A Reflective Newton Method for Minimizing
a Quadratic Function Subject to Bounds
on Some of The Variables*

Thomas F. Coleman
Yuying Li

TR 92-1315  
November 1992

Department of Computer Science  
Cornell University  
Ithaca, NY 14853-7501

*Research partially supported by the Applied Mathematical Sciences Research Program (KC-04-02) of the Office of Energy Research of the U.S. Department of Energy under grant DE-FG02-86ER25013.A000, and by the Computational Mathematics Program of the National Science Foundation under grant DMS-8706133, and by the Cornell Theory Center, which receives major funding from the National Science Foundation and IBM Corporation, with additional support from New York State and members of its Corporate Research Institute.

This report also appears as an Advanced Computing Research Institute (Cornell Theory Center) report CTC-92-TR111 and as a Cornell Computational Optimization Project (CCOP) report 92-24.
A reflective Newton method for minimizing a quadratic function subject to bounds on some of the variables

Thomas F. Coleman and Yuying Li  
Computer Science Department  
and  
Advanced Computing Research Institute  
Cornell University  
Ithaca, New York 14853

November 16, 1992

Abstract. We propose a new algorithm, a reflective Newton method, for the minimization of a quadratic function of many variables subject to upper and lower bounds on some of the variables. The method applies to a general (indefinite) quadratic function, for which a local minimizer subject to bounds is required, and is particularly suitable for the large-scale problem. Our new method exhibits strong convergence properties, global and quadratic convergence, and appears to have significant practical potential. Strictly feasible points are generated. Experimental results on moderately large and sparse problems support the claim of practicality for large-scale problems.

---

1 Research partially supported by the Applied Mathematical Sciences Research Program (KC-04-02) of the Office of Energy Research of the U.S. Department of Energy under grant DE-FG02-86ER25013.A000, and by the Computational Mathematics Program of the National Science Foundation under grant DMS-8706133, and by the Cornell Theory Center which receives major funding from the National Science Foundation and IBM Corporation, with additional support from New York State and members of its Corporate Research Institute.

2 The Advanced Computing Research Institute is a unit of the Cornell Center for Theory and Simulation in Science and Engineering (Theory Center).
1. Introduction. In this paper we propose a new algorithm for solving the box-constrained quadratic programming problem

\[(1.1) \quad \min_x \{ q(x) \overset{\text{def}}{=} c^T x + \frac{1}{2} x^T H x : \ l \leq x \leq u \}. \]

The matrix $H$ is symmetric and, in general, indefinite; $l \in \{ \mathbb{R} \cup \{-\infty\}\}^n$, $u \in \{ \mathbb{R} \cup \{\infty\}\}^n$, $l \leq u$. We denote the feasible region $\mathcal{F} = \{ x : \ l \leq x \leq u \}$ and the strict interior, $\text{int}(\mathcal{F}) = \{ x : \ l < x < u \}$. When $H$ is indefinite we are interested in locating a local minimizer.

Problem (1.1) arises as a subproblem when minimizing general nonlinear functions subject to bounds and as a problem in its own right. The box-constrained quadratic programming problem represents an important class of optimization problems and has been the subject of considerable recent work (e.g., [1, 5, 12, 13, 16, 19, 21, 24, 26, 27, 32]). A special subclass deserves mention: the box-constrained least-squares problem,

\[(1.2) \quad \min_x \{ \| A x - b \|_2 : \ l \leq x \leq u \}, \]

where $A$ is a rectangular $m$-by-$n$ matrix with $m > n$. Our proposed algorithm can of course be applied to this special case if we form $H = A^T A$ and $c = -A^T b$. The determination of a version of our algorithm which does not involve the formation of the matrix $H$ is an open question.

We propose a new approach, a reflective Newton algorithm. The algorithm generates a sequence of strictly feasible iterates, $\{ x_k \}$, which converges under standard assumptions to a local solution of (1.1), $x_*$, at a quadratic convergence rate. Coleman and Li [10] establish theoretical properties of the reflective Newton approach applied to the general nonlinear box-constrained problem – as we indicate in Section 5 these results apply directly to the reflective Newton procedure proposed here for the quadratic minimization problem (1.1). In this paper we discuss the nature of the reflective transformation (Section 2); we discuss the reflective Newton approach as applied to problem (1.1) with emphasis on a specialized line search to exploit the special structure of this problem (Section 3); numerical experiments involving an implementation of the reflective Newton method applied to box-constrained quadratic minimization (1.1) are discussed (Section 4).

The sequence $\{ x_k \}$ generated by the algorithm is strictly feasible: therefore, the algorithm can be regarded as an “interior-point” algorithm. However, this is a misleading classification. The algorithm differs markedly from methods commonly referred to as “interior-point” algorithms. For example, the proposed algorithm does not use a barrier function to ensure feasibility. The algorithm generates descent directions for $q$ and then follows a piecewise linear path, reflecting off constraints as they are encountered. Most interior point methods, on the other hand, generate descent directions (for some function) and then restrict the step, along this straight-line direction, to ensure feasibility.

The algorithm most similar to our current proposal is probably the recent method due to Coleman and Hulbert [6]. (There is also a strong connection to previous work
by Coleman and Li [7, 8, 9, 20] on various norm minimization problems.) Both are driven by the nonlinear system of equations representing first-order optimality conditions. Both methods require piecewise quadratic minimization. The methods differ in that our new algorithm is more general: positive definiteness of \( H \) is not required and it is not necessary to have finite upper and lower bounds on all the variables – the Coleman/Hulbert method requires both these restrictive properties. Finally, the Coleman/Hulbert method is an exterior-point method, requiring strict decrease in a piecewise quadratic "dual" function, whereas the new method generates feasible iterates, requiring strict decrease in the original quadratic function \( q \).

There are four key observations that underpin our new approach.

First, it is possible to change the constrained problem (1.1) to an unconstrained problem without using a penalty parameter. We can replace (1.1) with an unconstrained problem,

\[
\min_{y \in \mathbb{R}^n} \tilde{q}(y)
\]

where \( \tilde{q}(y) \) is a continuous piecewise quadratic function of \( y \), and \( y \in \mathbb{R}^n \) is unrestricted. Details of this transformation are given in the next section including a result, Theorem 1, proving the equivalence of (1.3) with (1.1). One view of our algorithm is that it is designed to find a local minimizer of \( \tilde{q}(y) \). Variables \( x \) and \( y \) are related by a piecewise linear transformation, a reflective transformation, \( x = R(y) \). Transformation \( R \) yields a feasible sequence \( \{x_k\} \). Moreover, evaluation of \( \tilde{q}(y) \) corresponds to evaluation of \( q(R(y)) \).

Alternatively, one can view our approach entirely in the original variables \( x \). Then, instead of describing the method as a descent algorithm for the transformed problem \( \tilde{q}(y) \), our method can be described as a method that generates feasible iterates by following a piecewise linear path induced by the reflective mapping \( R \). We discuss this below.

The second key observation is that the first-order optimality conditions for (1.1), or equivalently (1.3), can be expressed as a single system of nonlinear equations,

\[
F(y) = 0
\]

and a Newton step for this system is a descent direction for \( \tilde{q} \) in a neighbourhood of a local solution \( y_* \). Moreover, in a neighbourhood of a local solution to (1.3) a full Newton step for (1.4), i.e., a unit step size in the Newton direction, yields decrease in \( \tilde{q}(y) \). This is a very important point because it suggests that a Newton process for (1.4) is compatible with (1.3), at least in the neighbourhood of a solution. It suggests that ultimate second-order convergence can be achieved while decreasing \( \tilde{q}(y) \).

The third observation leads to globalization of the Newton process. It turns out that the Newton equation for (1.4), the nonlinear system representing first-order optimality conditions, can be written in the form:

\[
M(y)s = -\nabla_y \tilde{q}
\]
where $M$ is a symmetric matrix. Moreover, it turns out that $M$ is positive definite in a neighbourhood of a minimizer of $\hat{q}$ and $M$ can be interpreted, loosely, as a second derivative matrix for $\hat{q}(y)$. This suggests the use of an ellipsoidal constraint to ensure a descent direction when far from the solution. Specifically, solve

$$
\min \{ s^T \nabla_y \hat{q} + \frac{1}{2}s^T M(y)s : \|D^{-1}s\|_2 \leq \Delta \}
$$

where $D$ is a positive diagonal scaling matrix and $\Delta$ is positive. As we discuss in [10], a good choice for matrix $D(x)$ is

$$
D(x) = \text{diag}(|v(x)|^\frac{1}{2}),
$$

i.e., $D$ is a diagonal matrix with the $i^{th}$ diagonal component equal to $|v_i(x)|^\frac{1}{2}$. Vector $v_i(x)$ is defined in Figure 1 where

$$
g(x) \overset{\text{def}}{=} \nabla q(x) = Hx + c.
$$

Diagonal matrix $D$ plays an important role in this paper – henceforth we reserve the notation $D$, without superscript, to refer to definition (1.7). Of course $D_k$ refers to (1.7) with all quantities defined at the current point $x_k$.

(i) If $g_i < 0$ and $u_i < \infty$ then $v_i = x_i - u_i$.
(ii) If $g_i \geq 0$ and $l_i > -\infty$ then $v_i = x_i - l_i$.
(iii) If $g_i < 0$ and $u_i = \infty$ then $v_i = -1$.
(iv) If $g_i \geq 0$ and $l_i = -\infty$ then $v_i = 1$.

**Fig. 1. Definition of $v(x)$**

Solving (1.6) involves solving a symmetric positive definite system,

$$
(DMD + \lambda I)\dot{s}^y = -D\nabla_y \hat{q}
$$

for a suitable $\lambda$, and then $s \leftarrow D\dot{s}^y$. Thus it is easy to see that (1.6) leads to a descent direction for $\hat{q}$ at the current point. It may be felt that solving (1.6) is an expensive way to determine a descent direction in the large-scale setting. With this in mind a restricted version of (1.6) is used in our algorithm. In particular, similar to [2] we usually restrict $s$ to be in a low-dimensional subspace $S$. So (1.6) is replaced with

$$
\min \{ s^T \nabla_y \hat{q} + \frac{1}{2}s^T M(y)s : \|D^{-1}s\|_2 \leq \Delta, s \in S \}
$$

where $S$ is a low-dimensional subspace of $\mathbb{R}^n$. Provided the ellipsoidal constraint $\|D^{-1}s\|_2 \leq \Delta$ becomes inactive near the solution, and the low-dimensional subspace $S$

---

3 The function $\hat{q}$ is a piecewise quadratic function of $y$: therefore, $\nabla_y \hat{q}$ does not always exist. However, the proposed algorithm only generates points where $\nabla_y \hat{q}$ is defined.
ultimately includes the Newton direction, the solution to (1.9) will eventually be the Newton step (1.5).

The fourth major ingredient of our approach is the line search. Once a descent direction \( s_k \) is determined, a one-dimensional line search is performed to approximately locate a minimizer of \( \hat{q}_k(\alpha) \equiv \hat{q}_k(y_k + \alpha s_k^k) \). But \( \hat{q}_k \) has structure: \( \hat{q}_k \) is a one-dimensional piecewise quadratic function and so an efficient specialized line search procedure can be used. (Alternative view: A one-dimensional piecewise linear search is performed along a “reflective path”, \( p_k(\alpha) \), defined by the reflective transformation \( R \) and beginning at \( x_k \).)

We conclude the introduction with a short review of optimality conditions for problem (1.1).

The first-order optimality conditions can be written: If a feasible point \( x_* \) is a local minimizer of (1.1) then

\[
(1.10) \quad D_*^2 g_* = 0.
\]

Let \( \text{Free}_* \) denote the set of indices corresponding to “free” variables at point \( x_* \):

\[
\text{Free}_* = \{ i : l_i < (x_*)_i < u_i \}.
\]

Second-order necessary conditions can be written\(^4\): If a feasible point \( x_* \) is a local minimizer of (1.1) then \( D_*^2 g_* = 0 \) and \( H^{\text{Free}_*} \geq 0 \) where \( H^{\text{Free}_*} \) is the submatrix of \( H \) corresponding to the index set \( \text{Free}_* \).

These conditions are necessary but not sufficient. To state practical sufficiency conditions we first need a definition of degeneracy.

---

**DEFINITION 1.** A point \( x \in \mathbb{R}^n \) is nondegenerate if, for each index \( i \):

\[
g_i = 0 \implies l_i < x_i < u_i.
\]

---

With this definition we can state second-order sufficiency conditions: If a nondegenerate feasible point \( x_* \) satisfies \( D_*^2 g_* = 0 \) and \( H^{\text{Free}_*} > 0 \), then \( x_* \) is a local minimizer of (1.1).

2. The Reflective Transformation. One interpretation of our approach to solving the box-constrained quadratic programming problem (1.1) involves a transformation to an unconstrained piecewise quadratic minimization problem (1.3). The purpose of this section is to introduce this transformation. Since some of the ideas involved are more generally applicable, we begin our discussion at a more abstract level and gradually work our way back to the box-constrained quadratic programming situation.

\(^4\) Notation: If a matrix \( A \) is a symmetric matrix then we write \( A > 0 \) to mean \( A \) is positive definite; \( A \geq 0 \) means \( A \) is positive semi-definite.
Consider the problem

\[(2.1)\quad \min_{x} \{f(x) : x \in C\}\]

where \(f\) is a continuous function, \(f : \mathbb{R}^n \rightarrow \mathbb{R}^1\), and \(C\) is a closed connected region of \(\mathbb{R}^n\). We consider when the constrained problem (2.1) can be replaced with an unconstrained minimization problem of the form,

\[(2.2)\quad \min_{y \in \mathbb{R}^n} f(R(y)),\]

where \(R\) is a continuous onto mapping, \(R : \mathbb{R}^n \rightarrow C\).

What further restrictions on the mapping \(R\) make this an acceptable transformation? To see that continuity is not enough consider the following one-dimensional example. Let \(f(x) = -x\) and let \(C = [0, 1]\). Obviously there is only one local solution (the global solution), \(x_* = 1\). However, let \(R(y)\) be any continuous function, mapping \(\mathbb{R}\) onto \([0, 1]\), with a strict local maximizer at \(\bar{y}\) with \(\bar{x} = R(\bar{y}) \in (0, 1)\). It is easy to see that \(\bar{y}\) is a local minimizer of \(f(R(y))\) but \(\bar{x}\) is clearly not a local solution to the original problem.

The following property plays the key role in answering this question.

---

Open Mapping: Let \(R : \mathbb{R}^n \rightarrow C\). Then \(R\) is an open mapping if for each \(\epsilon > 0\) and pair \(\{\bar{y}, \bar{x} = R(\bar{y})\}\) there exists \(\delta > 0\) such that

\[(2.3)\quad \{R(y) : y \in N_\epsilon(\bar{y})\} \supseteq N_\delta(\bar{x}) \cap C.\]

---

See Munkres [25], for example, for a discussion of open mappings. We can now answer our question.

**Theorem 1.** Let \(R : \mathbb{R}^n \rightarrow C\) be a continuous onto mapping. Further, assume \(R\) is an open mapping. Then,

(i) If \(y_*\) is a local minimizer of (2.2) then \(\bar{x} = R(y_*)\) is a local minimizer of (2.1).

(ii) If \(x_*\) is a local minimizer of (2.1) then for each \(\bar{y} \in \mathbb{R}^n\) such that \(R(\bar{y}) = x_*\), \(\bar{y}\) is a local minimizer of (2.2). Moreover, there exists at least one \(\bar{y}\) such that \(R(\bar{y}) = x_*\).

(iii) Problem (2.1) is unbounded below if and only if problem (2.2) is unbounded below.

**Proof.** (i) Assume \(y_*\) is a local minimizer of (2.2). Let \(\bar{x} = R(y_*)\). Since \(y_*\) is a local minimizer, there exists \(\epsilon > 0\) such that

\[f(R(y)) \geq f(R(y_*)), \quad \forall y \in N_\epsilon(y_*).\]
But by (2.3) there exists $\delta > 0$ such that

$$\{ R(y) : y \in N_\varepsilon(x^*) \} \supseteq N_\delta(\bar{x}) \cap C. \tag{2.4}$$

Hence for each $x^* \in N_\delta(\bar{x}) \cap C$ there exists $\tilde{y} \in N_\varepsilon(x^*)$ such that $R(\tilde{y}) = x$ and

$$f(x) = f(R(\tilde{y})) \geq f(R(y^*)) = f(\bar{x}).$$

Therefore, $\bar{x}$ is a local minimizer of (2.1).

(ii) Assume $x^*$ is a local minimizer of (2.1). But $R$ is an onto mapping and therefore there exists $\tilde{y} \in \mathbb{R}^n$ such that $x = R(\tilde{y})$. Since $x^*$ is a local minimizer, there exists $\varepsilon > 0$ such that

$$f(x) \geq f(x^*), \quad \forall x \in N_\varepsilon(x^*) \cap C.$$ 

By continuity there exists $\delta > 0$ such that $y \in N_\delta(\tilde{y})$ implies $R(y) \in N_\varepsilon(x^*) \cap C$. Therefore, for all $y \in N_\delta(\tilde{y})$,

$$f(R(y)) \geq f(x^*) = f(R(\tilde{y})).$$

Hence, $\tilde{y}$ is a local minimizer of (2.2).

(iii) Suppose $\{y_k\}$ is a sequence such that

$$\lim_{k \to \infty} f(R(y_k)) = -\infty.$$ 

Clearly if $x_k = R(y_k), \; k = 1 : \infty$, then $\{x_k\} \in C$ and $\lim_{k \to \infty} f(x_k) = -\infty$.

Alternatively, assume $\{x_k\} \in C$ and

$$\lim_{k \to \infty} f(x_k) = -\infty.$$ 

But $R$ is an onto mapping; therefore, for each $x_k$ there exists $y_k$ such that $R(y_k) = x_k$ and

$$\lim_{k \to \infty} f(R(y_k)) = -\infty,$$

and therefore (iii) is established.

To illustrate, consider the problem

$$\min_{x} \{f(x) : x \geq 0\}, \tag{2.5}$$

i.e., $C = \{x : x \geq 0\}$. A definition of $R$ that clearly satisfies the open mapping property is $R(y) = \frac{x^* y}{2}$ where "\( \ast \)" denotes component-wise multiplication. Note that $R$ is
differentiable and the Jacobian of $R$, $J^R(y)$, is nonsingular if and only if $R(y) \in \text{int}(C)$, where $\text{int}(C)$ is the interior of $C$. Specifically, $\nabla_y f = J^R \nabla_x f = D^y \nabla_x f$ and

$$
\nabla_x^2 f(y) = \nabla_x f \nabla_x^2 R + (J^R)^T \nabla_x^2 R J^R = D^y + D^y H D^y,
$$

where $D^y$ is the diagonal matrix $\text{diag}(\nabla_x f)$ and $D^y$ is the diagonal matrix $\text{diag}(y)$. (Note that $\nabla_y R$ is a tensor term and $\nabla_x f \nabla_y R$ is a matrix – diagonal in this case.) Therefore, this definition of $R$ leads to an unconstrained twice-differentiable minimization problem and standard techniques can be used to solve (2.2). Unfortunately, our numerical experience with this approach has been mixed: In particular, as problems become large and ill-conditioning (and near-degeneracy) increases, the number of iterations required by standard minimization algorithms, to achieve good accuracy, becomes quite large. We feel this is due in part to the fact that this transformation causes an increase in the complexity of the function to be minimized: e.g., a quadratic function becomes a quartic. Our objection to this approach is largely numerical – ill-conditioning in the original problem is accentuated when the problem is transformed to a more complex form. Note also that the transformed problem may have many more local minimizers – by Theorem 1 this, in itself, is not a problem. However, along with this increase in the number of local minimizers comes an increase in negative curvature and this may cause some optimization algorithms some difficulty. In any event, our experience with this simple differentiable transformation has not been satisfactory: the subject of this paper is an alternative definition of $R$.

For problem (2.5) consider $R(y) = |v|$, where if $v$ is a vector, $|v|$ denotes the vector whose components are the absolute values of the vector $v$. It is clear that the open mapping property holds. Note that $R$ is not everywhere differentiable. In particular, $R$ is differentiable at point $y$ if and only if $R(y) \in \text{int}(C)$, i.e., $y; \neq 0$. In this case $J^R(y) = \text{diag}(\text{sgn}(y))$ and $J^R(y)$ is obviously nonsingular. Using this transformation, $f(R(y))$ has a piecewise differentiable nature as a function of $y$. For example, if $f(x)$ is a quadratic function then $f(R(y))$ is a piecewise quadratic function of $y$.

We now extend the absolute value approach, $R(y) = |y|$, to handle the more general situation,

$$
(2.6) \quad C = \mathcal{F} = \{ x : l \leq x \leq u \}
$$

where for each index $i$ either $u_i$ is finite or $u_i = \infty$. Similarly, for each index $i$ either $l_i$ is finite or $l_i = -\infty$. We assume $l < u$.

---

For simplicity we assume that the finite values of $u$ are all equal to unity and the finite values of $l$ are all equal to zero (a simple translation and scaling can achieve this form\(^5\)).

---

The transformation we propose, $x = R(y)$, is a diagonal transformation, i.e., for

---

\(^5\) A definition of the reflective transformation applied directly to the general problem is given in [10]
Fig. 2. The 1-Dimensional Reflective Transformation (Finite Upper and Lower Bound)

each index \(i\), \(x_i\) depends only on \(y_i\). This transformation, \(x = R(y)\), induces a piecewise linear "reflective" path in \(x\).

For example, if \(u_i = 1\) and \(l_i = 0\) then \(R_i\) is illustrated in Figure 2.

If \(l_i = 0\) and \(u_i = \infty\) then \(R_i\) is the absolute value function; if \(l_i = -\infty\) and \(u_i = 1\) then \(R_i\) is illustrated in Figure 3.

Finally, if \(u_i = -\infty\) and \(l_i = +\infty\) then \(R_i(y_i) = x_i\). The four cases are described more formally in Figure 4.

It is easy to verify that \(R\) satisfies the requirements of Theorem 1; therefore, use of \(R\) does not introduce extraneous local minimizers nor does it restrict the set of local minimizers.

Using the reflective transformation, problem (1.1) can be replaced with the unconstrained piecewise quadratic minimization (1.3). In principle, problem (1.3) can be solved by a descent direction quadratic algorithm, e.g., Algorithm 1 in Figure 5.

An advantage of using this piecewise linear transformation \(R\) is the linear aspect of the transformation: when \(y\) is a differentiable point the local complexity of \(\hat{f}(y) \overset{df}{=} f(R(y))\) is the same as the local complexity of \(f(x)\). When \(f = q\) is a quadratic function, \(\hat{f} = \hat{q}\) is a piecewise quadratic function. The apparent disadvantage is the piecewise nature of \(\hat{f}(y)\). This lack of differentiability means that conventional nonlinear minimization methods cannot be used.

In particular, in order to guarantee convergence, restrictions on the nature of the descent direction \(s^y\) must be imposed. To see this suppose that \(y_k\) is very close to a hyperplane \(x_j = R(y_j) = l_j\) and \(s^y\) is a descent direction for \(\hat{q}\) at \(y_k\). If \(s^y\) is nearly perpendicular to this hyperplane then the usual descent condition, \(\nabla_y \hat{q}^T s^y < 0\), may only result in a very small step since \(\nabla_y \hat{q}\) is not continuous at \(x_j = R(y_j) = l_j\). In [10] we describe two properties, "constraint-compatibility" and "consistency", which help guarantee sufficiently long steps and, consequently, global convergence. We discuss this
briefly in Section 3.

The straight-line direction $s_k^v$ corresponds to a piecewise linear path in $x$. This piecewise linear path can be described as follows. For simplicity, and without loss of generality, assume $y_k = x_k$. Let $s_k^x = s_k^v$. Define the vector\(^6\)

$$BR_k = \max[(l - x_k) / s_k^x, (u - x_k) / s_k^x].$$

Component $i$ of vector $BR_k$ records the positive distance from $x_k$ to the breakpoint corresponding to variable $x_{ki}$ in the direction $s_k^x$. The piecewise linear (reflective) path is defined by Algorithm 2 in Figure 6. Since only a single outer iteration is considered, we do not include the subscript $k$ with the variables in our description of Algorithm 2 - dependence on $k$ is assumed.

Given the current point $x_k$ and a descent direction $s_k^x$, let $p_k(\alpha)$ denote the piecewise linear path defined by Algorithm 2: For $\beta_k^{i-1} \leq \alpha < \beta_k^i$,

$$p_k(\alpha) = b_k^{i-1} + (\alpha - \beta_k^{i-1})p_k^i.$$

Note that it is now possible to describe Algorithm 1 entirely in $x$-space without explicitly introducing either the function $\hat{q}$ or the variables $y$. We do this in Algorithm 3–Figure 8.

The difference between Algorithm 1 and Algorithm 3 is notational. The view presented by Algorithm 3 has the advantage that it is in the original space – visualization

---

\(^6\) For the purpose of computing $BR$ we assume the following rules regarding arithmetic with infinities. If $a$ is a finite scalar then $a + \infty = \infty$, $a - \infty = -\infty$, $\frac{a}{\infty} = \infty \cdot \text{sgn}(a)$, $\frac{\infty}{a} = -\infty \cdot \text{sgn}(a)$, $\frac{a}{0} = \text{sgn}(a) \cdot \infty$, $\frac{\infty}{0} = \infty$, and $\frac{-\infty}{0} = -\infty$, where $\text{sgn}(a) = +1$ if $a \geq 0$, $\text{sgn}(a) < 0$ if $a < 0$. 

10
Case 1: \((l_i = 0, \ u_i = 1)\)

To evaluate \(x_i = R(y)_i\):

\[
w_i = |y_i| \mod 2, \quad x_i = \min(w_i, 2 - w_i).
\]

If \(x_i \neq 0, 1\) then we can differentiate \(R\) to obtain the \(i^{th}\) diagonal element of the diagonal Jacobian matrix \(J^R\):

\[
\text{If } w_i < 2 - w_i, \ J^R_{ii} = \text{sgn}(y_i), \ \text{else } J^R_{ii} = -\text{sgn}(y_i).
\]

Case 2: \((l_i = 0, \ u_i = \infty)\)

To evaluate \(x_i = R(y)_i\):

\[
x_i = |y_i|.
\]

If \(x_i \neq 0\) then we can differentiate \(R\) to obtain the \(i^{th}\) diagonal element of the Jacobian matrix \(J^R\):

\[
J^R_{ii} = \text{sgn}(y_i).
\]

Case 3: \((l_i = -\infty, \ u_i = 1)\)

To evaluate \(x_i = R(y)_i\):

\[
\text{If } y_i \leq 1, \ x_i = y_i, \ \text{else } x_i = 2 - y_i.
\]

If \(x_i \neq 1\) then we can differentiate \(R\) to obtain the \(i^{th}\) diagonal element of the Jacobian matrix \(J^R\):

\[
\text{If } y_i < 1 \begin{cases} J^R_{ii} = 1, & \text{else } J^R_{ii} = -1. \end{cases}
\]

Case 4: \((l_i = -\infty, \ u_i = \infty)\).

In this case there are no constraints on \(x_i\) and so \(x_i = y_i, \ J^R_{ii} = 1\).

\textbf{Fig. 4.} \textit{The Reflective Transformation }R
Algorithm 1
Choose $y_1 \in \text{int}(\mathcal{F})$.
For $k = 1, 2, ...$
1. Determine a descent direction $s_k^y$ for $\hat{q}(y)$ at $y_k$
2. Perform an approximate line minimization of $\hat{q}(y_k + \alpha s_k^y)$, with respect to $\alpha$, to determine an acceptable stepsize $\alpha_k$ (such that $\alpha_k$ does not correspond to a breakpoint)
3. $y_{k+1} = y_k + \alpha_k s_k^y$

**FIG. 5. Descent dir'n algorithm for $\hat{f}(y)$**

Algorithm 2 [Let $\beta^0 = 0$, $p^1 = s^x$, set $b^0 = x_k$]
[i is a finite upper bound on the number of segments of the path to be determined]

For $i = 1 : i_u$
1. Let $\beta^i$ be the distance to the nearest breakpoint along $p^i$:

   $$\beta^i = \min\{BR : BR > 0\}$$

2. Define $i^{th}$ breakpoint: $b^i = b^{i-1} + (\beta^i - \beta^{i-1})p^i$.
3. Reflect to get new dir'n and update $BR$:
   (a) $p^{i+1} = p^i$
   (b) For each $j$ such that $(b^i)_j = u_j$ (or $(b^i)_j = l_j$)
       - $BR(j) = BR(j) + \frac{|(u_j - l_j)|}{(s^x)_j}$
       - $(p^{i+1})_j = -(p^i)_j$

**FIG. 6. Determine the linear reflective path $p$**

of the reflective process is natural. The advantage of the first view, Algorithm 1, is that the algorithm is a straight line descent direction algorithm, a familiar structure. It is probably useful for the reader to keep both views in mind. In this paper we will primarily work in the $x$-space and Algorithm 3. For simplicity we now drop the superscripts $y$ and $x$ (e.g., $s^y$ becomes $s$).

It is well known that a descent direction algorithm demands sufficient decrease at every step in order to achieve reasonable convergence properties. We use conditions suggested by Goldfarb [18] for use in the unconstrained setting: Given $0 < \sigma_l < \sigma_u < 1$ and a descent direction $s_k$, $\alpha_k$ satisfies our approximate line search conditions if

\begin{equation}
q(x_{k+1}) < q(x_k) + \sigma_l (\alpha_k g_k^T s_k + \frac{1}{2}\alpha_k^2 \min(s_k^T H s_k, 0))
\end{equation}

and

\begin{equation}
q(x_{k+1}) > q(x_k) + \sigma_u (\alpha_k g_k^T s_k + \frac{1}{2}\alpha_k^2 \min(s_k^T H s_k, 0)),
\end{equation}
Algorithm 3

Choose $x_1 \in \text{int}(\mathcal{F})$.

For $k = 1, 2, \ldots$

1. Determine an initial descent dir'n $s_k^r$ for $q$ at $x_k \in \text{int}(\mathcal{F})$. Determine the piecewise linear reflective path $p_k(\alpha)$ via Algorithm 2.

2. Perform an approximate piecewise line minimization of $q(x_k + p_k(\alpha))$, with respect to $\alpha_k$, to determine an acceptable stepsize $\alpha_k$ (such that $\alpha_k$ does not correspond to a breakpoint).

3. $x_{k+1} = x_k + p_k(\alpha_k)$.

FIG. 8. A REFLECTIVE PATH ALGORITHM

where $g_k \overset{\text{def}}{=} \nabla q(x_k)$. Note that $x_{k+1} = x_k + p_k(\alpha_k) = R(y_k + \alpha_k s_k)$ where $p_k$ is the piecewise linear path defined by (2.8). Condition (2.9) can be interpreted as restricting the step length from being too large relative to the decrease in $f$; condition (2.10) can be interpreted as restricting the step length from being relatively too small.

A basic reflective path algorithm for problem (1.1) can now be stated, Algorithm 4. To allow for flexibility, especially with regard to the Newton step, we do not always require that both (2.9) and (2.10) be satisfied. Instead, we demand that either both these conditions are satisfied or (2.9) is satisfied and $\alpha_k$ is bounded away from zero, e.g., $\alpha_k > \rho > 0$. The latter conditions are used to allow for the liberal use of Newton steps and do not weaken the global convergence results.

Note that Algorithm 4 generates strictly feasible points; i.e., since $x_1 \in \text{int}(\mathcal{F})$, it follows that $x_k \in \text{int}(\mathcal{F})$. 

13
Algorithm 4 [ $\rho$ is a positive scalar.]
Choose $x_1 \in \text{int}(\mathcal{F})$.
For $k = 1, 2, \ldots$
1. Determine an initial descent dir'n $s_k$ for $q$ at $x_k$. Note that the piecewise linear path $p_k$ is defined by $x_k, s_k$.
2. Perform an approximate piecewise line minimization of $q(x_k + p_k(\alpha))$, with respect to $\alpha$, to determine $\alpha_k$ such that:
   (a) $\alpha_k$ does not correspond to a breakpoint
   (b) condition (2.9) is satisfied
   (c) Either
      i. $\alpha_k$ satisfies condition (2.10), or
      ii. $\alpha_k > \rho > 0$
3. $x_{k+1} = x_k + p_k(\alpha_k)$.

Fig. 9. A reflective path algorithm satisfying line search conditions

3. Algorithm Specifics. A framework for our reflective Newton approach was presented in the previous section, Algorithm 4. In this section we specify more precisely how the search directions will be generated as well as the mechanics of the line search, specialized to the quadratic problem (1.1).

The convergence analysis given in [10] uses two important properties of the sequence of search directions, “constraint-compatibility” and “consistency”. “Constraint-compatibility” is needed to guarantee that a sufficiently long step is taken before the first constraint is encountered. The usual descent condition, that $g_k^T s_k$ is sufficiently negative, is not enough in the context of a reflective algorithm because this condition takes no account of the proximity of the constraints. “Consistency” is a more standard notion capturing the idea that first-order descent, represented by the term $g_k^T s_k$, be consistent with first-order convergence. Following (1.7), define $D_k^2 = D^2(x_k) = \text{diag}(|v_k|)$.

Definition 2. A sequence of vectors $\{w_k\}$ is constraint-compatible if the sequence $\{D_k^{-2}w_k\}$ is bounded.

Definition 3. A sequence of vectors $\{w_k\}$ satisfies the consistency condition if $\{w_k^T g_k\} \to 0$ implies $\{D_k g_k\} \to 0$.

Central to our approach, both in terms of achieving quadratic convergence and the satisfaction of constraint-compatibility and consistency, is the frequent use of a reduced trust region problem to determine $s_k$:

$$
\min_s \left\{ s^T g_k + \frac{1}{2} s^T M_k s : \|D_k^{-1} s\|_2 \leq \Delta_k, \ s \in S_k \right\},
$$

(3.1)

14
where $S_k$ is a subspace of $R^n$, $D_k$ is a positive diagonal scaling matrix, and $\Delta_k > 0$. The matrix $M_k$ is defined:

$$M(x) = [H + J^v D_k^2]$$

where $J^v$ is the Jacobian of $v$, where $v$ is defined in Figure 9. Matrix $D_k^2$ is a diagonal matrix with component $i$ defined $D^2_{ii} = \frac{g^T_i(x)}{|v_i(x)|}$, for $i = 1 : n$; vector $g^+(x)$ is an “extended gradient”, extended to deal with possible degeneracy. In particular,

$$g^+_i = \begin{cases} |g_i| + \tau_g & \text{if } |g_i| + |v_i|^{\frac{1}{2}} \leq \tau_g \\ |g_i| & \text{otherwise,} \end{cases}$$

where $\tau_g$ is a small positive constant. Clearly if $x$ is a nondegenerate point and $\tau_g$ is sufficiently small then $g^+ = |g|$. If both $g_i = 0$ and $v_i = 0$ (which implies that $x$ is degenerate) then $g^+_i = \tau_g > 0$.

The diagonal matrix $D(x)$, used in (3.1), is defined by (1.7), i.e.,

$$D(x) = \text{diag}(|v(x)|^{\frac{1}{2}}).$$

This choice yields a well-defined trust region problem (3.1). To see this note that using (3.4), (3.1) becomes

$$\min_{\bar{s}} \{ \bar{s}^T \bar{g}_k + \frac{1}{2} \bar{s}^T \bar{M}_k \bar{s} : \|\bar{s}\|_2 \leq \Delta_k, \ D_k \bar{s} \in S_k \}$$

where

$$\bar{M}_k = D_k M_k D_k = D_k H_k D_k + J^v_k D^2_k, \quad \bar{g}_k = D_k g_k, \quad \bar{s} = D_k^{-1} s,$$

and $D^2_k$ is a diagonal matrix, $D^2 = \text{diag}(g^+)$, Note that $\bar{M}_k$ is positive definite in a neighbourhood of a nondegenerate point satisfying the second-order sufficiency conditions. Moreover, unlike $\{M_k\}$, $\{\bar{M}_k\}$ is bounded. Matrix $\bar{M}_k$ is a featured performer in our reflective Newton algorithm. A Newton step is defined when $\bar{M}_k$ is positive definite: $s^N_k \stackrel{df}{=} D_k \bar{s}_k^N = -D_k (\bar{M}_k)^{-1} \bar{g}_k$.

A final remark on the choice of scaling matrix (3.4). If we assume that $D$ has the form $D = \text{diag}(|v(x)|^p)$, then $p = \frac{1}{2}$ is the only reasonable choice. To see this suppose $D = \text{diag}(|v(x)|^p)$ and consider that

$$DMD = DHD + J^v \text{diag} \left( \frac{g^+}{|v|^{1-2p}} \right)$$

---

7 Matrix $J^v$ is a diagonal matrix with each diagonal component equal to zero or unity. For example, if all the components of $u$ and $v$ are finite then $J^v = I$. If variable $z_i$ has a finite lower bound and an infinite upper bound (or vice-versa) then strictly speaking $v_i$ is not differentiable at a point $g_i = 0$; we define $J^v_i = 0$ at such a point. Note that $v_i$ is discontinuous at such a point but $v_i \cdot g_i$ is continuous.

8 Notation: If $z$ is a vector then $|z|^\frac{1}{2}$ denotes a vector with the $i$th component equal to $|z_i|^\frac{1}{2}$.\n
15
If $p < \frac{1}{2}$ then $1 - 2p > 0$ and the calculation of $DMD$ involves division by $|v(x)|^{1-2p}$ which includes components which go to zero as $x \rightarrow x_*$. On the other hand, if $p > \frac{1}{2}$ then $2p - 1 > 0$ and $DMD$ approaches singularity as $x \rightarrow x_*$ (consider $v^*_i = 0$).

We will specify subspace $S_k$ below; it is important to realize that the cardinality of $S_k$, $|S_k|$, satisfies $|S_k| \leq 2$ in our implementation. Therefore, the cost of solving (3.1) is negligible. Given $S_k$, the subspace trust region problem (3.5) can be approached in the following way. Let $S_k$ be defined by the $t_k$ independent columns of an $n$-by-$t_k$ matrix $V_k$, i.e., $S_k = \langle V_k \rangle$; Therefore, $s = V_k s_{V_k}$ for some vector $s_{V_k}$. Let $Y_k$ be an orthonormalization of the columns of $D_k^{-1}V_k$. Hence,

$$D_k^{-1}s = D_k^{-1}V_k s_{V_k} = Y_k s_{Y_k}$$

for some vector $s_{Y_k}$. Therefore problem (3.5) becomes

$$\min_{s_{Y_k}} \left\{ s_{Y_k}^T Y_k^T g_k + \frac{1}{2} s_{Y_k}^T \bar{M}_k Y_k s_{Y_k} : \| s_{Y_k} \|_2 \leq \Delta_k \right\}$$

and set $s_k = D_k Y_k s_{Y_k}$. The solution to (3.7) is of negligible cost once the matrices are formed, provided $|S_k|$ is small (see Appendix).

Algorithm 5 in Figure 10 presents a second-order reflective path algorithm.

**Algorithm 5**

Choose $x_1 \in \text{int}(\mathcal{F})$.

For $k = 1, 2, \ldots$,

1. $\Delta_k = \min \{ \max \{ \Delta_l, \| v_k \| \}, \Delta_u \}$

2. Determine initial descent dir'n $s_k$ for $q$ at $x_k$: If $\bar{M}_k$ is positive definite and $\| \bar{s}_k \| \leq \Delta_k$, choose $s_k = D_k \bar{s}_k$. If $\bar{M}_k$ is not positive definite choose $\Delta_k \in [\Delta_l, \Delta_u]$, choose subspace $S_k$, and solve (3.1) to get $s_k$.

3. Determine $\alpha_k$: If $s_k = s_k^N$ and $x_k + p_k(1)$ satisfies (2.9), then set $\alpha_k = 1$; otherwise, perform an approximate piecewise line minimization of $q(x_k + p_k(\alpha))$, with respect to $\alpha$, to determine $\alpha_k$ such that
   (a) $\alpha_k$ is not a breakpoint
   (b) $\alpha_k$ satisfies (2.9) and (2.10).

4. $x_{k+1} = x_k + p_k(\alpha_k)$.

**Fig. 10. A second-order reflective path algorithm**

**Note:** If $\alpha_k = 1$ is accepted by the line search but corresponds to a breakpoint, then modify $\alpha_k$:

$$\alpha_k = \tilde{\alpha}_k \overset{def}{=} 1 - \epsilon_k$$

where $\tilde{\alpha}_k$ is not a breakpoint, $\tilde{\alpha}_k$ satisfies (2.9), and $\epsilon_k < \chi_\alpha \| D_k g_k \|$ for some $\chi_\alpha > 0$.

---

9 If $A$ is a matrix then $\langle A \rangle$ denotes the space spanned by the columns of $A$. 

16
It remains to be more precise about the determination of \( s_k \) and \( S_k \) and to fully specify the line search. We begin with \( s_k \) and \( S_k \). Algorithm 6 in Figure 11 describes our procedure.

---

**Algorithm 6** [Let \( \tau_1, \tau_2, \tau_3, \epsilon_{nc} \) be positive constants.]

**Case 0:** \( \tilde{M}_k \) is positive definite and \( \|\tilde{s}_k^N\| \leq \Delta_k \).
Set \( s_k = \tilde{s}_k^N = -D_k \tilde{M}_k^{-1} \bar{g}_k = D_k \tilde{s}_k^N \).

**Case 1:** \( \tilde{M}_k \) is positive definite and \( \|\tilde{s}_k^N\| > \Delta_k \).

if \( \|r(\tilde{s}_k^N, \bar{g}_k)\| > \tau_1 \)
\[ S_k = <D_k^2 g_k, \tilde{s}_k^N >, \text{ solve (3.1) to get } s_k. \]
else
set \( s_k = -D_k^2 g_k \)
end

**Case 2:** \( \tilde{M}_k \) is not positive definite. Compute \( w_k = D_k \tilde{w}_k \), where \( \tilde{w}_k \) is a unit vector such that \( \{w_k\} \) is constraint-compatible and

\[
\tilde{w}_k^T M_k \tilde{w}_k \leq \max\{-\epsilon_{nc}, \tau_2 \lambda_{\min}(\tilde{M}_k)\}
\]

Let \( \tilde{z}_k = \frac{D_k \text{sgn}(g_k)}{\|D_k \text{sgn}(g_k)\|} \).

if \( \|r(\tilde{w}_k, \tilde{z}_k)\| < \max(\|D_k g_k\|, -\tau_3 \tilde{w}_k^T \tilde{M}_k \tilde{w}_k) \)
\[ S_k = <D_k^2 \text{sgn}(g_k) >, \text{ solve (3.1) to get } s_k. \]
else
\[ S_k = <D_k^2 \text{sgn}(g_k), D_k \tilde{w}_k >, \text{ solve (3.1) to get } s_k. \]
end

**Fig. 11. Determination of the descent direction \( s_k \)**

---

**Remark on Case 2:** We determine an appropriate negative curvature direction in the following way. If a (sparse) Cholesky factorization of \( \tilde{M}_k \) does not complete then \( \tilde{M}_k \) is not positive definite and a unit direction of non-positive curvature, \( \tilde{w}_k \), is readily available and easily computable (e.g., [17]). Algorithm 6 can make use of \( \tilde{w}_k \) provided sufficient negative curvature is displayed by \( \tilde{w}_k \), i.e.,

\[
(3.9) \quad \tilde{w}_k^T \tilde{M}_k \tilde{w}_k \leq -\epsilon_{nc}
\]

and \( \{w_k = D_k \tilde{w}_k\} \) is constraint-compatible. A constraint-compatibility test is imple-
mented by introducing a large constant, $\chi_{CP}$, and requiring,
\begin{equation}
\frac{|\nu_{ki}|}{|\nu_{ki}|} < \chi_{CP}, \quad i = 1 : n.
\end{equation}

If either condition (3.9) or condition (3.10) is not satisfied then $\tilde{w}_k$ must be rejected. In this case we can turn to a Lanczos process [10] to get a unit vector $\tilde{u}_k$ such that both (3.9) and (3.10) are satisfied. It is interesting to note that in our extensive numerical experimentation, with results reported in Section 4, conditions (3.9) and (3.10) were always satisfied by the (partial) Cholesky factorization method – the backup Lanczos procedure was never required.

**The Line Search.** We have designed a specialized approximate line search procedure to efficiently exploit the structure of this problem and to guarantee the line search conditions in Algorithm 5. Before describing the approximate procedure, we develop an exact line search procedure – this is possible because the problem is to find a local minimizer of a quadratic function along a piecewise linear path. In the end we do not use the exact line search per se but rather we use a truncated version of it, subroutine “improve”, within an overall approximate strategy. But we begin with the exact search.

**The Exact Line Search.** We are initially concerned with the exact determination of $\alpha_k^*$ where $\alpha_k^*$ is a local minimizer of $q(x_k + p_k(\alpha))$. Note: It is convenient to describe the exact line search in terms of the $y$-variables, i.e., $y_{k+1} = y_k + \alpha_k s_k$ and $x_{k+1} = R(y_{k+1}) = x_k + p_k(\alpha_k)$. With this view we have a straight-line minimization of a piecewise quadratic function $\hat{q}_k(\alpha)$:
\begin{equation}
\hat{q}_k(\alpha) = \hat{q}(y_k + \alpha s_k) = q(R(y_k + \alpha s_k)) = q(x_k + p_k(\alpha)).
\end{equation}
Henceforth in this section we omit the major iteration subscript.

The function $\hat{q}(\alpha) \triangleq \hat{q}_k(\alpha)$ is a continuous piecewise quadratic function. The ray $y + \alpha s$, $\alpha \geq 0$ can be divided into intervals from left to right, $I_1 = [\beta^0, \beta^1]$, $I_2 = [\beta^1, \beta^2]$, ..., where $\beta^0 = 0$ and $\hat{q}(\alpha)$ is smooth on each interval. Denote the restriction of $\hat{q}(\alpha)$ to the $j^{th}$ interval by $q^j(\alpha)$ – note that $q^j(\alpha)$ is a quadratic function of a single variable.

Our exact line search algorithm visits the intervals $I_1, I_2, ..., $ in a left-to-right fashion in an attempt to locate the first local minimizer of $\hat{q}_k(\alpha)$. Assume we have not located a local minimizer on intervals $I_1, I_2, ..., I_{j-1}$ and assume that $(q^j)'(\beta^{j-1}) < 0$. There are two possibilities: either $q^j(\alpha)$ has a minimizer strictly within the interval $I_j$ or it does not. If it does, i.e., $\alpha^j \in (\beta^{j-1}, \beta^{j})$ where $(q^j)'(\alpha^j) = 0$, then $(\alpha_k)_j \leftarrow \alpha^j$. However, if $q^j$ does not admit a minimizer within $I_j$ then we must consider the possibility that $\beta^{j}$ is a minimizer of $\hat{q}_k(\alpha)$. This is now the case if $(q^{j+1})'(\beta^{j}) \geq 0$. If $\alpha^j$ is not in $\text{int}(I_j)$ and $(q^{j+1})'(\beta^{j}) < 0$ then this process is repeated on interval $I_{j+1}$.

Algorithm 7 in Figure 12 presents a compact description of the exact line search algorithm we have sketched above.

**Step (1.2)** in Algorithm 7 follows from the observation that if $(\alpha^*_k = \infty \text{ and } \beta^*_k = \infty$ then $s$ is a direction of infinite descent for $\hat{q}$, beginning at $y$ and therefore, by Theorem 1, $x(\alpha) = R(y + \alpha s)$ yields infinite (feasible) descent for (1.1).
Algorithm 7 [Exact line search along direction \( s \) beginning at point \( y \)]

(0) Determine the array of breakpoints \( BR \) according to (2.7). \( \beta^0 \leftarrow 0 \).

(1) For \( k = 1, 2, \ldots \),

(1.0) \( \beta^k \leftarrow \min\{BR : BR > 0\} \).

(1.1) Determine \( \alpha_*^k \), the minimizer of \( q^k \), if it exists; otherwise, set \( \alpha_*^k = \infty \).

(1.2) If \( \alpha_*^k = \infty \) and \( \beta^k = \infty \) then exit (problem (1.1) is unbounded).

(1.3) If \( \alpha_*^k \in \text{int}(I_k) = (\beta^{k-1}, \beta^k) \) then \( \alpha_* \leftarrow \alpha_*^k \), exit.

(1.4) If \( (q^{k+1})(\beta^k) \geq 0 \) then \( \alpha_* \leftarrow \beta^k \), exit.

(1.5) If \( j \) is the index such that \( BR(j) = \beta^k \), update \( BR(j) \) according to Algorithm 2.

Fig. 12. The Exact Line Search Algorithm

Our final concern, with regard to the exact line search, is an efficient implementation of step (1.1). In theory this computation is straightforward. Assume \( q^k(\alpha) = a_0^k + a_1^k \alpha + \frac{1}{2} a_2^k \alpha^2 \). If \( a_2^k > 0 \), then \( \alpha_*^k = -\frac{a_1^k}{a_2^k} \); if \( a_2^k < 0 \), then \( q^k(\alpha) \) is unbounded below; if \( a_2^k = 0 \), then \( q^k(\alpha) \) is unbounded below (unless \( a_1^k = 0 \) in which case \( q^k \) is constant). However, the challenge is to determine \( a_1^k \) and \( a_2^k \) efficiently, for \( k = 1, 2, \ldots \). (Note that \( a_0^k \) is not required.)

The key to efficiency here is the observation that the reflective transformation \( R \) is linear on each interval \( I_k = [\beta^{k-1}, \beta^k] \), i.e., \( J^R \) is constant within each interval. For each interval \( I_k \), define \( \sigma^k \) to be the vector of diagonal elements of \( J^R \) evaluated at any point in the interval \( (y + \beta^{k-1}s, y + \beta^k s) \). Then, if \( x^k = R(y + \beta^{k-1}s) \), it follows that \( R(y + \beta^{k-1}s + \alpha s) = x^k + \alpha D_s \sigma^k \) for \( \alpha \in [0, \beta^k - \beta^{k-1}] \), where \( D_s = \text{diag}(s) \). Therefore,

\[
\tilde{q}(y + \beta^{k-1}s + \alpha s) = q(R(y + \beta^{k-1}s + \alpha s)) = q(x^k + \alpha D_s \sigma^k)
\]

for \( \alpha \in [0, \beta^k - \beta^{k-1}] \). It follows that

\[
q^k(\alpha) = q(x^k) + \alpha (\sigma^k)^T D_s \nabla_x q(x^k) + \frac{\alpha^2}{2} (\sigma^k)^T D_s H D_s \sigma^k.
\]

Therefore, in the terminology used above, \( a_0^k = q(x^k) \), \( a_1^k = (\sigma^k)^T D_s \nabla_x q(x^k) \), and \( a_2^k = (\sigma^k)^T D_s H D_s \sigma^k \).

A straightforward implementation for determining \( a_1^k, a_2^k \) requires \( O(n^2) \) work per breakpoint (\( a_0^k \) is not needed). However, there is considerable structure that can be exploited; in particular, \( \sigma^{k+1} \), a vector with each component equal to \( \pm 1 \), differs from \( \sigma^k \) in exactly one component. We can exploit this to reduce the work in the line search to \( O(n) \) per breakpoint.

Suppose we have determined that \( I_k \) does not contain \( \alpha_*^k \) and we have at hand the following quantities:

\[
w^k = [D_s H D_s] \sigma^k,
\]

(3.12)
where $D_s = \text{diag}(s)$,

$$
(3.13) \quad a_1^k = (\sigma^k)^T D_s \nabla x q(x^k),
$$

$$
(3.14) \quad a_2^k = (\sigma^k)^T D_s H D_s \sigma^k = (\sigma^k)^T w^k,
$$

and $x^k$, the value of $x$ at the $k^{th}$ breakpoint. If $j$ is the index such that $BR(j) = \beta^k$ then

$$
(3.15) \quad \sigma^{k+1} = \sigma^k - 2(\sigma^k)_j e_j.
$$

The vector $w^k$ can be updated as follows:

$$
(3.16) \quad w^{k+1} = [D_s H D_s] \sigma^{k+1},
\quad = [D_s H D_s](\sigma^k - 2(\sigma^k)_j e_j),
\quad = w^k - [D_s H D_s] 2(\sigma^k)_j e_j,
\quad = w^k - 2s_j(\sigma^k)_j D_s H_j,
$$

where $H_j$ is column $j$ of $H$. Coefficient $a_2^{k+1}$ is simply computed:

$$
(3.17) \quad a_2^{k+1} = (\sigma^{k+1})^T w^{k+1}.
$$

Coefficient $a_1^{k+1}$ can be efficiently computed by considering the following equalities:

$$
(3.18) \quad a_1^{k+1} = (\sigma^{k+1})^T [D_s (H x^{k+1} + c)],
\quad = (\sigma^{k+1})^T [D_s (H x^k + c)] + \delta_k (\sigma^{k+1})^T D_s H D_s \sigma^k,
\quad = (\sigma^{k+1})^T [D_s (H x^k + c)] + \delta_k (\sigma^{k+1})^T w^k,
\quad = a_1^k - 2(\sigma^k)_j s_j H_j^T x^k - 2(\sigma^k)_j s_j c_j + \delta_k (\sigma^{k+1})^T w^k,
$$

where $\delta_k = \beta^k - \beta^{k-1}$. Finally, $x^{k+1}$ is computed:

$$
(3.19) \quad x^{k+1} = x^k + \delta_k D_s \sigma^k.
$$

In summary, the coefficients $a_1^{k+1}, a_2^{k+1}$ and the intermediate quantities $x^{k+1}, w_{k+1}$, can be computed, given $a_1^k, a_2^k, x^k, w^k$, using (3.16),(3.17),(3.18) and (3.19). This amounts to approximately $4n$ work. Of course the initial quantities, $w^0, a_1^0, a_2^0$ must be computed from scratch requiring $O(n^2)$ work. Therefore, if $k_{br}$ denotes the number of breakpoints crossed, the total cost of the exact line search is:

(i) $O(n^2)$ for initialization of $w^0, a_1^0, a_2^0$,

(ii) $O(n)$ for determination of $BR$,

(iii) $O(k_{br}n)$ for steps (1.0) - (1.5).
The Approximate Line Search. The exact line search described above is not practical, for two reasons. First, an exact minimizer along a line may correspond exactly to a breakpoint, i.e., a boundary point, and the algorithm requires strictly feasible points. This is actually not a serious problem since a small perturbation would yield strict feasibility and the reflective Newton method is not very sensitive to boundary proximity.

A more serious objection to the exact line search is economy: despite the efficient implementation described in the previous section, the relative cost can be high if the number of breakpoints crossed, \( k_{br} \), is large. Certainly if there are a large number of tight variables at the solution, say something close to \( n \), then the total cost of the exact line search algorithm ultimately becomes \( O(n^2) \) per line search. This is unsatisfactory and unnecessary since an economical approximate line search can be just as effective.

In this section we describe an approximate line search, henceforth refer to as subroutine \( \text{improve} \), which uses the exact line search, described above, in a limited fashion—beginning at an approximate minimizer, subject to a bound, \( k_u \), on the number of breakpoints permitted to cross. In particular, \( \text{improve} \) is used in a cleanup role: after determining an initial approximate minimizer by a bisection strategy, \( \text{improve} \) is called upon to apply the exact line search strategy. Thereby the approximate minimizer is further improved at cost \( O(k_u n) \), where \( k_u \) is typically chosen to be small, e.g., \( k_u = 20 \).

(In \( \text{improve} \) we also impose an approximate upper bound \( \alpha_{\text{max}} \) on the size of \( \alpha \). That is, the size of the improvement is bounded by \( \alpha_{\text{max}} \).

Subroutine \( \text{improve} \) has the following calling sequence:

\[
\alpha = \text{improve}(y, k_u, \alpha_{\text{max}}, s)
\]

A more precise description of the approximate line search algorithm is given in Figure 13, Algorithm 8. The basic idea is as follows. First, if the direction \( s_k \) is a Newton direction \( s_k^N \) then a unit step is attempted. If (2.9) is satisfied then the full Newton step is accepted subject to further improvement by subroutine \( \text{improve} \) and possible (slight) adjustment to avoid a breakpoint. Second, if \( s_k \) does not correspond to a Newton direction or if it does but a unit step does not satisfy (2.9), then a bisection procedure is executed on the interval \((0, \alpha_u)\) where \( \alpha_u \geq 1 \) is an upper bound on the step size. A point is located satisfying both (2.9) and (2.10) and then possibly further improved with subroutine \( \text{improve} \).
Algorithm 8 [Approximate line search along direction \( s \) beginning at point \( y \)]

If \( s_k = s_k^N \) and a unit step along \( s_k \) satisfies (2.9)

- set \( \hat{\alpha}_k = 1, \alpha_{\text{max}} = \chi_\alpha \|D_k g_k\| \)
- set \( \hat{\alpha}_k = \text{improve}(y_k + \hat{\alpha}_k s_k, k_u, \alpha_{\text{max}}, s_k) \)
- if \( \hat{\alpha}_k \) satisfies (2.9), set \( \hat{\alpha}_k = \hat{\alpha}_k \)
- if \( \hat{\alpha}_k \) corresponds to a breakpoint, set \( \alpha_k = \hat{\alpha}_k - \epsilon_k \) where \( \alpha_k \) is not a breakpoint and \( 0 < \epsilon_k < \{ \chi_1 \min \{ \|D_k g_k\|, \alpha_k - \rho \} \} \); else, set \( \alpha_k = \hat{\alpha}_k \)

else \{ \( s_k \neq s_k^N \) or a unit step along \( s_k^N \) does not satisfy (2.9)\}

- use bisection to find \( \hat{\alpha}_k \in (0, \alpha_u) \) satisfying (2.9) and (2.10) such that \( \hat{\alpha}_k \) is not a breakpoint
- set \( \alpha_{\text{max}} = \alpha_u - \hat{\alpha}_k \)
- \( \hat{\alpha}_k = \text{improve}(y_k + \hat{\alpha}_k s_k, k_u, \alpha_{\text{max}}, s_k) \)
- if \( \hat{\alpha}_k \) satisfies (2.9), \( \hat{\alpha}_k = \hat{\alpha}_k \)
- if \( \hat{\alpha}_k \) corresponds to a breakpoint, set \( \alpha_k = \hat{\alpha}_k - \epsilon_k \) where \( \alpha_k \) is not a breakpoint and \( 0 < \epsilon_k < \hat{\alpha}_k - \hat{\alpha}_k \); else, set \( \alpha_k = \hat{\alpha}_k \)

end

Fig. 13. Approximate Line Search Algorithm

4. Numerical Experiments. We have implemented our algorithm in a version of Matlab which allows for sparse matrix data structures [15], now Matlab 4.0. In this section we present some preliminary numerical results.

With the exception of the results reported on Table 12, all experiments were performed on Sun Sparc workstations in the Matlab environment [22]. Experiments reported in Table 12 were performed in a heterogeneous environment involving an Intel IPSC/860 32-node multiprocessor as the “backend”, and a Sun Sparc workstation as the “frontend”. Matrix factorizations and solves were performed on the “backend”, in C, while the main Matlab program executed on the “frontend”. Communication between “frontend” and “frontend” over ethernet was implemented through the use of Matlab “MEX” files. We used this environment to facilitate the solving of very large problems. (Details on this heterogeneous environment are given in [3].)

Starting and Stopping: In all the experiments reported in this paper the starting value of \( x \), i.e., \( x_1 \), is as follows. For component \( j \) where both upper and lower bounds are finite, choose the midpoint, \( (x_1)_j = \frac{l_j + u_j}{2} \). If both upper and lower bound corresponding to component \( j \) are infinite in size, choose \( (x_1)_j = 0 \). If \( l_j \) is finite and \( u_j = \infty \), choose \( (x_1)_j = l_j + 1 \); if \( l_j = -\infty \) and \( u_j \) is finite, choose \( (x_1)_j = u_j - 1 \). (Note: The reflective Newton approach is not particularly sensitive to starting value. For example,
we repeated many of the experiments reported here using a random (strictly) feasible starting point – very little difference in behaviour was detected.)

Choosing a robust stopping rule in optimization is not easy. Our primary stopping rule is based on the relative difference in function value. This is reasonable partly because strict feasibility is always maintained, and partly because often the real objective in practical optimization is to achieve a point of relatively low function value. Specifically, our primary stopping rule is:

\[
q(x_k) - q(x_{k+1}) \leq tol \times (1 + |q(x_k)|).
\]

We choose \( tol = 100 \times \mu \) where \( \mu \) is unit roundoff (machine epsilon), i.e., in Matlab on a Sun Sparc workstation, \( \mu = 2.2204 \times 10^{-16} \). We do have secondary stopping criteria as well - designed to determine when progress is deemed too slow. This secondary rule tends to kick in when solving degenerate or ill-conditioned problems and a very flat region around the solution has been entered.

**Parameter settings:** There are a number of parameters in the algorithm: most are either in the very large or very small category. Here are the settings we used in our experiments:

- \( \tau_g \): Used in the determination of scaling matrix \( D \), see (3.3): \( \tau_g = 10^{-12} \)
- \( \sigma_l \): Used in the line search, see (2.9): \( \sigma_l = .1 \)
- \( \sigma_u \): Used in the line search, see (2.10): \( \sigma_u = .9 \).
- \( k_u \): A bound on the number of breakpoints crossed in subroutine improve: \( k_u = 20 \).
- \( \rho \): A lower bound on the stepsize, see Algorithm 8: \( \rho = .1 \).
- \( \chi_\alpha \): If the line search produces a unit step which turns out to be a breakpoint, this point is perturbed by an amount bounded by \( \chi_\alpha \|D_k g_k\| \), see (3.8): \( \chi_\alpha = 1 \).
- \( \chi_{cp} \): Used to test for constraint-compatibility, see (3.10): \( \chi_{cp} = 10^4 \).
- \( \epsilon_{nc} \): Used in the negative curvature test, see (3.9): \( \epsilon_{nc} = .0001 \).
- \( \tau_1, \tau_2, \tau_3 \): Used in Algorithm 6: \( \tau_1 = \tau_2 = \tau_3 = \mu \), where \( \mu \) is unit roundoff.
- \( \chi_1 \): Used in Algorithm 8: \( \chi_1 = 1 \).
- \( \alpha_u \): An upper bound on the bisection process used in Algorithm 8: \( \alpha_u = 1.9 \).

### 4.1. Positive Definite Problems.

We have generated a number of quadratic test problems with certain properties. In the first set of results we concentrate on the case where \( H \) is symmetric positive definite. In the results reported below we use sparse matrices \( H \) with sparsity patterns representing 3-dimensional grid using a 7-point difference scheme. The Moré/Toraldo [24] QP-generator was adapted to generate problems with a given sparsity pattern (see also [6]). We will not review the generator characteristics here: our generator is a straightforward adaptation of the Moré/Toraldo scheme to the sparse setting. We use several sparse Matlab functions (e.g., “sprandsym”, “sprand”).

In Tables 1-3, the dimension of the test problems is \( n = 1000 \) in each case. The parameter “pctbnd” indicates the percentage of variables tight at the solution – ap-
proximately evenly divided between upper and lower bounds. Parameter "deg" reflects
the degree to which the solution is (nearly) degenerate - the larger the value of "deg",
the greater the amount of (near) degeneracy. Technical details of "deg" are discussed in
[6]. Parameter "cond" reflects the conditioning of the matrix $H$: the condition number
of $H$ is approximately $10^{\text{cond}}$.

The upper and lower bound vectors, $u$ and $l$, were generated as follows. Approximately 75% of the components of $l$ were chosen to be finite and assigned the value of zero – the index assignment was made in a random fashion. Similarly, approximately 75% of the components of $u$ were chosen to be finite and assigned the value of unity. Again, the index assignment was made in a random fashion, independent of the assignment of $l$.

Each row of Tables 1-3 reflects the results of 10 independent runs with the same parameter settings. The third column, labelled "max", indicates the maximum number of iterations required, over the set of 10 independent runs, to achieve the stopping criteria; the fourth column, labelled "avg" records the average number of iterations required to reach the stopping criteria over the 10 problems; the last column, labelled "acc", records the number of digits of accuracy achieved in the function value (the true solution is known).

<table>
<thead>
<tr>
<th>deg</th>
<th>cond</th>
<th>max</th>
<th>avg</th>
<th>acc</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3</td>
<td>15</td>
<td>14</td>
<td>15</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>16</td>
<td>15.6</td>
<td>15</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>15</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>14</td>
<td>12.7</td>
<td>15</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>16</td>
<td>15.3</td>
<td>15</td>
</tr>
<tr>
<td>9</td>
<td>6</td>
<td>15</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>13</td>
<td>12.7</td>
<td>15</td>
</tr>
<tr>
<td>6</td>
<td>9</td>
<td>16</td>
<td>15.3</td>
<td>15</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>16</td>
<td>15.7</td>
<td>15</td>
</tr>
</tbody>
</table>

Observations on Tables 1-3: First, we observe the remarkable consistency of our reflective Newton method on these problems. In terms of iterations required to achieve the stopping criteria and accuracy attained in the function value, there is apparently very little sensitivity to degeneracy, conditioning, or number of variables tight at the solution. Of course we do not claim that accuracy in $x$ is independent of condition/degeneracy – it surely is not. However, it is usually acceptable in optimization to attain a point with nearly optimal function value and we have been quite successful in that (on this test collection).

Second, the absolute number of iterations required to obtain a very accurate solution (in terms of the function value $q$) is modest in every case, i.e., less than 20.
**Table 2**  
*Positive Definite Problems, pctbnd = .5, n = 1000*

<table>
<thead>
<tr>
<th>deg</th>
<th>cond</th>
<th>max</th>
<th>avg</th>
<th>acc</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3</td>
<td>15</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>17</td>
<td>17</td>
<td>15</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>17</td>
<td>16.7</td>
<td>15</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>16</td>
<td>15.3</td>
<td>15</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>18</td>
<td>17.3</td>
<td>15</td>
</tr>
<tr>
<td>9</td>
<td>6</td>
<td>17</td>
<td>17</td>
<td>15</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>15</td>
<td>14.3</td>
<td>15</td>
</tr>
<tr>
<td>6</td>
<td>9</td>
<td>17</td>
<td>17</td>
<td>15</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>17</td>
<td>16.3</td>
<td>15</td>
</tr>
</tbody>
</table>

**Table 3**  
*Positive Definite Problems, pctbnd = .9, n = 1000*

<table>
<thead>
<tr>
<th>deg</th>
<th>cond</th>
<th>max</th>
<th>avg</th>
<th>acc</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3</td>
<td>17</td>
<td>16.7</td>
<td>15</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>18</td>
<td>17.3</td>
<td>15</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>17</td>
<td>16.3</td>
<td>15</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>16</td>
<td>15.7</td>
<td>15</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>18</td>
<td>17.3</td>
<td>15</td>
</tr>
<tr>
<td>9</td>
<td>6</td>
<td>18</td>
<td>17.3</td>
<td>15</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>16</td>
<td>15.3</td>
<td>15</td>
</tr>
<tr>
<td>6</td>
<td>9</td>
<td>17</td>
<td>17</td>
<td>15</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>17</td>
<td>16.7</td>
<td>15</td>
</tr>
</tbody>
</table>
This is very encouraging considering the dimension of the problems \((n = 1000)\) and the spectrum of problem characteristics being considered.

It is important to know where the algorithm spends its time. To this end we generated larger problems, with the same structure, and we have broken down the timing information. In Table 4 we consider a representative positive definite problem with "average characteristics", i.e., "deg" = 6, "cond" = 6, "pctbnd" = .5, and vary the problem dimension \(n\). (The sparsity structure remains the same.) The second column, labelled "it", records the number of iterations required to achieve the stopping criteria; "totM" records the total number of flops used by the (partial) Cholesky factorization ("m" represents a million); "totls" records the number of flops used in the approximate line search algorithm. Over 95% of the total flop count on these problems is represented by the sum of the "totM" and "totls" columns — the remaining work in the algorithm, such as the 2-dimensional trust region solution, is negligible in comparison (see Appendix for more detail on the solution of trust region problems).

Observations on Table 4: First, there is no significant growth in number of iterations as the problem dimension \(n\) increases. High accuracy is maintained for larger values of \(n\) as well. As \(n\) increases the sparse matrix factorization work, "totM", increases relative to the lines search cost, "totls". Therefore, speedup of the (partial) sparse Cholesky factorization aspect of the algorithm (e.g., use of parallelism, exploitation of specific particular structure) will have significant impact on the overall computing time. Conversely, improving the approximate line search (in terms of cost) is not a crucial computing issue, at this point, for large-scale problems.

In addition to these randomly generated, but structured, positive definite problems, we have experimented with three specific test cases. Two of these problems are from the literature (e.g., [12, 24]) and the third example is new. In Tables 5 and 6 we report on the "obstacle" problem — in the first case there are lower bounds only, in the second case there are lower and upper bounds. In defining the specific example used we have chosen the same parameter settings and specific functions used in [24]. Table 7 reports on the elastic-plastic torsion problem. Again we used the same parameters as reported in [24] to define the problem.

In Table 8 we report on a linear spline approximation problem. This type of problem arises, for example, in a particle method approach to turbulent combustion simulation [28]. The problem results in a large sparse least-squares problem subject to
nonnegativity constraints on the variables. To set up a sample problem we assume an $m$-by-$m$-by-$m$ 3-dimensional grid. Within each cell are a set of particles randomly located (we use approximately 10 particles per cell in our experiments). Each particle $p$ has a known function value, $\phi(p)$. Associate with each grid intersection point a linear basis function and determine the best set of coefficients, $x$, for the basis functions, in the least-squares sense, subject to nonnegativity constraints on $x$. The function $\phi$ we used in our experiments is defined: given a point in 3-space, $p = (p_1, p_2, p_3)$:

$$\phi(p) = .3 \sin(9.2p_1) \sin(9.3p_2) \sin(9.4p_3).$$

<table>
<thead>
<tr>
<th>Table 5</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Obstacle Problem: Lower Bounds Only</strong></td>
</tr>
<tr>
<td>m</td>
</tr>
<tr>
<td>30</td>
</tr>
<tr>
<td>40</td>
</tr>
<tr>
<td>50</td>
</tr>
<tr>
<td>60</td>
</tr>
<tr>
<td>100</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 6</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Obstacle Problem: Lower and Upper Bounds</strong></td>
</tr>
<tr>
<td>m</td>
</tr>
<tr>
<td>30</td>
</tr>
<tr>
<td>40</td>
</tr>
<tr>
<td>50</td>
</tr>
<tr>
<td>60</td>
</tr>
<tr>
<td>100</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 7</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Elastic-plastic Torsion Problem</strong></td>
</tr>
<tr>
<td>m</td>
</tr>
<tr>
<td>30</td>
</tr>
<tr>
<td>40</td>
</tr>
<tr>
<td>50</td>
</tr>
<tr>
<td>60</td>
</tr>
<tr>
<td>100</td>
</tr>
</tbody>
</table>

**Observations on Tables 5-8.** The most noteworthy observation is the apparent insensitivity of our method to problem size for each of these problems. The number of
Table 8
Linear Spline Approximation

<table>
<thead>
<tr>
<th></th>
<th>m</th>
<th>n</th>
<th>its</th>
<th>norm</th>
</tr>
</thead>
<tbody>
<tr>
<td>18</td>
<td>5832</td>
<td>16</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>8000</td>
<td>17</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>10,648</td>
<td>17</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>13,824</td>
<td>17</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>15,625</td>
<td>16</td>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

iterations does not grow, for a given problem class, as the dimension of the problem increases. For example, for the linear spline problem, 16 iterations are required when \( n = 5832 \), 16 iterations are required when \( n = 15,625 \). Moreover, the number of iterations is always modest, on this test set, i.e., less than 20. High accuracy is achieved in all cases.

4.2. Indefinite Problems. We have adapted the Moré/Toraldo QP generation scheme, in combination with sparse matrix functions in Matlab 4.0, to generate large sparse indefinite matrices with a given sparsity pattern and given approximate set of approximate eigenvalues. In the indefinite case we chose finite upper and lower bound vectors, \( l = 0, u = 1 \). (This is to avoid the generation of unbounded problems.)

In each of the problems in Tables 9-12 roughly 10% of the eigenvalues of \( H \) are negative. The column labels are the same as before however here “acc” does not represent the number of accurate digits compared with the true solution since the true solution is unknown due to indefiniteness of \( H \). Instead, “acc” records the number of matching digits in the objective function \( q \) in the last 2 iterations. (In each case the optimality conditions were verified to hold at the final point.)

Table 9
Indefinite problems. pctbnd = .1, n = 1000

<table>
<thead>
<tr>
<th></th>
<th>deg</th>
<th>cond</th>
<th>max</th>
<th>avg</th>
<th>acc</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3</td>
<td>18</td>
<td>16.7</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>19</td>
<td>17</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>23</td>
<td>19.3</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>14</td>
<td>13.7</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>32</td>
<td>22.7</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>6</td>
<td>26</td>
<td>21.7</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>15</td>
<td>14</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>9</td>
<td>16</td>
<td>15.7</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>16</td>
<td>15.7</td>
<td>15</td>
<td></td>
</tr>
</tbody>
</table>

Observations on Tables 9-11: Iteration counts indicate that our method is not quite as consistent or efficient on indefinite problems compared to the performance on positive definite problems. Still, the overall efficiency seems very good – the average number of
Table 10
Indefinite Problems. $pctbd = .5, n = 1000$

<table>
<thead>
<tr>
<th>deg</th>
<th>cond</th>
<th>max</th>
<th>avg</th>
<th>acc</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3</td>
<td>17</td>
<td>15.7</td>
<td>15</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>19</td>
<td>18</td>
<td>15</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>18</td>
<td>16.7</td>
<td>15</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>15</td>
<td>13.3</td>
<td>15</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>18</td>
<td>17.3</td>
<td>15</td>
</tr>
<tr>
<td>9</td>
<td>6</td>
<td>19</td>
<td>17.7</td>
<td>15</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>14</td>
<td>11.3</td>
<td>15</td>
</tr>
<tr>
<td>6</td>
<td>9</td>
<td>16</td>
<td>15.7</td>
<td>15</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>25</td>
<td>18.3</td>
<td>15</td>
</tr>
</tbody>
</table>

Table 11
Indefinite problems. $pctbd = .9, n = 1000$

<table>
<thead>
<tr>
<th>deg</th>
<th>cond</th>
<th>max</th>
<th>avg</th>
<th>acc</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>3</td>
<td>16</td>
<td>13.3</td>
<td>15</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>18</td>
<td>16</td>
<td>15</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>16</td>
<td>11</td>
<td>15</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>13</td>
<td>12</td>
<td>15</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>14</td>
<td>13</td>
<td>15</td>
</tr>
<tr>
<td>9</td>
<td>6</td>
<td>16</td>
<td>14.3</td>
<td>15</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>12</td>
<td>11.3</td>
<td>15</td>
</tr>
<tr>
<td>6</td>
<td>9</td>
<td>15</td>
<td>13.7</td>
<td>15</td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>15</td>
<td>12.7</td>
<td>15</td>
</tr>
</tbody>
</table>

29
iterations required for any problem category is always less than 23.

In Table 12 we indicate where the algorithm spends its time on indefinite problems by considering a representative example and increasing the dimension.

**Table 12**

*Indefinite Problems: Timing Breakdown*

<table>
<thead>
<tr>
<th>n</th>
<th>it</th>
<th>acc</th>
<th>totM</th>
<th>totls</th>
</tr>
</thead>
<tbody>
<tr>
<td>216</td>
<td>9</td>
<td>15</td>
<td>.6m</td>
<td>.7m</td>
</tr>
<tr>
<td>512</td>
<td>12</td>
<td>15</td>
<td>7.4m</td>
<td>2.9m</td>
</tr>
<tr>
<td>1000</td>
<td>10</td>
<td>16</td>
<td>20.5m</td>
<td>4.9m</td>
</tr>
<tr>
<td>1728</td>
<td>17</td>
<td>16</td>
<td>96.5m</td>
<td>11.1m</td>
</tr>
<tr>
<td>2744</td>
<td>14</td>
<td>15</td>
<td>347m</td>
<td>19.4m</td>
</tr>
</tbody>
</table>

**Remark on Table 12:** We see no apparent growth in required iterations as n increases. Clearly the “totM” column dominates the “totls” column as n increases. Recall that “totM” represents the matrix factorization flop count while “totls” represents the number of total flops required by the line search procedure. Therefore, to obtain further improvements in efficiency for this type of approach it is best to focus on the matrix factorization aspect of the overall procedure.

5. **Theory and Conclusions.** The numerical results obtained to date strongly support the notion that a reflective Newton method represents an efficient way to accurately locate local minimizers of large-scale (indefinite) quadratic functions subject to bounds on some of the variables. The theory is supporting also: our reflective Newton method is globally and quadratically convergent. Coleman and Li [10] present important theoretical properties of reflective Newton methods for general nonlinear functions, subject to bounds on some of the variables. The method in this paper is a specialization of the general method to the quadratic case. Therefore, the general theory applies.

We make a compactness assumption before formally stating the main result.

---

**Compactness Assumption:** Given initial point $x_1 \in \mathcal{F}$, it is assumed that the level set $\mathcal{L} = \{x : x \in \mathcal{F} \text{ and } q(x) \leq q(x_1)\}$ is compact.

---

**Theorem 2.** Let $\{x_k\}$ be generated by Algorithm 5 with $\{s_k\}$ generated by Algorithm 6 and with $\{\alpha_k\}$ determined by the approximate line search algorithm (Algorithm 7). If $\tau_3 < \frac{1}{5 \rho_M}$, then\(^\text{10}\),

- Every limit point of $\{x_k\}$ is a first-order point.

\(^{10}\rho_M is the maximum spectral radius of $\tilde{M}(x)$ on $\mathcal{L} = \{x : x \in \mathcal{F} \text{ and } q(x) \leq q(x_1)\}$. Since $\rho(\tilde{M}(x))$ is continuous on $\mathcal{L}$, a compact set, the upper bound $\rho_M$ exists. Recall that $\tau_3$ is a constant used in Algorithm 6.
• Every nondegenerate limit point satisfies the second-order necessary conditions.
• If a nondegenerate limit point $x_*$ satisfies second-order sufficiency conditions then, provided $\tau_g$ is sufficiently small, $\{x_k\}$ is convergent to $x_*$; the convergence rate is quadratic, i.e.,

$$\|x_{k+1} - x_*\| = O(\|x_k - x_*\|^2).$$

Proof. This algorithm is in the class of algorithm described in [10] and all the assumptions of Theorem 20 in [10] are satisfied. The result follows from Theorem 20 in [10].

In conclusion, strong theoretical and computational results indicate that a reflective Newton method is an efficient and reliable way to solve problem (1.1) to high accuracy. The computational results reported in this paper support this claim.

6. Acknowledgement. We thank our colleagues Jianguo Liu and Danny Ralph for many helpful discussions on this work. Danny Ralph drew our attention to the topology reference [25].
7. Appendix: The trust region problem. The trust region problem is

\[ \min_s \{ g^T s + \frac{1}{2} s^T A s : \|s\| \leq \Delta \}, \]

where \( A \) is a real symmetric matrix and \( \| \cdot \| \) denotes the 2-norm. The purpose of this section is to review the nature of problem (7.1) and discuss a possible solution suitable for low-dimensional problems. (In the context of our reflective Newton method for problem (1.1), \( A \) is matrix \( \tilde{M}(x) \), a symmetric matrix of order 2. The computational cost of the procedure we describe to solve (7.1) is negligible compared to the other required computations in the reflective Newton algorithm we propose.) For larger problems a more approximate procedure is usually preferred, e.g., [14, 23, 30, 31]. Much of the material in this section can be found elsewhere, e.g., [2, 4, 11, 14, 23, 29, 30, 31].

**Diagonalization.** We begin with an extremely useful characterization of the global solution to (7.1).

**Theorem 3.** Vector \( s \) solves (7.1) if and only if there exists a scalar \( \lambda \geq 0 \) such that

- (a) \( (A + \lambda I)s = -g \),
- (b) \( (A + \lambda I) \) is positive semidefinite,
- (c) \( \|s\| \leq \Delta \),
- (d) \( (\|s\| - \Delta)\lambda = 0 \).

**Proof.** A proof is given, for example, in [30]. \( \blacksquare \)

The usefulness of this result is best revealed after diagonalization. Suppose \( A = V \Lambda V^T \) where the columns of \( V \) are the orthonormal eigenvectors of \( A \) and

\[ \Lambda = \text{diag}(\lambda_1, ..., \lambda_n), \quad \lambda_1 \leq \lambda_2 \leq ... \leq \lambda_n. \]

Obviously then \( A + \lambda I = V(\Lambda + \lambda I)V^T \) and so (a) is equivalent to

\[ (\Lambda + \lambda I)\bar{s} = \alpha \overset{\text{def}}{=} -V^T g, \quad s \leftarrow V\bar{s}. \]

By (b), \( \lambda \geq -\lambda_1 \), and so all vectors \( s \) satisfying (a, b) are of the form

\[ s = \sum_{\{i : \lambda_i + \lambda > 0\}} (\frac{\alpha_i}{\lambda_i + \lambda})v_i + \sum_{\{i : \lambda_i + \lambda = 0\}} \beta_i v_i. \]

The vector \( \beta \) is arbitrary with respect to (a, b) but can sometimes help with respect to satisfying (c, d). A basis for an algorithmic approach to this problem is to assume the form given in (7.4) and strive to satisfy (c, d) by choosing \( \lambda \), and in some cases \( \beta \), appropriately. \( \beta \) plays a role only if \( \lambda_* = -\lambda_1 \), where \( \lambda_* \) is the value of \( \lambda \) at the solution.

The situation where (c, d) can be satisfied with \( \|s\| < \Delta \) and \( \lambda = 0 \) is easily dispensed with (first half of Case 1 below). Therefore the primary task, assuming form (7.4), is to determine \( \lambda \), and sometimes \( \beta \), to satisfy

\[ \|s\| - \Delta = 0. \]
We divide our approach into three possibilities. Define \( I = \{ i : \lambda_i = \lambda_1 \} \).

**Case 1:** \( \lambda_1 > 0 \). In this case either the Newton step is within the sphere \( \| s \|_2 \leq \Delta \) or it is not. If \( \| A^{-1} g \| \leq \Delta \) then the optimal solution \( s_\ast = -A^{-1} g, \lambda_\ast = 0 \). Otherwise, for \( \lambda \geq 0 \) define \( s(\lambda) = \sum_{i=1}^{n} (\frac{\alpha_i}{\lambda_i + \lambda}) v_i \).

**Case 2:** \( \lambda_1 \leq 0 \) and for some \( i \in I, \alpha_i \neq 0 \). For \( \lambda > -\lambda_1 \), define \( s(\lambda) = \sum_{i \in I} (\frac{\alpha_i}{\lambda_i + \lambda}) v_i \). Figure 2 illustrates \( \| s(\lambda) \| \) for \( \lambda \) around \( -\lambda_1 \). Obviously \( \| s(\lambda) \| \to \infty \) as \( \lambda \to -\lambda_1^+ \), and \( \| s(\lambda) \| \to 0 \) as \( \lambda \to \infty \). Moreover, \( \| s(\lambda) \| \) is convex; therefore, \( \| s(\lambda) \| \) intersects \( \Delta \) in exactly one place for \( \lambda > -\lambda_1 \).

**Case 3:** \( \lambda_1 \leq 0 \) and \( \forall i \in I, \alpha_i = 0 \). Let \( s_1(\lambda) = \sum_{i \in I} (\frac{\alpha_i}{\lambda_i + \lambda}) v_i \). Clearly \( \| s_1(-\lambda_1) \| \) is finite. Consider figures 3.4. There are now two possibilities. If \( \| s_1(-\lambda_1) \| \geq \Delta \) then there is a solution to (7.5) to the right of \( -\lambda_1 \), \( s(\lambda) = s_1(\lambda) \). Otherwise, \( s = s_1(-\lambda_1) + \sum_{i \in I} \beta_i v_i \) with \( \beta \) chosen to ensure (7.5), and \( \lambda_\ast = -\lambda_1 \). Note that if \( |I| > 1 \) then the null space component of \( s \) may not be unique.

**The Reciprocal Secular Equation.** In theory we can build an algorithm on the remarks given above. However it is better, numerically, to replace condition (7.5) with

\[
(7.6) \quad \mathrm{rsec}(\lambda) \overset{\text{def}}{=} \frac{1}{\Delta} - \frac{1}{\| s(\lambda) \|} = 0.
\]

Equation (7.6) is more linear in shape than equation (7.5); therefore, equation (7.6) is more amenable to solution via Newton’s method.

Considering the definition of \( \mathrm{rsec}(\lambda) \),

\[
(7.7) \quad \mathrm{rsec}(\lambda) = \frac{1}{\Delta} - \frac{1}{\left[ (\frac{\alpha_1}{\lambda_1 + \lambda})^2 + (\frac{\alpha_2}{\lambda_2 + \lambda})^2 + \ldots + (\frac{\alpha_n}{\lambda_n + \lambda})^2 \right]^{1/2}},
\]

it is easy to verify the following:

1. \( \mathrm{rsec}(\lambda) \) is convex on \( (-\lambda_1, \infty) \) and \( \lim_{\lambda \to \infty} \mathrm{rsec}(\lambda) = -\infty \).
2. \( \lim_{\lambda \to -\lambda_1^+} \mathrm{rsec}(\lambda) \) is finite.
3. If \( \alpha_i \neq 0 \), for some \( i \in I \), then \( \mathrm{rsec}(-\lambda_1) \overset{\text{def}}{=} \lim_{\lambda \to -\lambda_1^+} \mathrm{rsec}(\lambda) = \frac{1}{\Delta} \). Obviously a single zero of \( \mathrm{rsec} \) exits to the right of \( -\lambda_1 \) in this case.
4. If \( \forall i \in I, \alpha_i = 0 \), then \( \mathrm{rsec}(-\lambda_1) \overset{\text{def}}{=} \lim_{\lambda \to -\lambda_1^+} \mathrm{rsec}(\lambda) = \frac{1}{\Delta} - \frac{1}{(\sum_{i \in I} (\frac{\alpha_i}{\lambda_i - \lambda_1})^2)^{1/2}} \).

**Algorithmic and Numerical Concerns.** We assume that a solution to (7.1) is sought and we are willing and able to compute full eigenvalue information, \( A = V \Lambda V^T \). (If this is not the case, perhaps due to the cost, then it is possible to approximately solve (7.6) using an iterative scheme involving the Cholesky factorization of \( A + \lambda I \), [14, 23].)
The method using $A = VΛV^T$ appears to be straightforward.

**Case 1:** $λ_1 > 0$. If $\|A^{-1}g\| \leq Δ$ then the Newton step, $-A^{-1}g$, is the solution with $λ_* = 0$. If $\|A^{-1}g\| > Δ$ then we can determine the zero of $rsec(λ)$, $λ > 0$.

**Case 2:** $λ_1 \leq 0$ and for some $i \in I, α_i \neq 0$. Then $\lim_{λ \rightarrow -λ_1^+} rsec(λ) = \frac{1}{Δ}$ and $rsec$ admits a single solution to the right of $-λ_1$.

**Case 3:** $λ_1 \leq 0$ and $∀i \in I, α_i = 0$. Let $s = s_1(λ) + βv_1$ where $s_1(λ) = \sum_{i∈I} (α_i / λ_i^2) v_i$. If $\|s_1(-λ_1)\| ≥ Δ$ then there is a zero of $rsec(λ)$ to the right of $-λ_1$ with $s(λ) = s_1(λ)$. Otherwise, a solution to (7.1) is given by (7.4), $s = s_1(-λ_1) + \sum_{i∈I} β_i v_i$, with $\sum β_i^2 = Δ^2 - \|s_1(-λ_1)\|^2$, and $λ_* = -λ_1$.

Unfortunately, the situation is not quite so clean from a numerical point of view: there is fuzziness between the second and third cases. In particular, if $α_1$ is small the equation $rsec(λ) = 0$ is very ill-conditioned for $λ$ near $-λ_1$ and it can be quite difficult (nigh impossible) to compute $λ$ such that $rsec(λ)$ is small. This extreme ill-conditioning is due to the following “disagreement”.

Assume $I = \{1\}$, for simplicity, and note that, if $α_1 = 0$,

$$
(7.8) \quad rsec(-λ_1) \overset{def}{=} \lim_{λ \rightarrow -λ_1^+} rsec(λ) = \frac{1}{Δ} - \frac{1}{\left[\left(\frac{α_1^2}{λ_2^2 - λ_1^2}\right)^2 + \cdots + \left(\frac{α_n^2}{λ_n^2 - λ_1^2}\right)^2\right]^{\frac{1}{2}}}.
$$

On the other hand, if $α_1 \neq 0$, $rsec(-λ_1) \overset{def}{=} \lim_{λ \rightarrow -λ_1^+} rsec(λ) = \frac{1}{Δ}$ which is not, in general, the limiting value of (7.8). Therefore nearby problems ($α_1 = 0$ vs. $α_1 = ε$) can yield very different solutions to (7.6), and this is the cause of the ill-conditioning of (7.6). Our solution to this ill-conditioning problem (trust.m) is to first attempt to find a solution to $rsec(λ) = 0, λ ≥ -λ_1$. However, if $|rsec(λ)|$ is not small, where $\tilde{λ}$ is the computed “zero” returned by the zero-finder, then we set $λ_* = -λ_1$ and compute a solution to (7.1) via (7.4).

This strategy works because the solution to (7.6) with $α_i$ small for $i \in I$ is close to a solution of (7.1) with the corresponding $α_i$ at zero. To see this, initially assume that $I = \{1\}$; that is, $λ_1 < λ_2$.

Now first consider the case where $α_1 = 0$. Then the solution to (7.1) is:

$$
(7.9) \quad s = \sum_{i>1} \left(\frac{α_i}{λ_i - λ_1}\right) v_i + β v_1,
$$

where $β^2 = Δ^2 - \sum_{i>1} (\frac{α_i}{λ_i - λ_1})^2$ (we assume $Δ^2 < \sum_{i>1} (\frac{α_i}{λ_i - λ_1})^2$; otherwise, the result is obvious). Define $s_1 = \sum_{i>1} (\frac{α_i}{λ_i - λ_1}) v_i$; hence, the solution to (7.1) can be written $s = s_1 + β v_1$.

Next consider $α_1 = ε$, a small number. We can write the solution to (7.1) as

$$
(7.10) \quad s(λ) = \sum_{i>1} \left(\frac{α_i}{λ_i + λ}\right) v_i + \left(\frac{α_1}{λ_1 + λ}\right) v_1.
$$
But as $\alpha_1 \to 0$, $\lambda \to -\lambda_1$ (to keep $\|s(\lambda)\| = \Delta$). But as $\lambda \to -\lambda_1$, $\sum_{i>1} \left( \frac{\alpha_i}{\lambda_i + \lambda} \right) v_i \to s_1$, which implies

$$ (\frac{\alpha_1}{\lambda_1 + \lambda})^2 \to \beta^2, \text{ as } \alpha_1 \to 0. $$

(7.11)

Therefore, the solution to (7.1) with $\alpha_1 = \epsilon$ is near to the solution to (7.1) with $\alpha_1 = 0$.

In general, if $|I| > 1$ a solution to (7.1), with some components $\alpha_i$ near to zero, is near to a solution of (7.1) with those components set to zero. In this case, where several coefficients $\alpha_i$ equal to zero, $i \in I$, problem (7.1) does not enjoy a unique solution [see (7.4)] but the range space component is unique.
REFERENCES


[23] J. J. Moré and D. Sorensen, Computing a trust region step, SIAM Journal on Scientific and


