A PRACTICAL ALGORITHM FOR GENERAL LARGE SCALE NONLINEAR OPTIMIZATION PROBLEMS

PAUL T. BOGGS *, ANTHONY J. KEARSLEY † AND JON W. TOLLE ‡

Abstract. We provide an effective and efficient implementation of a sequential quadratic programming (SQP) algorithm for the general large scale nonlinear programming problem. In this algorithm the quadratic programming subproblems are solved by an interior point method that can be prematurely halted by a trust region constraint. Numerous computational enhancements to improve the numerical performance are presented. These include a dynamic procedure for adjusting the merit function parameter and procedures for adjusting the trust region radius. Numerical results and comparisons are presented.

Key words: nonlinear programming, interior point, SQP, merit function, trust region, large scale

1. Introduction. In a series of recent papers, [3], [6], and [8], the authors have developed a new algorithmic approach for solving large, nonlinear, constrained optimization problems. This proposed procedure is, in essence, a sequential quadratic programming (SQP) method that uses an interior point algorithm for solving the quadratic subproblems and achieves global convergence through the application of a special merit function and a trust region strategy. Over the past several years the theory supporting this approach has been analyzed and strengthened. This theory is presented in a companion paper [4]. In addition, implementations of the algorithm have been extensively tested on a variety of large problems, including standard test problems and problems of engineering and scientific origin, ranging in size from several hundred to several thousand variables with up to several thousand constraints. Specific strategies have been developed for handling the parameters utilized by the algorithm and for dealing with nontrivial pathologies (e. g. , linearly dependent active constraint gradients or inconsistent linearized constraints in the quadratic subproblem) that often occur in large scale problems. In this paper we present the results of these efforts.

Based on its theoretical foundation and on our numerical experience we are confident that this algorithm provides an efficient means for attacking a large, sparse, nonlinear program with equality and/or inequality constraints. Rigorous comparisons of algorithms for large nonlinear problems is notoriously difficult, especially given the extensive set of options typically available in codes for such problems. Nevertheless, our algorithm, with the (conservative) default parameter settings, has been successful on problems that have caused difficulties for other algorithms and, consequently, we are encouraged to believe that it is competitive at the current stage in the development of methods for solving these large problems.

Below we give an outline of our basic procedure and in the succeeding sections we provide more specific detail on the component parts of the implemented algorithm, including the strategies and safeguards that we have used. We also exhibit and comment on the results of some of our numerical tests. This paper relies heavily on the results from the paper on the theory for motivation of the basic ideas.

* Applied and Computational Mathematics Division, National Institute of Standards and Technology, Gaithersburg, MD 20899

† Department of Mathematical Sciences, Carnegie Mellon University, Pittsburgh, PA 15213-3890

‡ Mathematics Department, University of North Carolina, Chapel Hill, NC 27599

We assume the general nonlinear programming problem to be of the form

$$\begin{array}{l} \min\limits_{x} f(x) \\ \text{subject to:} \quad g(x) \leq 0 \end{array} \tag{NLP}$$

where $f : \mathcal{R}^n \to \mathcal{R}^1$, and $g : \mathcal{R}^n \to \mathcal{R}^m$ are smooth functions. Nonlinear equality constraints are not included in our description here in order to avoid distracting technicalities. The modifications necessary for their insertion can be inferred from [6]. Nonlinear equality constraints are included in our code and in some of the problems we tested. The sequential quadratic programming method is the backbone of our algorithm. (See [7] for a review of these techniques.) At the kth step we have an iterate, x^k, denoting the current approximation to the solution of (NLP). In addition to the x-iterate we also maintain a non-negative iterate, $z^k \in \mathcal{R}^m$, which measures the infeasibility at x^k. At this stage (NLP) is modeled by a quadratic program of the form

$$\begin{array}{l} \min\limits_{\delta} \nabla f(x^k)^\mathsf{T}\delta + \frac{1}{2}\delta^\mathsf{T} B^k \delta \\ \text{subject to:} \quad \nabla g(x^k)^\mathsf{T}\delta + g(x^k) \leq 0. \end{array} \tag{QP}$$

Here B^k is taken to be an appropriate approximation to the Hessian of the Lagrangian for (NLP), i.e.,

$$B^k \approx H_{xx}\ell(x^k, \lambda^k),$$

where

$$\ell(x, \lambda) = f(x) + g(x)^\mathsf{T}\lambda$$

and H_{xx} represents the Hessian with respect x of the function to which it is applied. (See Section 4.5 for a discussion of the choice of B^k used in our numerical experiments.) In this form (QP) generates a step that provides a search direction for improving the current iterate.

There are two significant points to be made concerning this phase of our algorithm. First, we apply an interior point quadratic program solver to (QP); more specifically, we use the method found in [1] where solutions are calculated by solving a sequence of low dimensional quadratic programs. Pertinent details of this solver and its properties relative to its use in our SQP method can be found in Section 2. Second, we do not try to solve (QP) with complete accuracy at each iteration; rather, we often terminate the interior point method prematurely. In particular, we halt the quadratic program solver when the steplength exceeds a "trust region radius" that is modified at each iteration according to how well the improvement in our merit function is predicted. Thus our algorithm can be said to be a "truncated Newton method" in the sense of [18] (see also [15]). This particular merit function and a more useful "working version" are discussed in Section 3 and our strategy for updating the trust region radius is given in Section 4.2.

The output of the (QP) solver is a vector that determines the direction of the step in the x-variable, which in turn yields a step direction for the "slack" variable z as explained in Section 3. The combined step direction of these two variables is a descent direction for the working version of the merit function and also for constraint infeasibility; thus we can choose the steplength in this direction to decrease the merit function and/or the infeasibility of the iterate. The choice of steplength determines the new iterate x^{k+1} and also the new value z^{k+1}. The strategy for choosing the steplength and other algorithmic details, including the modifications and safeguards necessary to make an implementation robust, are given in Section 4.

The results of our numerical tests are contained in Section 5. These results demonstrate the overall effectiveness of the procedure and highlight the beneficial effect of our trust region strategy and other procedures. Finally, in Section 6 we briefly consider weaknesses in the current version of the algorithm and suggest possible avenues of research to improve its efficiency.

For a discussion of the theoretical and practical questions related to large scale nonlinear programming see the recent surveys [12], [14] and [21].

2. An Interior Point QP Solver. Interior point methods for linear programming have been demonstrated to be very successful, especially on large problems, and recent research has lead to their extension to quadratic programs. A particular method, the method of optimizing over low-dimensional subspaces, has performed well on linear programs and has been extended to the quadratic programming case (see [1], and [2] and the references contained therein). This method, for which good numerical results for quadratic programs have been reported, has properties that make it particularly compatible with the SQP algorithm we are describing in this paper. A brief description of the essential features of this method and their importance for our purposes follow. The many details of the actual algorithm that are not reported here may be found in the above references.

The quadratic program that we solve, (QP), has the form

$$\begin{aligned} &\min_{s} \; c^{\mathsf{T}}s + \tfrac{1}{2}s^{\mathsf{T}}Qs \\ &\text{subject to:} \quad A^{\mathsf{T}}s + b \leq 0 \end{aligned} \tag{2.1}$$

where $c, s \in \mathcal{R}^n$, $Q \in \mathcal{R}^{n\times n}$, $A \in \mathcal{R}^{n\times m}$, and $b \in \mathcal{R}^m$. The assumptions on (2.1) that are necessary to apply the interior point algorithm are that the problem be bounded, that A have full column rank, and that there exist feasible points (i.e., that the constraints be consistent). Note that Q can be indefinite and that no assumption of a full-dimensional interior is required. If equality constraints are present, they are handled by writing them as two inequalities.

An important prerequisite for solving (2.1) by an interior point method is a feasible initial point. Our algorithm uses a "Big M" method to construct the Phase I problem

$$\begin{aligned} &\min_{s,\theta} \; c^{\mathsf{T}}s + \tfrac{1}{2}s^{\mathsf{T}}Qs + M\theta \\ &\text{subject to:} \quad A^{\mathsf{T}}s + b - e\theta \leq 0 \end{aligned} \tag{2.2}$$

where e is a vector of ones and θ is the "artificial" variable. Clearly for θ^* large enough the point $(s, \theta) = (0, \theta^*)$ is feasible for (2.2) and if M is sufficiently large the algorithm applied to (2.2) will reduce θ until the artificial variable is nonpositive, at which point the current value of s is feasible and the $M\theta$ and $e\theta$ terms are dropped. If no such value of the artificial variable can be found, then (2.2) is not consistent and the algorithm stops. As discussed below, we make use of the step obtained from (2.2) even if it is not feasible for (QP). Note that when equality constraints are present, the entire solution procedure takes place in Phase I and θ will always be present.

The defining characteristic of the algorithm is that it proceeds by solving a sequence of low-dimensional subspace approximations to (2.1). In our application we follow the reported results in which the dimension of the subspace is taken as three. The following is an outline of the O3D (for Optimizing over 3-Dimensional subspaces) version of the algorithm. As the variable θ is treated essentially the same as the components of s in the O3D algorithm (see, however, Step 6 below) the dependence on θ is incorporated into the formulation given in (2.1).

O3D Algorithm for Quadratic Programming

1. Given a feasible point, s^0; set $j := 0$.
2. Generate 3 independent search directions $p_i,\ i = 1, 2, 3$ and let P^j be the matrix whose columns are p_i.
3. Form and solve the restricted quadratic program

$$\begin{aligned} &\min_{\zeta} \ c^{\mathsf{T}}\tilde{s} + \tfrac{1}{2}\tilde{s}^{\mathsf{T}}Q\tilde{s} \\ &\text{subject to:} \quad A^{\mathsf{T}}\tilde{s} + b \leq 0 \end{aligned}$$

   where $\tilde{s} = s^j + P^j\zeta$ and $\zeta \in \mathcal{R}^3$. Call the solution ζ^*.
4. Set $s^{j+1} := s^j + \rho P^j\zeta^*$ for an appropriate value of the steplength $\rho \in (0, 1)$.
5. If stopping criteria are met, exit.
6. Go to 2. (At this step, if the component of the vector s corresponding to the artificial variable θ has become nonpositive, it is eliminated from the problem.)

The search directions in Step 2 are solutions to

$$\left[AD^2A^{\mathsf{T}} + Q/\beta\right] p_i = t_i, \qquad i = 1, 2, 3, \tag{2.3}$$

where β is a scalar depending on the current iterate,

$$D = \operatorname{diag}\{1/r_k,\ k = 1, \ldots, m\}$$

with $r_k = -(As + b)_k$, and the t_i are particular values chosen such that one of these directions is always a descent direction with respect to the objective function. The steplength ρ is set to the lesser of 99% of the distance to the boundary and the distance to the minimum of the objective function.

The form of the matrix in (2.3) allows for efficient exploitation of the sparsity. Note that if Q is positive semi-definite, then the matrix in (2.3) is positive definite for all interior points; otherwise, it may not be. In the latter case, a modification similar to that in [20] is used. In our application of this algorithm, using this procedure obviates the need for the matrix B^k to be positive definite, which in turn allows us to use the Hessian of the Lagrangian or a finite difference approximation thereof.

The standard stopping criterion for the algorithm is that at least one of the following holds: (a) the relative change in two successive values of the objective function is small; (b) the relative difference between the primal and the dual objective function values is small; or (c) the difference between two successive iterates is small. For use in our SQP algorithm we have added: (d) the length of the solution vector exceeds a specified value. This additional condition has been implemented to allow for trust region strategies; in particular, this criterion will cause the algorithm to halt if (QP) is unbounded. In any case, the terminal vector will be a useful direction in the context of our purposes; this point will be discussed in the next section.

The most recent version of O3D described in [1] contains an option to perform a special "recentering step" after each subspace optimizing step (Step 4) that has generally improved the efficiency. This option is not used in the results reported here. (See Section 6 for a further comment.)

3. Updating the Iterates: the Merit Functions. In this section we review the definitions and properties of our merit functions and provide formulas for updating the iterates. The reader is referred to the companion paper for proofs and motivations of these concepts.

As stated in Section 1, at each iteration our algorithm yields a pair (x^k, z^k) where x^k is an approximation to the solution of (NLP) and z^k is the corresponding approximate slack vector. The step directions for the updated values of these approximations are based on the (approximate) solution, (δ^k, θ^k), to the quadratic program

$$\begin{aligned} &\min_{\delta,\theta} \ \nabla f(x^k)^{\mathsf{T}}\delta + \tfrac{1}{2}\delta^{\mathsf{T}} B^k \delta + M\theta \\ &\text{subject to:} \ \ \nabla g(x^k)^{\mathsf{T}}\delta + g(x^k) - e\,\theta \leq 0. \end{aligned} \tag{3.1}$$

obtained as described in the preceding section. The vector δ^k gives the step direction for x^k and we determine the step direction, q^k, for the slack vector z^k by the formula

$$q^k = -\left[\nabla g(x^k)^{\mathsf{T}}\delta^k + g(x^k) + z^k - e\,\theta^k\right]. \tag{3.2}$$

Note that if δ^k is feasible for (QP) then $\theta^k = 0$ and hence

$$q^k = -\left[\nabla g(x^k)^{\mathsf{T}}\delta^k + g(x^k) + z^k\right].$$

In this case $z^k + q^k$ is the slack vector for (QP) corresponding to δ^k and thus is the slack variable for the linear approximation of $g(x^{k+1})$. Given the step direction we then update the iterate by means of the formulas

$$\begin{aligned} x^{k+1} &= x^k + \alpha\delta^k \\ z^{k+1} &= z^k + \alpha q^k \end{aligned}$$

for some value of the steplength parameter α. Observe that if $z^k \geq 0$ then the fact that (δ^k, θ^k) is feasible for (3.1) means that z^{k+1} will be non-negative if $\alpha \in [0, 1]$. In our algorithm the non-negativity of the slack vector iterates is preserved and, in fact, it sometimes turns out to be useful to maintain the z^k at a positive level (see Section 4.8).

It is important to emphasize that the δ^k are determined by (QP), the quadratic approximation to (NLP), and are not dependent on the choice of z^k. The z^k are generated solely for use with the merit function described below. That is, we *do not* solve the slack variable problem. A comment on the notation is also in order at this point: We denote the iterate by (x^k, z^k) and the step by (δ^k, q^k), whereas conventional notation would be to use

$$\begin{pmatrix} x^k \\ z^k \end{pmatrix} \ \text{and} \ \begin{pmatrix} \delta^k \\ q^k \end{pmatrix}.$$

It should be clear from the context what is meant.

In optimization algorithms the value of a steplength parameter is generally chosen so as to reduce the value of a suitably chosen merit function. Typically, a merit function for (NLP) is a scalar-valued function that has an unconstrained minimum at x^*, a solution to (NLP). Because a reduction in this function implies that progress is being made towards the solution, it can be used to determine an appropriate steplength in a given search direction.

In [5] and [6] a merit function for equality-constrained problems was derived that has important properties *vis-a-vis* the steps generated by the SQP algorithm. Using a slack-variable formulation of (NLP) a merit function for the inequality constrained problem can be constructed having the form

$$\psi_d(x, z) = f(x) + \bar{\lambda}(x, z)^{\mathsf{T}}\bar{c}(x, z) + \tfrac{1}{d}\bar{c}(x, z)^{\mathsf{T}}\bar{A}(x, z)^{-1}\bar{c}(x, z) \tag{3.3}$$

where z is nonnegative, d is a scalar,

$$\begin{aligned}\bar{c}(x,z) &= g(x) + z,\\ \bar{A}(x,z) &= \nabla g(x)^{\mathsf{T}}\nabla g(x) + Z,\\ \bar{\lambda}(x,z) &= -\bar{A}(x,z)^{-1}\nabla g(x)^{\mathsf{T}}\nabla f(x),\end{aligned}$$

and

$$Z = \operatorname{diag}\{z_1, \ldots, z_m\}.$$

We use this merit function (and its approximations defined below) for choosing the value of the steplength parameter α. As noted above, the approximate slack vectors generated by our algorithm, z^k, always remain non-negative; thus the non-negativity constraint on the z for ψ_d imposes no theoretical difficulty.

The function $\bar{c}(x,z)$ defined above plays an important role in our algorithm as it is used to measure the feasibility of the pair (x,z). That is, if we define the function

$$r(x,z) = \|\bar{c}(x,z)\|^2, \tag{3.4}$$

where $\|\cdot\|$ denotes the standard Euclidean norm and set

$$\mathcal{C}_\eta = \{(x,z) \,:\, r(x,z) \le \eta \text{ and } z \ge 0\}, \tag{3.5}$$

then $\mathcal{C}_0$ corresponds to the feasible set of (NLP) and hence (x^k, z^k) is close to feasible if it is in $\mathcal{C}_\eta$ for small η.

For d sufficiently small the merit function ψ_d has the desirable property that a solution of (NLP) corresponds to a (constrained) minimum of ψ_d. In addition, if d is small and δ^k is the exact solution to (QP) (which implies that $\theta^k = 0$) then the step (δ^k, q^k) is a descent direction for ψ_d when (x^k, z^k) is sufficiently close to feasibility. Despite these useful properties, ψ_d has two deficiencies that limit its use in an efficient algorithm. First, (δ^k, q^k) is a descent direction of ψ_d only near feasibility, and, second, the evaluation of ∇f and ∇g and additional nontrivial computational algebra are required to assess a prospective point. In order to overcome these difficulties, the *approximate merit function*

$$\psi_d^k(x,z) = f(x) + \bar{c}(x,z)^{\mathsf{T}}\bar{\lambda}^k + \frac{1}{d}\bar{c}(x,z)^{\mathsf{T}}(\bar{A}^k)^{-1}\bar{c}(x,z)$$

where

$$\begin{aligned}\bar{A}^k &= \nabla g(x^k)^{\mathsf{T}}\nabla g(x^k) + Z^k\\ \bar{\lambda}^k &= -(\bar{A}^k)^{-1}\nabla g(x^k)^{\mathsf{T}}\nabla f(x^k)\end{aligned}$$

is developed as a "working" version of ψ_d at (x^k, z^k). As the values of $\bar{\lambda}^k$ and $\bar{A}^k$ are fixed, ψ_d^k can be more easily evaluated than ψ_d in a line search algorithm for choosing an appropriate value of α. This approximate merit function, ψ_d^k, not only has essentially the same properties as ψ_d with respect to the step (δ^k, q^k) but it has the stronger property that the step is a descent direction for ψ_d^k *everywhere*. Moreover, for η sufficiently small and (x^k, z^k) outside of a ball around the solution a "sufficient" reduction in ψ_d^k implies a "sufficient" reduction in ψ_d. (We mean by "sufficient" reduction that a Wolfe condition is satisfied.) Thus we are able to use ψ_d^k as a surrogate for ψ_d for testing the progress of our iterates towards a minimum.

A further important property of the step δ^k, under the assumption that it is the exact solution to (QP), is that it is a descent direction for the function r defined by (3.4). Thus a basic algorithm for the case where the (QP) can be solved exactly is as follows: Given an initial value of η use the steps (δ^k, q^k) to reduce r until the iterates are in $\mathcal{C}_\eta$. Once the iterates are contained in $\mathcal{C}_\eta$ if a sufficient reduction in ψ_d^k does not yield a sufficient reduction in ψ_d then reduce η. If, in the course of the algorithm, η remains bounded away from zero, then convergence follows from the fact that the Wolfe condition is satisfied for ψ_d. If η goes to zero, then convergence follows from the observation that the radius of the ball in which the Wolfe condition is not satisfied also goes to zero. This is essentially the algorithm for which global convergence is proved in the paper on the theory.

In this paper we are primarily interested in enhancements that convert the theoretical algorithm into one that is practical and efficient. This requires that we make provisions for situations when the assumptions under which we performed the convergence analysis are not valid and that we adopt numerical procedures to reduce the computational effort. As we note below, not all of these modifications have been (or even can be) theoretically justified, but we believe that the firm foundation of the underlying algorithm and the evidence accumulated in extensive numerical testing validate their use.

In the implementation of our algorithm a trust region constraint is used that possibly truncates the quadratic programming algorithm before an exact solution is achieved. In this case the theory described above does not apply for the step (δ^k, q^k) obtained from the approximate solution, (δ^k, θ^k), to (3.1). Although a general convergence theory based on this step is not yet available, it is shown in the theory paper that if the approximate solution is obtained from the O3D algorithm and if θ^k is not too large then the resulting step has the appropriate descent properties for the functions r, ψ_d, and ψ_d^k at (x^k, z^k). In particular, convergence can be achieved if θ^k goes to zero in a suitable manner. These properties justify our use of the truncation procedure to speed up the algorithm. It is important to note that this approximation procedure also allows us to handle the difficulty that arises in sequential quadratic programming methods when the quadratic subproblem is inconsistent.

4. The Truncated SQP Algorithm. In this section we give a somewhat detailed description of our algorithm. Initially we assume that the Hessian approximations, B^k, are positive definite, the matrices $\bar{A}^k$, are nonsingular, and the linearized constraints in (QP) are consistent. In real-world applications these assumptions are not always valid so we have tried to make our algorithm flexible enough to perform well in situations where these assumptions fail to hold. We describe some of these adaptations at the end of this section.

The implementation of the algorithm depends upon four important parameters that need to be either computed or modified throughout the course of the algorithm. The *globalization parameter*, η, was introduced in (3.5). It is a measure of the size of the domain about the feasible region in which the direction (δ^k, q^k) is a descent direction for the true merit function ψ_d. A current estimate of η is maintained in the algorithm. The *trust region parameter*, τ, is an upper bound on the (weighted) norm of our approximate solution to (QP),

$$\|D\delta\| \leq \tau,$$

where D is a positive definite diagonal matrix. The trust region radius τ is updated at every iteration. The parameter, α, is the *steplength parameter*. It determines

the length of the step in the variables (x, z) in the direction (δ^k, q^k). It is chosen to guarantee progress towards the solution in decreasing either the merit function or infeasibility. Finally, d, the *merit function parameter*, must be small enough to guarantee that the theoretical properties described in the preceding section are valid. Although the theory allows arbitrarily small values of d, the algorithm becomes very slow if d is too small, thus it is monitored throughout the algorithm and either increased or decreased as appropriate.

The outline of the algorithm is followed by specific comments on the procedures and their justifications. This version contains some of the practical modifications described above. To simplify the notation we define

$$(x_\alpha, z_\alpha) = (x^k + \alpha\delta^k, z^k + \alpha q^k).$$

Recall that r is given by (3.4).

Basic Truncated SQP Algorithm

1. Initialization: Given x^0, B^0, τ, η, and d
   a. Initialize the slack variable $z^0 \geq 0$;
   b. Set $k := 0$.
2. Calculation of the basic trust region step:
   a. While $\|\delta\| < \tau$, iterate (using O3D) on

$$\begin{aligned} &\min_{\delta} \nabla f(x^k)^{\mathsf{T}}\delta + \tfrac{1}{2}\delta^{\mathsf{T}}B^k\delta + M\theta \\ &\text{subject to:} \quad \nabla g(x^k)^{\mathsf{T}}\delta + g(x^k) - e\theta \leq 0 \end{aligned}$$

   to obtain δ^k and θ^k.
   b. Set

$$q^k = \begin{cases} -\left[\nabla g(x^k)^{\mathsf{T}}\delta^k + g(x^k) + z^k - e\theta^k\right] \text{ if } \theta^k > 0 \\ -\left[\nabla g(x^k)^{\mathsf{T}}\delta^k + g(x^k) + z^k\right] \qquad \text{otherwise} \end{cases}.$$

   c. Decrease d if necessary.
3. Computation of the steplength parameter:
   a. Choose $\alpha \in (0, 1]$ such that ψ_d^k is sufficiently reduced.
   b. If $(x^k, z^k) \notin \mathcal{C}_\eta$ then reduce α if necessary until r is sufficiently reduced.
   c. If $(x^k, z^k) \in \mathcal{C}_\eta$ then reduce α if necessary so that $(x_\alpha, z_\alpha) \in \mathcal{C}_\eta$.
4. Update of the estimate of the globalization parameter:
   a. If

$$\psi_d(x_\alpha, z_\alpha) > \psi_d(x^k, z^k),$$

   set $\eta = \frac{1}{2}r(x^k, z^k)$.
5. Update of the variables and check for termination:
   a. Set

$$\begin{aligned} x^{k+1} &:= x^k + \alpha\delta^k \\ z^{k+1} &:= z^k + \alpha q^k. \end{aligned}$$

   b. If convergence criteria are met, quit.
   c. Update B^k to B^{k+1}.
6. Adjustment of the merit function and trust region parameters:
   a. Update d if necessary.
   b. Adjust the trust region radius τ.

7. Return:
   a. Set $k := k + 1$.
   b. Go to Step 2.

4.1. The Globalization Parameter. The globalization step is based on work in [6] and [4]. In Step 3 we require that the approximate merit function be reduced and, in addition, if the current iterate lies outside the set $\mathcal{C}_\eta$ we require that the constraint infeasibilities also be reduced. This is possible as a result of the descent properties described in Section 3. If we have a good estimate of η and $(x^k, z^k) \in \mathcal{C}_\eta$ then the true merit function can also be reduced; if this is not the case, then our estimate of η is too large and we reduce its value in Step 4. This procedure will eventually lead to a sufficiently small value of η. Note that this arrangement allows steps that may increase the merit function, but only in a controlled way. It also allows steps that may increase the constraint infeasibilities, but only when inside of $\mathcal{C}_\eta$.

4.2. Updating τ. Our procedure for updating τ, the trust region radius, in Step 6b is similar to the standard strategy used in trust region algorithms (see [17] or [31]) in that we base the decision on how to change τ on a comparison of a predicted relative reduction, $pred_k$, and an actual relative reduction, $ared_k$, in a function used to measure the progress toward the solution. (Various formulas for the predicted relative reduction, $pred_k$, have been suggested for different merit functions, especially for equality constrained programming problems; see, for example, [19]). What is distinctive about our procedure is that we use different functions for computing $pred_k$ and $ared_k$ depending on the current status of the algorithm. When the linearized constraints are satisfied we use the approximate merit function to compute the predicted and actual reductions. When the trust region constraint causes O3D to terminate in Phase I, i.e., when the linearized constraints are not satisfied, predicted and actual reductions in infeasibility are used.

In the case when a feasible solution to (QP) is obtained then ψ_d^k is used to compute the predicted and actual reductions. Our method for defining $pred_k$ differs from the standard methods used in unconstrained optimization because the step-finding subproblem is not based solely on the merit function and, moreover, the trust region constraint does not appear explicitly in the subproblem. Nevertheless in updating τ we want to assess how well an approximation to ψ_d^k agrees with ψ_d^k in the direction (δ^k, q^k). Since (QP) uses a quadratic approximation of the Lagrangian for the objective function with linearized constraints, we form our approximation to ψ_d^k based on a quadratic approximation to the function ψ_1^k given by

$$\psi_1^k(x, z) = f(x) + \bar{c}(x, z)^\mathsf{T} \bar{\lambda}^k$$

and a linear approximation to

$$\psi_2^k(x, z) = \bar{c}(x, z)^\mathsf{T} (\bar{A}^k)^{-1} \bar{c}(x, z).$$

Note that $\psi_d^k(x, z) = \psi_1^k(x, z) + (1/d)\psi_2^k(x, z)$. Based on these considerations and the results of [16] we define the predicted relative reduction by

$$\begin{aligned} pred_k = \Big\{ & -\alpha^k \nabla \psi_1^k(x^k, z^k)^\mathsf{T} (\delta^k, q^k) - \frac{(\alpha^k)^2}{2} (\delta^k, q^k)^\mathsf{T} \nabla^2 \psi_1^k(x^k, z^k)(\delta^k, q^k) \\ & - \frac{\alpha^k}{d} \nabla \psi_2^k(x^k, z^k)^\mathsf{T} (\delta^k, q^k) \Big\} / \psi_d^k(x^k, z^k) \end{aligned} \tag{4.1}$$

where the derivatives are with respect to x and z and the steplength parameter α^k is the size of the most recently accepted step. The value of the actual relative reduction, $ared_k$, is taken to be the difference in the values of ψ_d^k at the points (x^{k+1}, z^{k+1}) and (x^k, z^k) divided by the value of $\psi_d^k(x^k, z^k)$. A valid criticism of the formula for $pred_k$ is its dependence on higher order derivatives. Therefore we use the available approximation of the Hessian of the Lagrangian for $\nabla^2\psi_1^k$. For example, cell-centered finite difference approximations to the Hessian of the Lagrangian function were used in the numerical results presented here, unless analytic second derivative formulas were readily available.

The above choice for $pred_k$ is not used when the step returned by O3D is not feasible. In these situations the resulting step is dominated by a feasibility improving component and it makes little sense for the adjustment to τ to be determined by ψ_d^k; rather, a comparison of the predicted and actual improvement in constraint infeasibility seems more appropriate. Therefore, in this case the function $r(x, z)$ is used for comparison purposes. The values of $pred_k$ and $ared_k$ are given as follows for the case when the O3D algorithm terminates in Phase I:

$$pred_k = \left\{ r(x^k, z^k) - \left\| \alpha^k \nabla g(x^k)^\mathsf{T} \delta^k + g(x^k) + z^{k+1} \right\|^2 \right\} / r(x^k, z^k)$$

and

$$ared_k = \left\{ r(x^k, z^k) - r(x^{k+1}, z^{k+1}) \right\} / r(x^k, z^k).$$

These heuristics for choosing $pred_k$ and $ared_k$ appear to work well. Specifically, they allow the trust region radius, τ, to be increased even in the event that the step returned by O3D does not satisfy linearized constraints or it results in an increase in the true merit function. In our experience, the alternative formulas based solely on constraint violations never are employed close to the solution. Indeed, the iterates preceding convergence have always been observed to be well inside $\mathcal{C}_\eta$ where satisfying the linearized constraints and decreasing the merit functions usually pose no problem.

4.3. The steplength α. The steplength α is determined in Step 3 of the algorithm. The "sufficient decrease" referred to in 3a and 3b requires that the Wolfe condition be satisfied. For a given function ϕ and potential step w from point v this condition requires that α satisfy

$$\phi(v + \alpha w) \leq \phi(v) + \sigma\, \alpha\, \nabla\phi(v)^\mathsf{T} w$$

for some fixed $\sigma \in (0, 1)$. In the numerical experiments reported in Section 5 we employed a simple backtracking procedure (with factor one-half) to find α to satisfy this condition for both ψ_d^k and for r. We have also experimented with more sophisticated line search methods motivated by unconstrained optimization techniques as in [18], but the observations to date suggest that the more complicated line searches result in very little improvement of our algorithm, except when the iterates are quite far from the solution.

4.4. Adjusting d. Choosing an effective value for the merit function parameter d is essential in our algorithm. While it is clear that (in a compact set) a sufficiently small value of d will assure that the results given in [4] are valid, there are three very important practical reasons why the parameter must be adjusted rather than fixed. First, if the angle between the direction generated by O3D and the gradient of the approximate merit function becomes nearly orthogonal the steps might become too small. We adjust d to avoid this possibility. Second, the *approximate* merit

function, ψ_d^k, is changing at each iteration and it is possible a previous iterate might be acceptable to the current ψ_d^k, i.e., cycling might occur. This worry can also be alleviated by adjusting d. A third reason for changing d is to allow for larger steps. It is seen from the theory and has been verified by numerical experience that if d is too small then the form of the merit function forces the path of the iterates to follow the "nearly active" constraints closely. This causes the algorithm to take very small steps and, in particular, to be slow in moving away from a nonoptimal active set. By making it possible to increase d we can significantly improve the algorithm's performance.

In the implementation of our algorithm there are two opportunities to adjust d: in Step 2, after solving the quadratic subproblem, and in Step 6, after the step has been taken. In the first of these adjustments d can only be decreased; in the second, the parameter may be increased or decreased.

In Step 2, the angle between the gradient of the approximate merit function $\nabla\psi_d^k$ and the step direction (δ^k, q^k) is computed. If these two vectors become nearly orthogonal, we conclude that d is not small enough to ensure a good decrease in ψ_d^k, and we decrease the parameter. To be more specific, we compute

$$w(d) = \frac{(\nabla\psi_d^k(x_k, z_k))^{\mathsf{T}}(\delta^k, q^k)}{\left\|\nabla\psi_d^k(x_k, z_k)\right\| \cdot \|(\delta^k, q^k)\|}.$$

If $w(d) \geq -.1$ we calculate a value $\hat{d}$ so that $w(\hat{d}) \approx -.5$. We safeguard the procedure by not allowing more than a certain percentage decrease in d. In the current version we use 50%.

If d was not decreased in Step 2 we consider modifying it after a step has been taken (Step 6). Here the primary concern is to avoid cycling. To do so we compute an interval for the penalty parameter as follows. For a fixed integer κ we seek a value of the parameter, $\bar{d}$, such that

$$\psi_{\bar{d}}^k(x^k, z^k) < \psi_{\bar{d}}^k(x^{k-i}, z^{k-i}), \qquad i = 1, \ldots, \kappa. \tag{4.2}$$

Inequality (4.2) implies that none of the past κ iterates will be acceptable to the approximate merit function with the new value of $\bar{d}$. (Thus if $\kappa = k$ no cycling would be possible). To accomplish this, we use the decomposition

$$\psi_d^k = \psi_1^k + \frac{1}{d}\psi_2^k \tag{4.3}$$

where ψ_1^k and ψ_2^k are defined in Section 4.2. We then compute the values of $\psi_1^k(x^{k-i}, z^{k-i})$ and $\psi_2^k(x^{k-i}, z^{k-i})$, $i = 1, \ldots, \kappa$, and consider the inequalities

$$\psi_1^k(x^k, z^k) + \frac{1}{d}\psi_2^k(x^k, z^k) < \psi_1^k(x^{k-i}, z^{k-i}) + \frac{1}{d}\psi_2^k(x^{k-i}, z^{k-i}). \tag{4.4}$$

We define d_i^u and d_i^l to be the upper and lower values of d that ensure that inequality (4.4) is satisfied. Then letting

$$d^u = \min\{d_i^u : i = 1, \ldots, \kappa\} \tag{4.5}$$

and

$$d^l = \max\{d_i^l : i = 1, \ldots, \kappa\} \tag{4.6}$$

we obtain an interval (d^l, d^u). Assuming that this interval exists it is the case that if the value of d for the next step is chosen in this interval, the next iterate will not return to one of the previous κ iterates. In practice a value of $\kappa \approx 5$ is usually more than sufficient to prevent cycling. If the interval doesn't exist, then we make no change.

Given that we can choose d to avoid cycling, our second objective at this juncture is to increase d to allow bigger steps. If the d^u is larger than the current d then we can safely increase d without worrying about possible cycling. However, we safeguard this increase in two ways. First, we require that the predicted reduction based on the approximate merit function must be greater than the predicted reduction of infeasibility in the linearized constraints. This restriction prevents d from being increased prematurely due primarily to a large decrease in constraint infeasibilities. Specifically, writing the predicted reduction in ψ_d^k (see (4.1)) as

$$\mathcal{P}_{\mathcal{Q}} + \frac{1}{d}\mathcal{P}_{\mathcal{L}},$$

we insist that for a new value of d

$$\mathcal{P}_{\mathcal{Q}} + \frac{1}{d}\mathcal{P}_{\mathcal{L}} > \mathcal{P}_{\mathcal{L}}. \tag{4.7}$$

Second, we use a maximum allowable change (currently a factor of 2) to limit the growth of d. Computationally, these simple procedures for updating d appear to be effective, especially in the presence of highly nonlinear constraints and poorly scaled problems.

4.5. The Hessian Approximation. In the numerical experimentation reported here, we have used a finite difference approximation to the Hessian of the Lagrangian as B^k. Although the Hessian of the Lagrangian at a strong solution is positive definite on the appropriate subspace, it may be indefinite in general. Even if it is positive definite the finite difference approximation may not be. We experimented with two approaches for handling this possibility. First, we simply modified the approximate Hessian matrix by adding non-negative elements to the diagonal ensuring that the Cholesky factorization of the matrix had positive elements along its diagonal (see [20]). This modification was easy to implement, but it was observed to slow convergence on some problems. While this modification guarantees that a positive definite matrix will be delivered to the (QP) solver, if it takes place when the iterates get close to the solution, it generally precludes local q-superlinear convergence.

An alternative to modifying the approximate Hessian of the Lagrangian is simply to allow O3D to iterate on the indefinite QP subproblem, halting the iterations when the solution exceeds the trust region radius. We implemented this approach and it seemed to yield superior results to those obtained by making the approximate Hessian positive definite (especially when the iterates were close to a solution) even though, theoretically, we can only prove that we obtain a descent direction when the approximate Hessian is positive definite.

4.6. Convergence Criteria. The convergence criteria used are standard, and similar to those in [3]. We first insist that the constraints be satisfied to a close tolerance; specifically we require

$$\left\| \max(g(x^k), 0) \right\|_\infty \leq 10^{-6}. \tag{4.8}$$

We also require that either

$$\frac{\|\nabla f(x^k) + \nabla g(x^k)\lambda^k\|}{|f(x^k)|} \leq 10^{-7} \tag{4.9}$$

or

$$\left\|x^k - x^{k-1}\right\|_\infty \leq 10^{-8}(1 + \left\|x^k\right\|). \tag{4.10}$$

The criterion (4.9) is a stronger indication that a KKT point has been reached. The weaker criterion (4.10) suggests that progress slowed drastically and that iterates may or may not have drawn close to a solution. For this reason criterion (4.9) is usually preferable to criterion (4.10). The Lagrange multipliers returned by the quadratic program are used in (4.9) unless the trust region constraint determines the approximate solution of the (QP). In that case, we use the least squares approximation to the multipliers, replacing all negative multipliers with machine zeros. In all of the problems solved to date, the trust region never comes into play when the iterates get close to the solution; therefore the (QP) multipliers are used for the convergence test at the solution.

4.7. Inconsistent Quadratic Subproblems. One difficulty that can occur when making linear approximations to nonlinear constraints is that (QP) may be inconsistent. In this case O3D will, even if it runs to completion, not exit Phase I and will return a positive value of the artificial variable. (Note that this always occurs if equality constraints are present.) For small θ the resulting direction is a descent direction for ψ_d^k and for r. As a result, the step taken in this direction will generally decrease infeasibility, making it less likely that an inconsistent set of linearized constraints will be encountered during subsequent iterations.

More recent versions of our algorithm include a constraint relaxation procedure that appears to yield an acceptable step, δ_k, even in the event that inconsistent linearizations of constraints are encountered. Because this situation did not surface during the numerical experiments presented in this paper, we do not include a description of our perturbation procedure. We do note, however, that we have encountered important application problems where this procedure was crucial to the performance of our algorithm (see for example [24]).

4.8. Updating slack variables. One difficulty in our algorithm is the updating of slacks in the event that the SQP step does not satisfy the linearized constraints well enough, i.e., θ^k is not small enough. This can occur when (QP) is inconsistent or when a trust region bound is encountered during the solution of (QP). In this case our slack variable updating scheme would ensure that non-negative slacks remain non-negative, but the direction may not be one of descent. We resolve this dilemma by opting for descent, i.e., computing q^k with $\theta^k = 0$ and replacing any negative slacks using the following rule:

$$\text{If } \; z_i^{k+1} < 0 \text{ then set}$$

$$z_i^{k+1} = \begin{cases} \epsilon_{Mach} & g_i(x^{k+1}) \geq 0 \\ -g_i(x^{k+1}) & g_i(x^{k+1}) < 0 \end{cases}$$

where ϵ_{Mach} is machine epsilon. This is sometimes referred to as 'closing' the constraints (see for example [33]).

4.9. Linearly Dependent Constraint Gradients. Linearly dependent constraint gradients cause many theoretical and computational difficulties in constrained optimization. In our theoretical algorithm we obtain convergence even when there are linearly dependent constraint gradients provided the approximate multipliers do not become unbounded. In practice, even though O3D has no difficulty in dealing with this problem, evaluating the merit function and computing the least squares approximation to the Lagrange multipliers become problematical. Computational experience shows we solve many problems with degeneracy in the constraints. Simply maintaining slacks to to be positive as described above allows us to factor the crucial matrices and continue with the algorithm. However, the algorithm failed to solve some problems that had a large amount of degeneracy in the linearized constraint matrix. This was, of course, problem dependent but it was observed that the current implementation can usually solve problems where up to 25 percent of the constraint gradients are linearly dependent. This degeneracy causes the performance of the merit functions to deteriorate. In particular, the least squares approximation to Lagrange multipliers seems to be especially poor, resulting in only very small steps being allowed, even close to the solution.

5. Numerical Results. The modified algorithm was coded in Fortran and is installed on a SPARCstation 10 using *IEEE floating point arithmetic (64 bit)*. The current implementation is being used to solve a wide variety of medium to large scale problems. In this section we report the results of a set of performance tests designed specifically to answer questions about the trust region strategy and the procedure to update the penalty parameter, d. We conclude the section with the results of our algorithm applied to some test problems that are publicly available. We emphasize that all of the problems were solved with the same default settings of the parameters, (see Table 1), i.e., no attempt was made to pick parameter settings to optimize performance on individual problems.

Although in many of the applications some analytic derivatives were available, no use of analytic derivative information was used in these numerical experiments. When possible, first and second derivatives were computed using forward and central finite differences respectively. A costly one-time calculation provided a zero/non-zero stencil of the Hessian of the Lagrangian and the Jacobian matrix of the constraint function. These stencils were then used for the duration of the solution process. For some problems, these finite difference approximations are not convenient to use. This can be the case with control problems governed by partial differential equations (see [29] or [30]). If the partial differential equation is solved using a finite element method, with piecewise linear elements, then evaluating the derivative of the objective

Parameter	Value
M	$10 \min\{10^7, \|\nabla f(x_0)\|_\infty \min\{10^3, \|\nabla f(x_0)\|_\infty\}\}$
θ^*	$2\|g(x_0)\|_\infty$
τ_0	$(\|g(x_0)\|_\infty + \|x_0\|_2)$
η_0	$(1 + \|c(x_0, z_0)\|_\infty)^2$
z_0	$\epsilon_{Mach} + \max(-g(x_0), \epsilon_{Mach})$
σ	10^{-4}

TABLE 1
Numerical values of default parameters

function with respect to the control variables can be quite cumbersome. In such cases, which occurred in the control problems in our test suite, one can approximate the first derivatives of the objective function by solving an adjoint problem with a computational cost comparable to one function evaluation. (For examples, see [22].) The objective function portion of the Hessian of the Lagrangian can then be approximated with forward finite differences.

A set of eight problems was chosen as the first test suite. These problems ranged in size from 500–1000 variables and from 1000–2000 constraints. The first four are relatively straightforward nonlinear programming test examples, while the last four are from actual applications: two discretized control problems, a density estimation problem from statistics, and a "molecular distance" problem. A more complete description of these problems is found in the Appendix. The problems all have nonlinear inequality constraints and exploitable sparsity. Problem 4 (NLP4) was designed to have a controllable percentage of linear dependency in the constraint gradients to demonstrate any weaknesses in the algorithm associated with this difficulty. We ran three versions of our algorithm on each problem; using a positive definite modification of the Hessian matrix, as discussed in Section 4, with and without the trust region strategy and using the unmodified Hessian with the trust region. (Using the unmodified Hessian results in failure in most cases if no trust region strategy is employed.) In addition, each problem was run from two starting points; one, labeled "c", which was close to the solution in the sense that each of the variables was of the same order of magnitude as in the solution and a distant start, labeled "f".

The results of the numerical tests on these problems are summarized in Tables 1–3. The first two columns of each table gives the number of SQP iterations ("nl-i") and the total number of O3D iterations ("qp-i"). The next two columns contain the stopping criterion that was met and the value of the gradient of the Lagrangian at the solution. Unless the algorithm failed, (which is denoted by "Failure" in the tables) feasibility condition (4.8) was satisfied for all solutions. The stopping criterion is denoted by either a "1" or a "2" depending on whether (4.9) or (4.10) was satisfied. If both were conditions were satisfied, a "3" appears in the column. The remaining columns give information about the values of the parameter d for each run; columns five through eight giving the initial, maximum, minimum, and final values of this parameter and the final column giving the last iteration at which d was changed.

The results of the tests illustrate that using the unmodified Hessian with the trust region was most effective in reducing the number of O3D iterations *and* the number of SQP iterations. The trust region strategy prevented long, unprofitable steps from being generated when far from the solution and the use of the unmodified Hessian allowed the trust region to become inactive near the solution thus allowing rapid local convergence. Requiring the Hessian to be positive definite often precluded rapid local (q-superlinear) convergence and, when used in conjunction with the trust region strategy, resulted in the trust region's being active close to the solution.

The results also show that the value of the parameter d varied over several orders of magnitude. The procedures discussed in Section 4 that allowed the value of d to increase or decrease greatly enhanced the algorithm; earlier tests using either a fixed value of d or only allowing a reduction in d yielded inferior results.

Another modification in our algorithm, not reflected in the table or included in the description in the preceding section, was made to force the O3D algorithm to take a minimum number of steps. We found that when the trust region radius τ became small the algorithm would sometimes exit O3D after only one iteration, resulting in

a poor step direction. This poor step would result in a further decrease in τ, and eventually the algorithm would fail. When we required a minimum number of steps to be taken in O3D (our choice was 7) this problem disappeared.

Recently a collection of test problems has become available for the testing and comparing of optimization algorithms, (see [13]). The **C**onstrained and **U**nconstrained **T**esting **E**nviroment (CUTE), are quickly becoming standards with which researchers can establish the viability and effectiveness of their numerical algorithms. These problems are replacing the smaller and well scaled test problems of Hock and Schittkowski [25] and Schittkowski [32] which were not intended to be used to test large scale algorithms. Our results on the CUTE test problems are summarized in Tables 6, 7 and 8. These problems were solved to the same stopping conditions as the problems above. Likewise, the same table format was used to present these numerical results. For detailed description of these problems, structure, motivation, and sources see [9].

While it appears that the CUTE test problem set is rich in both large and small scale unconstrained and equality constrained test problems, at present there are not many large scale problems that include inequality constraints (and particularly nonlinear inequality constraints). We chose problems that reflected the class of problems our algorithm was designed to solve. At least one inequality constraint was present in each problem. The number of variables and/or constraints was large enough so that the exploitation of special sparsity structure was important. The problems we selected from CUTE to report on were "CORKSCREW, MANNE, SVANBERG" and "ZIGZAG". The associated problem sizes are recorded in Table 5.

It is worth commenting that much of the machinery developed in this paper deals with effectively handling nonlinear inequality constraints. The performance of our algorithm on the CUTE test problem set is, therefore, slightly deceiving since many of the constraints in these problems are simple bounds on the primal variables or purely linear. (For instance, approximately 83% of the constraints in CORKSCREW, 50% of the constraints in MANNE, and 66% of the constraints in ZIGZAG were linear and many of them were equality constraints). Although these caused no problem for our algorithm, the structure of these constraints was not completely exploited and the extra machinery of our code resulted in an overhead with no performance benefit. Clearly an algorithm designed specifically to deal with linear equality constraints should outperform our algorithm on these problems. The problem on which our algorithm appeared to perform best was SVANBERG, a problem with only inequality constraints (and a substantial number of them are nonlinear).

We succeeded in solving all four problems with a reasonable number of inner and outer iterations. However, many of our algorithmic enhancements contributed little to the solution process. The measure of distance to feasibility (the η–tube strategy), the nonmonotone updating of penalty parameter d, and the trust region strategy were essentially dormant during the solution process regardless of the iterates' proximity to the solution or to feasibility. In fact, the only evidence of our enhancements on the small number of CUTE test problems that we solved occurred when d was decreased slightly while solving the problem MANNE employing modified Hessians with a trust region strategy (see the third and fourth rows of Table 7). It is noteworthy that the iterates that resulted from solving this problem with the penalty parameter artificially held fixed at $d = 1$ were identical to iterates that resulted for the adjusted d solution. This appears to illustrate that in this case the adjustment of d was purely superficial.

Problem	nl-i	qp-i	conv	$\|\nabla_x l\|_\infty$	d_0	$\max d$	$\min d$	final d	last d-cha
NLP1 - c	37	1435	2	1.2e-7	1.00e00	2.08e00	4.45e-2	9.35e-1	34
NLP1 - f	49	1656	1	7.3e-8	1.00e00	1.07e00	5.89e-2	9.20e-1	46
NLP2- c	66	2211	1	1.2e-7	1.00e00	2.83e00	6.89e-2	1.72e00	61
NLP2- f	71	2369	1	3.1e-8	6.98e-1	3.00e00	8.31e-2	1.51e00	64
NLP3-c	29	983	1	6.7e-8	1.00e00	1.12e00	8.13e-1	1.03e00	22
NLP3-f	39	1314	1	4.2e-8	1.00e00	1.05e00	9.84e-1	1.03e00	31
NLP4- c	–	6		Failure	1.00e00	-	-	-	failure
NLP4 -f	–	6		Failure	1.00e00	-	-	-	failure
Truss - c	103	3561	1	4.4e-8	9.87e-1	1.01e00	9.57e-2	9.57e-1	100
Truss - f	110	3799	2	1.9e-7	1.00e00	1.08e00	8.93e-2	8.93e-1	106
Stat - c	135	4561	3	1.1e-8	1.00e00	2.33e00	8.25e-2	9.70e-1	129
Stat - f	144	4805	1	3.3e-8	1.00e00	2.16e00	7.77e-2	8.49e-1	140
BCHeat-c	257	5398	1	7.8e-8	9.18e-1	1.98e00	5.23e-2	1.24e00	254
BCHeat-f	289	5971	1	1.9e-7	1.00e00	4.21e00	4.92e-2	1.37e00	281
Molec-c	37	1376	1	9.8e-9	9.88e-1	1.21e1	1.17e-2	9.81e-1	34
Molec-f	41	1437	2	6.5e-6	1.00e00	2.38e0	1.49e-1	5.52e-1	39

TABLE 2
Modified Hessians with no trust region

Problem	nl-i	qp-i	conv	$\|\nabla_x l\|_\infty$	d_0	$\max d$	$\min d$	final d	last d-cha
NLP1- c	94	1412	2	3.2e-7	1.00e00	8.58e00	3.13e-3	6.55e-2	88
NLP1- f	108	2947	2	6.8e-8	1.00e00	7.50e00	1.23e-3	2.60e-2	99
NLP2 - c	213	2744	1	1.6e-7	1.00e00	3.85e00	1.28e-3	6.48e-1	209
NLP2-f	231	2963	1	5.4e-8	6.98e-1	1.10e01	5.67e-3	8.57e-1	221
NLP3-c	42	932	1	3.7e-8	1.00e00	1.64e00	4.27e-1	9.83e-1	39
NLP3-f	44	946	1	9.1e-8	1.00e00	1.53e00	2.71e-1	9.22e-1	38
NLP4- c	199	2582	2	9.2e-6	1.00e00	1.41e00	8.92e-1	1.18e00	49
NLP4-f	201	2599	2	5.2e-7	1.00e00	2.01e00	9.94e-1	1.21e00	55
Truss - c	195	3528	1	6.1e-8	9.87e-1	1.13e00	9.87e-1	1.11e00	189
Truss - f	195	3544	2	2.9e-7	1.00e00	1.47e00	1.00e00	1.47e00	188
Stat - c	144	4519	3	2.3e-8	1.00e00	2.39e00	1.00e00	1.53e00	140
Stat-f	150	4581	1	4.2e-8	1.00e00	2.48e00	1.00e00	1.89e00	144
BCHeat -c	257	2898	1	8.1e-8	9.18e-1	4.15e00	1.38e-1	4.10e00	249
BCHeat -f	289	3071	1	9.9e-8	1.00e00	3.74e00	2.44e-1	3.89e00	281
Molec-c	39	546	1	1.3e-7	9.88e-1	1.71e01	3.74e-2	2.22e00	36
Molec-f	44	621	1	6.6e-8	1.00e00	1.48e00	7.39e-2	9.52e-2	38

TABLE 3
Modified Hessians with trust region

Problem	nl-i	qp-i	conv	$\|\nabla_x l\|_\infty$	d_0	max d	min d	final d	last d-cha
NLP1-c	43	820	2	2.1e-7	1.00e00	3.69e00	5.13e-2	7.13e-1	38
NLP1-f	46	913	3	1.7e-8	1.00e00	6.58e00	6.27e-2	6.14e-1	39
NLP2-c	51	1330	1	9.8e-8	1.00e00	4.11e00	9.65e-2	5.95e-1	44
NLP2-f	53	1351	1	1.1e-7	6.98e-1	2.94e00	1.20e-1	2.47e-1	48
NLP3-c	35	832	1	4.5e-8	1.00e00	1.89e00	2.46e-2	3.79e-1	29
NLP3-f	39	867	1	7.3e-8	1.00e00	1.57e00	2.22e-2	4.52e-1	28
NLP4-c	–	–		Failure	-	-	-	-	failure
NLP4-f	–	–		Failure	-	-	-	-	failure
Truss-c	94	2242	1	3.9e-8	9.87e-1	1.03e00	1.26e-1	9.11e-1	87
Truss-f	96	2261	1	6.6e-8	1.00e00	1.33e00	5.67e-2	7.84e-1	85
Stat-c	121	1577	3	1.1e-8	1.00e00	2.58e00	1.57e-1	9.34e00	114
Stat-f	121	1585	1	4.7e-8	1.00e00	2.19e00	1.65e-1	7.03e00	111
BCHeat-c	231	2498	1	1.2e-7	9.18e-1	3.24e00	6.04e-2	8.83e-1	226
BCHeat-f	239	2871	3	2.4e-8	1.00e00	1.61e01	3.89e-2	4.98e-1	222
Molec-c	39	550	1	8.76e-8	9.88e-1	1.02e00	4.34e-2	6.04e-1	36
Molec-f	45	658	2	2.3e-7	1.00e00	8.78e00	1.35e-2	6.53e-1	42

TABLE 4
Un-modified Hessians with trust region

Problem	Variables	Constraints
CORKSCREW	96	159
MANNE	300	600
SVANBERG	500	1500
ZIGZAG	304	1206

TABLE 5
Minimization parameters

Problem	nl-i	qp-i	conv	$\|\nabla_x l\|_\infty$	d_0	max d	min d	final d	last d-cha
CORKSCREW - c	4	73	1	2.6e-8	1.d0	1.d0	1.d0	1.d0	0
CORKSCREW - f	5	90	1	3.3e-8	1.d0	1.d0	1.d0	1.d0	0
MANNE - c	8	144	1	1.9e-8	1.d0	1.d0	1.d0	1.d0	0
MANNE - f	8	146	2	1.3e-7	1.d0	1.d0	1.d0	1.d0	0
SVANBERG - c	6	111	1	1.9e-8	1.d0	1.d0	1.d0	1.d0	0
SVANBERG - f	6	111	1	3.9e-8	1.d0	1.d0	1.d0	1.d0	0
ZIGZAG - c	5	93	1	1.8e-8	1.d0	1.d0	1.d0	1.d0	0
ZIGZAG - f	6	99	1	9.9e-9	1.d0	1.d0	1.d0	1.d0	0

TABLE 6
Modified Hessians with no trust region

Problem	nl-i	qp-i	conv	$\|\nabla_x l\|_\infty$	d_0	max d	min d	final d	last d-cha
CORKSCREW - c	4	71	1	4.1e-8	1.d0	1.d0	1.d0	1.d0	0
CORKSCREW - f	5	87	1	5.2e-8	1.d0	1.d0	1.d0	1.d0	0
MANNE - c	8	141	1	2.8e-8	1.d0	1.d0	8.51d-1	8.51d-1	2
MANNE - f	8	142	1	5.4e-8	1.d0	1.d0	8.13d-1	8.13d-1	3
SVANBERG - c	5	91	1	2.5e-8	1.d0	1.d0	1.d0	1.d0	0
SVANBERG - f	6	100	1	1.3e-8	1.d0	1.d0	1.d0	1.d0	0
ZIGZAG - c	5	89	1	1.8e-8	1.d0	1.d0	1.d0	1.d0	0
ZIGZAG - f	5	91	1	1.6e-8	1.d0	1.d0	1.d0	1.d0	0

TABLE 7
Modified Hessians with trust region

Problem	nl-i	qp-i	conv	$\|\nabla_x l\|_\infty$	d_0	max d	min d	final d	last d-cha
CORKSCREW - c	3	39	1	1.1e-8	1.d0	1.d0	1.d0	1.d0	0
CORKSCREW - f	4	43	1	1.9e-8	1.d0	1.d0	1.d0	1.d0	0
MANNE - c	5	64	1	2.4e-8	1.d0	1.d0	1.d0	1.d0	0
MANNE - f	6	75	1	1.2e-8	1.d0	1.d0	1.d0	1.d0	0
SVANBERG - c	3	30	3	1.0e-8	1.d0	1.d0	1.d0	1.d0	0
SVANBERG - f	3	38	3	9.5e-9	1.d0	1.d0	1.d0	1.d0	0
ZIGZAG - c	3	38	1	9.3e-9	1.d0	1.d0	1.d0	1.d0	0
ZIGZAG - f	4	41	1	4.8e-8	1.d0	1.d0	1.d0	1.d0	0

TABLE 8
Un-modified Hessians with trust region

6. Future Directions. In this paper we have discussed in some detail an SQP algorithm for solving large scale nonlinear problems. The numerical results with default parameter settings indicate that the procedures that we have implemented are robust, effective, and efficient; the convergence theory in [4] provides a sound theoretical basis for the procedure. Nevertheless, there are several areas in which the techniques used here can be improved to allow the solution of larger and more difficult problems.

Algorithmically, we observe that the current implementation requires the factorization of both $(\nabla g^{\mathsf{T}} \nabla g + Z)$ and $(\nabla g \nabla g^{\mathsf{T}})$, the latter in O3D. While the sparse matrix package makes this reasonable for the problems that we have currently considered, it is clearly expensive to maintain both.

The results reported here use analytic or finite difference Hessian approximations. An examination of the details of O3D reveals that a limited memory BFGS or limited memory SR1 could be readily incorporated into the code. We have done some experimentation with such techniques; the results will be reported elsewhere [26].

Many of the problems that we have seen have been degenerate and this significantly slows the convergence of the method. The primary culprit is the extremely poor multiplier estimates provided by the least squares procedure. Improvements in this area are certainly required.

In some problems (not reported here) that have nonlinear equality constraints, we have occasionally observed significant difficulty in trying to satisfy the linearized equality constraints, i.e., in completing Phase I. In these cases we have had some success in relaxing the constraints [26]. In the context of O3D, this can be accomplished by simply fixing the artificial variable at some positive value and continuing the O3D iterations. In this approach, we often find that O3D converges and the "recentering" procedure mentioned in Section 2 has led to further improvements. The theory in [4] supports these ideas. The details will, again, be reported elsewhere.

REFERENCES

[1] P. T. Boggs, P. D. Domich, and J. E. Rogers, *An interior-point method for general large scale quadratic programming problems*, Annals of Operations Research, 62 (1996), pp. 419–437.

[2] P. T. Boggs, P. D. Domich, J. E. Rogers, and C. Witzgall, *An interior point method for linear and quadratic programming problems*, Mathematical Programming Society Committee on Algorithms Newsletter, 19 (1991), pp. 32–40.

[3] P. T. Boggs, A. J. Kearsley, and J. W. Tolle, *A merit function for inequality constrained nonlinear programming problems*, Internal Report 4702, National Institute of Standards and Technology, 1991.

[4] ———, *A global convergence analysis of an algorithm for large scale nonlinear programming problems*, SIAM Journal on Optimization, (to appear).

[5] P. T. Boggs and J. W. Tolle, *A family of descent functions for constrained optimization*, SIAM Journal on Numerical Analysis, 21 (1984), pp. 1146–1161.

[6] ———, *A strategy for global convergence in a sequential quadratic programming algorithm*, SIAM Journal on Numerical Analysis, 26 (1989), pp. 600–623.

[7] ———, *Sequential quadratic programming*, Acta Numerica, 1995 (1995), pp. 1–52.

[8] P. T. Boggs, J. W. Tolle, and A. J. Kearsley, *A truncated SQP algorithm for large scale nonlinear programming problems*, in Advances in Optimization and Numerical Analysis: Proceedings of the Sixth Conference on Numerical Analysis and Optimization, S. Gomez and J.-P. Hennart, eds., Dordrecht, 1994, Kluwer Academic Publishers, pp. 69–78.

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

[10] J. Burger and M. Pogu, *Functional and numerical solution of a control problem originating from heat transfer*, Journal of Optimization Theory and Applications, 68 (1991), pp. 49–73.
[11] C. Carthel, R. Glowinski, and J. L. Lions, *On exact and approximate boundary controllabilities for the heat equation. a numerical approach*, Jounal of Optimization Theory and Applications, 82 (1994), pp. 429–484.
[12] T. F. Coleman, *Large scale numerical optimization: Introduction and overview*, in Encyclopedia of Computer Science and Technology, Marcel Dekker (to appear), New York, 1992.
[13] A. R. Conn, N. I. M. Gould, and P. T. Toint, *Lancelot: A Fortran Package for Large-Scale Nonlinear Optimization*, vol. 17 of Series in Computational Mathematics, Springer-Verlag, Heidelberg and New York, 1992.
[14] ———, *Large-scale nonlinear constrained optimization*, in Proceedings of the Second International Conference on Industrial and Applied Mathematics, Philadelphia, PA, 1992, Society for Industrial adn Applied Mathematics, pp. 51–70.
[15] R. S. Dembo, S. C. Eisenstat, and T. Steihaug, *Inexact Newton methods*, SIAM Journal on Numerical Analysis, 19 (1982), pp. 400–408.
[16] J. Dennis, Jr., M. M. El-Alem, and M. C. Maciel, *A global convergence theory for general trust-region-based algorithms for equality constrained optimization*, SIAM Journal on Optimization, 7 (1997), pp. 177–207.
[17] J. E. Dennis, Jr. and R. B. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice-Hall, Englewood Cliffs, New Jersey, 1983.
[18] S. C. Eisenstat and H. Walker, *Globally convergent inexact Newton methods*, SIAM Journal on Optimization, 4 (1994), pp. 393–422.
[19] M. M. El-Alem, *A robust trust region algorithm with nonmonotonic penalty parameter scheme for constrained optimization*, SIAM Journal on Optimization, 5 (1995), pp. 348–378.
[20] P. E. Gill, W. Murray, and M. H. Wright, *Practical Optimization*, Academic Press, New York, 1981.
[21] P. E. Gill, M. A. Saunders, W. Murray, and M. H. Wright, *Constrained nonlinear programming*, in Optimization, G. L. Nemhauser, A. H. G. R. Kan, and M. J. Todd, eds., Amsterdam, 1989, North–Holland, pp. 171–210.
[22] R. Glowinski, *Numerical Methods for Nonlinear Variational Problems*, Springer-Verlag, Berlin, 1984.
[23] W. Glunt, T. L. Hayden, and M. Raydan, *Molecular conformations from distance matrices*, Journal of Computational Chemistry, 14 (1993), pp. 114–120.
[24] M. Gockenbach and A. J. Kearsley, *Optimal signal sets for non-gaussian detectors*, SIAM Journal on Optimization, (to appear) (1997).
[25] W. Hock and K. Schittkowski, *Test Examples for Nonlinear Programming Codes*, Lecture Notes in Economics and Mathematical Systems 187, Springer-Verlag, Berlin, 1981.
[26] A. J. Kearsley, *The Use of Optimization Techniques in the Solution of Partial Differential Equations from Science and Engineering*, PhD thesis, Rice University, 1996.
[27] A. J. Kearsley, R. A. Tapia, and M. Trosset, *The solution of the metric stress and sstress problems in multidimensional scaling using Newtons method*, Computational Statistics, 13 (1998), pp. 369–396.
[28] F.-S. Kupfer and E. W. Sachs, *Numerical solution of a nonlinear parabolic control problem by a reduced SQP method*, Computational Optimization and Applications, 1 (1992), pp. 113–135.
[29] J. L. Lions, *Optimal Control of Systems Governed by Partial Differential Equations*, Springer-Verlag, Berlin, 1971.
[30] ———, *Exact controllability, stabilization and perturbations for distributed systems*, SIAM Review, 30 (1988), pp. 1–68.
[31] J. Moré and D. C. Sorensen, *Computing a trust region step*, SIAM Journal of Scientific and Statistical Computing, 4 (1983), pp. 553–572.
[32] K. Schittkowski, *More Test Examples for Nonlinear Programming Codes*, Lecture Notes in Economics and Mathematical Systems 282, Springer-Verlag, Berlin, 1987.
[33] R. A. Tapia, *On the role of slack variables in quasi-Newton methods for constrained optimization*, in Numerical Optimization of Dynamical Systems, L. C. W. Dixon and G. P. Szegö, eds., Amsterdam, 1980, North-Holland, pp. 235–246.
[34] R. A. Tapia and J. R. Thompson, *Nonparametric Probability Density Estimation*, Johns Hopkins, Baltimore, 1978.
[35] J. R. Thompson and R. A. Tapia, *Nonparametric Function Estimation, Modeling, and Simulation*, SIAM, Philadelphia, 1990.

A. Problem Descriptions.

Nonlinear Program # 1 (NLP1)

$$\min f(x) = \frac{1}{2}((x_1 - x_{100})x_2 + x_{101})^2$$

subject to:

$$x_1 x_{i+1} + (1 + \frac{2}{i})x_i x_{100} + x_{101} \leq 0, \qquad i = 1, \ldots, 99$$

$$(\sin(x_j))^2 - \frac{1}{2} \leq 0, \qquad j = 1, \ldots, 100$$

$$\sin((x_j)^2) \geq 0, \qquad j = 1, \ldots, 100$$

$$x_j \leq j, \qquad j = 1, \ldots, 100$$

$$-x_j \leq 1, \qquad j = 1, \ldots, 100$$

$$(x_1 + x_{100})^2 = 1$$

Explanation: This problem with 101 variables and 500 constraints is taken from [13] where it was used to illustrate separability in nonlinear programming.

Nonlinear Program # 2 (NLP2)

$$\min f(x) = 1000 \left[\left(\sum_{i=1}^{n} x_i^3 \right)^2 - \left(\sum_{i=1}^{n} x_i^2 \right) \left(\sum_{i=1}^{n} x_i^4 \right) \right]$$

subject to:

$$x_1 \geq 0 \quad \text{and} \quad x_n \leq 1$$

$$x_i - x_{i+1} \leq 0, \qquad i = 1, \ldots, (n-1)$$

$$x_i^2 - x_i x_{i+1}^2 \leq 0, \qquad i = 1, \ldots, (n-1)$$

Explanation: There are many local extrema for this problem; we made no special effort to locate global minima. The objective function is highly nonlinear and has a dense Hessian, but the constraints have sparse banded first derivatives. Our example uses $n = 250$.

Nonlinear Program # 3 (NLP3)

$$\min f(x) = \sum_{i=1}^{n} \left[100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2 \right]$$

subject to:

$$x_1 \geq 0 \quad \text{and} \quad x_n \geq 0$$

$$x_i - x_{i+1} \leq 0, \qquad i = 1, 3 \ldots, (n-1)$$

$$4x_{i+1} - x_i^2 - 4 \leq 0 \qquad i = 1, 3 \ldots, (n-1)$$

$$2x_{i+1} + x_i - 1 \leq 0 \qquad i = 1, 3 \ldots, (n-1)$$

Explanation: The objective function here is Rosenbrock's function. The objective function is nonlinear and has a tridiagonal Hessian. The constraints have sparse banded first derivatives. We solved the problem with $n = 250$.

Nonlinear Program # 4 (NLP4)

$$\min f(x) = x^{\mathsf{T}} L^2 x$$
subject to:
$$i - (x_i^2 + x_{2i}^2) \leq 0, \qquad i = 1, \ldots, (n/2)$$
$$\sqrt{2i} - (x_i + x_{2i}) \leq 0, \qquad i = 1, \ldots, (n/2)$$
$$\log(x_i + x_{i+1} + x_{i+2}) - x_i + x_{i+1} + x_{i+2} \leq 0$$
$$i = 1, \ldots, n-2$$

Explanation: The matrix L is the discretized tridiagonal Laplacian operator so the objective function is convex and quadratic. The constraints are nonlinear and the gradients of the active constraints at the solution are linearly dependent. The problem on which we reported results has $n = 1000$.

Truss Problem (Truss)

$$\min_x \rho(c^{\mathsf{T}} x)$$
subject to:
$$S(x)^{-1} F - b \leq 0$$
$$X(x) G S(x)^{-1} F - \beta x \leq 0$$

Explanation: This problem chooses the state variables $x \in \mathcal{R}^n$ to minimize the weight of an optimal n-bar truss design, subject to constraints on the deflection and stress of the truss. The function ρ is the density of the material and in our problem was a non-convex polynomial, $\rho(\zeta) = \zeta^4 - \zeta^2 + 1$. c is a vector containing the lengths of the bars in the truss. The matrix S is the positive definite stiffness matrix, G is a matrix that represents the geometry of the truss and design and F is the vector of applied forces. The vector b and scalar β form bounds on the maximum allowable deflections in the state variables, and the maximum allowable stress in the truss. We solved a problem with $n = 500$ and 1500 constraints.

Maximum Penalized Likelihood Estimate (Stat)

$$\min_x f(x(t)) = -\prod_{i=1}^{n} x(t) e^{\phi_\mu(x(t))}$$
subject to:
$$x \in H^2(-\infty, \infty)$$
$$\int_{-\infty}^{\infty} x(t)^2 dt = 1$$
$$-x(t) \leq 0 \quad \text{for all} \quad t$$

Explanation: This particular maximum penalized likelihood estimator is sometimes referred to as 'the second estimate of Gaskins and Good' (see, e.g., [34] or [35]). We discretize this problem by taking a finite random sample of t_i's say, $t_i \in [\alpha, \beta]$. $\phi(x)$ is defined by

$$\phi(x) = \alpha \int_{-\infty}^{\infty} x'(t)^2 dt + \beta \int_{-\infty}^{\infty} x''(t)^2 dt \tag{A.1}$$

and given $\mu > 0$ the regularized function $\phi_\mu(x)$ is defined by

$$\phi_\mu(x) = \phi(x)\mu \int_{-\infty}^{\infty} x(t)^2 dt. \tag{A.2}$$

The discrete approximate of $x(t)$ was taken to be a cubic spline. The resulting problem had 500 variables and 1000 constraints.

Boundary Control of Heat Equation (BCHeat)

$$\min f(x,y) = \int_0^T \left[(x(1,t) - x_d(t))^2 + ay(t)\right] dt$$

subject to:

$$C(x(z,t))x_t(z,t) - \nabla(\lambda(x(z,t))\nabla x(z,t)) = f(z,t) \quad \text{on} \quad \Omega \times [0,T]$$
$$\lambda(x(z,t))\nabla x(z,t) = b(z,t) \quad \text{on} \quad \partial\Omega \times [0,T]$$
$$x(z,0) = x_0 \quad \text{on} \quad \Omega$$
$$x \in L^2(0,T;H^1(\Omega))$$
$$y \in L^2(0,T)$$

Explanation: The desired profile is denoted by $x_d(t)$ and Ω is a square in $\mathcal{R}^2$. The inequality constraints are quadratic and linear, and arise from enforcing the space conditions $x \in L^2(0,T) \times H^1(\Omega))$ and $y \in L^2(0,T)$. Our discretization results in 500 variables and 1200 constraints. Similar problems have been solved by Newton's method (see [10]), conjugate gradient methods (see [11]) and by reduced methods (see [28]).

Molecule Distance Problem (Molec)

$$\min_{x \in \mathcal{R}^{3d}} \|\Delta - D(X)\|_F$$

subject to:

$$a_{ij} \leq D_{ij} \leq b_{ij}$$

Explanation: Here $\Delta, D(X), a, b \in \mathcal{R}^{m \times m}$ and $X \in \mathcal{R}^{n \times 3}$ where n is the number of atoms and m is the number of interatomic distances ($2m = n^2 - n$). Δ is a set of observed data, X is a configuration of atoms (their locations in $\mathcal{R}^3$, and D is a transformation into the space of "distance matrices". The bound matrices a, b are upper and lower bounds based on estimating errors in measurements. This problem arises in the processing of NMR data for visualization of large proteins and organic molecules (see, e.g., [23] and [27]). The results in the tables correspond to a problem we solved with 100 variables and 5000 constraints.