.

[15] Kearfott, R. B. and Xing, Z., *An Interval Step Control for Continuation Methods*, SIAM J. Numer. Anal. **31** (3), pp. 892–914, 1994.

[16] Kristinsdottir, B. P., Zabinsky, Z. B., Csendes, T., Tuttle, M. E., *Methodologies for Tolerance Intervals*, Interval Computations (3), pp. 133–147, 1993.

[17] Moore, R. E., *Methods and Applications of Interval Analysis*, SIAM, Philadelphia, 1979.

[18] Neumaier, A., *Interval Methods for Systems of Equations*, Cambridge University Press, Cambridge, England, 1990.

[19] Ratschek, H., and Rokne, J., *New Computer Methods for Global Optimization*, Wiley, New York, 1988.

[20] Rump, S. M., *Kleine Fehlerschranken bei Matrixproblemen*, Ph.D. dissertation, Universität Karlsruhe, 1980.

[21] 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.

[22] 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.

[23] Wolfe, M. A., *An Interval Algorithm for Constrained Global Optimization*, J. Comput. Appl. Math. **50**, pp. 605–612, 1994.