SIAM J. OPTIM.
Vol. 8, No. 3, pp. 682–706, August 1998
© 1998 Society for Industrial and Applied Mathematics
004

ON THE IMPLEMENTATION OF AN ALGORITHM FOR LARGE-SCALE EQUALITY CONSTRAINED OPTIMIZATION*

MARUCHA LALEE†, JORGE NOCEDAL†, AND TODD PLANTENGA‡

Abstract. This paper describes a software implementation of Byrd and Omojokun's trust region algorithm for solving nonlinear equality constrained optimization problems. The code is designed for the efficient solution of large problems and provides the user with a variety of linear algebra techniques for solving the subproblems occurring in the algorithm. Second derivative information can be used, but when it is not available, limited memory quasi-Newton approximations are made. The performance of the code is studied using a set of difficult test problems from the CUTE collection.

Key words. minimization, nonlinear optimization, large-scale optimization, constrained optimization, trust region methods, quasi-Newton methods

AMS subject classifications. 65K05, 90C30

PII. S1052623493262993

1. Introduction.

This paper describes the implementation and testing of an algorithm for solving large nonlinear optimization problems with equality constraints. We write this problem as

$$\min_{x \in \mathbf{R}^n} f(x) \qquad \text{subject to} \quad c(x) = 0, \tag{1.1}$$

where $f : \mathbf{R}^n \to \mathbf{R}$ and $c : \mathbf{R}^n \to \mathbf{R}^m$ are twice continuously differentiable functions, and where $n \geq m$. The algorithm is an extension of the trust region method proposed by Byrd [2] and Omojokun [32], and our implementation is designed for large-scale applications. By this we mean problems where the matrices of function derivatives are more profitably treated using sparse data structures. We have in mind problems with a thousand or more unknowns, and the number of constraints may be small, large, or even equal to n—the algorithm treats all cases in a uniform manner.

Every iteration of the algorithm of Byrd and Omojokun requires the solution of two trust region subproblems of smaller dimension. We provide both iterative and direct solvers for these subproblems, allowing the user to select the most efficient solver for a particular problem. If second derivatives of f and c are not provided, our code automatically generates limited memory BFGS (ℓ-BFGS) approximations.

Inequality constraints are not considered in this study for two reasons. First, we want to determine if the method of Byrd and Omojokun is robust, and we wish to give careful attention to its proper implementation in the setting of large-scale optimization, starting with the simpler case of equality constraints. As we will discuss below, this case raises many difficult questions and requires a sophisticated implementation. The second reason is that the algorithm developed in this paper forms the core of two new algorithms for handling general inequality constraints that are currently under investigation [34], [3].

*Received by the editors February 15, 1994; accepted for publication (in revised form) March 3, 1997; published electronically June 3, 1998. This work was supported by National Science Foundation grants CCR-9101359 and ASC-9213149 and by Department of Energy grant DE-FG02-87ER25047.
http://www.siam.org/journals/siopt/8-3/26299.html

†Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL 60208 (nocedal@ece.nwu.edu).

‡Sandia National Laboratories, MS 9214, P.O. Box 969, Livermore, CA 94551-0969 (tdplant@ca.sandia.gov).

The algorithm is motivated and described in section 2. Two trust region subproblems arise in the algorithm, and section 3 digresses to develop some techniques for finding approximate solutions for them. In section 4 the details of our implementation are presented. The results of numerical experiments, which suggest that the method holds great promise, are discussed in section 5. A summarizing discussion and ideas for further enhancements presented in section 6 conclude the paper.

2. General description of the algorithm. We begin by introducing some notation. The gradient of the objective function f is denoted by g, and A denotes the $n \times m$ matrix of constraint gradients, i.e.,

$$A(x) = [\nabla c^1(x), \ldots, \nabla c^m(x)],$$

where c^i, $i = 1, ..., m$ are the components of the vector c. We will assume that $A(x_k)$ has full column rank for all x_k. The Lagrangian of problem (1.1) is $L(x, \lambda) = f(x) - \lambda^T c(x)$, where λ is the vector of Lagrange multipliers. Throughout the paper $\|\cdot\|$ denotes the ℓ_2 norm.

The algorithm of Byrd [2] and Omojokun [32] can be interpreted as a sequential quadratic programming method with a trust region, and it is derived from earlier work by Vardi [42]; Celis, Dennis, and Tapia [7]; Powell and Yuan [38], and Byrd, Schnabel, and Schultz [5]. (Although motivated differently, [45] and [12] present related methods.) The algorithm decomposes each constrained SQP subproblem into two smaller unconstrained trust region subproblems which are easier to solve. This makes the Byrd–Omojokun method attractive for large-scale optimization, which was one of the reasons why we chose to base our implementation on it.

The algorithm is simple to explain and motivate. At a current iterate x_k we choose a trust region radius Δ_k and Lagrange multipliers λ_k, and attempt to generate a new step d_k by solving

$$\min_{d \in \mathbf{R}^n} \quad d^T g_k + \frac{1}{2} d^T \nabla_x^2 L(x_k, \lambda_k)\, d \tag{2.1}$$

$$\text{subject to} \quad A_k^T d + c_k = 0, \tag{2.2}$$

$$\|d\| \le \Delta_k, \tag{2.3}$$

where the subscript denotes evaluation of a function at x_k. However, as is well known, restricting the size of the step by (2.3) may preclude us from satisfying the linear constraint (2.2). Therefore Byrd and Omojokun first compute a step that lies well within the trust region and that satisfies the linear constraint (2.2) as much as possible. This is done by defining a relaxation parameter $\zeta \in (0, 1)$ and computing a step v_k that solves the *vertical* (or *normal*) *subproblem*

$$\min_{v \in \mathbf{R}^n} \quad \|A_k^T v + c_k\| \tag{2.4}$$

$$\text{subject to} \quad \|v\| \le \zeta \Delta_k. \tag{2.5}$$

This problem can have many solutions, but we will show in section 4.1 that it always has a solution v_k in the range space of A_k; i.e., v_k will be expressed as a linear combination of the columns of A_k. This allows us to completely decouple this subproblem from the next one.

The algorithm is designed so that the full step d need not move any closer to the feasible manifold than v_k does, so we next reformulate (2.1)–(2.3) as

$$\min_{d\in\mathbf{R}^n} \quad d^T g_k + \frac{1}{2} d^T \nabla_x^2 L(x_k, \lambda_k)\, d \tag{2.6}$$

$$\text{subject to} \quad A_k^T d = A_k^T v_k, \tag{2.7}$$

$$\|d\| \le \Delta_k. \tag{2.8}$$

This problem, unlike (2.1)–(2.3), always has a nonempty feasible region (for instance, $d = v_k$ satisfies all constraints). Byrd and Omojokun solve for d by seeking a step complementary to v_k. To this end we compute an $n \times (n-m)$ matrix Z_k that spans the null space of A_k (so that $A_k^T Z_k = 0$), and we define the total step of the algorithm as $d = v_k + Z_k u$, where the vector $u \in \mathbf{R}^{n-m}$ is yet to be determined. Substituting for d in (2.6)–(2.8), noting that v_k and $Z_k u$ are orthogonal, and ignoring constant terms, we obtain

$$\min_{u\in\mathbf{R}^{n-m}} \quad (g_k + \nabla_x^2 L_k v_k)^T Z_k u + \frac{1}{2} u^T Z_k^T \nabla_x^2 L_k Z_k u \tag{2.9}$$

$$\text{subject to} \quad \|Z_k u\| \le \sqrt{\Delta_k^2 - \|v_k\|^2}. \tag{2.10}$$

We denote the solution of this *horizontal* (or *tangential*) *subproblem* by u_k, and we define the total step as

$$d_k = v_k + Z_k u_k. \tag{2.11}$$

We then set

$$x_{k+1} = x_k + d_k,$$

provided x_{k+1} gives a reduction in the merit function; otherwise, the trust region is reduced and a new trial step is computed. (The merit function will be described later in section 4.6.)

The approach of Byrd–Omojokun thus consists of replacing (2.1)–(2.3) with two trust region subproblems of smaller dimension, each with just a single quadratic constraint. The vertical subproblem (2.4)–(2.5) has a spherical trust region in $\mathbf{R}^n$, whereas the horizontal subproblem (2.9)–(2.10) has an ellipsoidal trust region in $\mathbf{R}^{n-m}$. Both subproblems appear to be much easier to solve than the trust region formulations studied by Celis, Dennis, and Tapia [7], and by Powell and Yuan [38], which include an additional quadratic constraint.

Let us compare the Byrd–Omojokun algorithm with some other methods for large-scale constrained optimization. The trust region method implemented in LANCELOT [9] is based on the minimization of an augmented Lagrangian and is quite different from the SQP approach. The SL1QP method developed by Fletcher [18] also poses a single unconstrained trust region subproblem of dimension n, but it differs markedly from our approach in that it requires the minimization of a nondifferentiable model. SNOPT [23], a new line search implementation of the sequential quadratic programming method, appears to hold much promise, but the computational issues for large-scale problems are fundamentally different from those arising in our trust region method.

The second reason for our interest in the Byrd–Omojokun method (the first reason is the simplicity of the trust region subproblems) is its ability to handle nonlinear constraints. Note that when $n = m$, i.e., when the problem reduces to that of finding the root of a system of nonlinear equations, the Byrd–Omojokun algorithm coincides

with the Levenberg–Marquardt method, which is known to be robust and efficient. Most of the algorithms mentioned in the previous paragraph do not have this property, and we believe that our code will prove to be very effective at handling highly nonlinear constraints.

An outline of the algorithm studied in this paper is given below (section 4.9 contains the detailed version).

ALGORITHM 2.1. *General description of the Byrd–Omojokun algorithm*

Constants $\epsilon > 0$ and $\eta \in (0, 1)$ are given
Choose x_0 and $\Delta_0 > 0$
loop, starting with $k = 0$
  Compute f_k, c_k, g_k, A_k, and Z_k
  Compute multipliers λ_k
  if $\|g_k - A_k\lambda_k\|_\infty < \epsilon$ **and** $\|c_k\|_\infty < \epsilon$ **then stop**
  Compute v_k by solving the vertical subproblem (2.4)–(2.5) approximately
  Compute $\nabla_x^2 L_k(x_k, \lambda_k)$ or update its ℓ-BFGS approximation
  Compute u_k by solving the horizontal subproblem (2.9)–(2.10) approximately
  Set $d_k = v_k + Z_k u_k$
  Compute the actual reduction in the merit function, a_red,
  and the predicted reduction, p_red
  if $\dfrac{\text{a_red}}{\text{p_red}} \geq \eta$
    then set $x_{k+1} = x_k + d_k$, $\Delta_{k+1} \geq \Delta_k$
    else set $x_{k+1} = x_k$, $\Delta_{k+1} < \|d_k\|$
continue loop, after incrementing k

We show in the rest of the paper that this algorithm can be converted into efficient and robust software for large-scale computations. Since the key to good performance lies in finding approximate solutions to the subproblems without incurring a high computational cost, we begin by studying a formulation of the trust region problem that is general enough for our purposes.

3. Solving trust region subproblems. We wish to develop efficient methods for approximately solving the trust region problem

$$\min_{s \in \mathbf{R}^q} \phi(s) = g^T s + \frac{1}{2} s^T B s \qquad \text{subject to} \quad \|Cs\| \leq \Delta, \tag{3.1}$$

where B is symmetric, C is a $p \times q$ matrix of full rank with $p \geq q$, and where the number of variables q is assumed to be large. The vertical subproblem (2.4)–(2.5) can be written in this form with $p = q = n$ and $C = I$, and the horizontal subproblem (2.9)–(2.10) can be obtained by setting $C = Z_k$, $q = n - m$, and $p = n$.

The presence of C can make the trust region ellipsoidal, and it is therefore convenient to transform the problem so that it has a spherical trust region. Let us define

$$\tilde{s} \equiv (C^T C)^{1/2} s, \tag{3.2}$$

so that

$$s = (C^T C)^{-1/2} \tilde{s}. \tag{3.3}$$

Using (3.3) in (3.1) we obtain the equivalent problem

$$\min_{\tilde{s} \in \mathbf{R}^q} \tilde{\phi}(\tilde{s}) = \tilde{g}^T \tilde{s} + \frac{1}{2} \tilde{s}^T \tilde{B} \tilde{s} \qquad \text{subject to} \quad \|\tilde{s}\| \leq \Delta, \tag{3.4}$$

where we have defined

$$\tilde{g} \equiv (C^T C)^{-1/2} g \quad \text{and} \quad \tilde{B} \equiv (C^T C)^{-1/2} B (C^T C)^{-1/2}. \tag{3.5}$$

We will now state some well-known methods for approximately solving the spherically constrained trust region problem (3.4) and then transform them via (3.2) to obtain methods for solving the ellipsoidally constrained problem (3.1). We begin by discussing the Cauchy step, which plays a fundamental role in all of these methods.

Cauchy point. The Cauchy point for (3.4) is defined as the minimizer of $\tilde{\phi}$ in the direction of steepest descent at $\tilde{s} = 0$, subject to the trust constraint. Therefore, we solve the problem

$$\min_{\alpha>0} \; -\tilde{g}^T(\alpha\tilde{g}) + \frac{1}{2}(\alpha\tilde{g})^T \tilde{B}(\alpha\tilde{g}) \qquad \text{subject to} \quad \|\alpha\tilde{g}\| \le \Delta$$

to obtain

$$\tilde{s}_{cp} = -\tilde{\alpha}\tilde{g},$$

where

$$\tilde{\alpha} = \begin{cases} \dfrac{\tilde{g}^T\tilde{g}}{\tilde{g}^T\tilde{B}\tilde{g}} & \text{if } \tilde{g}^T\tilde{B}\tilde{g} > 0 \quad \text{and} \quad \dfrac{(\tilde{g}^T\tilde{g})^{3/2}}{\tilde{g}^T\tilde{B}\tilde{g}} \le \Delta, \\ \dfrac{\Delta}{\|\tilde{g}\|} & \text{otherwise.} \end{cases} \tag{3.6}$$

(If $\|\tilde{g}\| = 0$ then we take $\tilde{s}_{cp} = 0$.) We now apply the transformations (3.2) and (3.5) to obtain the Cauchy step for the ellipsoidal problem (3.1),

$$s_{cp} = -\tilde{\alpha}(C^T C)^{-1} g, \tag{3.7}$$

where $\tilde{\alpha}$ is defined in terms of g and B by (3.6) and (3.5). Thus the transformation results in a Cauchy step in the direction of scaled steepest descent, the scale factor $(C^T C)^{-1}$ coming from the ellipsoidal shape of the trust region constraint.

We next describe the two methods used in our code for approximately solving the trust region problem (3.1) when the number of unknowns q is large: Powell's dogleg method and Steihaug's conjugate gradient (CG) iteration.

Dogleg method. If the Hessian matrix B in (3.1) is positive definite, then Powell's dogleg method [35] can give an inexpensive approximate solution. The method calculates the Cauchy point s_{cp} and the Newton step $s_n = -B^{-1}g$ (even if s_n violates the trust region constraint). The dogleg path consists of the two line segments from $s = 0$ to $s = s_{cp}$ and from $s = s_{cp}$ to $s = s_n$. The dogleg method finds the minimizer of ϕ along this path subject to $\|s\| \le \Delta$. Since ϕ decreases monotonically along the path, we simply find the intersection point with the trust region boundary, or we use the Newton step if the path lies entirely inside the trust region [13, section 6.4.2]. It is well known that, in the unconstrained setting, the dogleg method is sufficiently accurate to ensure first-order global convergence and a quadratic convergence rate [39]; the same properties carry over to the Byrd–Omojokun algorithm [33]. The cost of computing the Newton step can be high in large problems, and an approximation to it is an attractive option.

Steihaug's method. If the matrix B is not positive definite, there is no unconstrained minimizer of ϕ, and the Newton step s_n is not defined. In this case one can use an extension of CG proposed by Steihaug [40] (some elements of this method were also suggested by Toint [41]). The method is shown for the spherically constrained problem (3.4) in Algorithm 3.1.

ALGORITHM 3.1. *Steihaug's method for problem* (3.4)
Constant $\tilde{\epsilon} > 0$ is given
Start with $\tilde{s}_0 = 0$, $\tilde{r}_0 = -\tilde{g}$, and $\tilde{p}_0 = \tilde{r}_0$
if $\|\tilde{r}_0\| < \tilde{\epsilon}$ **then return** $\tilde{s}_0$
loop, starting with $j = 0$
  if $\tilde{p}_j^T \tilde{B} \tilde{p}_j \leq 0$
    then find $\tilde{\tau}$ so that $\tilde{s} = \tilde{s}_j + \tilde{\tau}\tilde{p}_j$ minimizes $\tilde{\phi}(\tilde{s})$
      and satisfies $\|\tilde{s}\| = \Delta$, and **return** $\tilde{s}$
  $\tilde{\alpha}_j = \tilde{r}_j^T \tilde{r}_j \,/\, \tilde{p}_j^T \tilde{B} \tilde{p}_j$
  $\tilde{s}_{j+1} = \tilde{s}_j + \tilde{\alpha}_j \tilde{p}_j$
  if $\|\tilde{s}_{j+1}\| \geq \Delta$
    then find $\tilde{\tau} \geq 0$ so that $\|\tilde{s}_j + \tilde{\tau}\tilde{p}_j\| = \Delta$,
      and **return** $\tilde{s}_j + \tilde{\tau}\tilde{p}_j$
  $\tilde{r}_{j+1} = \tilde{r}_j - \tilde{\alpha}_j \tilde{B} \tilde{p}_j$
  if $\|\tilde{r}_{j+1}\| \,/\, \|\tilde{r}_0\| < \tilde{\epsilon}$ **then return** $\tilde{s}_{j+1}$
  $\tilde{\beta}_{j+1} = \tilde{r}_{j+1}^T \tilde{r}_{j+1} \,/\, \tilde{r}_j^T \tilde{r}_j$
  $\tilde{p}_{j+1} = \tilde{r}_{j+1} + \tilde{\beta}_{j+1} \tilde{p}_j$
continue loop, after incrementing j

Note that when the trust region constraint is inactive and $\tilde{B}$ is positive definite, Algorithm 3.1 reduces to a CG computation of the Newton step. Steihaug proved [40] that $\tilde{\phi}$ decreases monotonically with each step $\tilde{s}_j$, and that every step moves farther away from the start point $\tilde{s}_0 = 0$, in the sense that $\|\tilde{s}_{j+1}\| > \|\tilde{s}_j\|$. These two properties imply that stopping the iteration as soon as the trust region is encountered is a sensible strategy. Observe that the first step $\tilde{s}_1$ is simply the Cauchy step $\tilde{s}_{cp}$ (assuming $\|\tilde{g}\| \geq \tilde{\epsilon}$), and that the residual vector $\tilde{r}_j$ equals the gradient of $\tilde{\phi}$ at $\tilde{s}_j$.

To adapt Steihaug's method for ellipsoidally constrained problem (3.1), we apply the transformations (3.2) and (3.5). If we additionally define the vectors

$$r_j \equiv (C^T C)^{1/2} \tilde{r}_j \quad \text{and} \quad p_j \equiv (C^T C)^{-1/2} \tilde{p}_j,$$

then we obtain Algorithm 3.2.

ALGORITHM 3.2. *Steihaug's method for problem* (3.1)
Constant $\epsilon > 0$ is given
Start with $s_0 = 0$, $r_0 = -g$, and $p_0 = (C^T C)^{-1} r_0$
if $\sqrt{r_0^T (C^T C)^{-1} r_0} < \epsilon$ **then return** s_0
loop, starting with $j = 0$
  if $p_j^T B p_j \leq 0$
    then find τ so that $s = s_j + \tau p_j$ minimizes $\phi(s)$
      and satisfies $\|Cs\| = \Delta$, and **return** s
  $\alpha_j = r_j^T (C^T C)^{-1} r_j \,/\, p_j^T B p_j$
  $s_{j+1} = s_j + \alpha_j p_j$
  if $\|C s_{j+1}\| \geq \Delta$
    then find $\tau \geq 0$ so that $\|C(s_j + \tau p_j)\| = \Delta$,
      and **return** $s_j + \tau p_j$
  $r_{j+1} = r_j - \alpha_j B p_j$
  if $\sqrt{r_{j+1}^T (C^T C)^{-1} r_{j+1}} \,/\, \sqrt{r_0^T (C^T C)^{-1} r_0} < \epsilon$ **then return** s_{j+1}
  $\beta_{j+1} = r_{j+1}^T (C^T C)^{-1} r_{j+1} \,/\, r_j^T (C^T C)^{-1} r_j$
  $p_{j+1} = (C^T C)^{-1} r_{j+1} + \beta_{j+1} p_j$
continue loop, after incrementing j

Algorithms 3.1 and 3.2 are equivalent and generate the same set of iterates. The first step s_1 of Algorithm 3.2 is again s_{cp}, and each residual r_j is the gradient of ϕ at s_j. If the trust region constraint is inactive and B is positive definite, then Algorithm 3.2 reduces to a *preconditioned* CG computation of the Newton step, with C^TC acting as the preconditioning matrix (although this "preconditioner" does not necessarily make the problem better conditioned).

We are now ready to discuss the implementation of Algorithm 2.1.

4. Detailed description of the algorithm. We begin by applying the solution techniques of the previous section to the vertical and horizontal subproblems, paying close attention to the linear algebra costs. Following that, we discuss other components of the algorithm that also require a careful implementation.

4.1. Vertical step. Let us replace (2.4)–(2.5) by the equivalent problem

$$\min_{v \in \mathbf{R}^n} \quad c_k^T A_k^T v + \frac{1}{2} v^T A_k A_k^T v \tag{4.1}$$

$$\text{subject to} \quad \|v\| \le \zeta \Delta_k, \tag{4.2}$$

and remember that we want the solution v_k to lie in the range space of A_k (to make it orthogonal to the horizontal step). To show that such a solution always exists, we note that the exact solution of (4.1)–(4.2) satisfies the equation

$$(A_k A_k^T + \rho I) v_* = -A_k c_k \tag{4.3}$$

for some $\rho \ge 0$ [20, p. 101]. If $\rho = 0$, then $v_* = -A_k(A_k^T A_k)^{-1} c_k$ is a solution, and it is clearly in the range space of A_k. On the other hand, if $\rho > 0$ then premultiplying both sides of (4.3) by Z_k^T gives $Z_k^T v_* = 0$, showing that v_* is in the range of A_k. Our implementation provides two options for computing an approximate vertical step v_k—the dogleg method and Steihaug's method—and in both cases we ensure that v_k lies in the range space of A_k.

When we compare (4.1)–(4.2) with the general formulation (3.1), we see that $C = I$, $B = A_k A_k^T$ is positive semidefinite, $\Delta = \zeta \Delta_k$, and $g = A_k c_k$. Substituting into (3.7), we obtain the Cauchy point

$$v_{cp} = -\alpha A_k c_k,$$

where

$$\alpha = \min \left\{ \frac{\zeta \Delta_k}{\|A_k c_k\|} \ , \ \frac{\|A_k c_k\|^2}{(A_k c_k)^T A_k A_k^T (A_k c_k)} \right\}.$$

To formulate a dogleg method for (4.1)–(4.2) we need to define the Newton step. Since the matrix $A_k A_k^T$ has $n - m$ linearly independent null vectors, there is a whole manifold of minimizers for (4.1), each satisfying the linear equations (2.2). In our implementation we choose the shortest step, uniquely given by

$$v_n = -A_k(A_k^T A_k)^{-1} c_k. \tag{4.4}$$

Together, v_{cp} and v_n define a dogleg method for the vertical subproblem.

The main computational expense of the dogleg method lies in solving the linear system involving $A_k^T A_k$ to obtain v_n. We provide two strategies for performing

this calculation. One is to compute a sparse Cholesky factorization of $A_k^T A_k$ (using the subroutines of Ng and Peyton [30]), which gives an accurate answer (unless A_k is extremely ill conditioned), but could take $O(n^3)$ operations in the worst case. The alternative is to estimate v_n in (4.4) by the CG method (this is equivalent to Craig's method [10] applied to (4.1)), which can be a cheaper calculation, although less accurate. Preconditioners would clearly enhance the performance of this inner CG iteration, but none have been implemented yet in our software. Regardless of whether Cholesky or CGs are used for solving (4.4) we include the following safeguard: if $\|v_n\|$ is smaller than $\|v_{cp}\|$, then serious roundoff errors may have occurred in the computation of v_n and we revert to using just the Cauchy segment of the dogleg path. This is, in essence, our application of the dogleg method for the computation of the vertical step.

Steihaug's method as given in Algorithm 3.2 can also be applied to the vertical subproblem. The Hessian $B = A_k A_k^T$ is of size $n \times n$, but all search directions p_j in Algorithm 3.2 lie in the m-dimensional range space of A_k. Our test results indicate Steihaug's method sometimes performs better than the dogleg method and is certainly an alternative worth considering.

4.2. Horizontal step. For convenience let us write W_k for the Hessian of the Lagrangian or its limited memory BFGS approximation, and define $\bar{g}_k \equiv g_k + W_k v_k$ and $\bar{\Delta}_k^2 \equiv \Delta_k^2 - \|v_k\|^2$. Then the horizontal subproblem (2.9)–(2.10) can be written as

$$\min_{u \in \mathbf{R}^{n-m}} \quad \bar{g}_k^T Z_k u + \frac{1}{2} u^T Z_k^T W_k Z_k u \tag{4.5}$$

$$\text{subject to} \quad \|Z_k u\| \le \bar{\Delta}_k. \tag{4.6}$$

This is a trust region subproblem with ellipsoidal constraint; in terms of the general formulation (3.1) we have $C = Z_k$, $B = Z_k^T W_k Z_k$, $g = Z_k^T \bar{g}_k$, $p = n$, and $q = n - m$. Using (3.7) we see that the Cauchy point is given by

$$u_{cp} = -\alpha \bar{u}_k, \quad \text{with} \quad \bar{u}_k \equiv (Z_k^T Z_k)^{-1} Z_k^T \bar{g}_k \tag{4.7}$$

and α determined by the following rule:

if $\bar{u}_k^T Z_k^T W_k Z_k \bar{u}_k \le 0$
then

$$\alpha = \frac{\bar{\Delta}_k}{\|Z_k \bar{u}_k\|}$$

else

$$\alpha = \min \left\{ \frac{\bar{\Delta}_k}{\|Z_k \bar{u}_k\|}, \frac{\bar{g}_k^T Z_k \bar{u}_k}{\bar{u}_k^T Z_k^T W_k Z_k \bar{u}_k} \right\}.$$

It will be seen in section 4.3 that directly computing $(Z_k^T Z_k)^{-1}$ can be very expensive, but the cost of multiplying Z_k and Z_k^T by a vector is much lower; therefore, we solve for $\bar{u}_k$ approximately by the CG method.

This means that u_{cp} is just an approximation to the Cauchy step, raising the question whether global convergence of the Byrd–Omojokun algorithm is still assured. Analysis similar to that in [6] shows that an approximation to u_{cp} is adequate for global convergence as long as the error in its computation tends to zero as fast as $Z_k^T \bar{g}_k$. Therefore, the stop tolerance in the CG iteration used for computing (4.7) is set to a small fraction of $\|Z_k^T \bar{g}_k\|$.

The horizontal step should make at least as much a reduction in (4.5) as the Cauchy step u_{cp} does, and preferably more in order to accelerate convergence. The dogleg method is only applicable to (4.5)–(4.6) in cases where W_k can be guaranteed positive definite, but since Steihaug's method can be applied for any W_k, we adopt it as our sole option for computing the horizontal step. To define Steihaug's method we only need to make the appropriate substitutions from (4.5)–(4.6) into Algorithm 3.2. This generates a set of iterates $\{u_j\}$ which estimate the solution of (4.5)–(4.6); the final iterate is then multiplied by Z_k to define the horizontal step.

A close examination reveals that this Steihaug iteration can be made more efficient by directly generating iterates and search directions of the form $Z_k u_j$ and $Z_k p_j$. To see this, note that the matrices B and C appearing in Algorithm 3.2 are given in this case by $B = Z_k^T W_k Z_k$ and $C = Z_k$. Thus, it is necessary to first multiply the quantities u_j (or s_j in the notation of Algorithm 3.2) and p_j by Z_k at every place they appear in the iteration. Instead of computing u_j and p_j themselves, we can work with $Z_k u_j$ and $Z_k p_j$ directly, saving the cost of multiplying by Z_k at every iteration of Steihaug's method. These ideas are implemented in Algorithm 4.1, where the symbols $(Zu)_j$ and $(Zp)_j$ denote the iterates and search directions in the desired form. By construction, all search directions $(Zp)_j$ lie in the null space of the constraint matrix, i.e., in the span of Z_k. Algorithm 4.1 also expresses the ellipsoidally scaled residual explicitly as a vector t_j, a quantity we shall examine more closely in a moment.

ALGORITHM 4.1. *Computation of the horizontal step*

Constants $\epsilon_{h_0} > 0$ and $\epsilon_h > 0$ are given
Start with $(Zu)_0 = 0$ and $r_0 = -Z_k^T \bar{g}_k$
Solve $(Z_k^T Z_k) t_0 = r_0$ for t_0, and set $(Zp)_0 = Z_k t_0$
if $\sqrt{r_0^T t_0} < \epsilon_{h_0}$
  then use (4.7) and **return** the Cauchy step $Z_k u_{cp}$
loop, starting with $j = 0$
  if $(Zp)_j^T W_k (Zp)_j \le 10^{-8} (Zp)_j^T (Zp)_j$
    then find τ so that $(Zu) = (Zu)_j + \tau (Zp)_j$ minimizes (4.5)
      and satisfies $\|(Zu)\| = \bar{\Delta}_k$, and **return** (Zu)
  $\alpha_j = r_j^T t_j \,/\, (Zp)_j^T W_k (Zp)_j$
  $(Zu)_{j+1} = (Zu)_j + \alpha_j (Zp)_j$
  if $\|(Zu)_{j+1}\| \ge \bar{\Delta}_k$
    then find $\tau \ge 0$ so that $\|(Zu)_j + \tau (Zp)_j\| = \bar{\Delta}_k$
      and **return** $(Zu)_j + \tau (Zp)_j$
  $r_{j+1} = r_j - \alpha_j Z_k^T W_k (Zp)_j$
  Solve $(Z_k^T Z_k) t_{j+1} = r_{j+1}$ for t_{j+1}
  if $\sqrt{r_{j+1}^T t_{j+1}} < \epsilon_h \sqrt{r_0^T t_0}$ **then return** $(Zu)_{j+1}$
  $\beta_{j+1} = r_{j+1}^T t_{j+1} \,/\, r_j^T t_j$
  $(Zp)_{j+1} = Z_k t_{j+1} + \beta_{j+1} (Zp)_j$
continue loop, after incrementing j

As in the computation of the Cauchy step (4.7), we calculate t_{j+1} by approximately solving $(Z_k^T Z_k) t_{j+1} = r_{j+1}$ by the CG method. This inner loop is terminated when its residual becomes smaller than a fraction of $\epsilon_h \sqrt{r_0^T t_0}$, which is the stop tolerance in Algorithm 4.1. Since $r_0^T t_0$ is proportional to $\|Z_k^T g_k\|^2$, an analysis similar to that in [11] shows that this tolerance will give the algorithm a Q-quadratic asymptotic rate of convergence in the case when second derivatives are used for W_k.

Unscaled Steihaug iteration. The approximate solution to the horizontal subproblem found by Algorithm 4.1 turns out to be quite adequate for solving (1.1), but the calculation of the scaled residual $(Z_k^T Z_k)t_{j+1} = r_{j+1}$ is very time consuming in many problems. This CG subiteration is effectively a triply nested loop since Algorithm 4.1 lies within the main loop of Algorithm 2.1. Thus, we are motivated to search for less expensive alternatives for computing u_k that preserve the robustness of the trust region algorithm. Our most successful idea, which we call the *unscaled Steihaug iteration*, is to eliminate the matrix $(Z_k^T Z_k)^{-1}$ from Algorithm 4.1, thereby setting $t_{j+1} = r_{j+1}$. This can be viewed as preconditioning with the matrix $Z_k^T Z_k$; however, this "preconditioner" is not chosen for the usual purpose of improving the condition number of $Z_k^T W_k Z_k$ but only to reduce the cost of the iteration. Indeed, ignoring the matrix $(Z_k^T Z_k)^{-1}$ can be problematic, and must be done with certain precautions.

If $Z_k^T W_k Z_k$ is positive definite, and if the unscaled Steihaug iteration remains inside the scaled trust region $\|Zu\| \leq \bar{\Delta}$, then the solution obtained is simply the Newton step for (4.5). In this case the "preconditioner" has had no effect on the answer, and savings in computing time have been obtained. But if the unscaled Steihaug method terminates by reaching the trust region boundary, it is possible that the resulting step is poor compared with the solution obtained by the regular Steihaug iteration. The source of the difficulty is that, by removing the term $(Z_k^T Z_k)^{-1}$, the inequalities $\|(Zu)_{j+1}\| > \|(Zu)_j\|$ may no longer hold for all j. Therefore, an early iterate $(Zu)_j$ could leave the trust region and a later iterate $(Zu)_{j+s}$, giving a significantly lower value of (4.5), may subsequently return to the trust region. (It is still true that (4.5) is reduced monotonically by the sequence of steps $(Zu)_j$.)

To overcome this potential problem, we start with the unscaled Steihaug iteration but discard all the information generated if it leaves the scaled trust region, instead applying Algorithm 4.1. To minimize wasted effort, the unscaled Steihaug method is tried only if the last step d_k was accepted and Δ_{k+1} is the same as Δ_k. In particular, this ensures the method is tried as the iterates converge to a solution and the trust region becomes inactive. As a further precaution the stop test requires two successive residuals to be less than the tolerance ϵ_h. Numerical tests comparing both variations of the Steihaug iteration are given in section 5.1.

4.3. Computation of Z_k. Recall that Z_k is the $n \times (n-m)$ matrix spanning the manifold tangent to all constraints at x_k; this means it satisfies $A_k^T Z_k = 0$. A numerically stable way of calculating Z_k from A_k is by means of the QR factorization, which ensures that the columns of Z_k are orthogonal. Although a sparse QR factorization appears feasible, we suspect that it may be costly in many cases and have deferred exploring this option to a later date. The idea we have pursued is one of direct elimination (see for example [20, p. 234]). First, partition A_k^T into

$$A_k^T = [B_k \;\; N_k], \tag{4.8}$$

where the $m \times m$ basis matrix B_k is nonsingular (for simplicity (4.8) assumes that the basis B_k is formed by the first m columns of A_k^T). Now define Z_k to be

$$Z_k = \begin{bmatrix} -B_k^{-1} N_k \\ I \end{bmatrix}, \tag{4.9}$$

which satisfies $A_k^T Z_k = 0$, although the columns of Z_k are not orthogonal. Our software uses Harwell subroutine MA28 [15] to first choose m columns of A_k^T that

define the basis (MA28 can do this even though A_k^T is not square), and then compute the sparse LU factorization of B_k. MA28 selects the basis by considering both the resulting sparsity and the size of the pivots in the LU factorization of B_k. Forming Z_k explicitly could be very expensive since $B_k^{-1}N_k$ requires $n-m$ back-solves, and could destroy sparsity. Instead, the implicit representation (4.9) of Z_k is employed to compute matrix-vector products of the form $Z_k u$ and $Z_k^T x$ (Murtagh and Saunders use the same concept in [29]). For example, $Z_k u$ is obtained by computing $b = N_k u$, solving $B_k y = -b$ for y using the sparse LU factors of B_k, and appending u for the last components.

MA28 is designed to take advantage of situations where matrix values change but the sparsity pattern remains the same. This will be the case in our algorithm as long as we can keep the same m columns to serve as the basis matrix. At the beginning of the algorithm we do the costly work of choosing columns for B_0 and symbolically factoring it, and then use this same basis for subsequent iterates, saving many computations. But it is possible for iterates to change enough that B_k becomes very ill conditioned, in which case it is necessary to choose a different set of columns to define the basis. We have devised the following heuristic to determine when to change basis and refactor.

Heuristic I: Strategy for changing basis. When solving systems of the form $B_k y = b$ or $B_k^T y = b$ (which arise in the horizontal subproblem), we compute the ratio $\|b\|/\|y\|$. This is a number that lies somewhere between the largest and smallest singular values of B_k. During the lifetime of a particular choice of basis we remember the largest and smallest ratios seen, then divide these to estimate the condition number of the current B_k. If the estimate becomes greater than 10^2, then on the next iteration we let MA28 try and choose a new basis and symbolically refactor.

To prevent this heuristic from interfering with good algorithm behavior we overrule it, and do not change basis, when the step is inside the trust region and is accepted by the algorithm. The performance of the heuristic is documented in section 5.1.

4.4. Limited memory approximations. Our software supplies an alternative when second derivatives are unavailable or prohibitively expensive to calculate. A limited memory BFGS (ℓ-BFGS) approximation to $\nabla_x^2 L_k$ is maintained and used in the role of W_k. The estimate is initialized with a scalar multiple of the identity matrix, then is updated implicitly as described in [27]. Only information reflecting the last t updates is kept, which requires $O(tn)$ storage locations and $O(tn)$ operations for updating W_k and multiplying W_k by a vector. Limited memory approximations give rise to a linear convergence rate, so there can be a significant loss in performance compared with using second derivatives. Previous research [22], [27] has shown that the choice $t \in [4, 10]$ gives good performance, and our testing suggests ℓ-BFGS is a viable option within the framework of this algorithm.

The software implementation uses the compact representations of limited memory BFGS matrices described in [4]. Every (implicit) BFGS update is of the form

$$W_{k+1} = W_k - \frac{W_k d_k d_k^T W_k}{d_k^T W_k d_k} + \frac{y_k y_k^T}{y_k^T d_k}, \tag{4.10}$$

where $d_k = x_{k+1} - x_k$ and $y_k = (g_{k+1} - A_{k+1}\lambda_{k+1}) - (g_k - A_k\lambda_k)$. If $y_k^T d_k$ is not positive, we enforce this condition by modifying y_k according to Powell's damping strategy [36]:

if $d_k^T y_k < 0.2\, d_k^T W_k d_k$
then $y_k \leftarrow \theta y_k + (1-\theta) W_k d_k$,

where

$$\theta = \frac{0.8\, d_k^T W_k d_k}{d_k^T W_k d_k - d_k^T y_k}.$$

There is also a danger that the last term in (4.10) becomes too large. To avoid this the update is not performed if

$$10^{-8}\, y_k^T y_k \;\geq\; y_k^T d_k.$$

4.5. Lagrange multipliers. At every new iterate x_{k+1} we compute least squares Lagrange multiplier estimates λ_{k+1} by solving

$$(A_{k+1}^T A_{k+1})\lambda_{k+1} = A_{k+1}^T g_{k+1}. \tag{4.11}$$

This linear system is solved by either sparse Cholesky factorization or by the CG method. The accuracy of the Lagrange multipliers needs to be good enough to allow proper evaluation of the first-order KKT measure $\|g_k - A_k\lambda_k\|_\infty$. Therefore, when the CG method is used, the iteration is stopped when the residual in (4.11) becomes less than $10^{-2} * \max\{\|c_k\|_\infty, \|g_k - A_k\lambda_k\|_\infty\}$.

4.6. Merit function and choice of penalty parameter. The merit function is used to decide whether the step d_k makes sufficient progress toward the solution of problem (1.1). We follow the Byrd and Omojokun algorithm in [32] and use as merit function

$$\psi(x,\mu) = f(x) + \mu\|c(x)\|, \tag{4.12}$$

where $\mu > 0$ is called the penalty parameter; recall that $\|\cdot\|$ denotes the ℓ_2 norm. Since the last term in (4.12) is not squared, this merit function is not differentiable. It is also exact: in a neighborhood of a solution point x_*, and for $\mu > \|\lambda_*\|_\infty$, the minimizer of $\psi(x,\mu)$ is precisely x_*. We will see below that this merit function has the advantage that the ℓ_2 norm used to penalize the constraints is compatible with the ℓ_2 norm used in the formulation (2.4) of the vertical step.

We need to decide how the penalty parameter will be chosen at each iteration of the algorithm. As is commonly done (see, for example, [24], [37]) we first compute a step d_k and then choose μ large enough that d_k results in a reduction of the model. To be more precise, let us replace f by the model objective (2.1) and linearize the constraints in (4.12) to give the model merit function at x_k,

$$\hat{\psi}_k(d,\mu) \equiv d^T g_k + \frac{1}{2} d^T W_k d + \mu\|A_k^T d + c_k\|,$$

where, as before, W_k denotes $\nabla_x^2 L_k$ or an ℓ-BFGS approximation to it. We now define the predicted reduction p_red in the model merit function $\hat{\psi}_k$ by

$$\begin{aligned} \text{p_red} &= \hat{\psi}_k(0,\mu) - \hat{\psi}_k(d_k,\mu) \\ &= -d_k^T g_k - \frac{1}{2} d_k^T W_k d_k + \mu\left(\|c_k\| - \|A_k^T v_k + c_k\|\right), \end{aligned} \tag{4.13}$$

where we have used (2.11) and the fact that $A_k^T Z_k = 0$. The term inside the parentheses is nonnegative because the dogleg or Steihaug methods used in the vertical step computation reduce the objective $\|A_k^T v + c_k\|$ from its initial value $\|c_k\|$ at $v = 0$. We now compute the trial value

$$\mu^+ = \max\left\{\mu_k,\, 0.1 + \frac{d_k^T g_k + \frac{1}{2} d_k^T W_k d_k}{\|c_k\| - \|A_k^T v_k + c_k\|}\right\}, \tag{4.14}$$

where μ_k is the penalty parameter at the previous iteration. It is clear from (4.13) that for this value of the penalty parameter the predicted reduction p_red is positive. (The term inside the parentheses in (4.13) can be zero when $v_k = 0$, which only happens when $c_k = 0$. In this case (4.14) is not used because p_red is made positive from the decrease that d_k makes in the objective of the horizontal subproblem (4.5).)

Before accepting the step, Algorithm 2.1 tests whether this choice of the penalty parameter leads to an actual decrease in the merit function (and not just in the model). We define the actual reduction a_red in the merit function by

$$\begin{aligned} \text{a_red} &= \psi(x_k, \mu^+) - \psi(x_k + d_k, \mu^+) \\ &= f(x_k) - f(x_k + d_k) + \mu^+(\|c(x_k)\| - \|c(x_k + d_k)\|). \end{aligned} \tag{4.15}$$

If a_red is sufficiently positive in the sense that a_red $\geq \eta$ p_red, d_k is accepted and we set $\mu_{k+1} = \mu^+$; otherwise, the second-order correction procedure described in section 4.7 is invoked.

There is a danger of v_k being so small that the denominator of (4.14) approaches zero and μ^+ blows up, but this will normally happen only if the constraints are already nearly satisfied; in this case there is no point in further penalizing the constraints and we are justified in keeping $\mu^+ = \mu_k$. These guidelines for choosing the penalty parameter are sufficient for robust algorithm performance, but we have observed that the following modifications can often help the algorithm reach a solution faster.

Heuristic II: Update of penalty parameter. For certain problems with nonlinear constraints, convergence can be very slow because the penalty parameter is not increased enough. Formula (4.14) computes μ^+ using a linearization of the constraints, but if these are sufficiently nonlinear that $\|c(x_k)\| - \|c(x_k + d_k)\|$ is not well approximated by $\|c_k\| - \|A_k^T d_k + c_k\|$, then the step may increase the merit function and be rejected. In these cases it is advantageous to give more weight to constraint satisfaction by further increasing the penalty parameter. However, there is a risk that an unnecessarily large penalty parameter will cause iterates to follow the constraint manifold too closely, resulting in an excessive number of iterations for convergence. In the following, heuristic $k - s$, with $s \geq 1$, denotes the index of the most recent iterate at which the step d_{k-s} was accepted by the algorithm.

ALGORITHM 4.2. *Heuristic for modifying the penalty parameter*
Compute μ^+ using (4.14)
if $\mu^+ > \mu_k$ **and** $\mu^+ < 5\mu_k$ **and** $\mu_k > \mu_{k-s}$ **and** $\|c_k\| > \frac{1}{5}\|c_{k-s}\|$
and one or both of the last two steps were rejected
then $\mu^+ \leftarrow \min\{5\mu_k, \mu^+ + 25(\mu^+ - \mu_{k-s})\}$
if $\mu^+ = \mu_k$ **and** $\|v_k\| < \zeta\Delta_k/10$ **and** $\|c_k\|_\infty < 10^4\epsilon$
then $\mu^+ \leftarrow \max\{\mu_0, \bar{\mu}, \|\lambda_k\|\}$
(ϵ is the stop tolerance in Algorithm 2.1 and $\bar{\mu}$ is the second term inside the brackets in (4.14).)

The first **if** statement allows a further increase in the penalty parameter if not much progress ($\|c_k\| > \frac{1}{5}\|c_{k-s}\|$) has been made toward feasibility—a condition that

indicates that the constraint error is not being weighed strongly enough in the merit function—and if one of the last two steps failed, implying that the linearized constraint model is not a good fit. Safeguards are included to prevent the penalty parameter from changing too rapidly ($\mu^+ < 5\mu_k$) and to permit increases only when the penalty parameter is already on the rise ($\mu_k > \mu_{k-s}$). The second **if** statement permits reductions of the penalty parameter when iterates are close to satisfying the equality constraints; furthermore, the reduced μ^+ is never smaller than $\|\lambda_k\|$ (which is an estimate of the theoretical safe lower bound on μ_*) or smaller than the value $\bar{\mu}$ needed to assure that the step is a descent direction for the model merit function. In section 5.1 we present numerical results comparing this heuristic with the simpler rule (4.14).

4.7. Second-order correction. The merit function (4.12) is nondifferentiable and can give rise to the Maratos effect [28], which results in poor performance on some problems. The effect can be overcome by computing a second-order correction term (see [20, pp. 393–396] or [8] for details) and adding it to d_k, giving the new trial step

$$d_{\text{SOC}} = d_k - A_k(A_k^T A_k)^{-1} c(x_k + d_k). \tag{4.16}$$

The linear system involving $A_k^T A_k$ is solved just as in the computation of v_n in section 4.1: either by sparse Cholesky factorization or by CG. Computation of the second-order correction term is costly, so we only try d_{SOC} if the step d_k has been rejected, x_k is already nearly feasible, and $\|v_k\|$ is small compared with $\|Z_k u_k\|$. These conditions attempt to identify the occurrence of the Maratos effect and are much simpler than the rules given by Fletcher [19]. Combining this with the ideas of section 4.6, the final **if** statement of Algorithm 2.1 is more precisely defined by the following rules.

Compute μ^+ from Algorithm 4.2, a_red from (4.15), and p_red from (4.13)
if $\dfrac{\text{a_red}}{\text{p_red}} \geq \eta$
  then $x_{k+1} = x_k + d_k$, $\mu_{k+1} = \mu^+$, $\Delta_{k+1} \geq \Delta_k$
  else if $\|v_k\| \leq 0.8\zeta\Delta_k$ **and** $\|v_k\| \leq 0.1\|Z_k u_k\|$
    then compute d_{SOC} using (4.16)
      recalculate a_red with $d_k \leftarrow d_{\text{SOC}}$
      if $\dfrac{\text{a_red}}{\text{p_red}} \geq \eta$
        then $x_{k+1} = x_k + d_{\text{SOC}}$,
          $\mu_{k+1} = \mu^+$, $\Delta_{k+1} \geq \Delta_k$
        else $x_{k+1} = x_k$,
          $\mu_{k+1} = \mu_k$, $\Delta_{k+1} < \|d_k\|$
    else $x_{k+1} = x_k$, $\mu_{k+1} = \mu_k$, $\Delta_{k+1} < \|d_k\|$.

4.8. Modifying the trust region. We will now describe in detail how to update the trust region radius. Our numerical experience suggests that the following aggressive strategy often saves many iterations of the algorithm and is rarely harmful. When a step d_k (or d_{SOC}) is accepted we increase the trust radius according to the following rule:

if $\dfrac{\text{a_red}}{\text{p_red}} \geq 0.9$
  then $\Delta_{k+1} = \max\{10\,\|d_k\|, \Delta_k\}$
else if $\dfrac{\text{a_red}}{\text{p_red}} \geq 0.3$
  then $\Delta_{k+1} = \max\{2\,\|d_k\|, \Delta_k\}$
else $\Delta_{k+1} = \Delta_k$.

When the step is rejected, Δ_{k+1} is reduced to a fraction of the failed step length such that $\Delta_{k+1} \in [0.1\,\|d_k\|, 0.5\,\|d_k\|]$. The precise value is computed by assuming that the ratio of actual to predicted reduction is a linear function h of the step length $\|d\|$, satisfying $h(0) = 1$ and $h(\|d_k\|) = (\text{a_red}/\text{p_red})$, and then finding the value of $\|d\|$ where h equals η. Thus, the computation of Δ_{k+1} for a failed step is as follows:

Set $\Delta_{k+1} = \dfrac{1-\eta}{1-(\text{a_red}/\text{p_red})}\|d_k\|$
if $\Delta_{k+1} > 0.5\,\|d_k\|$
  then $\Delta_{k+1} \leftarrow 0.5\,\|d_k\|$
else if $\Delta_{k+1} < 0.1\,\|d_k\|$
  then $\Delta_{k+1} \leftarrow 0.1\,\|d_k\|$.

For other strategies of updating the trust region radius see [13, section 6.4.3] and [9].

4.9. Summary of the algorithm implementation. The previous discussion has provided a detailed description of our implementation of the Byrd–Omojokun algorithm, including several options for approximately solving the subproblems that arise. Below is an expanded description of Algorithm 2.1, which closely reflects the form of our code.

ALGORITHM ETR. *Trust region algorithm for equality constrained optimization*
Constants $\epsilon > 0$, and η, $\zeta \in (0,1)$ are given
Choose x_0 and $\Delta_0, \mu_0 > 0$
loop, starting with $k = 0$
  Compute f_k, c_k, g_k, and A_k
  if Heuristic I (see section 4.3) indicates the basis B_k for A_k is ill conditioned
    then Find a new basis and its LU factors
    else Compute the LU factors of the current basis
  (The null-space basis Z_k is now defined by (4.9))
  Compute multipliers λ_k by solving $(A_k^T A_k)\lambda_k = A_k^T g_k$
    Option 1: solve the linear system by Cholesky factorization
    Option 2: solve the system by CG
  if $\|g_k - A_k\lambda_k\|_\infty < \epsilon$ **and** $\|c_k\|_\infty < \epsilon$ **then stop**
  Compute the vertical step v_k
    Option 1: use dogleg method, obtain the Newton step v_n by Cholesky
    Option 2: use dogleg method, obtain the Newton step v_n by CG
    Option 3: use Steihaug's method
  Compute W_k
    Option 1: set $W_k = \nabla_x^2 L_k(x_k, \lambda_k)$
    Option 2: update the limited memory (ℓ-BFGS) approximation W_k
  Compute the horizontal step u_k with the unscaled Steihaug iteration (section 4.2)
  Set $d_k = v_k + Z_k u_k$
  Compute μ^+ using Heuristic II as given in Algorithm 4.2
  Compute a_red from (4.15) and p_red from (4.13)
  if $\dfrac{\text{a_red}}{\text{p_red}} \geq \eta$
    then $x_{k+1} = x_k + d_k$, $\mu_{k+1} = \mu^+$, and expand trust region
    else if $\|v_k\| \leq 0.8\zeta\Delta_k$ **and** $\|v_k\| \leq 0.1\|Z_k u_k\|$
      then Compute the second-order correction d_{SOC} from (4.16):
        use the same option as was chosen to compute λ_k

Recalculate a_red (4.15) at d_{SOC}
if $\frac{\text{a_red}}{\text{p_red}} \geq \eta$
then $x_{k+1} = x_k + d_{\text{SOC}},\ \mu_{k+1} = \mu^+$, expand trust region
else $x_{k+1} = x_k,\ \mu_{k+1} = \mu_k$, contract trust region
else $x_{k+1} = x_k,\ \mu_{k+1} = \mu_k$, and contract trust region
(The expansion and contraction of the trust region Δ_{k+1} is made according to the rules of section 4.8)
continue loop, after incrementing k

Let us reiterate our strategy for solving the linear systems involving the matrix $A_k^T A_k$ occurring in the algorithm. We provide two ways (Cholesky factorization and CG) of solving these linear systems, which occur in the dogleg component of the vertical subproblem v_k, in the computation of the Lagrange multipliers λ_k, and in the second-order correction step d_{SOC}. For these systems we are consistent in our choice of linear solver: if the vertical step is computed by Cholesky factorization, then the Cholesky factors are saved and used for solving the other systems; otherwise, CG is used to solve all three linear systems.

For testing purposes, the parameters occurring in the algorithm were selected as follows. The stop tolerance was $\epsilon = 10^{-5}$, and we set $\eta = 0.1$, $\zeta = 0.8$, and $\Delta_0 = 1$. The penalty parameter was always started at $\mu_0 = 1$. In the horizontal step computation (Algorithm 4.1) we used stop tolerances of $\epsilon_h = 10^{-2}$ and $\epsilon_{h_0} = 10^{-10}$. The computation of the horizontal Cauchy step in (4.7) requires approximately solving the system $Z_k^T Z_k$ by CG; this inner iteration was stopped when the residual was less than $\max\{10^{-10}, 10^{-8} * \min\{1, \|Z_k^T \bar{g}_k\|\}\}$. A more relaxed CG tolerance of $\max\{10^{-7}, 10^{-2} * \min\{1, r_0^T t_0\}\}$ was used to solve for t_{j+1} in Algorithm 4.1. When CG was used to obtain Lagrange multipliers λ_{k+1}, we stopped iterating when the residual became smaller than $10^{-2} * \max\{\|c_k\|_\infty, \|g_k - A_k\lambda_k\|_\infty\}$. When CG was used to obtain v_n, the residual had to be less than $\max\{10^{-8}, 10^{-2} * \min\{1, \|c_k\|\}\}$, while the more stringent tolerance $\max\{10^{-8}, 10^{-7} * \min\{1, \|c(x_k + d_k)\|\}\}$ was used to compute d_{SOC}.

5. Test results. We tested our software implementation on a set of difficult nonlinear equality constrained problems drawn from the CUTE collection [1]. The problems are divided into two groups, large and small, based on the number of unknowns. Table 5.1 presents a brief overview of the large problems. An asterisk by the problem name means the original CUTE problem has been modified for our use; for example, nonlinear inequalities may have been dropped or changed into equalities, or the problem parameters may have been altered to obtain a more ill conditioned or badly scaled problem. (All problems can be obtained from the authors in electronic form.) The number of nonzero elements in A is denoted by $nnz(A)$. The value of the objective at the solution is given under $f(x_*)$ to the number of significant figures that all optimization methods agreed upon. Note that in some problems $n = m$; in these we seek the root of a system of nonlinear equations. The CUTE software interface computes all derivatives by analytic formulae and provides a means of checking results by comparing them with solutions obtained with other software packages.

Our main concern in this paper is the development of an algorithm suitable for large-scale optimization; therefore, we begin by examining performance for the problems in Table 5.1 and defer the testing of small optimization problems until section 5.2. We first present numerical results that explore the performance of three options introduced in the previous sections: the use of the unscaled Steihaug iteration; Heuristic I—the strategy for automatically choosing a new basis; and Heuristic II—the aggres-

TABLE 5.1
Description of large test problem set.

Problem name	n	m	$nnz(A)$	*Objective*	*Constraints*	$f(x_*)$
GENHS28	300	298	894	nonlinear	linear	3.31481×10^{1}
HAGER2	10001	5001	15001	nonlinear	linear	4.3208×10^{-1}
HAGER3	8001	4001	12001	nonlinear	linear	1.4096×10^{-1}
ORTHREGA	2053	1024	7168	nonlinear	nonlinear	5.66143×10^{2}
ORTHREGC	1005	500	3500	nonlinear	nonlinear	1.87906×10^{1}
ORTHREGD	203	100	500	nonlinear	nonlinear	3.05079×10^{1}
∗ORTHRGDS	203	100	500	nonlinear	nonlinear	4.037×10^{1}
∗ORTHRGDM	4003	2000	10000	nonlinear	nonlinear	1.55533×10^{2}
∗ORTHRGFB	1205	400	3200	nonlinear	nonlinear	1.62006×10^{1}
∗OPTCTRL3	122	83	283	nonlinear	nonlinear	1.34806×10^{3}
∗OPTCTRL6	122	83	283	nonlinear	nonlinear	2.04802×10^{3}
DTOC1ND	2998	2000	13976	nonlinear	nonlinear	2.37703×10^{1}
DTOC2	2998	2000	7988	nonlinear	nonlinear	4.9723×10^{-1}
DTOC3	14999	10000	34995	nonlinear	linear	2.3526×10^{2}
DTOC4	14999	10000	34995	nonlinear	nonlinear	2.868×10^{0}
DTOC5	9999	5000	14998	nonlinear	nonlinear	1.534×10^{0}
DTOC6	2001	1001	3001	nonlinear	nonlinear	1.7176×10^{0}
EIGENA2	110	55	1000	nonlinear	nonlinear	0.00000×10^{0}
EIGENC2	462	231	9261	nonlinear	nonlinear	0.00000×10^{0}
ARTIF	1002	1002	3002	-	nonlinear	-
BRATU3D	27000	27000	158712	-	nonlinear	-
BROYDNBD	5000	5000	34984	-	nonlinear	-
HTRODEQ	4656	4656	27361	-	nonlinear	-

* Test problem was created specifically for this paper.

sive strategy for updating the penalty parameter. After we have established that these options perform well on a carefully chosen sample of our test problems, we describe the performance of the fully developed algorithm on the whole test set. All our computations were performed in FORTRAN double precision on a Silicon Graphics workstation running IRIX 5.2 using a 150 MHz MIPS R4400 processor and 64Mbytes RAM.

5.1. Testing of heuristics. Let us begin by considering the unscaled Steihaug iteration introduced in section 4.2 to reduce the cost of the horizontal step computation. Since this iteration ignores the matrix $(Z_k^T Z_k)^{-1}$ it generally performs more steps than the regular Steihaug method. However, Table 5.2 shows that the many steps of the unscaled Steihaug iteration are so much cheaper to compute that there is still an overall savings in execution time. These experiments used the same stop tolerance $\epsilon_h = 10^{-2}$ for both iterations, although the preconditioned algorithm requires that two consecutive residuals be small enough before stopping, as discussed in section 4.2. Table 5.2 gives the results on a selection of problems that highlight the differences between the two approaches and considers the use of both exact Hessians and ℓ-BFGS approximations. The numbers in the table give the total CPU time for solving the problem (*CPU*) and the cumulative total number of horizontal step iterations (*h itr*) of Algorithm 4.1. (The numbers were obtained by using the fastest option for computing the vertical step.)

We observe that execution time nearly always decreases, sometimes dramatically, using the unscaled Steihaug iteration. Since the restrictions described in section 4.2 on when to use the unscaled Steihaug iteration ensure that its application does not alter the number of function and gradient evaluations needed to solve a problem, we performed all subsequent experiments with the unscaled Steihaug iteration.

TABLE 5.2
Comparison of unscaled Steihaug and regular Steihaug iterations.

Problem name	Exact Hessians				ℓ-BFGS approximations			
	Unscaled Steihaug		Steihaug		Unscaled Steihaug		Steihaug	
	h itr	*CPU*	*h itr*	*CPU*	*h itr*	*CPU*	*h itr*	*CPU*
HAGER2	11	18.7 sec	11	18.7 sec	13	19.3 sec	9	20.2 sec
HAGER3	12	14.4 sec	12	14.4 sec	16	14.3 sec	9	15.8 sec
ORTHREGA	128	168.9 sec	118	170.9 sec	485	275.0 sec	313	292.2 sec
ORTHREGC	179	23.3 sec	111	30.1 sec	424	42.8 sec	160	56.0 sec
ORTHRGDM	40	33.2 sec	16	35.7 sec	41	42.6 sec	19	46.9 sec
ORTHRGFB	71	27.7 sec	71	27.7 sec	213	32.6 sec	128	37.3 sec
DTOC1ND	33	39.0 sec	26	41.6 sec	83	85.8 sec	60	93.5 sec
DTOC2	7139	398.3 sec	331	949.9 sec	18008	1311.2 sec	962	3023.3 sec
DTOC5	11	78.6 sec	10	79.3 sec	14	105.7 sec	9	104.7 sec
DTOC6	243	134.3 sec	246	139.1 sec	321	206.8 sec	314	201.8 sec
EIGENC2	582	38.6 sec	408	101.3 sec	5022	234.0 sec	2039	506.7 sec

TABLE 5.3
Performance of strategies for choosing a basis.

Problem name	Never change basis				Always new basis				Heuristic I			
	nb	*bas*	*hor*	*CPU*	*nb*	*bas*	*hor*	*CPU*	*nb*	*bas*	*hor*	*CPU*
ORTHREGA	1	16%	67%	164.3	20	67%	17%	194.6	15	63%	20%	168.9
ORTHREGC	1	3%	93%	165.0	15	40%	35%	27.9	4	24%	48%	23.3
ORTHRGDM	1	70%	14%	33.1	7	84%	7%	57.9	1	70%	14%	33.2
ORTHRGFB	1	2%	95%	242.5	15	38%	36%	27.2	3	17%	57%	27.7
DTOC6	1	8%	89%	141.7	25	9%	87%	135.5	21	9%	88%	134.3
EIGENC2	1	16%	76%	38.8	10	32%	63%	48.7	1	16%	76%	38.6

Let us now test Heuristic I—the strategy for choosing a new basis with Harwell subroutine MA28 described in section 4.3. We compare it with the alternative strategies of always using the same basis and always letting MA28 choose a basis freely. We expect the last strategy to produce a better conditioned basis and therefore a better-conditioned horizontal subproblem (4.5)–(4.6). Heuristic I seeks to balance the cost of using MA28 at every new iterate against the cost of solving badly conditioned horizontal subproblems.

Table 5.3 shows a comparison between the three strategies for those test problems in which it made a difference. (In the other problems either the basis chosen by MA28 never varied for the three strategies, or the relative cost of executing MA28 was insignificant.) For each strategy we list the number of times that a new basis was computed (*nb*), the percent of execution time spent by MA28 choosing and factoring a basis (*bas*), the percent of execution time spent computing the horizontal step, and the total number of CPU seconds used to solve the problem (*CPU*). All problems were run using exact Hessian information and the fastest option for computing v_k.

The data clearly shows a substantial tradeoff between the time spent finding a good basis and the time needed to solve the horizontal subproblem. Table 5.3 suggests that Heuristic I adequately controls the condition number of the basis and results in an acceptable computing time. Therefore, even though our sample of results is small, we have incorporated it into our code.

Next we study the performance of Heuristic II: the strategy for updating the penalty parameter described in section 4.6. Table 5.4 compares the number of function and constraint evaluations required for convergence using first the updating rule

TABLE 5.4
Comparison of two strategies for updating the penalty parameter.

	Exact Hessians		ℓ-BFGS approximations	
Problem name	*Rule* (4.14)	*Heuristic* II	*Rule* (4.14)	*Heuristic* II
ORTHREGA	70	44	138	115
ORTHREGC	36	26	93	73
ORTHRGFB	28	27	60	53
OPTCTRL3	7	7	64	63
OPTCTRL6	9	8	150	164
DTOC1ND	7	7	17	17
DTOC3	6	6	10	10
DTOC4	5	5	7	7
DTOC5	6	6	8	8
DTOC6	> 1000	43	> 1000	164

(4.14) and then Heuristic II, as given in Algorithm 4.2. We present results only for those problems for which the two strategies differed, that is, problems in which μ^+ is increased by Algorithm 4.2 at some iteration. (Again, each problem was run using the fastest option for computing v_k.)

The results in Table 5.4 show that Heuristic II is significantly beneficial for several problems while degrading performance in only one instance (OPTCTRL6). Our sample of results is again quite small, but since the heuristic appears to work rather well, it has been incorporated into the code.

5.2. Comprehensive tests. We are now ready to describe the performance of the algorithm on the full set of test problems. The values of all tolerances and parameters used by the algorithm during testing are given at the end of section 4.9. First we give results for many small nonlinear problems to test the robustness of the algorithm. Table 5.5 lists the number of unknowns (n) and constraints (m) in each problem, the number of times that the function and constraints were evaluated to obtain a solution (f,c), and the number of evaluations of gradients (g,A). As in Table 5.1, an asterisk by the problem name means the original CUTE problem has been modified for our use. We report the best results for the algorithm from among the three different options for computing the vertical step. Note that in problems where $n = m$ there is no horizontal subproblem and W_k plays no role in the algorithm.

The results given in Table 5.5 suggest that our implementation of the Byrd–Omojokun algorithm is robust. All the problems were solved successfully using the default algorithm parameters described in section 4.9, except for HS104LNP and HS111LNP. These two problems originally contained bound constraints that were inactive at the solution. We deleted the bounds, but to prevent iterates from straying into a region where f was not defined we had to limit the maximum growth of the trust region to a factor of 2 instead of 10 (see section 4.8).

To further test robustness of the algorithm we duplicated the experiments described in [43], in which problems HS6, HS7, HS8, HS9, HS26, HS27, HS39, HS40, HS42, HS60b, HS77, HS78, and HS79 were posed with a set of more difficult starting points. Out of 102 different test cases our algorithm (using exact Hessians and trying all three options for computing the vertical step) failed to reach a solution in seven instances. The failures were caused by rank deficiency in the constraint Jacobian A_k. Thus our code appears to be extremely robust, although it could be improved by better handling linear dependencies among the constraints.

TABLE 5.5
Performance on small test problems.

			Exact Hessians		ℓ-BFGS approximations	
Problem	n	m	*Number f,c*	*Number g,A*	*Number f,c*	*Number g,A*
BT1	2	1	51	18	12	8
BT2	3	1	13	13	16	16
BT4	3	2	7	7	12	11
BT5	3	2	7	7	10	10
BT6	5	2	17	13	19	15
BT7	5	3	36	22	361	170
BT8	5	2	10	10	13	13
BT10	2	2	7	7	no Hessian when $n = m$	
BT11	5	3	7	7	12	12
BT12	5	3	6	6	9	7
HS6	2	1	30	13	20	12
HS7	2	1	8	7	15	11
HS8	2	2	6	6	no Hessian when $n = m$	
HS9	2	1	6	4	6	6
HS26	3	1	16	16	42	24
HS27	3	1	39	18	25	17
HS39	4	2	17	13	26	19
HS40	4	3	4	4	7	5
HS42	4	2	5	5	8	8
∗HS60b	3	1	6	6	13	11
HS77	5	2	11	9	23	16
HS78	5	3	5	5	8	7
HS79	5	3	6	6	12	12
∗HS100LNP	7	2	11	8	18	15
∗HS104LNP	8	4	12	11	23	21
∗HS111LNP	10	3	11	11	30	24
METHANB8	31	31	4	4	no Hessian when $n = m$	
METHANL8	31	31	6	6	no Hessian when $n = m$	

* Test problem was created specifically for this paper.

Let us move on to our main goal in this section, testing algorithm performance on the large-scale problems of Table 5.1. Table 5.6 presents results using exact second derivatives to define the Hessian of the Lagrangian W_k and Table 5.7 the results using ℓ-BFGS approximations. Both tables characterize performance by listing the number of times that the function and constraints were evaluated (f, c), the number of evaluations of the gradients (g, A), and the total CPU time. Data is presented for all three methods for computing the vertical step: dogleg using the sparse Cholesky factorization of $A_k^T A_k$, Steihaug's method, and dogleg using CG to obtain the point v_n in (4.4). Since W_k plays no role in problems where $n = m$, those problems are omitted from Table 5.7.

In problems HAGER2, HAGER3, DTOC3, DTOC4, and DTOC5 the constraint Jacobian A_k is ill conditioned and large; consequently, the CG iterations used in the Steihaug and dogleg-CG options for computing v_k were very inefficient. On the other hand, a great amount of fill-in occurs during the formation or Cholesky factorization of $A_k^T A_k$ in the ORTHREG family of problems and BRATU3D. In three instances the fill-in even exceeded the 100Mbyte virtual memory workspace of our workstation. As these problems illustrate, the choice of iterative versus direct methods for computing the vertical step depends on the nature of the problem's constraints and can be critical to performance. We therefore make both options available to the user in our software.

The results of Table 5.7 indicate that using limited memory approximations for second derivatives gives rise to a slower algorithm that needs more function and first

TABLE 5.6
Performance results using exact Hessians.

	v_k by dogleg-Cholesky			v_k by Steihaug			v_k by dogleg-CG		
Problem	*f,c*	*g,A*	*CPU time*	*f,c*	*g,A*	*CPU time*	*f,c*	*g,A*	*CPU time*
GENHS28	4	4	0.2 sec	6	6	0.4 sec	6	6	0.4 sec
HAGER2	5	5	18.7 sec	5	5	884.7 sec	6	6	649.8 sec
HAGER3	5	5	14.4 sec	5	5	479.4 sec	6	6	399.5 sec
ORTHREGA	A^TA too dense			42	19	164.2 sec	44	20	168.9 sec
ORTHREGC	53	20	119.6 sec	26	15	23.3 sec	53	20	41.1 sec
ORTHREGD	14	10	1.4 sec	8	8	0.6 sec	15	10	1.6 sec
ORTHRGDM	A^TA too dense			8	8	37.9 sec	7	7	33.2 sec
ORTHRGFB	32	17	63.7 sec	27	15	27.7 sec	32	17	36.6 sec
OPTCTRL3	7	7	0.6 sec	7	7	0.8 sec	7	7	0.8 sec
OPTCTRL6	8	8	0.7 sec	9	9	1.1 sec	9	9	1.1 sec
DTOC1ND	7	7	39.0 sec	7	7	42.5 sec	7	7	42.2 sec
DTOC2	6	6	398.3 sec	7	7	502.1 sec	6	6	486.2 sec
DTOC3	6	6	68.9 sec	7	7	2505.3 sec	7	7	2991.1 sec
DTOC4	5	5	292.3 sec	12	9	6368.6 sec	5	5	3752.8 sec
DTOC5	6	6	78.6 sec	21	14	3190.6 sec	6	6	1081.1 sec
DTOC6	43	25	134.3 sec	27	18	171.9 sec	43	25	230.8 sec
EIGENA2	4	4	0.2 sec	4	4	0.2 sec	4	4	0.2 sec
EIGENC2	14	10	41.4 sec	14	10	38.7 sec	14	10	38.6 sec
ARTIF	10	10	1.0 sec	10	10	6.2 sec	14	11	14.7 sec
BRATU3D	factors of A^TA too dense			5	5	216.5 sec	5	5	281.6 sec
BROYDNBD	A^TA rank deficient			8	8	6.7 sec	did not converge		
HTRODEQ	12	10	36.3 sec	did not converge			did not converge		

TABLE 5.7
Performance results using ℓ-BFGS approximations.

	v_k by dogleg-Cholesky			v_k by Steihaug			v_k by dogleg-CG		
Problem	*f,c*	*g,A*	*CPU time*	*f,c*	*g,A*	*CPU time*	*f,c*	*g,A*	*CPU time*
GENHS28	8	8	0.3 sec	9	9	0.6 sec	9	9	0.6 sec
HAGER2	8	8	18.9 sec	8	8	1452.1 sec	8	8	937.4 sec
HAGER3	8	8	14.2 sec	8	8	803.4 sec	8	8	582.4 sec
ORTHREGA	A^TA too dense			95	41	189.8 sec	115	49	215.3 sec
ORTHREGC	118	47	256.8 sec	73	36	43.0 sec	100	41	57.4 sec
ORTHREGD	12	11	1.1 sec	11	11	0.6 sec	13	12	0.7 sec
ORTHRGDM	A^TA too dense			9	9	43.0 sec	9	9	41.7 sec
ORTHRGFB	28	21	64.6 sec	53	25	26.6 sec	46	24	29.5 sec
OPTCTRL3	63	45	2.1 sec	65	58	4.4 sec	68	47	3.7 sec
OPTCTRL6	did not converge			191	187	12.0 sec	164	154	9.7 sec
DTOC1ND	17	16	84.1 sec	22	18	105.7 sec	24	18	105.9 sec
DTOC2	102	93	1034.0 sec	195	99	3140.5 sec	101	88	2526.3 sec
DTOC3	10	8	70.3 sec	9	9	4402.1 sec	10	8	4433.4 sec
DTOC4	7	7	404.3 sec	7	7	4699.6 sec	7	7	6144.5 sec
DTOC5	8	8	103.2 sec	8	8	1795.1 sec	8	8	1610.7 sec
DTOC6	164	88	197.8 sec	89	52	364.2 sec	200	96	567.7 sec
EIGENA2	4	4	0.2 sec	4	4	0.2 sec	4	4	0.2 sec
EIGENC2	208	192	230.4 sec	217	203	193.6 sec	235	212	218.4 sec

derivative evaluations. This demonstrates the advantage of exploiting exact Hessian information. However, second derivatives are difficult to obtain in some problems, and then the availability of compact ℓ-BFGS matrices is a valuable algorithm feature.

We monitored the speed of convergence of the algorithm, and when second derivatives were used, all the problems exhibited quadratic contraction in the error during the final iterations. The ORTHREG family of problems have the kind of constraint curvature that gives rise to the Maratos effect, and our second-order corrections were

successful in maintaining a quadratic convergence rate. When ℓ-BFGS approximations were used for W_k the rate of convergence was linear, as expected.

The algorithm failed to converge on problems BROYDNBD and HTRODEQ for some of the vertical subproblem options. This is because the constraint Jacobian A in these problems is nearly rank deficient, preventing us from reducing constraint infeasibility beyond a certain point. We constructed problem ORTHRGDS (see Table 5.1) to investigate the limits of algorithm robustness when A_k becomes rank deficient. By a judicious choice of the starting point we can make the algorithm converge to a solution point where any desired number of the columns of A_* vanish. As we increased the number of zero columns, the algorithm took more and more CPU time to find a solution satisfying the stop tolerance of $\epsilon = 10^{-5}$. When more than 5 columns of A_* vanished at the solution, the algorithm was unable to meet this stop tolerance. There is clearly a great need for modifying the algorithm so that it can cope with linearly dependent constraint gradients, but this is a task that we have not yet addressed.

We modified the original OPTCNTRL problem so as to introduce ill conditioning by adding a quadratic term to the objective function f. The resulting problems, OPTCTRL3 and OPTCTRL6, have Hessian matrices with condition numbers of order 10^3 and 10^6, respectively. In these problems the merit function penalty parameter becomes very large, but the algorithm is nevertheless able to reach the solution, except for one case: when ℓ-BFGS approximations are used with the dogleg-Cholesky option, rounding errors prevent the stop tolerance $\epsilon = 10^{-5}$ from being satisfied.

A detailed analysis of other test results, and a discussion of the properties of the test problems is given by Plantenga [33].

We also solved the large test problems with LANCELOT A (7/94) [9], using second derivatives and the same stop tolerance of 10^{-5} (specified for both `constraint-accuracy-required` and `gradient-accuracy-required`). The `infinity-norm-trust-region` was applied, and an `exact-cauchy-point-required`. LANCELOT provides a variety of preconditioners to choose from, and the results we state in Table 5.8 are the best performance by LANCELOT out of a set of 11 preconditioners tried. More specifically, for each problem we report the fewest function evaluations obtained over the set of all preconditioning options and the smallest execution time and note that the best performance with respect to these two measures was often obtained by different options. The results for our code were tabulated in the same manner. We do this because in some applications a user may want to minimize function evaluations, while in others execution time may be the top priority.

We certainly do not yet consider our software to be as general or robust as LANCELOT, and our tests show that it sometimes requires much more CPU time. But the data in Table 5.8 indicate that our algorithm has great promise; perhaps the most encouraging statistic is the smaller number of function and gradient evaluations required by our algorithm in these tests. In the next section we mention various possibilities for decreasing the CPU cost of an iteration.

6. Final remarks. During the last ten years much work has been devoted to the development of trust region methods for equality constrained optimization based on sequential quadratic programming ([42], [7], [38], [5], [12], [16], [17], [25], [43], [44], [46]). However, to our knowledge, none of these methods has received a careful implementation, and it was unknown to us if they would perform well in practice. We therefore undertook the task of implementing one of these methods and observing its performance both in the large-scale case and in the solution of ill-conditioned problems. We have chosen the trust region method of Byrd [2] and Omojokun [32] because

TABLE 5.8
Performance comparison with LANCELOT A (7/94) .

	LANCELOT			Algorithm ETR		
Problem	*f,c*	*g,A*	*CPU time*	*f,c*	*g,A*	*CPU time*
GENHS28	6	7	0.6 sec	4	4	0.2 sec
HAGER2	8	9	21.1 sec	5	5	18.7 sec
HAGER3	8	9	19.2 sec	5	5	14.4 sec
ORTHREGA	118	107	97.4 sec	42	19	164.2 sec
ORTHREGC	30	28	11.8 sec	26	15	23.3 sec
ORTHREGD	44	40	1.4 sec	8	8	0.6 sec
ORTHRGDM	29	26	26.0 sec	7	7	33.2 sec
ORTHRGFB	42	38	10.9 sec	27	15	27.7 sec
OPTCTRL3	37	38	0.8 sec	7	7	0.6 sec
OPTCTRL6	46	47	1.1 sec	8	8	0.7 sec
DTOC1ND	15	15	21.2 sec	7	7	39.0 sec
DTOC2	20	21	17.9 sec	6	6	398.3 sec
DTOC3	22	23	76.4 sec	6	6	68.9 sec
DTOC4	13	14	54.2 sec	5	5	292.3 sec
DTOC5	21	22	41.0 sec	6	6	78.6 sec
DTOC6	54	55	19.3 sec	27	18	134.3 sec
EIGENA2	13	14	0.5 sec	4	4	0.2 sec
EIGENC2	174	148	62.0 sec	14	10	38.6 sec
ARTIF	23	17	3.3 sec	10	10	1.0 sec
BRATU3D	3	4	169.2 sec	5	5	216.5 sec
BROYDNBD	12	12	16.0 sec	8	8	6.7 sec
HTRODEQ	did not converge			12	10	36.3 sec

we feel that it is well suited for large problems and because of its attractive convergence properties. Our implementation can incorporate second derivative information without an explicit Hessian matrix—only Hessian-vector products are needed. Alternatively, the software can supply an ℓ-BFGS Hessian approximation. Our numerical experience with a small but challenging set of test problems has convinced us that this algorithm is quite effective and merits further development.

The following refinements and options that we plan to investigate may significantly improve its efficiency:

1. Applying preconditioners to the many subproblems solved by CG iterations.

2. Replacing the system $Z_k^T Z_k$ in the horizontal subproblem with a full-space orthogonal projection, based on the identity $Z(Z^TZ)^{-1}Z^T = I - A(A^TA)^{-1}A^T$. Preliminary tests indicate this significantly reduces computing time when the Cholesky factorization of A^TA can be obtained cheaply and accurately [26].

3. Using a sparse QR factorization of A_k to compute Z_k. This eliminates the difficulties associated with ill-conditioned bases in (4.8). The QR factorization may not always be economical, but it is likely to be very effective on some problems.

4. Solving linear systems involving $A_k^T A_k$ by the augmented system approach [14], [21]. This provides an interesting alternative to the normal equations and extends the class of problems for which a direct solution of the vertical problem is practical.

5. Computing the horizontal step more efficiently when using ℓ-BFGS approximations. Our current code approximates the full Hessian of the Lagrangian W_k, projects it onto the tangent space of the constraints, and applies Steihaug's method via Algorithm 4.1. The solution of this subproblem takes up to 60% of total execution time, far too high given that the ℓ-BFGS approximation is not very accurate. A promising alternative currently under investigation [31] is to solve the horizontal subproblem using a dual method.

6. Our current implementation requires that the matrices A_k of constraint gradients be of full rank, but this is not intrinsic to the Byrd–Omojokun approach. An important enhancement of our code will be the development of solvers that do not require this full-rank assumption.

But the most pressing need is to adapt the algorithm so that it can accept inequality constraints. There are various ways of doing this, and they are currently the subject of investigation [34], [3].

Acknowledgments. The authors would like to express their gratitude to Richard Byrd for his insights. We also thank the referees and the associate editor for many valuable comments and constructive suggestions.

REFERENCES

[1] I. Bongartz, A. R. Conn, N. I. M. Gould, and Ph. L. Toint, *CUTE: Constrained and unconstrained testing environment*, ACM Trans. Math. Software, 21 (1995), pp. 123–160.

[2] R. H. Byrd, *Robust trust region methods for constrained optimization*, Third SIAM Conference on Optimization, Houston, TX, May 1987.

[3] R. H. Byrd, J. Ch. Gilbert, and J. Nocedal, *A Trust Region Method Based on Interior Point Techniques for Nonlinear Programming*, Tech. report OTC 96/02, Optimization Technology Center, Northwestern University, Evanston, IL, 1996.

[4] R. H. Byrd, J. Nocedal, and R. B. Schnabel, *Representations of quasi-Newton matrices and their use in limited memory methods*, Math. Programming, Ser. A, 63 (1994), pp. 129–156.

[5] R. H. Byrd, R. B. Schnabel, and G. A. Schultz, *A trust region algorithm for nonlinearly constrained optimization*, SIAM J. Numer. Anal., 24 (1987), pp. 1152–1170.

[6] R. G. Carter, *On the global convergence of trust region algorithms using inexact gradient information*, SIAM J. Numer. Anal., 28 (1991), pp. 251–265.

[7] M. R. Celis, J. E. Dennis, and R. A. Tapia, *A trust region strategy for nonlinear equality constrained optimization*, in Numerical Optimization 1984, P. T. Boggs, R. H. Byrd, and R. B. Schnabel, eds., SIAM, Philadelphia, 1985, pp. 71–82.

[8] T. F. Coleman and A. R. Conn, *Nonlinear programming via an exact penalty function: Global analysis*, Math. Programming, 24 (1982), pp. 137–161.

[9] A. R. Conn, N. I. M. Gould, and Ph. L. Toint, *LANCELOT : A Fortran Package for Large-Scale Nonlinear Optimization (Release A)*, Springer Ser. Comput. Math., Springer-Verlag, Berlin, 1992.

[10] E. J. Craig, *The N-step iteration procedure*, J. Math. Phys., 34 (1955), pp. 65–73.

[11] R. S. Dembo, S. C. Eisenstat, and T. Steihaug, *Inexact Newton methods*, SIAM J. Numer. Anal., 19 (1982), pp. 400–408.

[12] J. E. Dennis, M. El-Alem, and M. C. Maciel, *A Global Convergence Theory for General Trust-Region-Based Algorithms for Equality Constrained Optimization*, Tech. report 92-28, Dept. of Mathematical Sciences, Rice University, Houston, TX, 1992.

[13] J. E. Dennis Jr. and R. B. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice–Hall, Englewood Cliffs, NJ, 1983.

[14] I. S. Duff and J. K. Reid, *A comparison of some methods for the solution of sparse overdetermined systems of linear equations*, J. Inst. Maths. Applics., 17 (1976), pp. 267–280.

[15] I. S. Duff and J. K. Reid, *Some design features of a sparse matrix code*, ACM Trans. Math. Software, 5 (1979), pp. 18–35.

[16] M. M. El-Alem, *A global convergence theory for the Celis-Dennis-Tapia trust region algorithm for constrained optimization*, SIAM J. Numer. Anal., 28 (1991), pp. 266–290.

[17] M. M. El-Alem, *A Robust Trust-Region Algorithm with a Non-monotonic Penalty Parameter Scheme for Constrained Optimization*, Tech. report, Dept. of Mathematical Sciences, Rice University, Houston, TX, 1993.

[18] R. Fletcher, *An ℓ_1 penalty method for nonlinear constraints*, in Numerical Optimization 1984, P. T. Boggs, R. H. Byrd, and R. B. Schnabel, eds., SIAM, Philadelphia, 1985, pp. 26–40.

[19] R. Fletcher, *Second order corrections for nondifferentiable optimization*, in Numerical Analysis, Dundee 1981, G. A. Watson, ed., Lecture Notes in Math. 912, Springer-Verlag, Berlin, 1982.

[20] R. Fletcher, *Practical Methods of Optimization*, 2nd ed., Wiley & Sons, Chichester, UK, 1990.

[21] R. Fourer and S. Mehrotra, *Solving symmetric indefinite systems in an interior-point method for linear programming*, Math. Programming, Ser. B, 62 (1993), pp. 15–39.
[22] J. Ch. Gilbert and C. Lemaréchal, *Some numerical experiments with variable-storage quasi-Newton algorithms*, Math. Programming, Ser. B, 45 (1989), pp. 407–435.
[23] P. E. Gill, W. Murray, and M. A. Saunders, *SNOPT: An SQP Algorithm for Large-Scale Constrained Optimization*, Numerical Analysis Report 96-2, Dept. of Mathematics, University of California, San Diego, La Jolla, CA, 1996.
[24] S. P. Han, *A globally convergent method for nonlinear programming*, J. Optim. Theory Appl., 22 (1977), pp. 297–309.
[25] M. Heinkenschloss, *On the solution of a two-ball trust region subproblem*, Math. Programming, Ser. A, 64 (1994), pp. 249–276.
[26] M. E. Hribar, *Large-Scale Constrained Optimization*, Ph.D. thesis, Dept. of Electrical Engineering and Computer Science, Northwestern University, Chicago, IL, 1996.
[27] D. C. Liu and J. Nocedal, *On the limited memory BFGS method for large scale optimization*, Math. Programming, Ser. B, 45 (1989), pp. 503–528.
[28] N. Maratos, *Exact Penalty Function Algorithms for Finite Dimensional and Control Optimization Problems*, Ph.D. thesis, University of London, UK, 1978.
[29] B. A. Murtagh and M. A. Saunders, *Large-scale linearly constrained optimization*, Math. Programming, 14 (1978), pp. 41–72.
[30] E. G. Ng and B. W. Peyton, *Block sparse Cholesky algorithms on advanced uniprocessor computers*, SIAM J. Sci. Statist. Comp., 14 (1993), pp. 1034–1056.
[31] J. Nocedal and T. D. Plantenga, *Solving KKT systems using limited memory BFGS matrices*, manuscript.
[32] E. O. Omojokun, *Trust Region Algorithms for Optimization with Nonlinear Equality and Inequality Constraints*, Ph.D. thesis, Dept. of Computer Science, University of Colorado, Boulder, 1989.
[33] T. D. Plantenga, *Large-Scale Nonlinear Constrained Optimization Using Trust Regions*, Ph.D. thesis, Dept. of Electrical Engineering and Computer Science, Northwestern University, Chicago, IL, 1994.
[34] T. D. Plantenga, *A trust region method for nonlinear programming based on primal interior-point techniques*, SIAM J. Sci. Comput., to appear.
[35] M. J. D. Powell, *A hybrid method for nonlinear equations*, in Numerical Methods for Nonlinear Algebraic Equations, P. Rabinowitz, ed., Gordon and Breach, London, UK, 1970, pp. 87–114.
[36] M. J. D. Powell, *The convergence of variable metric methods for nonlinearly constrained optimization calculations*, in Nonlinear Programming 3, O. Mangasarian, R. Meyer, and S. Robinson, eds., Academic Press, New York, London, 1978, pp. 27–63.
[37] M. J. D. Powell and Y. Yuan, *A recursive quadratic programming algorithm that uses differentiable exact penalty functions*, Math. Programming, 35 (1986), pp. 265–278.
[38] M. J. D. Powell and Y. Yuan, *A trust region algorithm for equality constrained optimization*, Math. Programming, Ser. A, 49 (1991), pp. 189–211.
[39] G. A. Schultz, R. B. Schnabel, and R. H. Byrd, *A family of trust-region-based algorithms for unconstrained minimization with strong global convergence properties*, SIAM J. Numer. Anal., 22 (1985), pp. 47–67.
[40] T. Steihaug, *The conjugate gradient method and trust regions in large scale optimization*, SIAM J. Numer. Anal., 20 (1983), pp. 626–637.
[41] Ph. L. Toint, *Towards an efficient sparsity exploiting Newton method for minimization*, in Sparse Matrices and Their Uses, I. S. Duff, ed., Academic Press, New York, 1981, pp. 57–87.
[42] A. Vardi, *A trust region algorithm for equality constrained minimization: Convergence properties and implementation*, SIAM J. Numer. Anal., 22 (1985), pp. 575–591.
[43] K. A. Williamson, *A Robust Trust Region Algorithm for Nonlinear Programming*, Ph.D. thesis, Dept. of Mathematical Sciences, Rice University, Houston, TX, 1991.
[44] Y. Yuan, *On a subproblem of trust region algorithms for constrained optimization*, Math. Programming, Ser. A, 47 (1990), pp. 53–63.
[45] J. Z. Zhang and D. T. Zhu, *Projected quasi-Newton algorithm with trust region for constrained optimization*, J. Optim. Theory Appl., 67 (1990), pp. 369–393.
[46] Y. Zhang, *Computing a Celis-Dennis-Tapia trust region step for equality constrained optimization*, Math. Programming, Ser. A, 55 (1992), pp. 109–124.