Summer 2007

On the Use of Quasi-Newton Methods for the Minimization of Convex Quadratic Splines

William Howard Thomas II
Old Dominion University

ON THE USE OF QUASI-NEWTON METHODS FOR THE MINIMIZATION OF CONVEX QUADRATIC SPLINES

by

William Howard Thomas II
B.S. May 1983, Northeast Louisiana University
M.S. June 1992, Naval Postgraduate School

A Dissertation Submitted to the Faculty of
Old Dominion University in Partial Fulfillment of the
Requirement for the Degree of

DOCTOR OF PHILOSOPHY

COMPUTATIONAL AND APPLIED MATHEMATICS

OLD DOMINION UNIVERSITY
August 2007

Approved by:

John J. Swetits (Director)

Wu Li (Member)

Hideaki Kaneko (Member)

Przemek Bogacki (Member)

ABSTRACT

ON THE USE OF QUASI-NEWTON METHODS FOR THE MINIMIZATION OF CONVEX QUADRATIC SPLINES

William Howard Thomas II
Old Dominion University, 2007
Director: Dr. John J. Swetits

In reformulating a strictly convex quadratic program with simple bound constraints as the unconstrained minimization of a strictly convex quadratic spline, established algorithms can be implemented with relaxed differentiability conditions. In this work, the positive definite secant update method of Broyden, Fletcher, Goldfarb, and Shanno (BFGS) is investigated as a tool to solve the unconstrained minimization problem. It is shown that there is a linear convergence rate and, for nondegenerate problems, the process terminates in a finite number of iterations. Numerical examples are provided.

ACKNOWLEDGMENTS

I would like to thank my wife, Katy, for her support and Dr. John Swetits for his patience and guidance.

Soli Deo Gloria.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

INTRODUCTION

Background*

A very common mathematical problem involves finding an extremum (i.e., maximum or minimum) of a given objective function. Since we can recast maximization problems as minimization problems, we will focus on minimization problems without loss of generality.

In simplest form, a quadratic programming problem takes the form

$$\min\left\{\frac{1}{2}x^TAx - b^Tx\right\}$$

where $A \in \mathbf{R}^{n\times n}$ (i.e., A is an $n \times n$ real matrix) and $b, x \in \mathbf{R}^n$ (real vectors of n components). We adopt the convention that a vector x is a column vector and x^T denotes a row vector. When A is symmetric and positive semidefinite, then the objective function $\frac{1}{2}x^TAx - b^Tx$ is convex. For A symmetric and positive definite, the objective function is strictly convex.

At this point, there are no restrictions or constraints on the independent variable. These problems are called *unconstrained minimization problems.* We make this distinction because the introduction of constraints on the independent variable often has a significant impact on our ability to solve the problem. For example, a constrained quadratic programming problem may take the form

$$\min\left\{\frac{1}{2}x^TAx - b^Tx\right\} \text{ subject to } \ell \leq Mx \leq u$$

---

* The model journal for this dissertation is *SIAM Journal on Optimization.*

where $A \in \mathbf{R}^{n \times n}$ is a symmetric positive definite matrix; $M \in \mathbf{R}^{m \times n}$; $b, x \in \mathbf{R}^n$; and $\ell, u \in \mathbf{R}^m$.

There are many techniques for solving the unconstrained problem. Methods for solving the general quadratic programming problem have been broadly classified as finite or iterative (see [28,24] and references therein). Among the finite methods, i.e., those that terminate using a finite number of arithmetic operations, Pang [28] identified four families of algorithms including methods based on simplicial pivoting, active set methods, simplicial decomposition methods, and methods based on shrinking ellipsoids.

In a later survey, Lin and Pang [24] focused on iterative methods which generate an infinite sequence converging to a limit point solution. Starting from Hildreth's procedure [17], the survey describes the evolution of iterative methods including the successive overrelaxation methods, matrix splitting techniques, and a Lagrangian relaxation algorithm presented in [6]. More recently, Wright [36] and Nesterov [27] addressed interior-point methods which generated a great deal of activity beginning with Karmarkar's paper [18].

By reformulating a constrained problem as an unconstrained problem, the task of solving the constrained problem is only as difficult as solving the corresponding unconstrained problem and, if necessary, interpreting the solution in terms of the original problem. For certain classes of constrained problems, techniques have been developed that yield unconstrained minimization problems having additional

structure that can be exploited. We will examine one of these classes under a particularly elegant reformulation.

Specifically, we are interested in quadratic programming problems of the form

$$\min\left\{\frac{1}{2}x^T Ax - b^T x\right\}$$
$$\text{subject to } \ell \leq x \leq u \tag{1.1}$$

where A is an $n \times n$ positive definite symmetric matrix; x,b are n-dimensional real vectors; and ℓ,u are n-dimensional real vectors with $\ell_i \leq u_i$. Some components of ℓ or u may be $\pm\infty$.

This is a constrained minimization problem when ℓ or u have any finite components. The transformation referenced above is based on the Karush-Kuhn-Tucker optimality conditions. The result is fascinating in that the constrained problem (1.1) becomes an unconstrained problem with a richly structured objective function, namely, a quadratic spline. A function Φ on $\mathbf{R}^n$ is a quadratic spline if and only if Φ is differentiable and there are finitely many convex polyhedral subsets $\{W_i\}_{i=1}^r$ such that $\bigcup_{i=1}^r W_i = \mathbf{R}^n$, and Φ is a quadratic function on each W_i. We will address these polyhedral subsets at some length in the next section.

Not only can we apply iterative solution techniques to this unconstrained problem, but we can customize the algorithm to exploit the structure of the spline itself. This is the essence of work done by Li and Swetits [22,23]. In turning their attention to this new problem

$$\min\{\Phi(x)\} \tag{1.2}$$

where $\Phi(x)$ is a quadratic spline, Li and Swetits developed an algorithm using Newton's method to find the minimizer. A subsequent paper by Li [21] uses an algorithm based on the conjugate gradient method. The present work investigates the use of the positive definite secant update (BFGS) method independently derived by Broyden [3], Fletcher [12], Goldfarb [16], and Shanno [32] to solve (1.2) .

Simply Bound Quadratic Programs

We now turn to some of the details of reformulating a quadratic program with simply bound constraints into an unconstrained minimization problem of a quadratic spline. Throughout this work let $\|\cdot\|$ denote the ℓ_2 vector norm or its induced matrix norm unless otherwise specified.

Li and Swetits demonstrated in [22] and [23] that the reformulation of the quadratic program (1.1) stems from the Karush-Kuhn-Tucker conditions. As outlined in [21], x^* is a solution of (1.1) if and only if there exists $w^* \in \mathbf{R}^n$ satisfying

$$w^* = Ax^* - b,$$

$$x_i^* = \ell_i \text{ for } w_i^* > 0,$$

$$x_i^* = u_i \text{ for } w_i^* < 0,$$

$$\ell_i \leq x_i^* \leq u_i \text{ for } w_i^* = 0.$$

We will also adopt the definition of nondegeneracy given in [21]. Specifically, a solution x^* of (1.1) is said to be nondegenerate if and only if

$$\begin{aligned} x_i^* &= \ell_i \text{ for } (Ax^* - b)_i > 0, \\ x_i^* &= u_i \text{ for } (Ax^* - b)_i < 0, \\ \ell_i &< x_i^* < u_i \text{ for } (Ax^* - b)_i = 0. \end{aligned} \tag{1.3}$$

Based on these conditions, [23] shows that x^* is a solution of (1.1) if and only if

$$x^* = (x^* - \alpha w^*)_l^u \tag{1.4}$$

where $w^* = Ax^* - b$, α is any positive constant, and $(v)_l^u$ is the vector whose i-th component is $\max\{\min\{v_i, u_i\}, l_i\}$. For our purposes, we substitute for w^* directly into (1.4) obtaining

$$x^* = (x^* - \alpha(Ax^* - b))_l^u = ((I - \alpha A)x^* + \alpha b)_l^u.$$

Letting $E := I - \alpha A$ and $h := \alpha b$, then x^* is a solution to (1.1) if and only if x^* satisfies the piecewise linear equation

$$x = (Ex + h)_l^u. \tag{1.5}$$

Restricting α to the interval $0 < \alpha < 1/\|A\|$, where $\|A\|$ is the spectral radius of A, then [21] and [23] give the explicit forms with which we will work. Namely, that

$$\nabla\Phi(x) = E(x - (Ex + h)_l^u) \tag{1.6}$$

is the gradient of the following strictly convex quadratic spline

$$\begin{aligned}\Phi(x) = \frac{1}{2}x^T(E - E^2)x - x^T Eh + \frac{1}{2}\|(\ell - (Ex + h))_+\|^2 \\ + \frac{1}{2}\|((Ex + h) - u)_+\|^2.\end{aligned} \tag{1.7}$$

Here $(v)_+$ is the vector whose i-th component is $\max\{v_i, 0\}$. Strict convexity follows from the positive definiteness of A. This restriction on α also guarantees that the

eigenvalues of the symmetric positive definite matrix E are contained in the interval $(0, 1]$. With this explicit representation of Φ, we can directly address the closed convex polyhedral subsets $\{W_i\}_{i=1}^r$ mentioned earlier.

The polyhedral subsets arise from the final terms $\|(\ell - (Ex + h))_+\|^2$ and $\|((Ex + h) - u)_+\|^2$ in (1.7). By definition,

$$\left[(\ell - (Ex + h))_+\right]_i = \begin{cases} (\ell - (Ex + h))_i, & \text{if } (\ell - (Ex + h))_i > 0; \\ 0, & \text{otherwise} \end{cases} \tag{1.8}$$

and

$$\left[((Ex + h) - u)_+\right]_i = \begin{cases} ((Ex + h) - u)_i, & \text{if } ((Ex + h) - u)_i > 0; \\ 0, & \text{otherwise.} \end{cases} \tag{1.9}$$

As x changes, $(Ex + h)_i$ changes, and so the corresponding components (1.8) and (1.9) will change as x moves across the hyperplanes defined by

$$(\ell - (Ex + h))_i = 0 \tag{1.10}$$

or

$$((Ex + h) - u)_i = 0. \tag{1.11}$$

These hyperplanes are the boundaries of the polyhedrons formed by the intersection of the associated half-spaces. Thus, the polyhedrons are convex sets. Letting E_i denote the i^{th} row of the matrix E, we can rewrite (1.10) as

$$E_i x = \ell_i - h_i. \tag{1.12}$$

Similarly, we can rewrite (1.11) as

$$E_i x = u_i - h_i. \tag{1.13}$$

Here, E_i is the normal vector to the hyperplane. Since E is positive definite, the rows of E are linearly independent. Thus $\mathbf{R}^n = \text{span}\{E_1, \ldots, E_n\}$. From this we can conclude that for $-\infty < \ell_i < u_i < \infty$, (1.12) and (1.13) give two distinct and parallel hyperplanes. In the dimension represented by E_i, the space $\mathbf{R}^n$ is partitioned into three subsets.

If for each $i = 1, \ldots, n$, we have $-\infty < \ell_i < u_i < \infty$, then (1.12) and (1.13) give n pairs of distinct parallel hyperplanes. It follows that the maximum number of polyhedrons defined by these hyperplanes is 3^n.

If, for some $1 \leq i \leq n$, one component of the pair ℓ_i, u_i is infinite (i.e., $\ell_i = -\infty$ or $u_i = \infty$) and the other is finite, then only one of the hyperplanes (1.12) or (1.13) will partition $\mathbf{R}^n$ into two subsets. Essentially, the hyperplane associated with the infinite component exists at ∞. For m such pairs, the maximum number of polyhedrons defined is $3^{n-m}2^m$.

When $\ell_i = -\infty$ and $u_i = \infty$ for some $1 \leq i \leq n$, then there is no partitioning of $\mathbf{R}^n$ in the dimension represented by E_i. For p such pairs, the maximum number of polyhedrons is $3^{n-m-p}2^m$.

Finally, when $\ell_i = u_i$ for some $1 \leq i \leq n$, we note that this component of the solution vector x^* is fixed and the minimization problem exists in $\mathbf{R}^{n-1}$. In this sense, it is not unreasonable to require that $\ell_i \neq u_i$. With this condition, each of the polyhedrons has a nonempty interior. A constructive argument is provided below.

Suppose that $-\infty < \ell_i < u_i < \infty$ for $i = 1, \ldots, n$. Since the hyperplanes (1.12) and (1.13) are distinct and parallel, we can define three new parallel and distinct hyperplanes, $E_i x = u_i - h_i + 1$, $E_i x = \frac{1}{2}(u_i + \ell_i) - h_i$, and $E_i x = \ell_i - h_i - 1$. Each of these new hyperplanes exists entirely within a subset of $\mathbf{R}^n$ as partitioned by (1.12) and (1.13) . Letting i range from 1 to n, we obtain 3^n systems of equations in the form

$$Ex = v \tag{1.14}$$

where $v_i \in \{u_i - h_i + 1, \frac{1}{2}(u_i + \ell_i) - h_i, \ell_i - h_i - 1\}$. Since E is symmetric positive definite, then E is invertible. Thus $x = E^{-1}v$ generates 3^n points, each of which lies in the interior of some $\{W_i\}_{i=1}^{3^n}$.

At this point, we note that $\nabla^2\Phi(x)$ does not exist uniquely for all points on the quadratic spline Φ. In the collection of closed convex polyhedral subsets $\{W_i\}_{i=1}^{r}$ described earlier, $\Phi(x)$ is a quadratic function on each W_i and $\nabla^2\Phi(x)$ exists uniquely for each x in the interior of W_i. In fact, the explicit form of $\nabla^2\Phi(x)$ is straightforward when it exists.

From (1.6) we have

$$\nabla\Phi(x) = Ex - E(Ex + h)_\ell^u. \tag{1.15}$$

By definition,

$$[(Ex + h)_\ell^u]_i = \begin{cases} \ell_i, & \text{if } (Ex + h)_i \leq \ell_i; \\ (Ex + h)_i, & \text{if } \ell_i < (Ex + h)_i < u_i; \\ u_i, & \text{if } (Ex + h)_i \geq u_i. \end{cases} \tag{1.16}$$

Since the polyhdral boundaries emerge from the relationship between $(Ex + h)_i$ and ℓ_i or $(Ex + h)_i$ and u_i, we will find it convenient to keep track of where

$(Ex+h)_i$ lies in relation to ℓ_i and u_i. This will allow us to detect movement across polyhedral boundaries. Thus, we define the vector

$$\xi_i(x) := \begin{cases} -1, & \text{if } (Ex+h)_i \leq \ell_i; \\ 0, & \text{if } \ell_i < (Ex+h)_i < u_i; \\ 1, & \text{if } (Ex+h)_i \geq u_i. \end{cases} \tag{1.17}$$

The constant components of $(Ex+h)_\ell^u$ contribute nothing to $\nabla^2\Phi(x)$, so we define

$$\sigma_{ii}(x) := \begin{cases} 1, & \text{if } \ell_i < (Ex+h)_i < u_i; \\ 0, & \text{if } (Ex+h)_i \leq \ell_i \text{ or } (Ex+h)_i \geq u_i \end{cases} \tag{1.18}$$

and

$$\sigma(x) = \operatorname{diag}\left(\sigma_{ii}(x)\right). \tag{1.19}$$

Then

$$\nabla^2\Phi(x) = E - E\sigma(x)E. \tag{1.20}$$

However, for x on the boundary of W_i, i.e., when $(Ex+h)_m = \ell_m$ or $(Ex+h)_m = u_m$ for some m, then x is also on the boundary of W_j where $i \neq j$. Thus, $\nabla^2\Phi(x)$ may not exist because it is not uniquely defined for x on the boundary of W_i.

Example

A relatively simple example in $\mathbf{R}^2$ will help to clarify the previous discussion. Consider the following simply bound quadratic programming problem in the form of (1.1):

$$\begin{aligned} &\min\{\frac{1}{2}x^TAx - b^Tx\} \\ &\text{subject to } \ell \leq x \leq u \end{aligned} \tag{1.21}$$

where

$$A = \begin{bmatrix} 4 & 2 \\ 2 & 5 \end{bmatrix}, b = \begin{bmatrix} 3 \\ 1 \end{bmatrix}, \ell = \begin{bmatrix} 2 \\ -1 \end{bmatrix}, \text{ and } u = \begin{bmatrix} 3 \\ 2 \end{bmatrix}.$$

The surface $F(x_1, x_2) = \frac{1}{2}x^TAx - b^Tx$ is plotted in Figure 1.

and states of materials. In 1963, spectrochemical analysis of surfaces using lasers was first presented, followed by the observation of optical induced breakdown in a gas in 1964. Then, during the 1970s, development continued in several directions. In 1972, Felske et al. described the analysis of steel by means of a Q-switched laser. In the early 1980s, there was a renewed interest in spectrochemical applications of LIBS, driven by its unique advantages and applications in different media. Important applications were the detection of hazardous gases and vapors in air and small amounts of beryllium in air or on filters. A repetitively pulsed Nd:YAG laser at 1.06 μm was used to excite effluent gases from an experimental fixed-bed coal gasifier. Although alkalis at the parts per billion levels were not detected, the major constituents, including sulfur, were easily seen and quantified. Liquids were analyzed either by excitation at the surface or in the volume. Solutions of ten different elements were analyzed and atomic and ionic uranium spectra were seen by exciting a flowing solution of uranium in nitric acid. Uranium could not be detected by focusing into the liquid, only through focusing on the liquid-air interface. As a progressing technique, LIBS was used by Poulain and Alexander [64] to measure the salt concentration in seawater aerosol droplets. Then, Aguilera [65] applied LIBS to determine carbon content in molten and solid steel, while the elemental analysis of aluminum alloy targets was studied by Sabsabi and Cielo. [66]. During the 1990s, the applications turned to very practical problems, such as monitoring environmental contamination, control of materials processing, and sorting of materials to put them in proper scrap bins. More concentrated work was directed to develop a rugged, moveable instrumentation. Optical fibers were built into LIBS systems, primarily for carrying the spark light to the spectrometer and occasionally for the delivery of the laser pulse as well.

convex polyhedral subsets into which $\mathbf{R}^2$ is partitioned. Namely,

$$W_1 = \{x \in \mathbf{R}^2 : E_1 x + h_1 \geq u_1\} \cap \{x \in \mathbf{R}^2 : E_2 x + h_2 \geq u_2\}$$

$$W_2 = \{x \in \mathbf{R}^2 : E_1 x + h_1 \geq \ell_1\} \cap \{x \in \mathbf{R}^2 : E_1 x + h_1 \leq u_1\} \cap \{x \in \mathbf{R}^2 : E_2 x + h_2 \geq u_2\}$$

$$W_3 = \{x \in \mathbf{R}^2 : E_1 x + h_1 \leq \ell_1\} \cap \{x \in \mathbf{R}^2 : E_2 x + h_2 \geq u_2\}$$

$$W_4 = \{x \in \mathbf{R}^2 : E_1 x + h_1 \geq u_1\} \cap \{x \in \mathbf{R}^2 : E_2 x + h_2 \geq \ell_2\} \cap \{x \in \mathbf{R}^2 : E_2 x + h_2 \leq u_2\}$$

$$W_5 = \{x \in \mathbf{R}^2 : E_1 x + h_1 \geq \ell_1\} \cap \{x \in \mathbf{R}^2 : E_1 x + h_1 \leq u_1\} \cap \{x \in \mathbf{R}^2 : E_2 x + h_2 \geq \ell_2\} \cap \{x \in \mathbf{R}^2 : E_2 x + h_2 \leq u_2\}$$

$$W_6 = \{x \in \mathbf{R}^2 : E_1 x + h_1 \leq \ell_1\} \cap \{x \in \mathbf{R}^2 : E_2 x + h_2 \geq \ell_2\} \cap \{x \in \mathbf{R}^2 : E_2 x + h_2 \leq u_2\}$$

$$W_7 = \{x \in \mathbf{R}^2 : E_1 x + h_1 \geq u_1\} \cap \{x \in \mathbf{R}^2 : E_2 x + h_2 \leq \ell_2\}$$

$$W_8 = \{x \in \mathbf{R}^2 : E_1 x + h_1 \geq \ell_1\} \cap \{x \in \mathbf{R}^2 : E_1 x + h_1 \leq u_1\} \cap \{x \in \mathbf{R}^2 : E_2 x + h_2 \leq \ell_2\}$$

$$W_9 = \{x \in \mathbf{R}^2 : E_1 x + h_1 \leq \ell_1\} \cap \{x \in \mathbf{R}^2 : E_2 x + h_2 \leq \ell_2\}$$

Figure 2 shows the bounding hyperplanes (i.e., lines) and Figure 3 shows the resulting polyhedral subsets.

tool for instantaneous, multi-elemental analysis of any kind of sample, solid, liquid, or gas.

1.2 Advantages and disadvantages of LIBS

Comparing the spectroscopic techniques in terms of their analytical figure of merits, simplicity, cost, and applications, we conclude some of the distinguishing advantages of LIBS such as:

1. Minimum or no sample preparation results in a reduction of time-consuming procedures.
2. Both conducting and non-conducting materials can be tested.
3. Very small amounts of sample (0. 1 pg to 0. 1 mg) are vaporized.
4. Hard materials that can be difficult to get into solution can be analyzed (e.g. ceramics, glasses, and superconductors).
5. Multiple elements can be determined simultaneously.
6. The direct determination of aerosols or ambient air is possible.
7. The analysis is simple, rapid and produces no waste.
8. Remote sensing is possible with the use of fiber optics.
9. Samples can be analyzed in a hostile environment.
10. Underwater analysis is possible.

In addition, LIBS has some drawbacks such as:

1. Current systems are expensive and complex.
2. Obtaining suitable matrix-matched standards is difficult.
3. Interference (matrix) effects can be large.

Obtaining the 3^2 corresponding interior points from (1.14) involves solving the following systems of equations:

$$\begin{cases} E_1x + h_1 = u_1 + 1 \\ E_2x + h_2 = u_2 + 1 \end{cases} \Rightarrow \begin{cases} \frac{1}{2}x_1 - \frac{1}{4}x_2 = \frac{29}{8} \\ -\frac{1}{4}x_1 + \frac{3}{8}x_2 = \frac{23}{8} \end{cases} \Rightarrow \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 133/8 \\ 75/4 \end{bmatrix} \in W_1$$

$$\begin{cases} E_1x + h_1 = \frac{1}{2}(\ell_1 + u_1) \\ E_2x + h_2 = u_2 + 1 \end{cases} \Rightarrow \begin{cases} \frac{1}{2}x_1 - \frac{1}{4}x_2 = \frac{17}{8} \\ -\frac{1}{4}x_1 + \frac{3}{8}x_2 = \frac{23}{8} \end{cases} \Rightarrow \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 97/8 \\ 63/4 \end{bmatrix} \in W_2$$

$$\begin{cases} E_1x + h_1 = \ell_1 - 1 \\ E_2x + h_2 = u_2 + 1 \end{cases} \Rightarrow \begin{cases} \frac{1}{2}x_1 - \frac{1}{4}x_2 = \frac{5}{8} \\ -\frac{1}{4}x_1 + \frac{3}{8}x_2 = \frac{23}{8} \end{cases} \Rightarrow \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 61/8 \\ 51/4 \end{bmatrix} \in W_3$$

$$\begin{cases} E_1x + h_1 = u_1 + 1 \\ E_2x + h_2 = \frac{1}{2}(\ell_2 + u_2) \end{cases} \Rightarrow \begin{cases} \frac{1}{2}x_1 - \frac{1}{4}x_2 = \frac{29}{8} \\ -\frac{1}{4}x_1 + \frac{3}{8}x_2 = \frac{3}{8} \end{cases} \Rightarrow \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 93/8 \\ 35/4 \end{bmatrix} \in W_4$$

$$\begin{cases} E_1x + h_1 = \frac{1}{2}(\ell_1 + u_1) \\ E_2x + h_2 = \frac{1}{2}(\ell_2 + u_2) \end{cases} \Rightarrow \begin{cases} \frac{1}{2}x_1 - \frac{1}{4}x_2 = \frac{17}{8} \\ -\frac{1}{4}x_1 + \frac{3}{8}x_2 = \frac{3}{8} \end{cases} \Rightarrow \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 57/8 \\ 23/4 \end{bmatrix} \in W_5$$

$$\begin{cases} E_1x + h_1 = \ell_1 - 1 \\ E_2x + h_2 = \frac{1}{2}(\ell_2 + u_2) \end{cases} \Rightarrow \begin{cases} \frac{1}{2}x_1 - \frac{1}{4}x_2 = \frac{5}{8} \\ -\frac{1}{4}x_1 + \frac{3}{8}x_2 = \frac{3}{8} \end{cases} \Rightarrow \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 21/8 \\ 11/4 \end{bmatrix} \in W_6$$

$$\begin{cases} E_1x + h_1 = u_1 + 1 \\ E_2x + h_2 = \ell_2 - 1 \end{cases} \Rightarrow \begin{cases} \frac{1}{2}x_1 - \frac{1}{4}x_2 = \frac{29}{8} \\ -\frac{1}{4}x_1 + \frac{3}{8}x_2 = -\frac{17}{8} \end{cases} \Rightarrow \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 53/8 \\ -5/4 \end{bmatrix} \in W_7$$

$$\begin{cases} E_1x + h_1 = \frac{1}{2}(\ell_1 + u_1) \\ E_2x + h_2 = \ell_2 - 1 \end{cases} \Rightarrow \begin{cases} \frac{1}{2}x_1 - \frac{1}{4}x_2 = \frac{17}{8} \\ -\frac{1}{4}x_1 + \frac{3}{8}x_2 = -\frac{17}{8} \end{cases} \Rightarrow \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 17/8 \\ -17/4 \end{bmatrix} \in W_8$$

$$\begin{cases} E_1x + h_1 = \ell_1 - 1 \\ E_2x + h_2 = \ell_2 - 1 \end{cases} \Rightarrow \begin{cases} \frac{1}{2}x_1 - \frac{1}{4}x_2 = \frac{5}{8} \\ -\frac{1}{4}x_1 + \frac{3}{8}x_2 = -\frac{17}{8} \end{cases} \Rightarrow \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} -19/8 \\ -29/4 \end{bmatrix} \in W_9.$$

For this example, $\Phi(x)$ can be given explicitly. On W_1,

$$\Phi(x) = \frac{1}{4}x_1^2 - \frac{1}{4}x_1x_2 + \frac{3}{16}x_2^2 - x_1 + \frac{333}{64}.$$

On W_2,

$$\Phi(x) = \frac{1}{8}x_1^2 - \frac{1}{8}x_1x_2 + \frac{5}{32}x_2^2 + \frac{5}{16}x_1 - \frac{21}{32}x_2 + \frac{225}{128}.$$

On W_3,

$$\Phi(x) = \frac{1}{4}x_1^2 - \frac{1}{4}x_1x_2 + \frac{3}{16}x_2^2 - \frac{1}{2}x_1 - \frac{1}{4}x_2 + \frac{197}{64}.$$

On W_4,

$$\Phi(x) = \frac{7}{32}x_1^2 - \frac{5}{32}x_1x_2 + \frac{15}{128}x_2^2 - \frac{47}{32}x_1 + \frac{45}{64}x_2 + \frac{441}{128}.$$

On W_5,

$$\Phi(x) = \frac{3}{32}x_1^2 - \frac{1}{32}x_1x_2 + \frac{11}{128}x_2^2 - \frac{5}{32}x_1 + \frac{3}{64}x_2.$$

On W_6,

$$\Phi(x) = \frac{7}{32}x_1^2 - \frac{5}{32}x_1x_2 + \frac{15}{128}x_2^2 - \frac{31}{32}x_1 + \frac{29}{64}x_2 + \frac{169}{128}.$$

On W_7,

$$\Phi(x) = \frac{1}{4}x_1^2 - \frac{1}{4}x_1x_2 + \frac{3}{16}x_2^2 - \frac{7}{4}x_1 + \frac{9}{8}x_2 + \frac{261}{64}.$$

On W_8,

$$\Phi(x) = \frac{1}{8}x_1^2 - \frac{1}{8}x_1x_2 + \frac{5}{32}x_2^2 - \frac{7}{16}x_1 + \frac{15}{32}x_2 + \frac{81}{128}.$$

On W_9,

$$\Phi(x) = \frac{1}{4}x_1^2 - \frac{1}{4}x_1x_2 + \frac{3}{16}x_2^2 - \frac{5}{4}x_1 + \frac{7}{8}x_2 + \frac{125}{64}.$$

Figure 4 shows a contour plot of $\Phi(x)$ overlaid with the bounding hyperplanes.

References of Chapter I

[1] N. Omenetto, In "*Analytical laser spectroscopy*," Wiley Interscience, New York, (1979).

[2] J. A. Broekaert, In "*Analytical atomic spectrometry with flames and plasmas*," Wiley-VCH, Germany, (2002).

[3] M. Cullin, In "*Atomic spectroscopy in elemental analysis*," Blackwell Publishing Ltd, UK, (2004).

[4] J. Robinson, In "*Atomic spectroscopy*," Marcel Dekker Inc., New York, (1996).

[5] R. K. Marcus, In "*Glow discharge spectroscopy*," Plenum Press, New York, (1993).

[6] G. R. Kirchoff and R. Bunsen, "Chemical analysis by spectrum observation," Philos. Mag. **20**, 89-98 (1860).

[7] S. J. Weeks, H. Haraguchi, and J. D. Winefordner, "Improvement of detection limits in laser-excited atomic fluorescence flame spectrometry," Anal. Chem. **50**, 360-68 (1978).

[8] S. Sjostrom and P. Mauchien, "Laser atomic spectroscopic techniques-The analytical performance for trace element analysis of solid and liquid samples," Spectrochim. Acta B. **15**, 153-180 (1991).

[9] F. Capitilli, F. Colao, M. R. Provenzano, R. Fantoni, G. Brunetti, and N. Sensi, "Detection of heavy metals in soils by Laser Induced Breakdown Spectroscopy," Geoderma **106**, 45-62 (2002).

ALGORITHM

Newton Methods

To properly understand quasi-Newton methods, we should first understand Newton's method for minimizing a function of several variables. Following the development of Bazaraa, Sherali, and Shetty [1, p. 308], let $f : \mathbf{R}^n \to \mathbf{R}$ be a continuously twice-differentiable function with a local minimum at $x^* \in \mathbf{R}^n$. Consider the Taylor's series representation of f in the vicinity of a point $x_k \in \mathbf{R}^n$

$$f(x) = f(x_k) + \nabla f(x_k)^T(x - x_k) + \frac{1}{2}(x - x_k)^T \nabla^2 f(x_k)(x - x_k) + o\left(\|x - x_k\|^2\right)$$

where $\nabla f(x_k)$ is the gradient of f evaluated at x_k and $\nabla^2 f(x_k)$ is the Hessian of f evaluated at x_k. A quadratic model of f in the vicinity of x_k can be extracted from the Taylor's series as

$$f_m(x) = f(x_k) + \nabla f(x_k)^T(x - x_k) + \frac{1}{2}(x - x_k)^T \nabla^2 f(x_k)(x - x_k).$$

In searching for the minimizer of f, it seems reasonable to begin with $\overline{x} \in \mathbf{R}^n$, the minimizer of f_m, where necessarily $\nabla f_m(\overline{x}) = 0$. Thus

$$\nabla f(x_k) + \nabla^2 f(x_k)(\overline{x} - x_k) = 0.$$

If the inverse $\left(\nabla^2 f(x_k)\right)^{-1}$ exists, then a bit of algebra yields

$$\overline{x} = x_k - \left(\nabla^2 f(x_k)\right)^{-1} \nabla f(x_k). \tag{2.1}$$

The point $\overline{x}$ that minimizes f_m should better approximate x^* than x_k. Repeating this process, we generate a sequence of iterates that converge to x^*. Thus, as a recursion formula, we write (2.1) as

$$x_{k+1} = x_k - \left(\nabla^2 f(x_k)\right)^{-1} \nabla f(x_k). \tag{2.2}$$

If $\nabla f(x^*) = 0$ and that $\nabla^2 f(x^*)$ is positive definite, the sequence is well-defined for x_k close enough to x^*.

For a general function f, close enough is defined as the neighborhood about x^* for which the Hessian matrix is positive definite when evaluated at a point in the neighborhood. If $\nabla^2 f(x_k)$ is singular at some point x_k then the process cannot generate x_{k+1}.

Quasi-Newton Methods

In moving from x_k to x_{k+1} in (2.2), the direction of movement is a deflection of the steepest descent direction, $-\nabla f(x_k)$, by $\left(\nabla^2 f(x_k)\right)^{-1}$. If instead we deflect the steepest descent direction by a symmetric positive definite matrix D_k approximating $\left(\nabla^2 f(x_k)\right)^{-1}$ in each step, then the procedure is generally classified as a quasi-Newton method. Requiring each D_k to be positive definite ensures that the resulting search direction $s_k = -D_k \nabla f(x_k)$ is a descent direction whenever $\nabla f(x_k) \neq 0$.

An important part of any quasi-Newton algorithm is the generation of a symmetric positive definite matrix D_k as an approximation to the inverse of the Hessian matrix. From the original DFP method presented by Davidon [8] and later refined by Fletcher and Powell [13], quasi-Newton algorithms evolved until the BFGS algorithm was independently derived in 1970 by Broyden [3], Fletcher [12], Goldfarb [16], and Shanno [32]. Subsequent study has placed the BFGS method as a special case of the Broyden family of parameterized updating schemes [1, p. 325].

Implementation

Recall that we began this discussion assuming f to be continuously twice-differentiable. From the problem definition leading to (1.2), we know that we cannot make this assumption for the strictly convex quadratic spline Φ. While Φ is continuously twice-differentiable on the interior of each polyhedron, the algorithm must be able to accommodate changes in the Hessian as the search moves into other polyhedrons. Our implementation of the BFGS algorithm includes a restarting mechanism for polyhedron changes. This is necessary for the proof of linear convergence and finite termination.

We begin with an initial point x_1 and an initial symmetric positive definite matrix $B_1 = I$. For $k > 1$,

$$\text{Solve } B_k s_k = -\nabla\Phi(x_k) \text{ for } s_k. \tag{2.3}$$

$$\text{Solve } s_k^T \nabla\Phi(x_k + \lambda_k s_k) = 0 \text{ for } \lambda_k > 0. \tag{2.4}$$

$$\text{Set } x_{k+1} := x_k + \lambda_k s_k \text{ and } u_k := \lambda_k s_k. \tag{2.5}$$

$$\text{Set } y_k := \nabla\Phi(x_{k+1}) - \nabla\Phi(x_k). \tag{2.6}$$

If x_{k+1} belongs to the same polyhedral set as x_k then we update B_k to B_{k+1} according to

$$B_{k+1} := B_k + \frac{y_k y_k^T}{y_k^T u_k} - \frac{B_k u_k u_k^T B_k}{u_k^T B_k u_k}, \tag{2.7}$$

otherwise,we restart by setting

$$B_{k+1} := I.$$

Here B_k is approximating $\nabla^2\Phi(x_k)$ in each step. Recalling (1.17), we can detect whether or not x_{k+1} and x_k are in the same polyhedral subset by comparing $\xi(x_{k+1})$ to $\xi(x_k)$.

On a given polyhedron, the Hessian $\nabla^2\Phi$ is unique and constant. The update (2.7) should generate a better approximation to this Hessian in each subsequent iteration. However, when the search for the minimizer moves into a new polyhedron, the Hessian may change. As mentioned in the previous section, if the search moves to a boundary, i.e., $(Ex_k + h)_i = \ell_i$ or $= u_i$ for some $i = 1, \ldots, n$, then x_k belongs to two or more polyhedrons. Thus $\nabla^2\Phi(x_k)$ may not exist because of a lack of uniqueness. With the restart feature of (2.7) this presents no difficulty. In moving to this boundary, the i-th component of $\xi(x_k)$ will differ from that of $\xi(x_{k-1})$ triggering a restart, $B_{k+1} = I$. If $k = 1$, then having set $B_1 = I$ creates the same scenario. In either case, the search direction s_k as determined in (2.3) becomes a steepest descent step, $s_k = -\nabla\phi(x_k)$. Since the gradient is continuous, this direction will be the same regardless of which polyhedral representation of the gradient is used.

CONVERGENCE

In this section, we establish the global convergence of the BFGS algorithm acting on a strictly convex quadratic spline. Since the spline is strictly convex, it is coercive and bounded below. Consequently, by the Frank-Wolfe theorem [7, p. 114, 14], the quadratic spline has and attains a unique minimizer. In order to show linear convergence to the spline minimizer, we will need a theorem due to Wolfe as given in Dennis and Schnabel [11, p. 121 Copyright © 1996 Society for Industrial and Applied Mathematics. Reprinted with permission].

THEOREM 3.1 (Wolfe [35]) *Let* $f : \mathbf{R}^n \to \mathbf{R}$ *be continuously differentiable on* $\mathbf{R}^n$, *and assume there exists* $\gamma \geq 0$ *such that*

$$\|\nabla f(z) - \nabla f(x)\|_2 \leq \gamma \|z - x\|_2$$

for every $x, z \in \mathbf{R}^n$. *Then, given any* $x_0 \in \mathbf{R}^n$, *either* f *is unbounded below, or there exists a sequence* $\{x_k\}$, $k = 0, 1, \ldots$, *obeying*

$$f(x_k + \lambda s_k) \leq f(x_k) + \alpha \lambda s_k^T \nabla f(x_k) \text{ for } \alpha \in (0, 1), \lambda > 0, \tag{3.1}$$

$$s_k^T \nabla f(x_k + \lambda_k s_k) \geq \beta s_k^T \nabla f(x_k) \text{ for } \beta \in (\alpha, 1), \tag{3.2}$$

where s_k *is the descent direction, and either*

$$\nabla f(x_k)^T s_k < 0, \tag{3.3}$$

or

$$\nabla f(x_k) = 0 \text{ and } s_k = 0, \tag{3.4}$$

for each $k \geq 0$, *where*

$$s_k := x_{k+1} - x_k. \tag{3.5}$$

Furthermore, for any such sequence, either

$$\nabla f(x_k) = 0 \text{ for some } k \geq 0, \text{ or}$$

$$\lim_{k \to \infty} f(x_k) = -\infty, \text{ or}$$

$$\lim_{k \to \infty} \frac{\nabla f(x_k)^T s_k}{\|s_k\|_2} = 0.$$

In applying this theorem to the quadratic spline, we address first the conditions on Φ. The spline $\Phi : \mathbf{R}^n \to \mathbf{R}$ is continuously differentiable since the gradient, $\nabla\Phi$, as given in (1.6), is continuous. Dennis and Schnabel [11, p. 123] note that while the theorem assumes that Φ is Lipschitz on all $\mathbf{R}^n$, this condition is necessary only in a neighborhood of the solution x^*. The convexity of Φ guarantees this as shown in [31].

Conditions (3.1) and (3.2) are imposed to ensure the adequacy of an inexact line search. Since our implementation of the BFGS algorithm uses an exact line search, these conditions are satisfied in every iteration.

It is well known [11, pp. 199–201] that if the initial approximating matrix is positive definite, then the sequence of matrices $\{B_k\}$ generated by the BFGS method are positive definite. As noted in the previous section, the BFGS algorithm generates search directions, s_k, from

$$B_k s_k = -\nabla\Phi(x_k) \tag{3.6}$$

where B_k approximates the Hessian in each iteration. Hence, (3.6) implies

$$0 < s_k^T B_k s_k = -s_k^T \nabla\Phi(x_k)$$

which gives

$$s_k^T \nabla\Phi(x_k) < 0$$

confirming s_k as a descent direction and satisfying (3.3).

We say that a matrix A *dominates* a matrix G if $A - G$ is symmetric positive semidefinite, i.e., $z^T(A-G)z \geq 0$ for all $z \in \mathbf{R}^n$.

LEMMA 3.2 *Let $A \in \mathbf{R}^{n\times n}$ be a symmetric positive definite matrix. Let $G \in \mathbf{R}^{n\times n}$ be a symmetric positive definite matrix with eigenvalues $0 < \lambda_n^G \leq \ldots \leq \lambda_1^G < \infty$. If A dominates G, then $z^T A z \geq z^T \lambda_n^G z$ for all $z \in \mathbf{R}^n$.*

Proof. If A dominates G, then $z^T(A-G)z \geq 0$ for all $z \in \mathbf{R}^n$. Thus $z^T A z \geq z^T G z \geq z^T \lambda_n^G z$.

LEMMA 3.3 *On any given polyhedron with Hessian $\nabla^2\Phi$, let B_1 be the initial approximation of the Hessian on that subregion. If B_1 dominates $\nabla^2\Phi$, then each of the matrices B_k for $k > 1$ generated by the BFGS method dominates $\nabla^2\Phi$.*

Proof. Let $\nabla^2\Phi$ be the Hessian on a given polyhedron. Then $\nabla^2\Phi$ is an $n \times n$ symmetric positive definite matrix and the gradient $\nabla\Phi$ has the form $\nabla\Phi(x) = (\nabla^2\Phi)x - b$. Let $B_1 \in \mathbf{R}^{n\times n}$ be symmetric positive definite and let x_1 be some point in $\mathbf{R}^n$.

From (2.4) we have

$$\begin{aligned} 0 &= s_k^T \nabla\Phi(x_k + \lambda_k s_k) \\ &= s_k^T \left[\left(\nabla^2\Phi\right)(x_k + \lambda_k s_k) - b\right] \\ &= s_k^T \left[\left(\nabla^2\Phi\right) x_k - b\right] + \lambda_k s_k^T \left(\nabla^2\Phi\right) s_k \\ &= s_k^T \nabla\Phi\left(x_k\right) + \lambda_k s_k^T \left(\nabla^2\Phi\right) s_k. \end{aligned}$$

Thus

$$-s_k^T \nabla\Phi\left(x_k\right) = \lambda_k s_k^T \left(\nabla^2\Phi\right) s_k$$

which by (2.3) becomes

$$s_k^T B_k s_k = \lambda_k s_k^T \left(\nabla^2\Phi\right) s_k$$

or, equivalently, by (2.5)

$$u_k^T B_k u_k = \lambda_k u_k^T \left(\nabla^2\Phi\right) u_k. \tag{3.7}$$

The collection $\{u_1, \ldots, u_k\}$ is $\nabla^2\Phi$-conjugate. Since $\nabla^2\Phi$ is positive definite, we can extend to a $\nabla^2\Phi$-conjugate basis $\{u_1, \ldots, u_k, \ldots, u_n\}$.

On the given polyhedron,

$$\begin{aligned} y_k &= \nabla\Phi\left(x_{k+1}\right) - \nabla\Phi\left(x_k\right) \\ &= \left(\nabla^2\Phi\right) x_{k+1} - \left(\nabla^2\Phi\right) x_k \\ &= \left(\nabla^2\Phi\right)\left(x_{k+1} - x_k\right) \\ &= \left(\nabla^2\Phi\right) u_k \end{aligned}$$

so that (2.7) can be written as

$$B_{k+1} = B_k + \frac{(\nabla^2\Phi)u_k u_k^T(\nabla^2\Phi)}{u_k^T(\nabla^2\Phi)u_k} - \frac{B_k u_k u_k^T B_k}{u_k^T B_k u_k} \tag{3.8}$$

giving

$$B_{k+1}u_k = (\nabla^2\Phi)u_k.$$

Therefore, for $i \neq k$,

$$u_i^T B_{k+1} u_k = 0 = u_i^T(\nabla^2\Phi)u_k. \tag{3.9}$$

Let $z \in \mathbf{R}^n$. Then $z = \sum_{i=1}^n c_i u_i \equiv w + c_k u_k$ and by (3.9) $w^T B_{k+1} u_k = 0$ and $w^T(\nabla^2\Phi)u_k = 0$. This allows us to write

$$\begin{aligned} z^T B_{k+1} z =& w^T B_{k+1} w + c_k^2 u_k^T B_{k+1} u_k \\ =& w^T B_k w - \frac{\left(w^T B_k u_k\right)^2}{u_k^T B_k u_k} + c_k^2 u_k^T(\nabla^2\Phi)u_k \\ =& w^T\left(B_k - \nabla^2\Phi\right) w - \frac{\left(w^T\left(B_k - \nabla^2\Phi\right) u_k\right)^2}{u_k^T B_k u_k} + z^T(\nabla^2\Phi)z. \end{aligned} \tag{3.10}$$

Now suppose that $B_k - \nabla^2\Phi$ is positive semi-definite. Then

$$u_k^T \left(B_k - \nabla^2\Phi\right) u_k \geq 0$$

so that

$$\frac{u_k^T B_k u_k}{u_k^T \left(\nabla^2\Phi\right) u_k} \geq 1.$$

We conclude from (3.7) that $\lambda_k \geq 1$. By the Cauchy-Schwarz inequality and (3.7),

$$\begin{aligned}
\frac{\left(w^T \left(B_k - \nabla^2\Phi\right) u_k\right)^2}{u_k^T B_k u_k} \leq & \frac{\left(w^T \left(B_k - \nabla^2\Phi\right) w\right) \left(u_k^T \left(B_k - \nabla^2\Phi\right) u_k\right)}{u_k^T B_k u_k} \\
= & \frac{\left(w^T \left(B_k - \nabla^2\Phi\right) w\right) \left(\left(\lambda_k - 1\right) u_k^T (\nabla^2\Phi) u_k\right)}{\lambda_k u_k^T (\nabla^2\Phi) u_k} \\
= & \frac{\lambda_k - 1}{\lambda_k} \left(w^T \left(B_k - \nabla^2\Phi\right) w\right) \\
\leq & w^T \left(B_k - \nabla^2\Phi\right) w.
\end{aligned}$$

Thus from (3.10),

$$z^T B_{k+1} z \geq z^T (\nabla^2\Phi) z.$$

Getting an acceptable upper bound is easier. For this part there is no need to assume $B_k - \nabla^2\Phi$ is positive semi-definite. Since a quasi-Newton method terminates in at most $n+1$ iterations on a quadratic function, we have

$$\begin{aligned}
z^T B_{k+1} z = & z^T B_k z + \frac{\left(z^T \nabla^2\Phi u_k\right)^2}{u_k^T \nabla^2\Phi u_k} - \frac{\left(z^T B_k u_k\right)^2}{u_k^T B_k u_k} \\
\leq & z^T B_k z + \frac{\left(z^T \nabla^2\Phi u_k\right)^2}{u_k^T (\nabla^2\Phi) u_k} \\
\leq & z^T B_k z + \frac{\left(z^T \nabla^2\Phi z\right) \left(u_k^T \nabla^2\Phi u_k\right)}{u_k^T (\nabla^2\Phi) u_k} \\
= & z^T B_k z + z^T (\nabla^2\Phi) z \\
\leq & z^T B_1 z + k z^T (\nabla^2\Phi) z \\
\leq & z^T \left(B_1 + n\nabla^2\Phi\right) z.
\end{aligned}$$

This completes the proof.

We now demonstrate that the sequence of minimal eigenvalues is bounded away from zero and that the sequence of maximal eigenvalues is bounded above.

LEMMA 3.4 *Let $x_1 \in \mathbf{R}^n$ be given and let $B_1 = I$. Let $\{B_k\}$ be the sequence of matrices generated by the BFGS algorithm as given in (2.3) – (2.7). Then $\{\mu_k\}$, the set of smallest eigenvalues of $\{B_k\}$, is bounded below away from zero, and $\{\nu_k\}$, the set of largest eigenvalues of $\{B_k\}$, is bounded above.*

Proof. For each polyhedron W_i, the Hessian is given by (1.20). Recall that the matrix E was given as $E = I - \alpha A$ where $0 < \alpha < 1/\|A\|$. Let $\xi_n \leq \ldots \leq \xi_1$ be the n real positive eigenvalues of A. Then

$$0 < \xi_n x^T x \leq x^T A x \leq \xi_1 x^T x$$

$$0 < \alpha\xi_n x^T x \leq x^T \alpha A x \leq \alpha\xi_1 x^T x < x^T x$$

$$-x^T x < -\alpha\xi_1 x^T x \leq -x^T \alpha A x \leq -\alpha\xi_n x^T x < 0$$

$$0 < (1 - \alpha\xi_1) x^T x \leq x^T (I - \alpha A) x \leq (1 - \alpha\xi_n) x^T x < x^T x$$

$$0 < x^T E x < x^T x,$$

so the eigenvalues of E are contained in the interval $(0, 1)$. By similar argument, the eigenvalues of $\nabla^2\Phi(x) = E(I - \sigma(x)E)$ are also contained in the interval $(0, 1)$. Therefore, $B_1 = I$ dominates $\nabla^2\Phi$ on W_i, and by Lemma 3.3, each of the matrices B_k generated by the algorithm dominates $\nabla^2\Phi$ for x_k remaining in W_i. Thus for $k \geq 1$,

$$x^T B_k x \geq x^T \left(\nabla^2\Phi\right) x \geq \lambda_n^i x^T x \tag{3.11}$$

for every $x \in W_i$ where λ_n^i is the smallest eigenvalue of $\nabla^2\Phi$. Therefore λ_n^i is a lower bound for $\{\mu_k\}$ generated on the polyhedron W_i.

By Lemma 3.3,

$$x^T B_k x \leq x^T \left(B_1 + n\nabla^2\Phi\right) x \leq \left(\nu_1 + n\lambda_1^i\right) x^T x \tag{3.12}$$

for every $x \in W_i$ where λ_1^i is the largest eigenvalue of $\nabla^2\Phi$. Therefore $\nu_1 + n\lambda_1^i = 1 + n\lambda_1^i$ is an upper bound for $\{\nu_k\}$ generated on the polyhedron W_i.

Recall that the algorithm restarts by setting the initial approximating matrix to the identity whenever x moves into a different polyhedron. Thus (3.11) and (3.12) establish lower and upper bounds on each W_i. Since there are a finite number of convex polyhedral subsets $\{W_i\}_{i=1}^r$, then $\mu := \min_{1\leq i\leq r}\{\lambda_n^i\}$ is a uniform lower bound for the sequence $\{\mu_k\}$. Since μ is the smallest eigenvalue for the Hessian of some W_i, then $\mu > 0$. Likewise $\nu := \max_{1\leq i\leq r}\{1 + n\lambda_1^i\}$ is a uniform upper bound for the sequence ν_k. From this we can conclude that the condition numbers of $\{B_k\}$ are bounded above. This completes the proof.

LEMMA 3.5 *For $k \geq 1$ and $\nabla\Phi(x_k) \neq 0$, the quantity $\nabla\Phi(x_k)^T s_k$ is bounded away from zero.*

Proof. Suppose that B_k as defined in Lemma 3.4 has eigenvalues $\xi_n \leq \ldots \leq \xi_1$ which are bounded as so that

$$\mu \leq \xi_n \leq \ldots \leq \xi_1 < \nu.$$

Then B_k^{-1} has eigenvalues $\xi_1^{-1} \leq \ldots \leq \xi_n^{-1}$ which are bounded by

$$\nu^{-1} < \xi_1^{-1} \leq \ldots \leq \xi_n^{-1} \leq \mu^{-1}.$$

Rewriting (3.6) as

$$s_k = -B_k^{-1}\nabla\Phi(x_k),$$

we have

$$\nabla\Phi(x_k)^T s_k = -\nabla\Phi(x_k)^T \left(B_k^{-1}\right)\nabla\Phi(x_k).$$

Multiplying through by -1 gives

$$-\nabla\Phi(x_k)^T s_k = \nabla\Phi(x_k)B_k^{-1}\nabla\Phi(x_k)$$

so that

$$\nu^{-1}\|\nabla\Phi(x_k)\|^2 < \xi_1^{-1}\|\nabla\Phi(x_k)\|^2 \leq -\nabla\Phi(x_k)^T s_k$$

and, therefore,

$$\nabla\Phi(x_k)^T s_k < -\nu^{-1}\|\nabla\Phi(x_k)\|^2 < 0.$$

This completes the proof.

LEMMA 3.6 *If* $\lim_{k\to\infty} \frac{\nabla\Phi(x_k)^T s_k}{\|s_k\|} = 0$, *then* $\lim_{k\to\infty} \|\nabla\Phi(x_k)\| = 0$.

Proof. Suppose that $\lim_{k\to\infty} \frac{\nabla\Phi(x_k)^T s_k}{\|s_k\|} = 0$. From Lemma 3.5 we have

$$\nabla\Phi(x_k)^T s_k < -\nu^{-1}\|\nabla\Phi(x_k)\|^2 < 0.$$

Dividing through by $\|s_k\|$ gives

$$\frac{\nabla\Phi(x_k)^T s_k}{\|s_k\|} < -\frac{\|\nabla\Phi(x_k)\|^2}{\nu\|s_k\|} < 0.$$

Now $s_k = -B_k^{-1}\nabla\Phi(x_k)$ allows us to write

$$\|s_k\| = \|B_k^{-1}\nabla\Phi(x_k)\| \leq \|B_k^{-1}\|\|\nabla\Phi(x_k)\| \leq \xi_n^{-1}\|\nabla\Phi(x_k)\| \leq \mu^{-1}\|\nabla\Phi(x_k)\|.$$

Therefore

$$\frac{1}{\|s_k\|} \geq \frac{\mu}{\|\nabla\Phi(x_k)\|}$$

so that

$$\frac{\|\nabla\Phi(x_k)\|^2}{\|s_k\|} \geq \frac{\mu\|\nabla\Phi(x_k)\|^2}{\|\nabla\Phi(x_k)\|}$$

which becomes

$$-\frac{\|\nabla\Phi(x_k)\|^2}{\nu\|s_k\|} \leq -\frac{\mu\|\nabla\Phi(x_k)\|^2}{\nu\|\nabla\Phi(x_k)\|} = -\frac{\mu}{\nu}\|\nabla\Phi(x_k)\| < 0.$$

Thus

$$\frac{\nabla\Phi(x_k)^T s_k}{\|s_k\|} < -\frac{\mu}{\nu}\|\nabla\Phi(x_k)\| < 0 \tag{3.13}$$

and by the Sandwich theorem

$$\lim_{k\to\infty} \|\nabla\Phi(x_k)\| = 0.$$

This completes the proof.

THEOREM 3.7 *The BFGS algorithm is globally convergent to* $\nabla\Phi = 0$.

Proof. The spline Φ is continuously differentiable since the gradient $\nabla\Phi$ is continuous. The convexity of Φ guarantees that Φ is Lipschitz in a neighborhood of the minimizer and that Φ is bounded below. Since $\nabla\Phi(x_k)^T s_k < 0$, then by Theorem 3.1, we conclude that there exists a sequence $\{x_k\}, k = 1, 2, \ldots$ obeying

$$\Phi(x_k + \lambda s_k) \leq \Phi(x_k) + \alpha\lambda s_k^T \nabla\Phi(x_k) \text{ for } \alpha \in (0,1), \lambda > 0$$

and

$$s_k^T \nabla\Phi(x_k + \lambda_k s_k) \geq \beta s_k^T \nabla\Phi(x_k) \text{ for } \beta \in (\alpha, 1).$$

For this sequence, either $\nabla\Phi(x_k) = 0$ for some $k \geq 1$ or $\lim_{k\to\infty} \frac{\nabla\Phi(x_k)^T s_k}{\|s_k\|} = 0$. By Lemma 3.6, the latter implies $\lim_{k\to\infty} \|\nabla\Phi(x_k)\| = 0$. Thus either $\nabla\Phi(x_k) = 0$ for some $k \geq 1$ or $\{\nabla\Phi(x_k)\}_{k=1}^{\infty} \to 0$. This completes the proof.

To show that the sequence of iterates $\{x_k\}$ converges linearly to the unique solution of (1), we will need the following lemma from [20].

LEMMA 3.8 (Li [20, Corollary 2.8]) *Let X be a convex polyhedral subset of $\mathbf{R}^n$ and $\Phi(x)$ a convex piecewise quadratic function on X. Suppose that $X^* := \{x \in X : \Phi(x) = \Phi_{\min}\}$, where $\Phi_{\min} := \min_{x\in X} \Phi(x) > -\infty$. Then there exists a positive constant γ (depending only on $\Phi(x)$ and X) such that*

$$\operatorname{dist}(x, X^*) \leq \gamma\left(\Phi(x) - \Phi_{\min} + \sqrt{\Phi(x) - \Phi_{\min}}\right), \text{ for } x \in X.$$

THEOREM 3.9 *Let $\epsilon := \min\{\mu, \nu^{-1}\}$ as defined in Lemma 3.4 and let $\{x_k\}$ be the sequence of iterates generated by the BFGS algorithm. Then there exist two positive constants $\delta \equiv \delta(\Phi, \epsilon)$ and $\gamma \equiv \gamma(\Phi)$ such that*

$$\operatorname{dist}(x_k, X^*) \leq \gamma(\Phi_1 + \sqrt{\Phi_1})\left(\sqrt{1 - \frac{\delta}{(1+\Phi_1)^2}}\right)^k \text{ for } k \geq 1 \tag{3.14}$$

where $\Phi_1 := \Phi(x_1) - \Phi_{\min}$, and $\Phi_{\min} := \inf_{x\in\mathbf{R}^n} \Phi(x)$, and $X^* := \{x^* \in \mathbf{R}^n : \Phi(x^*) = \Phi_{\min}\}$.

Proof. For Φ a convex quadratic spline, Li and Swetits [23, Lemma 3.1] proved that there exists a positive constant α (depending only on Φ) such that

$$\left(\frac{s^T\nabla\Phi(x)}{\|s\|}\right)^2 \leq \alpha(\Phi(x) - \Phi(x + \lambda s))$$

whenever $s^T\nabla\Phi(x) < 0$ and $s^T\nabla\Phi(x + \lambda s) = 0$.

Having previously established that the search directions generated were descent directions and having noted that we are using exact line searches in each iteration, we can conclude that

$$\left(\frac{s_k^T \nabla\Phi(x_k)}{\|s_k\|}\right)^2 \leq \alpha(\Phi(x_k) - \Phi(x_k + \lambda_k s_k)) = \alpha(\Phi(x_k) - \Phi(x_{k+1})).$$

From (3.13), we have

$$\frac{s_k^T \nabla\Phi(x_k)}{\|s_k\|} \leq -\frac{\mu}{\nu}\|\nabla\Phi(x_k)\| < 0.$$

Thus,

$$\left(\frac{\mu}{\nu}\right)^2 \|\nabla\Phi(x_k)\|^2 \leq \left(\frac{s_k^T \nabla\Phi(x_k)}{\|s_k\|}\right)^2$$

so

$$\|\nabla\Phi(x_k)\|^2 \leq \left(\frac{\nu}{\mu}\right)^2 \left(\frac{s_k^T \nabla\Phi(x_k)}{\|s_k\|}\right)^2 \leq \alpha\left(\frac{\nu}{\mu}\right)^2 (\Phi(x_k) - \Phi(x_{k+1})).$$

By Theorem 2.1 of [19], there exists $\delta > 0$ depending only on Φ, μ, and ν such that

$$\Phi(x_k) - \Phi_{\min} \leq \Phi_1 \left(1 - \frac{\delta}{(1+\Phi_1)^2}\right)^k \text{ for } k \geq 1. \tag{3.15}$$

By Lemma 3.8, there exists a positive constant γ (depending only on Φ) such that

$$\operatorname{dist}(x_k, X^*) \leq \gamma\left(\Phi(x_k) - \Phi_{\min} + \sqrt{\Phi(x_k) - \Phi_{\min}}\right). \tag{3.16}$$

By (3.15), we have

$$\sqrt{\Phi(x_k) - \Phi_{\min}} \leq \sqrt{\Phi_1}\left(1 - \frac{\delta}{(1+\Phi_1)^2}\right)^k \leq \sqrt{\Phi_1}\left(1 - \frac{\delta}{(1+\Phi_1)^2}\right)^{k/2}. \tag{3.17}$$

Therefore, from (3.15)-(3.17), we have the estimate

$$\operatorname{dist}(x_k, X^*) \leq \gamma(\Phi_1 + \sqrt{\Phi_1})\left(\sqrt{1 - \frac{\delta}{(1+\Phi_1)^2}}\right)^k,$$

completing the proof.

THEOREM 3.10 *Let $\{x_k\}$ be the sequence of iterates generated by the BFGS algorithm (2.4) – (2.7) applied to the strictly convex quadratic spline Φ. Then $\{x_k\}$ converges linearly to the minimizer of Φ and converges finitely.*

Proof. For a strictly convex quadratic spline Φ, the set X^* of Lemma 3.8 is a singleton set containing the unique minimizer x^*. Linear convergence of $\{x_k\}$ follows from Theorem 3.9.

It is well known that the BFGS algorithm uses conjugate directions and therefore exhibits finite termination on a quadratic function (cf. Theorem 8.8.6 in [1]). For nondegenerate problems of the form (1.2), x^* exists in the interior of polyhedron $W^* = W_i$ for some $i = 1, \ldots, r$ and Φ is a quadratic function on W^*.

For $x \in W_j \neq W^*$, we know that $\nabla\Phi(x) \neq 0$ so the descent direction s_k generated for any $x_k \in W_j$ will force the sequence $\{x_k\}$ from W_j. Since there are a finite number of polyhedrons, $\{x_k\}$ must eventually move into W^* for some $k \geq 1$. Let k_0 be the smallest k for which $x_k \in W^*$. Then the algorithm terminates in at most $k_0 + n + 1$ iterations. This completes the proof.

NUMERICAL TESTING

Example Problem Revisited

To visualize the behavior of the BFGS algorithm, we can examine the trajectory to the known solution of our example problem (1.21) from a variety of starting points. This problem was well behaved over the entire selection of starting points including points at the intersection of the bounding lines, points on the bounding lines, and points in the interior of each polyhedral subregion of the spline. Sample trajectories are shown in Figures 5 through 7.

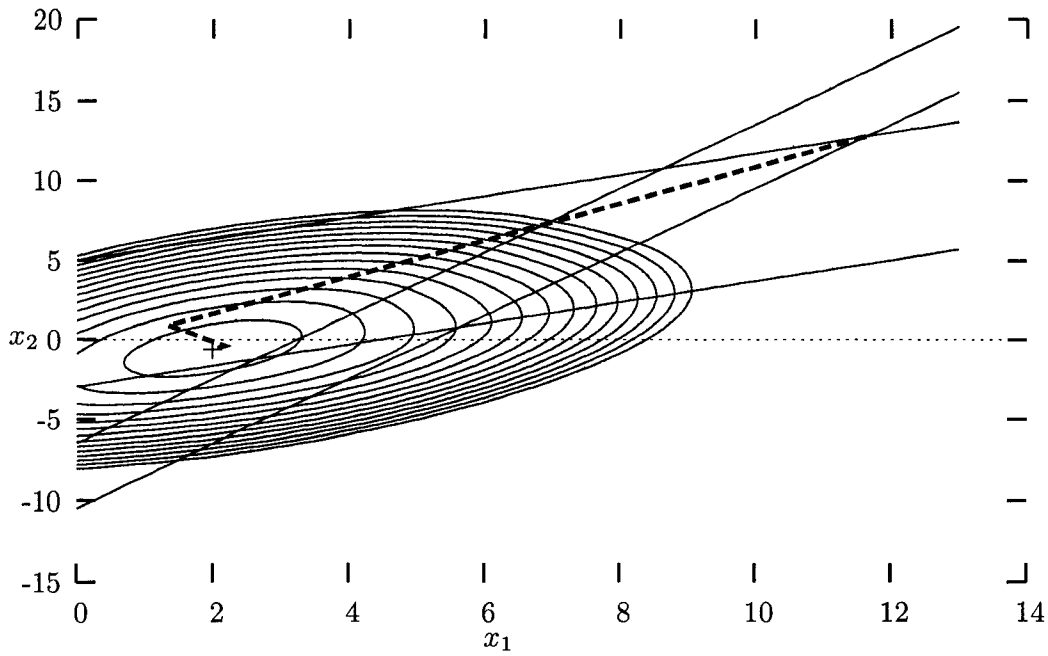

Figure 5. Trajectory From Point of Intersection

procedure deteriorates. The picture changes completely if a laser pulse with duration of approximately 100 fs or shorter is used. Now the laser interacts only with the electrons of a material. Before the material undergoes any changes in thermodynamic state, the laser pulse is over and most of the energy is deposited into the sample. Material removal occurs after the laser pulse. From this brief discussion it can be concluded that an fs-laser with pulse duration of approximately 100-200 fs and shorter should be closer to an optimal laser system than other systems. However, the importance of the leading edge rise time of the laser pulse has been studied [25]. The author showed that it should be fast enough to remove material layer at a speed equal to the heat conduction moving to the bulk. Lasers with different pulse duration have been applied to LIBS, for instance, Chichkov et al. studied laser ablation of solid targets by 0.2-5000 ps Ti: Sapphire laser pulses and introduced theoretical models and qualitative explanations of their experimental results [26]. They presented the advantages of femtosecond lasers for precise material processing, well defined patterns, and its pure ablation of metal targets in vacuum, which insures its ability as a promising tool for applications in precise material processing. However, Rieger et al. investigated the emission of laser-produced silicon and aluminum plasmas in the energy range from 0.1 to 100 μJ (0.5-500 J/cm^2) using 10 ns and 50 ps KrF laser pulses focused to a 5-μm diameter spot [27]. They showed that there is a little difference between 50 ps and 10 ns pulses in the plasma emission both in terms of the intensity of the emission lines and in terms of lifetime of the emission, while differences become important only at very low fluences approaching the plasma formation threshold. The effect of laser pulses of different durations has been compared mainly in terms of the amount of ablated material. Recently, studies have also compared

Unfortunately, (1.21) is so well behaved that none of the many starting points produced any erratic behavior that would help to explain later results. Shewchuk's analysis [33] of the Method of Steepest Descent sheds some light on our example problem. In Figure 8, we plot the quadratic form from Page 14 for W_2 over $\mathbf{R}^2$ and overlay the boundaries for W_2 with dashed lines (see Figure 3). The solid lines represent worst case starting points for Steepest Descent. A similar plot is rendered for W_7 and W_6 in Figures 9 and 10, respectively.

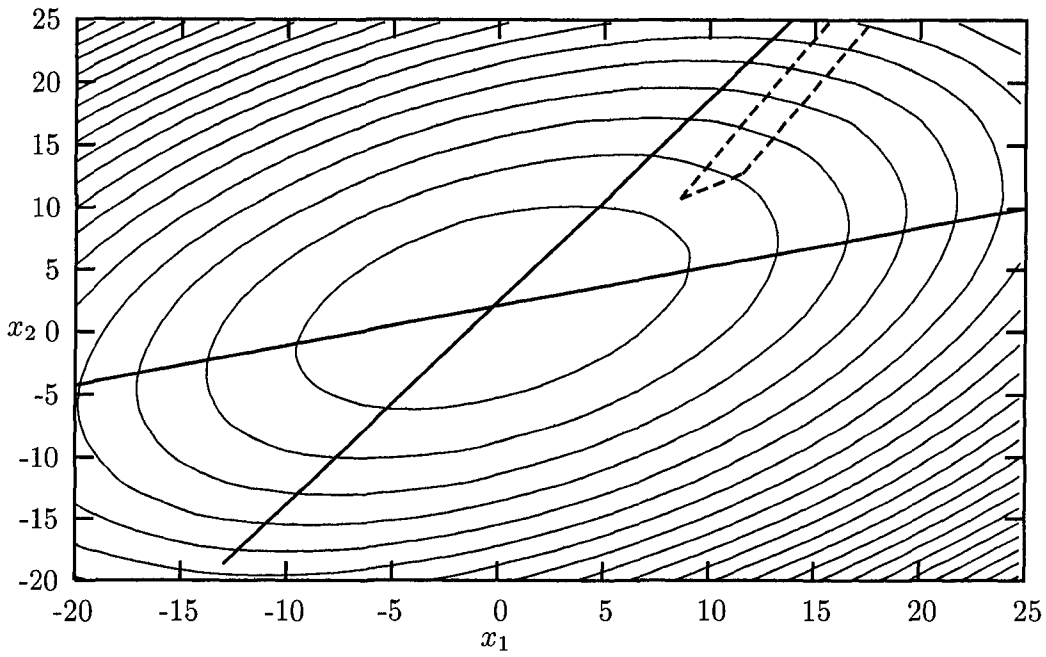

Figure 8. Worst case starting points in W_2

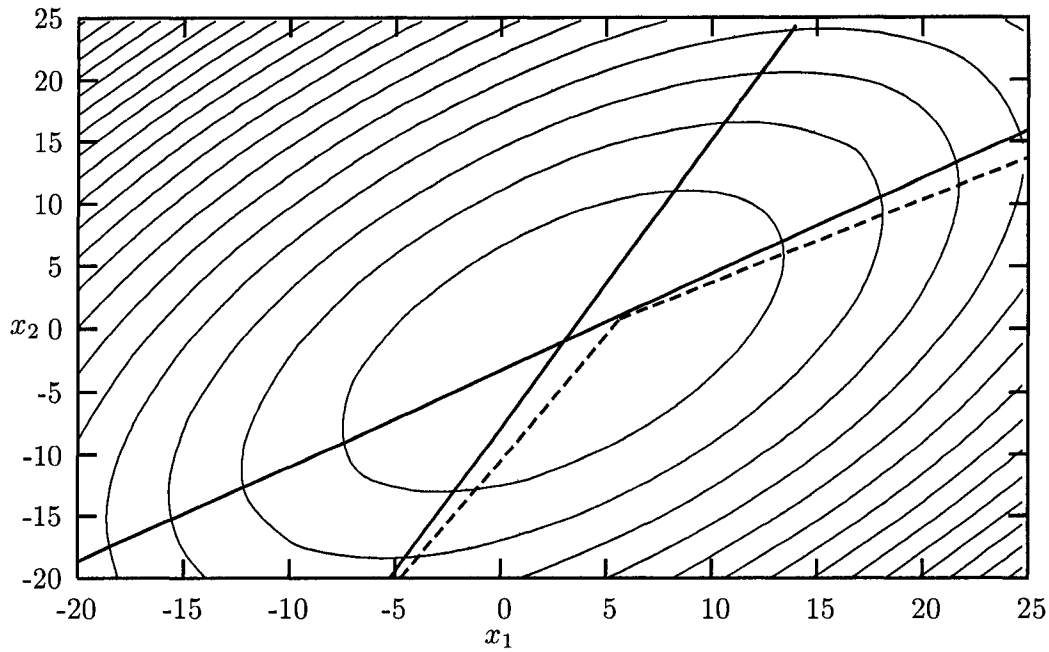

Figure 9. Worst case starting points in W_7

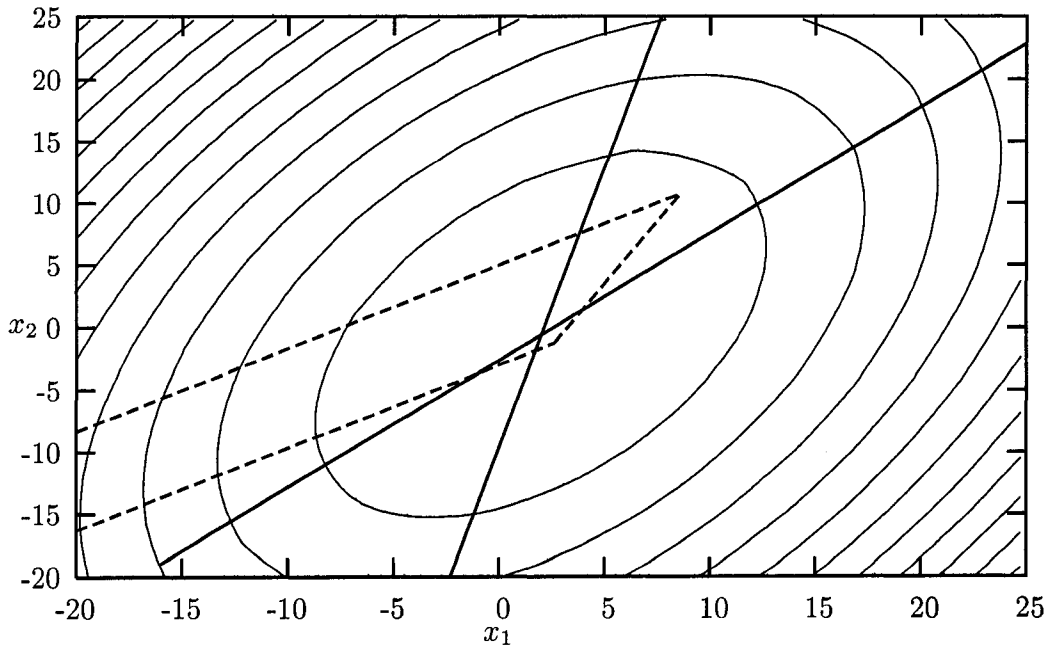

Figure 10. Worst case starting points in W_6

Because our first BFGS step is actually a Steepest Descent step, the pathologies of that method apply for each restart. In particular, when starting from a point on either worst case line, the next iterate falls on the other line. For Steepest Descent, the sequence of iterates bounces back and forth between the two lines making very slow progress toward the solution.

With the quadratic spline, this behavior is mitigated because starting from a worst case line in any region, except W_6, the next iterate lands in a new region. Because the worst case lines are different for each region, the restarted BFGS algorithm will not be restarting from a worst case point and performance improves as compared to Steepest Descent.

In W_6, both worst case lines exist in the region and a starting point on one line places the next iterate on the other line. At this stage, the BFGS method updates, because no restart is required, and we are no longer in the Steepest Descent model. Figure 11 plots a trajectory of this case.

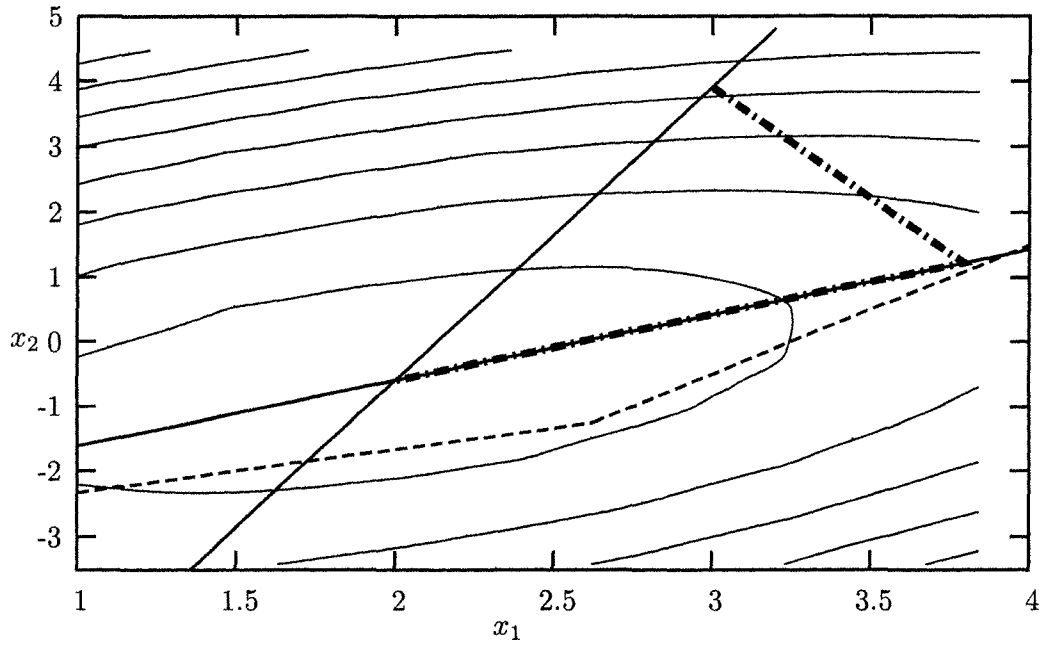

Figure 11. Trajectory from a Worst Case Starting Point in W_6

Test Problem Generation

The test problems are generated as simply bound quadratics of the form

$$\frac{1}{2}x^T Ax - b^T x$$

$$\text{subject to } -1 \leq x \leq 1$$

according to the method described in Moré and Toraldo [26] and presented in Li and Swetits [23]. The condition number of the matrix A, the degree of degeneracy of the solution, and the number of active constraints in the solution are the input parameters $ncond, ndeg$, and nax to the routine. Random numbers are generated using the RAN2 function defined in Press, Teukolsky, Vetterling, and Flannery [30].

The positive definite matrix A is constructed from a randomly generated orthogonal Householder matrix Y and a diagonal matrix D where the ith diagonal component d_i is given by

$$\log d_i = \left(\frac{i-1}{n-1}\right) \cdot ncond$$

for $i = 1, \ldots, n$. The matrix A is then formed as $A = YDY$ and has condition number 10^{ncond}, listed in Tables 1–15 as "Condition".

The exact solution x^* is randomly generated with $|x_i| < 1$ for $i = 1, \ldots, n$. To provide the required number of active constraints, we randomly generate a subset J of size nax from the set $\{1, \ldots, n\}$. For this active set J, we generate the Lagrange multiplier y using the input parameter $ndeg$ according to

$$|y_i| = 10^{-\mu_i \cdot ndeg} \text{ for } i \in J$$

where μ_i is generated randomly in the interval $(0, 1)$. The value 10^{-ndeg} is a measure of the numerical degeneracy of the problem and is listed in Tables 1–15 under "Degeneracy".

Having randomly generated A, x^*, y, and the active set J, we finish constructing the test problem by setting $b = Ax^* - y$. Bounds are set by defining $\ell_i = -1$, $u_i = 1$, and $y_i = 0$ for $i \notin J$. When $i \in J$, we define

$$\ell_i = x_i^* \text{ and } u_i = 1 \text{ for } y_i > 0$$

or

$$\ell_i = -1 \text{ and } u_i = x_i^* \text{ for } y_i < 0.$$

Numerical Results

Results of testing are given in Tables 1–12. The "Restart" column reflects the use of the restarting strategy given in (2.7). The "No Restart" column shows results with the restart completely suppressed. The "Accuracy" is measured in the ℓ_∞ norm by $\|\bar{x} - x^*\|_\infty$ where $\bar{x}$ is a solution generated by the BFGS algorithm. The number of iterations required to obtain $\bar{x}$ is reported under "Iterations". The number of variables and active constraints for each test set of problems is given at the top of the associated table. Algorithm termination criteria was set to $\|\nabla\Phi(x_k)\| \leq 0.5\left(10^{-16}\right)$ for Tables 1–9. The termination criteria was relaxed to $\|\nabla\Phi(x_k)\| \leq 0.5\left(10^{-13}\right)$ for Tables 10–12.

TABLE 1. Results for 2 Variables, 0 Constraints with Tight Tolerance

2 Variables					
0 Active Constraints					
Characteristics		Iterations		Accuracy	
Condition	Degeneracy	Restart	No Restart	Restart	No Restart
10^3	10^{-3}	4	3	0.13E-13	0.14E-13
10^3	10^{-6}	4	6	0.33E-14	0.71E-14
10^3	10^{-9}	4	4	0.22E-13	0.89E-14
10^3	10^{-12}	7	14	0.20E-13	0.14E-13
10^6	10^{-3}	5	4	0.22E-10	0.39E-10
10^6	10^{-6}	5	6	0.13E-10	0.53E-10
10^6	10^{-9}	5	3	0.34E-10	0.15E-10
10^6	10^{-12}	6	6	0.27E-11	0.12E-10
10^9	10^{-3}	11	5	0.23E-07	0.13E-07
10^9	10^{-6}	20	7	0.83E-08	0.22E-09
10^9	10^{-9}	6	4	0.49E-08	0.48E-10
10^9	10^{-12}	8	5	0.17E-07	0.32E-07
10^{12}	10^{-3}	50	9	0.48E-06	0.39E-05
10^{12}	10^{-6}	39	10	0.21E-04	0.33E-05
10^{12}	10^{-9}	16	8	0.34E-05	0.18E-04
10^{12}	10^{-12}	10	4	0.12E-04	0.18E-04

TABLE 2. Results for 2 Variables, 1 Constraint with Tight Tolerance

2 Variables					
1 Active Constraint					
Characteristics		Iterations		Accuracy	
Condition	Degeneracy	Restart	No Restart	Restart	No Restart
10^3	10^{-3}	7	5	0.00E+00	0.28E-16
10^3	10^{-6}	4	4	0.00E+00	0.11E-15
10^3	10^{-9}	8	6	0.10E-15	0.56E-16
10^3	10^{-12}	7	11	0.17E-15	0.56E-16
10^6	10^{-3}	6	8	0.33E-15	0.17E-14
10^6	10^{-6}	4	3	0.00E+00	0.28E-16
10^6	10^{-9}	5	5	0.26E-14	0.30E-14
10^6	10^{-12}	5	4	0.22E-15	0.11E-15
10^9	10^{-3}	4	8	0.31E-14	0.56E-14
10^9	10^{-6}	8	8	0.00E+00	0.00E+00
10^9	10^{-9}	11	5	0.22E-15	0.22E-15
10^9	10^{-12}	10	6	0.00E+00	0.00E+00
10^{12}	10^{-3}	9	8	0.14E-16	0.00E+00
10^{12}	10^{-6}	4	3	0.22E-15	0.67E-15
10^{12}	10^{-9}	8	9	0.00E+00	0.00E+00
10^{12}	10^{-12}	4	3	0.00E+00	0.00E+00

TABLE 3. Results for 2 Variables, 2 Constraints with Tight Tolerance

2 Variables					
2 Active Constraints					
Characteristics		Iterations		Accuracy	
Condition	Degeneracy	Restart	No Restart	Restart	No Restart
10^3	10^{-3}	7	3	0.00E+00	0.00E+00
10^3	10^{-6}	9	6	0.00E+00	0.43E-18
10^3	10^{-9}	6	6	0.00E+00	0.28E-16
10^3	10^{-12}	9	7	0.00E+00	0.00E+00
10^6	10^{-3}	6	3	0.00E+00	0.00E+00
10^6	10^{-6}	4	3	0.00E+00	0.00E+00
10^6	10^{-9}	7	6	0.00E+00	0.00E+00
10^6	10^{-12}	12	12	0.00E+00	0.00E+00
10^9	10^{-3}	5	6	0.00E+00	0.00E+00
10^9	10^{-6}	8	7	0.00E+00	0.00E+00
10^9	10^{-9}	15	11	0.00E+00	0.00E+00
10^9	10^{-12}	5	9	0.00E+00	0.00E+00
10^{12}	10^{-3}	4	3	0.00E+00	0.00E+00
10^{12}	10^{-6}	4	3	0.00E+00	0.00E+00
10^{12}	10^{-9}	13	7	0.00E+00	0.00E+00
10^{12}	10^{-12}	8	7	0.00E+00	0.00E+00

TABLE 4. Results for 10 Variables, 1 Constraint with Tight Tolerance

10 Variables					
1 Active Constraint					
Characteristics		Iterations		Accuracy	
Condition	Degeneracy	Restart	No Restart	Restart	No Restart
10^3	10^{-3}	52	27	0.28E-13	0.56E-13
10^3	10^{-6}	90	83	0.95E-13	0.13E-13
10^3	10^{-9}	137	214	0.13E-13	0.24E-13
10^3	10^{-12}	60	136	0.35E-13	0.58E-13
10^6	10^{-3}	129	141	0.18E-10	0.26E-11
10^6	10^{-6}	114	545	0.66E-10	0.53E-10
10^6	10^{-9}	68	53	0.90E-11	0.60E-11
10^6	10^{-12}	168	105	0.23E-11	0.17E-11
10^9	10^{-3}	215	318	0.44E-08	0.21E-08
10^9	10^{-6}	202	107	0.62E-10	0.27E-09
10^9	10^{-9}	348	186	0.22E-08	0.23E-08
10^9	10^{-12}	300	138	0.13E-08	0.56E-09
10^{12}	10^{-3}	5001	5001	0.72E+00	0.72E+00
10^{12}	10^{-6}	1408	800	0.47E-05	0.46E-05
10^{12}	10^{-9}	884	553	0.30E-06	0.21E-06
10^{12}	10^{-12}	5001	5001	0.52E+00	0.52E+00

TABLE 5. Results for 10 Variables, 5 Constraints with Tight Tolerance

10 Variables					
5 Active Constraints					
Characteristics		Iterations		Accuracy	
Condition	Degeneracy	Restart	No Restart	Restart	No Restart
10^3	10^{-3}	17	35	0.22E-14	0.15E-14
10^3	10^{-6}	28	25	0.27E-13	0.45E-13
10^3	10^{-9}	50	34	0.14E-13	0.12E-13
10^3	10^{-12}	48	31	0.63E-14	0.84E-14
10^6	10^{-3}	40	32	0.11E-12	0.14E-12
10^6	10^{-6}	55	78	0.67E-11	0.48E-11
10^6	10^{-9}	42	43	0.22E-12	0.93E-13
10^6	10^{-12}	85	51	0.16E-10	0.31E-10
10^9	10^{-3}	72	79	0.11E-08	0.10E-08
10^9	10^{-6}	48	78	0.12E-07	0.14E-07
10^9	10^{-9}	57	55	0.10E-10	0.27E-11
10^9	10^{-12}	69	223	0.25E-10	0.17E-11
10^{12}	10^{-3}	5001	5001	0.79E+00	0.79E+00
10^{12}	10^{-6}	33	41	0.31E-10	0.25E-10
10^{12}	10^{-9}	130	106	0.24E-09	0.38E-08
10^{12}	10^{-12}	171	275	0.36E-06	0.21E-06

TABLE 6. Results for 10 Variables, 9 Constraints with Tight Tolerance

10 Variables					
9 Active Constraints					
Characteristics		Iterations		Accuracy	
Condition	Degeneracy	Restart	No Restart	Restart	No Restart
10^3	10^{-3}	36	31	0.89E-15	0.21E-14
10^3	10^{-6}	45	40	0.56E-15	0.44E-15
10^3	10^{-9}	89	140	0.11E-15	0.11E-15
10^3	10^{-12}	176	153	0.00E+00	0.33E-15
10^6	10^{-3}	31	30	0.46E-14	0.72E-14
10^6	10^{-6}	25	19	0.33E-15	0.11E-15
10^6	10^{-9}	81	62	0.11E-14	0.13E-14
10^6	10^{-12}	378	368	0.31E-15	0.36E-15
10^9	10^{-3}	26	19	0.56E-16	0.11E-15
10^9	10^{-6}	66	72	0.33E-15	0.14E-14
10^9	10^{-9}	1097	51	0.56E-15	0.61E-15
10^9	10^{-12}	5001	5001	0.10E-02	0.10E-02
10^{12}	10^{-3}	37	35	0.33E-15	0.27E-14
10^{12}	10^{-6}	50	53	0.22E-15	0.56E-16
10^{12}	10^{-9}	96	118	0.22E-13	0.33E-14
10^{12}	10^{-12}	67	118	0.26E-14	0.43E-14

TABLE 7. Results for 100 Variables, 10 Constraints with Tight Tolerance

100 Variables					
10 Active Constraints					
Characteristics		Iterations		Accuracy	
Condition	Degeneracy	Restart	No Restart	Restart	No Restart
10^3	10^{-3}	5001	5001	0.17E-12	0.17E-12
10^3	10^{-6}	5001	5001	0.26E-12	0.26E-12
10^3	10^{-9}	5001	5001	0.61E-04	0.61E-04
10^3	10^{-12}	5001	5001	0.35E-12	0.35E-12
10^6	10^{-3}	5001	5001	0.20E+00	0.20E+00
10^6	10^{-6}	5001	5001	0.31E-01	0.31E-01
10^6	10^{-9}	5001	5001	0.42E+00	0.42E+00
10^6	10^{-12}	5001	5001	0.19E+00	0.19E+00
10^9	10^{-3}	5001	5001	0.77E+00	0.77E+00
10^9	10^{-6}	5001	5001	0.90E+00	0.90E+00
10^9	10^{-9}	5001	5001	0.11E+01	0.11E+01
10^9	10^{-12}	5001	5001	0.97E+00	0.97E+00
10^{12}	10^{-3}	5001	5001	0.97E+00	0.97E+00
10^{12}	10^{-6}	5001	5001	0.11E+01	0.11E+01
10^{12}	10^{-9}	5001	5001	0.96E+00	0.96E+00
10^{12}	10^{-12}	5001	5001	0.11E+01	0.11E+01

TABLE 8. Results for 100 Variables, 50 Constraints with Tight Tolerance

100 Variables					
50 Active Constraints					
Characteristics		Iterations		Accuracy	
Condition	Degeneracy	Restart	No Restart	Restart	No Restart
10^3	10^{-3}	5001	5001	0.13E-12	0.13E-12
10^3	10^{-6}	5001	5001	0.95E-13	0.95E-13
10^3	10^{-9}	5001	5001	0.27E-12	0.27E-12
10^3	10^{-12}	5001	5001	0.13E-12	0.13E-12
10^6	10^{-3}	5001	5001	0.42E+00	0.42E+00
10^6	10^{-6}	5001	5001	0.10E-01	0.10E-01
10^6	10^{-9}	5001	5001	0.87E-01	0.87E-01
10^6	10^{-12}	5001	5001	0.37E+00	0.37E+00
10^9	10^{-3}	5001	5001	0.10E+01	0.10E+01
10^9	10^{-6}	5001	5001	0.84E+00	0.84E+00
10^9	10^{-9}	5001	5001	0.90E+00	0.90E+00
10^9	10^{-12}	5001	5001	0.75E+00	0.75E+00
10^{12}	10^{-3}	5001	5001	0.95E+00	0.95E+00
10^{12}	10^{-6}	5001	5001	0.98E+00	0.98E+00
10^{12}	10^{-9}	5001	5001	0.90E+00	0.90E+00
10^{12}	10^{-12}	5001	5001	0.92E+00	0.92E+00

TABLE 9. Results for 100 Variables, 90 Constraints with Tight Tolerance

100 Variables					
90 Active Constraints					
Characteristics		Iterations		Accuracy	
Condition	Degeneracy	Restart	No Restart	Restart	No Restart
10^3	10^{-3}	144	112	0.47E-14	0.25E-13
10^3	10^{-6}	313	272	0.14E-12	0.23E-12
10^3	10^{-9}	5001	5001	0.95E-04	0.95E-04
10^3	10^{-12}	5001	5001	0.64E-04	0.64E-04
10^6	10^{-3}	993	1772	0.57E-10	0.51E-10
10^6	10^{-6}	5001	5001	0.77E-02	0.77E-02
10^6	10^{-9}	5001	5001	0.51E+00	0.51E+00
10^6	10^{-12}	5001	5001	0.69E+00	0.69E+00
10^9	10^{-3}	5001	5001	0.18E+00	0.18E+00
10^9	10^{-6}	5001	5001	0.39E+00	0.39E+00
10^9	10^{-9}	5001	5001	0.83E+00	0.83E+00
10^9	10^{-12}	5001	5001	0.92E+00	0.92E+00
10^{12}	10^{-3}	5001	5001	0.65E+00	0.65E+00
10^{12}	10^{-6}	5001	5001	0.91E+00	0.91E+00
10^{12}	10^{-9}	5001	5001	0.98E+00	0.98E+00
10^{12}	10^{-12}	5001	5001	0.10E+01	0.10E+01

TABLE 10. Results for 100 Variables, 10 Constraints with Relaxed Tolerance

100 Variables					
10 Active Constraints					
Characteristics		Iterations		Accuracy	
Condition	Degeneracy	Restart	No Restart	Restart	No Restart
10^3	10^{-3}	377	205	0.22E-09	0.37E-09
10^3	10^{-6}	428	1774	0.54E-09	0.20E-09
10^3	10^{-9}	698	529	0.15E-09	0.15E-09
10^3	10^{-12}	394	138	0.26E-09	0.75E-10
10^6	10^{-3}	5001	5001	0.20E+00	0.20E+00
10^6	10^{-6}	5001	5001	0.31E-01	0.31E-01
10^6	10^{-9}	5001	5001	0.42E+00	0.42E+00
10^6	10^{-12}	5001	5001	0.19E+00	0.19E+00
10^9	10^{-3}	5001	5001	0.77E+00	0.77E+00
10^9	10^{-6}	5001	5001	0.90E+00	0.90E+00
10^9	10^{-9}	5001	5001	0.11E+01	0.11E+01
10^9	10^{-12}	5001	5001	0.97E+00	0.97E+00
10^{12}	10^{-3}	5001	5001	0.97E+00	0.97E+00
10^{12}	10^{-6}	5001	5001	0.11E+01	0.11E+01
10^{12}	10^{-9}	5001	5001	0.96E+00	0.96E+00
10^{12}	10^{-12}	5001	5001	0.11E+01	0.11E+01

TABLE 11. Results for 100 Variables, 50 Constraints with Relaxed Tolerance

100 Variables					
50 Active Constraints					
Characteristics		Iterations		Accuracy	
Condition	Degeneracy	Restart	No Restart	Restart	No Restart
10^3	10^{-3}	1226	338	0.41E-09	0.16E-11
10^3	10^{-6}	519	310	0.32E-09	0.11E-09
10^3	10^{-9}	379	181	0.52E-09	0.21E-12
10^3	10^{-12}	560	477	0.17E-09	0.45E-10
10^6	10^{-3}	5001	538	0.42E+00	0.52E-07
10^6	10^{-6}	5001	5001	0.10E-01	0.10E-01
10^6	10^{-9}	3814	3814	0.68E-10	0.68E-10
10^6	10^{-12}	5001	5001	0.37E+00	0.37E+00
10^9	10^{-3}	5001	5001	0.10E+01	0.10E+01
10^9	10^{-6}	5001	5001	0.84E+00	0.84E+00
10^9	10^{-9}	5001	5001	0.90E+00	0.90E+00
10^9	10^{-12}	5001	5001	0.75E+00	0.75E+00
10^{12}	10^{-3}	5001	5001	0.95E+00	0.95E+00
10^{12}	10^{-6}	5001	5001	0.98E+00	0.98E+00
10^{12}	10^{-9}	5001	5001	0.90E+00	0.90E+00
10^{12}	10^{-12}	5001	5001	0.92E+00	0.92E+00

TABLE 12. Results for 100 Variables, 90 Constraints with Relaxed Tolerance

100 Variables					
90 Active Constraints					
Characteristics		Iterations		Accuracy	
Condition	Degeneracy	Restart	No Restart	Restart	No Restart
10^3	10^{-3}	122	105	0.14E-12	0.92E-13
10^3	10^{-6}	296	258	0.21E-11	0.21E-11
10^3	10^{-9}	5001	5001	0.95E-04	0.95E-04
10^3	10^{-12}	5001	5001	0.64E-04	0.64E-04
10^6	10^{-3}	789	224	0.74E-10	0.22E-08
10^6	10^{-6}	3949	3949	0.21E-09	0.21E-09
10^6	10^{-9}	5001	5001	0.51E+00	0.51E+00
10^6	10^{-12}	5001	5001	0.69E+00	0.69E+00
10^9	10^{-3}	5001	5001	0.18E+00	0.18E+00
10^9	10^{-6}	5001	5001	0.39E+00	0.39E+00
10^9	10^{-9}	5001	5001	0.83E+00	0.83E+00
10^9	10^{-12}	5001	5001	0.92E+00	0.92E+00
10^{12}	10^{-3}	5001	5001	0.65E+00	0.65E+00
10^{12}	10^{-6}	5001	5001	0.91E+00	0.91E+00
10^{12}	10^{-9}	5001	5001	0.98E+00	0.98E+00
10^{12}	10^{-12}	5001	5001	0.10E+01	0.10E+01

Tables 13–15 show the results of an experiment in handing off a solution from the BFGS method to a Newton method. The BFGS method routinely delivers gradients less than $5.0(10^{-4})$. Using this value as a handoff tolerance, the Newton method then completes the search in just a few additional iterations. In every instance, the handoff technique was able to converge to a satisfactory solution in fewer iterations than using the Newton method alone.

TABLE 13. Results for 100 Variables, 10 Constraints with Handoff to Newton Method

100 Variables					
10 Active Constraints					
Characteristics		Iterations		Accuracy	
Condition	Degeneracy	Restart	Newton	Restart	Newton
10^{3}	10^{-3}	121	2	0.93E-02	0.76E-13
10^{3}	10^{-6}	124	2	0.16E-01	0.67E-13
10^{3}	10^{-9}	149	5	0.27E-01	0.28E-12
10^{3}	10^{-12}	140	3	0.74E-02	0.19E-12
10^{6}	10^{-3}	303	2	0.15E+01	0.23E-10
10^{6}	10^{-6}	359	2	0.18E+01	0.38E-10
10^{6}	10^{-9}	286	5	0.18E+01	0.11E-09
10^{6}	10^{-12}	451	2	0.12E+01	0.36E-10
10^{9}	10^{-3}	318	2	0.18E+01	0.14E-07
10^{9}	10^{-6}	396	4	0.19E+01	0.40E-08
10^{9}	10^{-9}	342	5	0.20E+01	0.72E-07
10^{9}	10^{-12}	353	6	0.20E+01	0.61E-07
10^{12}	10^{-3}	204	3	0.19E+01	0.65E-04
10^{12}	10^{-6}	171	4	0.20E+01	0.49E-04
10^{12}	10^{-9}	228	5	0.19E+01	0.75E-05
10^{12}	10^{-12}	231	4	0.19E+01	0.29E-04

TABLE 14. Results for 100 Variables, 50 Constraints with Handoff to Newton Method

100 Variables					
50 Active Constraints					
Characteristics		Iterations		Accuracy	
Condition	Degeneracy	Restart	Newton	Restart	Newton
10^3	10^{-3}	129	9	0.14E-01	0.24E-12
10^3	10^{-6}	178	9	0.28E-01	0.34E-12
10^3	10^{-9}	293	7	0.71E-01	0.21E-12
10^3	10^{-12}	172	7	0.14E-01	0.19E-12
10^6	10^{-3}	368	10	0.12E+01	0.22E-09
10^6	10^{-6}	330	13	0.12E+01	0.59E-10
10^6	10^{-9}	414	12	0.14E+01	0.88E-10
10^6	10^{-12}	434	11	0.19E+01	0.96E-10
10^9	10^{-3}	229	11	0.20E+01	0.11E-06
10^9	10^{-6}	246	15	0.18E+01	0.12E-06
10^9	10^{-9}	594	14	0.18E+01	0.63E-07
10^9	10^{-12}	748	22	0.17E+01	0.75E-08
10^{12}	10^{-3}	147	13	0.19E+01	0.32E-04
10^{12}	10^{-6}	265	15	0.16E+01	0.90E-05
10^{12}	10^{-9}	330	20	0.19E+01	0.49E-04
10^{12}	10^{-12}	646	18	0.18E+01	0.11E-03

TABLE 15. Results for 100 Variables, 90 Constraints with Handoff to Newton Method

100 Variables					
90 Active Constraints					
Characteristics		Iterations		Accuracy	
Condition	Degeneracy	Restart	Newton	Restart	Newton
10^3	10^{-3}	99	8	0.62E-02	0.45E-13
10^3	10^{-6}	382	9	0.32E-01	0.77E-13
10^3	10^{-9}	132	12	0.75E-01	0.11E-12
10^3	10^{-12}	127	11	0.57E-01	0.11E-12
10^6	10^{-3}	117	14	0.16E+01	0.11E-09
10^6	10^{-6}	200	15	0.16E+01	0.19E-09
10^6	10^{-9}	722	11	0.90E+00	0.23E-11
10^6	10^{-12}	920	17	0.15E+01	0.24E-10
10^9	10^{-3}	87	20	0.10E+01	0.23E-08
10^9	10^{-6}	697	21	0.19E+01	0.77E-07
10^9	10^{-9}	965	18	0.16E+01	0.74E-07
10^9	10^{-12}	199	24	0.20E+01	0.66E-07
10^{12}	10^{-3}	87	28	0.15E+01	0.17E-06
10^{12}	10^{-6}	1512	36	0.18E+01	0.97E-04
10^{12}	10^{-9}	812	31	0.18E+01	0.88E-04
10^{12}	10^{-12}	411	31	0.17E+01	0.22E-06

CONCLUSIONS

The results of testing the BFGS algorithm with both the restarting and non-restarting strategies were rather disappointing. In the majority of the instances of failed convergence, the algorithm degenerated to a steepest descent behavior with the iterates alternating back and forth between two polyhedral regions. While (3.14) hints that a judicious choice of x_1 may improve the overall rate of convergence, the testing did not bear this out. Among a variety of initialization routines there was no significant difference in algorithm performance.

On the positive side, the numerical evidence lends strong support to the conjecture that the BFGS algorithm without restarting converges at least linearly on a strictly convex quadratic spline. Inspection of Tables 1-3 shows that the "No Restart" version was able to obtain a satisfactory solution in every case as the version with "Restart" and usually required fewer iterations.

The tight tolerance termination threshold turned out to be particularly important. While the initial testing of the algorithm showed a great deal of promise with reasonable rates of convergence, the final point generated as a solution almost always turned out to be infeasible. The combination of strigent feasibility testing and tight tolerance thresholds resulted in excellent accuracy when the algorithm converged. As evidenced in Tables 7 – 12, the algorithm performed poorly with larger sized problems.

Directions of Future Research

In this study, we have explored a narrow slice of a broad field. Several ideas for future study emerged from both tangential investigations to resolve nagging questions and recommendations from other people exposed to the work. A few of these ideas are presented below in no particular order.

Handoff Methods and Criteria

Upon examining the behavior of the algorithm when it starts to slow down, we found that it is relatively close to a solution in all cases. For problems up to and including 100 variables, the BFGS algorithm was able to obtain estimates to the actual solution as close as 10^{-4}. In these cases, handing the estimate off to a Newton method resulted in accurate and feasible solutions in every problem set. Tables 13 - 15 show the same 100 variable problem sets previously used with a column for the lowest magnitude gradient obtainable by the BFGS algorithm and a column giving the number of Newton iterations required to find an acceptable solution.

The handoff tolerance of $5.0(10^{-4})$ was a hueristic determined from examining the results of the BFGS performance. Additional work is required to determine proper switching criteria.

Improved Performance of the BFGS Algorithm

While several subroutines would immediately benefit from modifications to the underlying data structures or transfers of data between the structures, these

changes would make the algorithm faster but not necessarily more accurate. Additional research into the failure modes of the BFGS algorithm is needed to find ways to improve the performance of the algorithm. For example, an examination of the search direction vector revealed that several failures were related to the extremely small magnitude of the search vector and the correspondingly large magnitude step size required to compensate.

Pathology of a Common Failure Mode

One of the most common failure modes for the BFGS algorithm was a condition where the sequence of iterates alternated between two neighboring regions. In these cases, the restarting criteria introduced some aspects of the Method of Steepest Descent. With the extensive literature analyzing Steepest Descent as a starting point, a more complete analysis of this failure mode would be useful.

Large Sparse Problems

An improved BFGS algorithm may be most useful in large sparse problems. In these cases, a Newton method may be difficult to apply because the calculation of the matrix inverse results in a fully populated matrix. The BFGS method does not compute the inverse directly and could realize some efficiencies in both performance and storage by taking advantage of the sparsity.

Restarting

As noted above, the BFGS algorithm appears to work well without restarting. A proof of convergence without restarting is needed.

Genetic Algorithms

In some cases, the BFGS algorithm demonstrated a sensitivity to initial starting values. Using concepts from the study of genetic algorithms, several starting values could be used to initiate the search for a minimizer. Fitness functions that use the resulting gradients at each iteration would guide the algorithm by dropping the worst performing approaches. In addition, better avenues may result from the combination of the better performing approach vectors according to the basic operations of reproduction, crossover, and mutation as described in Goldberg [15].

REFERENCES

[1] M. S. Bazaraa, H. D. Sherali, and C. M. Shetty, *Nonlinear Programming: Theory and Algorithms, 2nd Edition,* John Wiley & Sons, New York, NY, 1993.

[2] C. G. Broyden, *A class of methods for solving nonlinear simultaneous equations,* Math. Comp., 19 (1965), pp. 577–593.

[3] C. G. Broyden, *The convergence of a class of double-rank minimization algorithms 2,* J. Inst. Math. Appl., 6 (1970), pp. 222–231.

[4] P. Brucker, *An $O(n)$ algorithm for quadratic knapsack problems,* Oper. Res. Lett., 3 (1984), pp. 163–166.

[5] P. H. Calamai and J. J. Moré, *Quasi-Newton updates with bounds,* SIAM J. Numer. Anal., 24 (1987), pp. 1434–1441.

[6] R. W. Cottle, S. G. Duvall, and K. Zikan, *A Lagrangian relaxation algorithm for the constrained matrix problem,* Naval Res. Logist. Quart., 33 (1986), pp. 55–76.

[7] R. W. Cottle, J.-S. Pang, and R. E. Stone, *The Linear Complementarity Problem,* Academic Press, San Diego, CA, 1992.

[8] W. C. Davidon, *Variable Metric Method for Minimization,* AEC Research Development Report, ANL-5990, 1959.

[9] J. E. DENNIS, JR. AND J. J. MORÉ, *A characterization of superlinear convergence and its application to quasi-Newton methods,* Math. Comp., 28 (1974), pp. 549–560.

[10] J. E. DENNIS, JR. AND J. J. MORÉ, *Quasi-Newton methods, motivation and theory,* SIAM Review, 19 (1977), pp. 46–89.

[11] J. E. DENNIS, JR. AND R. B. SCHNABEL, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations,* Prentice-Hall, Englewood Cliffs, NJ, 1983.

[12] R. FLETCHER, *A new approach to variable metric algorithms,* Comp. J., 13 (1970), pp. 317–322.

[13] R. FLETCHER AND M. J. D. POWELL, *A rapidly convergent descent method for minimization,* Comp. J., 6 (1963), pp. 163–168.

[14] M. FRANK AND P. WOLFE, *An algorithm for quadratic programming,* Naval Res. Logist. Quart., 3 (1956), pp. 95–110.

[15] D. E. GOLDBERG, *Genetic Algorithms in Search, Optimization & Machine Learning,* Addison Wesley Longman, Reading, MA, 1989.

[16] D. GOLDFARB, *A family of variable metric methods derived by variational means,* Math. Comp., 24 (1970), pp. 23–26.

[17] C. HILDRETH, *A quadratic programming procedure,* Naval Res. Logist. Quart., 4 (1957), pp. 79–85, Erratum, ibid., p. 361.

[18] N. KARMARKAR, *A new polynomial-time algorithm for linear programming,* Combinatorica, 4 (1984), pp. 373–395.

[19] W. Li, *Linearly convergent descent methods for the unconstrained minimization of convex quadratic splines,* J. Optim. Theory Appl., 86 (1995), pp. 145–172.

[20] W. Li, *Error bounds for piecewise convex quadratic programs and applications,* SIAM J. Control Optim., 33 (1995), pp. 1510–1529.

[21] W. Li, *A conjugate gradient method for the unconstrained minimization of strictly convex quadratic splines,* Math. Programming, 72 (1996), pp. 17–32.

[22] W. Li and J. Swetits, *A Newton method for convex regression, data smoothing, and quadratic programming with bounded constraints,* SIAM J. Optim., 3 (1993), pp. 466–488.

[23] W. Li and J. Swetits, *A new algorithm for solving strictly convex quadratic programs,* SIAM J. Optim., 7 (1997), pp. 595–619.

[24] Y. Y. Lin and J.-S. Pang, *Iterative methods for large convex quadratic programs: a survey,* SIAM J. Control Optim., 25 (1987), pp. 383–411.

[25] R. Mifflin, *Semismooth and semiconvex functions in constrained optimization,* SIAM J. Control Optim., 15 (1977), pp. 959–972.

[26] J. J. Moré and G. Toraldo, *On the solution of large quadratic programming problems with bound constraints,* SIAM J. Optim., 1 (1991), pp. 93–113.

[27] Y. Nesterov and A. Nemirovskii, *Interior-Point Polynomial Algorithms in Convex Programming,* SIAM, Philadelphia, PA, 1994.

[28] J.-S. Pang, *Methods for quadratic programming: a survey,* Computers Chem. Engineering, 7 (1983), pp. 583–594.

[29] P. M. Pardalos and N. Kovoor, *An algorithm for a singly constrained class of quadratic programs subject to upper and lower bounds,* Math. Programming, 46 (1990), pp. 321–328.

[30] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in Fortran 77: The Art of Scientific Computing, Second Edition,* Cambridge University Press, New York, NY, 1992.

[31] A. W. Roberts and D. E. Varberg, *Another proof that convex functions are locally Lipschitx,* Amer. Math. Monthly, 81 (1974), pp. 1014–1016.

[32] D. F. Shanno, *Conditioning of quasi-Newton methods for function minimization,* Math. Comp., 24 (1970), pp. 647–656.

[33] J. R. Shewchuk, *An Introduction to the Conjugate Gradient Method Without the Agonizing Pain,* CMU-CS-94-125, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1994.

[34] N. E. Shor, *A class of nearly-differentiable functions and a minimization method for functions of this class,* Cybernetics, 4 (1972), pp. 65–70.

[35] P. Wolfe, *Convergence conditions for ascent methods,* SIAM Review, 11 (1969), pp.226–235.

[36] S. J. Wright, *Primal-Dual Interior-Point Methods,* SIAM, Philadelphia, PA, 1997.

APPENDIX

CODE

```
      PROGRAM BFGS
      IMPLICIT DOUBLE PRECISION(A-H,O-Z)
      PARAMETER(N=100)
      COMMON/INVERSE/GRD(N,N),ERR(N),ISIGN(N),ISNOLD(N)
      COMMON/HESSIAN/BFGS(N,N),Q(N)
      COMMON/OTHER/GAM(2*N)
      COMMON/DATA/A(N,N),SOL(N),B(N),R(N)
      COMMON/UNKNOWN/X(N),Y(N)
      COMMON/BOUND/BL(N),UB(N),C(N)

      ICOUNT = 0
      ITMAX = 20000
      TOL = 0.5D-12

      DO 2100 NAX = N/10,(9*N)/10,(4*N)/10
      DO 2000 NCOND = 3,12,3
      DO 1900 NDEG = 3,12,3

      ICOUNT = ICOUNT + 1
      IDUM = -ICOUNT

      Generate the problem data

      CALL PRBGEN(N,NCOND,NDEG,NAX,A,B,BL,SOL,UB,
          R,C,Y,ISIGN,X,IDUM)

      Estimate the norm of A

      ALP = 0.0D0
      DO 200 I = 1,N
        S = 0.0D0
        DO 100 J = 1,N
          S = S + DABS(A(I,J))
100     CONTINUE
        ALP = DMAX1(ALP,S)
200   CONTINUE
```

```
      ALP = 1.0D0 / ALP
      DO 400 I = 1,N
        DO 300 J = 1,N
          GRD(I,J) = A(I,J)
300     CONTINUE
        ERR(I) = 0.0D0
        ISIGN(I) = 0
400   CONTINUE

      Set GRD = I - ALP * A and C = ALP * B

      DO 600 I = 1,N
        DO 500 J = 1,N
          GRD(I,J) = -ALP * A(I,J)
500   CONTINUE
        GRD(I,I) = 1.0D0 + GRD(I,I)
        C(I) = ALP * B(I)
600   CONTINUE

      IRESET = 1

      Initialize x, the starting point, and the matrix

      CALL RESTART
700   DO 800 I = 1,N
        X(I) = 1.0D0
800   CONTINUE
900   ERR1 = 1.0D0

      This is where the algorithm proper starts

      Compute the ERROR, X-(GRD*X+C)(SUB L, SUPER U)

      ITER = 1
      CALL ERROR(ERR1,IFLAG,ITER)
1000  CONTINUE

      If the maximum number of iterations is exceeded, then quit

      IF(ITER.GT.ITMAX) GO TO 1600
```

```
      IF(ITER.EQ.(ITMAX / 5)) THEN
        DO 1100 I = 1,N
          X(I) = -1.0D0
1100    CONTINUE
        CALL RESTART
      END IF

      IF(ITER.EQ.(2 * ITMAX / 5)) THEN
        DO 1200 I = 1,N
          X(I) = (-1.0D0)**N
1200    CONTINUE
        CALL RESTART
      END IF

      IF(ITER.EQ.(3 * ITMAX / 5)) THEN
        DO 1300 I = 1,N
          X(I) = 0.0D0
1300    CONTINUE
        CALL RESTART
      END IF

      IF(ITER.EQ.(4 * ITMAX / 5)) THEN
        DO 1400 I = 1,N
          X(I) = (BL(I) + UB(I)) / 2.0D0
1400    CONTINUE
        CALL RESTART
      END IF

      If the ERROR is sufficiently small, then quit

      IF(ERR1.LT.TOL) THEN
      DO 1500 I = 1,N
        IF(X(I).LT.BL(I)) X(I) = BL(I)
        IF(X(I).GT.UB(I)) X(I) = UB(I)
1500  CONTINUE
      END IF

      CALL ERROR(ERR1,IFLAG,ITER)

      IF(ERR1.LT.TOL) GO TO 1600
```

```
      IF((IRESET.EQ.1).AND.(IFLAG.NE.0)) CALL RESTART

      Compute the descent direction, Q

      CALL SEARCH

      Prepare for line minimization

      A1 = 0.0D0
      B1 = 0.0D0
      CALL LINEFUNCT(A1,B1)

      Do line minimization

      T = 0.0D0
      IRES = 1
      CALL UNIMIN(A1,B1,T,IRES,ITER)

      Update BFGS and X to X-T*Q

      CALL UPDATE(T,ERR1,ITER)

      Start another iteration

      ITER = ITER + 1
      GO TO 1000
      Exit calculations

1600  CONTINUE

      DO 1700 I = 1,N
        IF(X(I).LT.BL(I))X(I) = BL(I)
        IF(X(I).GT.UB(I))X(I) = UB(I)
1700  CONTINUE

      CALL ERROR(ERR1,IFLAG,ITER)
      ERRMAX = 0.0D0
      ERRM = 0.0D0

      DO 1800 I = 1,N
        ERRM = DABS(X(I) - SOL(I))
        ERRMAX = DMAX1(ERRMAX,ERRM)
1800  CONTINUE
```

```
      IF(IRESET.EQ.1) THEN
        ITERRS = ITER
        RSERMX = ERRMAX
        IRESET = 0
        GO TO 700
      END IF

1900  CONTINUE
2000  CONTINUE
2100  CONTINUE
      STOP
      END

      INCLUDE 'prbgen.f'
      INCLUDE 'ran2.f'
```

```
      SUBROUTINE ERROR(ERR1,IFLAG,ITER)
      IMPLICIT DOUBLE PRECISION(A-H,O-Z)
      PARAMETER(N=100)
      COMMON/INVERSE/GRD(N,N),ERR(N),ISIGN(N),ISNOLD(N)
      COMMON/HESSIAN/BFGS(N,N),Q(N)
      COMMON/BOUND/BL(N),UB(N),C(N)
      COMMON/UNKNOWN/X(N),Y(N)
      COMMON/OTHER/GAM(2*N)

      IFLAG = 0
      ERR1 = 0.0D0

      DO 100 I = 1,N
        ISNOLD(I) = ISIGN(I)
100   CONTINUE

      DO 300 I = 1,N
        GAM(I) = C(I)
        DO 200 J = 1,N
          GAM(I) = GAM(I) + GRD(I,J) * X(J)
200     CONTINUE
        IF(GAM(I).GE.BL(I).AND.GAM(I).LE.UB(I))THEN
          ISIGN(I) = 0
          ERR(I) = -GAM(I)
        END IF
        IF(GAM(I).LT.BL(I))THEN
          ISIGN(I) = -1
          ERR(I) = -BL(I)
        END IF
        IF(GAM(I).GT.UB(I))THEN
          ISIGN(I) = 1
          ERR(I) = -UB(I)
        END IF
        ERR(I) = X(I) + ERR(I)
300   CONTINUE

      DO 400 I = 1,N
        ERR1 = ERR1 + ERR(I) * ERR(I)
400   CONTINUE

      ERR1 = DSQRT(ERR1)
```

```
      DO 500 I = 1,N
        IF(ISIGN(I).NE.ISNOLD(I)) IFLAG = 1
500   CONTINUE

      RETURN
      END
```

```
      SUBROUTINE LINEFUNCT(A1,B1)
      IMPLICIT DOUBLE PRECISION(A-H,O-Z)
      PARAMETER(N=100)
      COMMON/INVERSE/GRD(N,N),ERR(N),ISIGN(N),ISNOLD(N)
      COMMON/HESSIAN/BFGS(N,N),Q(N)
      COMMON/UNKNOWN/X(N),Y(N)
      COMMON/OTHER/GAM(2*N)

      DO 100 I = 1,N
        Y(I) = 0.0D0
        DO 100 J = 1,N
          Y(I) = Y(I) + GRD(I,J) * Q(J)
100   CONTINUE

      DO 200 I = 1,N
        A1 = A1 - X(I) * Y(I) + Y(I) * GAM(I)
200   CONTINUE

      DO 300 I = 1,N
        B1 = B1 + Y(I) * Q(I) - Y(I) * Y(I)
300   CONTINUE

      RETURN
      END
```

```
      SUBROUTINE UNIMIN(A1,B1,T,IRES,ITER)
      IMPLICIT DOUBLE PRECISION(A-H,O-Z)
      PARAMETER(N=100)
      COMMON/OTHER/GAM(2*N)
      COMMON/DATA/A(N,N),SOL(N),B(N),R(N)
      COMMON/UNKNOWN/X(N),Y(N)
      COMMON/BOUND/BL(N),UB(N),C(N)
      COMMON/HESSIAN/BFGS(N,N),Q(N)
      DIMENSION W(2*N)

      N0 = 0
      Z = 1.0D-08
      ZZZ = 1.0D-14

      DO 100 I = 1,N
        W(I) = Y(I)
        GAM(I) = BL(I) - GAM(I)
        R(I) = BL(I) - UB(I)
100   CONTINUE

      DO 200 I = 1,N
        IF(DABS(W(I)).GT.Z)THEN
          N0 = N0+1
          GAM(N0) = GAM(I)
          W(N0) = W(I)
          R(N0) = R(I)
        END IF
200   CONTINUE

      DO 300 I = 1,N0
        W(N0 + I) = -W(I)
        GAM(N0 + I) = R(I) - GAM(I)
300   CONTINUE

      N0 = 2 * N0

400   CONTINUE

      IF(N0.EQ.0)GO TO 700

      IF(IRES.GE.2)THEN
        J = (N0 - 1) / 2 + 1
        T = -GAM(J) / W(J)
      END IF
```

```
      F = A1 + B1 * T

      DO 500 I = 1,N0
        TEMP = GAM(I) + W(I) * T
        IF(IRES.GT.2.AND.I.EQ.J) TEMP = 0.0D0
        IF(TEMP.GT.0.0D0)THEN
          F = F + W(I) * TEMP
        END IF
500   CONTINUE

      IF(F.GT.0.0D0)THEN
        SIGN = 1.0D0
      ELSE
        SIGN = -1.0D0
      END IF

      JJ = 0

      DO 600 I = 1,N0
        TEM = SIGN *(T + GAM(I) / W(I))
        IF(I.EQ.J.AND.IRES.GT.2) TEM = 0.0D0
        TEMP = SIGN * W(I)
        IF(TEMP.LT.0.0D0.AND.TEM.LE.0.0D0)THEN
          A1 = A1 + W(I) * GAM(I)
          B1 = B1 + W(I) * W(I)
        ELSE
        IF(TEMP.LT.0.0D0.OR.(TEMP.GT.0.0D0.AND.TEM.GT.0.0D0))THEN
          JJ = JJ + 1
          GAM(JJ) = GAM(I)
          W(JJ) = W(I)
        END IF
        END IF
600   CONTINUE

      N0=JJ
      IRES=IRES+1

      GO TO 400
```

```
700  CONTINUE

     IF(DABS(B1).GT.ZZZ)THEN
       T = -A1/B1
     ELSE
       T = -1.0D0
     END IF

     IF(T.LE.0.0D0) T = 1.0D0

800  CONTINUE

     RETURN
     END
```

```
      SUBROUTINE UPDATE(STEP,ERR1,ITER)
      Adapted from Algorithm A9.4.2 (BFGSFAC) [11].
      Copyright © 1996 Society for Industrial and Applied Mathematics.
      Reprinted with permission.
      IMPLICIT DOUBLE PRECISION(A-H,O-Z)
      PARAMETER(N=100)
      COMMON/INVERSE/GRD(N,N),ERR(N),ISIGN(N),ISNOLD(N)
      COMMON/HESSIAN/BFGS(N,N),Q(N)
      COMMON/UNKNOWN/X(N),Y(N)
      COMMON/OTHER/GAM(2*N)
      COMMON/BOUND/BL(N),UB(N),C(N)
      DIMENSION P(N),S(N),ERROLD(N),GC(N),GN(N),T(N)
      DIMENSION U(N),XOLD(N)

      TOL = 5.0D-15
      PNORM = 0.0D0
      SNORM = 0.0D0

      DO 100 I = 1,N
        ERROLD(I) = ERR(I)
        P(I) = -STEP*Q(I)
        XOLD(I) = X(I)
        PNORM = PNORM + P(I) * P(I)
        X(I) = X(I) + P(I)
  100 CONTINUE

      PNORM = DSQRT(PNORM)
      CALL ERROR(ERR1,IFLAG,ITER)

      IF(IFLAG.NE.0)THEN
        CALL RESTART
        GO TO 1500
      END IF

      DO 300 I=1,N
        GC(I) = 0.0D0
        GN(I) = 0.0D0
        DO 200 J=1,N
          GC(I) = GC(I) + GRD(I,J)*ERROLD(J)
          GN(I) = GN(I) + GRD(I,J)*ERR(J)
          S(I) = GN(I) - GC(I)
  200   CONTINUE
        SNORM = SNORM + S(I)*S(I)
  300 CONTINUE
```

```
      SNORM = DSQRT(SNORM)
      GNNORM = 0.0D0
      GCNORM = 0.0D0
      QNORM = 0.0D0
      PGN = 0.0D0
      QGN = 0.0D0

      DO 400 I = 1,N
        GNNORM = GNNORM + GN(I) * GN(I)
        GCNORM = GCNORM + GC(I) * GC(I)
        QNORM = QNORM + Q(I) * Q(I)
        PGN = PGN + P(I) * GN(I)
        QGN = QGN + Q(I) * GN(I)
400   CONTINUE

      GNNORM = DSQRT(GNNORM)
      GCNORM = DSQRT(GCNORM)
      QNORM = DSQRT(QNORM)
      TEMP1 = 0.0D0

      DO 600 I = 1,N
        TEMP1 = TEMP1 + S(I) * P(I)
        T(I) = 0.0D0
        DO 500 J = I,N
          T(I) = T(I) + BFGS(J,I) * P(J)
500     CONTINUE
600   CONTINUE

      IF(TEMP1.LT.(TOL * PNORM * SNORM))THEN
        GO TO 1500
      END IF

      TEMP2 = 0.0D0

      DO 700 I = 1,N
        TEMP2 = TEMP2 + T(I) * T(I)
700   CONTINUE

      ALPHA = DSQRT(TEMP1 / TEMP2)
```

```
      DO 900 I = 1,N
        TEMP3 = 0.0D0
        DO 800 J = 1,I
          TEMP3 = TEMP3 + BFGS(I,J) * T(J)
800     CONTINUE
        GC(I) = DABS(GC(I))
        GN(I) = DABS(GN(I))
        IF(DABS(S(I) - TEMP3).LT.(TOL * DMAX1(GC(I),GN(I))))THEN
          GO TO 1500
        END IF
        U(I) = S(I) - ALPHA * TEMP3
900   CONTINUE

      IF((DSQRT(TEMP1 * TEMP2)).EQ.0.0D0)GO TO 1500

      TEMP3 = 1.0D0 / (DSQRT(TEMP1 * TEMP2))

      DO 1000 I = 1,N
        T(I) = TEMP3 * T(I)
1000  CONTINUE

      DO 1200 I = 2,N
        DO 1100 J = 1,I-1
          BFGS(J,I) = BFGS(I,J)
1100    CONTINUE
1200  CONTINUE

      CALL QRUPDATE(T,U)

      DO 1400 I = 2,N
        DO 1300 J = 1,I-1
          BFGS(I,J) = BFGS(J,I)
1300    CONTINUE
1400  CONTINUE

1500  RETURN
      END
```

```
      SUBROUTINE QRUPDATE(T,U)
      Adapted from Algorithm A3.4.1 (QRUPDATE) [11].
      Copyright © 1996 Society for Industrial and Applied Mathematics.
      Reprinted with permission.
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      PARAMETER(N=100)
      COMMON/INVERSE/GRD(N,N),ERR(N),ISIGN(N),ISNOLD(N)
      COMMON/HESSIAN/BFGS(N,N),Q(N)
      DIMENSION T(N),U(N)

      DO 100 I=2,N
        BFGS(I,I-1) = 0.0D0
100   CONTINUE

      K = N
200   IF((T(K).NE.0.0D0).OR.(K.LE.1)) GO TO 300
      K = K - 1
      GO TO 200

300   DO 400 I = K-1,1,-1
        CALL JACROTATE(I,T(I),-T(I+1))
        IF(T(I).EQ.0.0D0) THEN
          T(I) = DABS(T(I+1))
        ELSE
          T(I) = DSQRT(T(I)**2 + T(I+1)**2)
        END IF
400   CONTINUE

      DO 500 J = 1,N
        BFGS(1,J) = BFGS(1,J) + T(1) * U(J)
500   CONTINUE

      DO 600 I = 1,K-1
        CALL JACROTATE(I,BFGS(I,I),-BFGS(I+1,I))
600   CONTINUE

      RETURN
      END
```

```
      SUBROUTINE JACROTATE(I,A,B)
      Adapted from Algorithm A3.4.1a (JACROTATE) [11].
      Copyright © 1996 Society for Industrial and Applied Mathematics.
      Reprinted with permission.
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      PARAMETER(N=100)
      COMMON/INVERSE/GRD(N,N),ERR(N),ISIGN(N),ISNOLD(N)
      COMMON/HESSIAN/BFGS(N,N),Q(N)

      IF(A.EQ.0.0D0)THEN
        C = 0.0D0
        S = 1.0D0
        S = DSIGN(S,B)
      ELSE
        DEN = DSQRT(A**2 + B**2)
        C = A/DEN
        S = B/DEN
      END IF

      DO 100 J = I,N
        Y = BFGS(I,J)
        W = BFGS(I+1,J)
        BFGS(I,J) = C * Y - S * W
        BFGS(I+1,J) = S * Y + C * W
  100 CONTINUE

      RETURN
      END
```

```
      SUBROUTINE RESTART
      IMPLICIT DOUBLE PRECISION(A-H,O-Z)
      PARAMETER(N=100)
      COMMON/INVERSE/GRD(N,N),ERR(N),ISIGN(N),ISNOLD(N)
      COMMON/HESSIAN/BFGS(N,N),Q(N)

      DO 200 I = 1,N
        DO 100 J = 1,N
          IF(I.EQ.J)THEN
          BFGS(I,J) = 1.0D0
          ELSE
          BFGS(I,J) = 0.0D0
        END IF
100     CONTINUE
200   CONTINUE

      RETURN
      END
```

```
      SUBROUTINE SEARCH
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      PARAMETER(N=100)
      COMMON/INVERSE/GRD(N,N),ERR(N),ISIGN(N),ISNOLD(N)
      COMMON/HESSIAN/BFGS(N,N),Q(N)
      DIMENSION GRDERR(N),Y(N)

      DO 200 I=1,N
        GRDERR(I) = 0.0D0
        DO 100 J=1,N
          GRDERR(I) = GRDERR(I) + GRD(I,J) * ERR(J)
100     CONTINUE
200   CONTINUE

      Solve BFGS*Y = GRDERR for Y

      DO 400 I = 1,N
        SUM = 0.0D0
        DO 300 J = 1,I-1
          SUM = SUM + BFGS(I,J) * Y(J)
300     CONTINUE
        Y(I) = (GRDERR(I) - SUM) / BFGS(I,I)
400   CONTINUE

      Solve BFGS(transpose) * Q = Y for Q

      Q(N) = Y(N) / BFGS(N,N)
      DO 600 I = N-1,1,-1
        SUM = 0.0D0
        DO 500 J = I+1,N
          SUM = SUM + BFGS(J,I) * Q(J)
500     CONTINUE
        Q(I) = (Y(I) - SUM) / BFGS(I,I)
600   CONTINUE

      RETURN
      END
```

```
      SUBROUTINE PRBGEN(N,NCOND,NDEG,NAX,A,B,
        BL,SOL,UB,R,C,Y,ISIGN,X,IDUM)
      IMPLICIT DOUBLE PRECISION(A-H,O-Z)
      DIMENSION A(N,N),B(N),BL(N),SOL(N),UB(N),
        Y(N),R(N),X(N),C(N),ISIGN(N)
      INTEGER N,NCOND,NDEG,NAX,IDUM
      REAL XRAY,RAN2

      This subroutine generates test problems for algorithms that
      solve strictly convex quadratic programming problems with
      simple bound constraints. The Hessian is the matrix A,
      BL and UB are the lower and upper bounds, XS is the solution,
      ISIGN is the set of indices of active constraints, X0 is the
      initial guess, NCOND is the condition number of A,
      NDEG is a measure of the degeneracy in the problem,
      NAX is the number of active constraints at the solution, and
      NAO is the number of "active" constraints in X0.

      The objective function is (XT)*A*X/2-(XT)*B.
      The constraints are BL .= X .= UB.
      See More and Toraldo, Numer. Math. 55(1989),
      377-400, especially pages 390-400.

      A has the form MDM where M is an orthogonal Householder
      matrix generated by the vector Y and D is a diagonal matrix
      with LOG(C(I))=(I-1)*NCOND/(N-1).
      The components of Y are random elements in (-1,1).

      DN = DBLE(N-1)
      DC = DBLE(NCOND)

      DO 100 I = 1,N
        DI = DBLE(I-1)
        C(I) = DI * DC / DN
        C(I) = 10.0D0**(-C(I))
        XRAY = RAN2(IDUM)
        DL = DBLE(XRAY)
        LI = IDNINT(DN * DL)
        LL = LI - 2 * (LI / 2)
        IF(LL.EQ.0) LL = -1
        Y(I) = DBLE(LL) * DL
  100 CONTINUE
```

```
      YNORM = 0.0D0

      DO 200 I = 1,N
        YNORM = YNORM + Y(I) * Y(I)
200   CONTINUE

      YNORM = DSQRT(YNORM)

      DO 300 I = 1,N
        Y(I) = Y(I) / YNORM
300   CONTINUE

      DO 400 L = 1,N
        DO 400 M = 1,N
          A(M,L) = 0.0D0
400   CONTINUE

      Z = 0.0D0

      DO 700 L = 1,N
        DO 600 M = 1,N
          DO 500 J = 1,N
          Z = 1.0D0 - 2.0D0 * Y(J) * Y(J)
          IF(J.EQ.L.AND.J.EQ.M)THEN
          A(M,L) = A(M,L) + Z * C(J) * Z
          END IF
          IF(J.EQ.L.AND.J.NE.M)THEN
          A(M,L) = A(M,L) - 2.0D0 * Y(M) * Y(J) * C(J) * Z
          END IF
          IF(J.EQ.M.AND.J.NE.L)THEN
          A(M,L) = A(M,L) - 2.0D0 * Y(L) * Y(J) * C(J) * Z
          END IF
          IF(J.NE.L.AND.J.NE.M)THEN
          A(M,L) = A(M,L) + 4.0D0 * Y(M) * Y(J) * C(J) * Y(J) * Y(L)
          END IF
500       CONTINUE
600     CONTINUE
700   CONTINUE

      DO 800 M = 1,N-1
        DO 800 L = M+1,N
          A(M,L) = A(L,M)
800   CONTINUE
```

```
      Generate the solution XS

      DO 900 I = 1,N
        XRAY = RAN2(IDUM)
        DL = DBLE(XRAY)
        LI = IDNINT(DN * DL)
        LL = LI - 2 * (LI / 2)
        IF(LL.EQ.0) LL = -1
        SOL(I) = DBLE(LL) * DL
900   CONTINUE

      Generate the active constraints and upper and
      lower bounds

      DNA = DBLE(NAX)
      Z = 0.0D0
      DUMMY = DNA / DN
      DNDEG = DBLE(NDEG)
      KOWNT = 0

      DO 1000 I = 1,N
        XRAY = RAN2(IDUM)
        DL = DBLE(XRAY)
        IF(KOWNT.EQ.NAX) DL = DL + DUMMY
        IF(NAX - KOWNT.EQ.N - I + 1) DL = -DL
        IF(DL.LE.DUMMY)THEN
          DL = DABS(DL)
          KOWNT = KOWNT + 1
          ISIGN(I) = I
          R(I) = 10.0D0**(-DL * DNDEG)
          LI = IDNINT(DN * DL)
          LL = LI - 2 * (LI / 2)
          IF(LL.EQ.0) LL = -1
          R(I) = DBLE(LL) * R(I)
```

```
            IF(LL.LT.0)THEN
              BL(I) = -1.0D0
              UB(I) = SOL(I)
            ELSE
              BL(I) = SOL(I)
              UB(I) = 1.0D0
            END IF
          ELSE
            R(I) = 0.0D0
            ISIGN(I) = 0
            BL(I) = -1.0D0
            UB(I) = 1.0D0
          END IF
1000    CONTINUE

        Generate the initial guess

        DO 1100 I = 1,N
          X(I) = (BL(I) + UB(I)) / 2.0D0
1100    CONTINUE

        Guarantee that XS is the solution by setting
        B = A * XS - R.

        DO 1300 I = 1,N
          B(I) = -R(I)
          DO 1200 J = 1,N
            B(I) = B(I) + A(I,J) * SOL(J)
1200      CONTINUE
1300    CONTINUE

        RETURN
        END
```

VITA

William Howard Thomas II was born in Charleston, South Carolina and grew up in Arlington and Tyler, Texas. He attended Northeast Louisiana University in Monroe, Louisiana and earned a Bachelor of Science degree in Mathematics on May 13, 1983.

Having volunteered for service in the United States Navy in August 1982, Mr. Thomas trained in Newport, Rhode Island and was commissioned October 1, 1983. After an additional 18 months of nuclear power and division officer training, he reported for duty aboard USS MISSISSIPPI (CGN-40). As the Reactor Electrical Division Officer, he helped bring the Virginia-class nuclear powered cruiser through a 20-month complex overhaul at Norfolk Naval Shipyard in Portsmouth, Virginia.

In 1988, Lieutenant Thomas transferred to the USS THEODORE ROOSEVELT (CVN-71) as the Chemistry and Radiological Controls Assistant. During this 26-month tour of duty, this Nimitz-class aircraft carrier made her maiden deployment to the Mediterranean and Lieutenant Thomas earned his Surface Warfare Officer qualification. Following the deployment, he married the former Mary Katherine Riedel of Norfolk, Virginia.

Lieutenant Thomas spent the next two years in Monterey, California attending the Naval Postgraduate School where he earned a Master of Science degree in Applied Mathematics. Both his daughter, Chelsea Anne, and son, William Howard III, were born there.

After Department Head training, Lieutenant Thomas reported aboard USS ANTRIM (FFG-20) as Combat Systems Officer. Homeported in Pascagoula, Mississippi, the Oliver Hazard Perry-class frigate conducted numerous drug interdiction operations throughout the Caribbean and participated in the naval blockade of Haiti in 1994. Lieutenant Thomas separated from active duty to begin doctoral study in 1995. His personal decorations include the Navy Achievement Medal and Navy Commendation Medal (2 awards).

While pursuing studies at Old Dominion University, Mr. Thomas served as a Graduate Teaching Assistant. In 1998, he was recognized as Outstanding Classroom Instructor by the College of Sciences. He completed coursework in 1998 and accepted full-time employment at the Naval Surface Warfare Center in Dahlgren, Virginia.

From 1998 to 2004, Mr. Thomas analyzed various commercial infrastructures combining his naval engineering experience with his mathematical training to help strengthen Department of Defense installations against terrorist attack. His work in automating infrastructure analysis resulted in a suite of software for which a patent is pending. More recently, Mr. Thomas has directed software development on several vehicle integration projects bringing an automated counter-sniper capability to soldiers and marines deploying overseas.

Mr. Thomas is an Eagle Scout.