



Computational Experience with a Safeguarded Barrier Algorithm for Sparse Nonlinear Programming

RAVINDRA S. GAJULAPALLI
L.S. LASDON
*MSIS Department, College of Business Administration, The University of Texas at Austin, Austin, TX,
78712-1175, USA*

gajulapalli@earthlink.net
lasdon@mail.utexas.edu

Received October 26, 1998; Accepted December 16, 1999

Abstract. We describe an enhanced version of the primal-dual interior point algorithm in Lasdon, Plummer, and Yu (ORSA Journal on Computing, vol. 7, no. 3, pp. 321–332, 1995), designed to improve convergence with minimal loss of efficiency, and designed to solve large sparse nonlinear problems which may not be convex. New features include (a) a backtracking linesearch using an L1 exact penalty function, (b) ensuring that search directions are downhill for this function by increasing Lagrangian Hessian diagonal elements when necessary, (c) a quasi-Newton option, where the Lagrangian Hessian is replaced by a positive definite approximation (d) inexact solution of each barrier subproblem, in order to approach the central trajectory as the barrier parameter approaches zero, and (e) solution of the symmetric indefinite linear Newton equations using a multifrontal sparse Gaussian elimination procedure, as implemented in the MA47 subroutine from the Harwell Library (Rutherford Appleton Laboratory Report RAL-95-001, Oxfordshire, UK, Jan. 1995). Second derivatives of all problem functions are required when the true Hessian option is used. A Fortran implementation is briefly described. Computational results are presented for 34 smaller models coded in Fortran, where first and second derivatives are approximated by differencing, and for 89 larger GAMS models, where analytic first derivatives are available and finite differencing is used for second partials. The GAMS results are, to our knowledge, the first to show the performance of this promising class of algorithms on large sparse NLP's. For both small and large problems, both true Hessian and quasi-Newton options are quite reliable and converge rapidly. Using the true Hessian, INTOPT is as reliable as MINOS on the GAMS models, although not as reliable as CONOPT. Computation times are considerably longer than for the other 2 solvers. However, interior point methods should be considerably faster than they are here when analytic second derivatives are available, and algorithmic improvements and problem preprocessing should further narrow the gap.

Keywords: non linear programming, barrier algorithm, SQP algorithm, large sparse problems

Introduction

Recently, several authors have described interior point algorithms for twice differentiable nonlinear programs. We focus here on approaches that are not restricted to convex problems. El-Bakry, Tapia, et al. [10] describe a primal-dual interior point algorithm which uses the squared L2 norm of the Kuhn-Tucker residuals as a linesearch merit function, and guarantees descent directions for this function by appropriate choice of the barrier parameter. Global convergence to a Kuhn-Tucker point is proved, and conditions on 2 key algorithm parameters which guarantee Q-superlinear and Q-quadratic convergence are

derived under standard smoothness, regularity, second order sufficiency, and strict complementarity assumptions. In [1], Argaez and Tapia study a similar algorithm, which uses a modified augmented Lagrangian as the linesearch objective. References [15] and [23] describe primal-dual interior point methods for optimal power flow problems, and demonstrate excellent computational results when analytic second derivatives are provided. Lasdon, Plummer, and Yu [18] present a similar algorithm, and apply it to a variety of small to medium size NLP's, approximating first and second derivatives by finite differencing. Almost all of the test problems are solved efficiently, even though there is no linesearch or trust region strategy to force convergence. In [4], Byrd et al. propose an interior point algorithm using a trust region strategy, and prove its global convergence.

This paper describes an extension of the algorithm in [18] designed to improve its reliability. That method, a direct extension of primal-dual interior point methods for LP, applies Newton's method to the Kuhn-Tucker conditions of a barrier problem, formulated so that these conditions are perturbed versions of the Kuhn-Tucker conditions of the original problem. The full Newton step is taken, limited only by the requirement that primal and dual variables remain within their bounds, and the barrier parameter is reduced after each Newton step. As in [1] and [10], the enhanced procedure described here uses a line search to force convergence from poor starting points, but the merit function is the L1 true penalty function, originally proposed in the context of SQP algorithms by Han [16] and Powell [22]. We describe mechanisms that ensure that the search directions generated are downhill for this function. We also allow several Newton steps to be taken before the barrier parameter is reduced. A sparsity exploiting Fortran implementation is discussed, including improved termination criteria. This implementation also includes efficient data structures for Jacobians, Hessians, and Newton matrix coefficients—for further details on these data structures, see Gajulapalli [13]. Extensive computational experience is described, both with smaller models coded in Fortran and with larger models implemented in the GAMS modeling language.

1. Problem and algorithm statement

We assume that slack variables have been used to convert all inequality constraints to equalities. Then, letting x denote the vector containing both structural and slack variables, the problem to be solved has the form

Problem NLP

$$\begin{aligned} &\text{minimize} && f(x) \\ &\text{subject to} && g(x) = b \\ & && x_i \leq u_i, \quad i \in U \quad x_i \geq l_i, \quad i \in L \end{aligned}$$

where $x \in R^n$, $g : x \rightarrow R^m$, and U and L are arbitrary subsets of $\{1, \dots, n\}$. The functions f and g are assumed to have continuous second partial derivatives at all points x satisfying the bounds. Adding slacks su_i and sl_i to convert the bounds to equalities, the problem

becomes

$$\begin{aligned} & \text{minimize} && f(x) \\ & \text{subject to} && g(x) = b \end{aligned} \tag{1}$$

$$x_i + su_i = u_i, \quad i \in U \tag{2}$$

$$x_i - sl_i = l_i, \quad i \in L \tag{3}$$

$$su_i \geq 0, \quad i \in U \quad sl_i \geq 0, \quad i \in L \tag{4}$$

The Lagrangian function for this problem is

$$\begin{aligned} L(x, y, w, z, sl, su) = & f(x) + y^t(g(x) - b) + \sum_{i=1}^{nw} w_i(x_{j_i} + su_{j_i} - u_{j_i}) \\ & - \sum_{i=1}^{nz} z_i(x_{k_i} - sl_{k_i} - l_{k_i}) \end{aligned} \tag{5}$$

where y is the vector of Lagrange multipliers for the equalities (1), j_i and k_i are the elements of U and L respectively (sorted in increasing order), and w_i, z_i are multipliers for the upper and lower bounds (2) and (3).

Incorporating the nonnegativities (4) into the objective using a logarithmic barrier function with positive barrier parameter μ , leads to the following barrier problem:

$$\text{minimize} \quad f(x) - \mu \left(\sum_{i \in U} \ln(su_i) + \sum_{i \in L} \ln(sl_i) \right) \tag{6}$$

$$\text{subject to} \quad g(x) = b \tag{7}$$

$$x_i + su_i = u_i, \quad i \in U \tag{8a}$$

$$x_i - sl_i = l_i, \quad i \in L \tag{8b}$$

Fiacco and McCormick [11] prove that, under very mild conditions, if this barrier problem is solved (globally) for a sequence of positive μ values converging to zero, the resulting sequence of solutions $\{x(\mu)\}$ converges to a (global) solution of the original problem. As is common with interior point procedures, our algorithm requires only an inexact solution to each barrier problem.

As is standard in primal-dual interior point algorithms, we write the Kuhn-Tucker conditions for this barrier problem, and linearize these to obtain a set of linear Newton equations in the change vectors (dx, dy, dw, dz) —see [17] for details. If dw and dz are eliminated from the Newton system, the equations for dx and dy are:

$$\begin{bmatrix} H + Dw + Dz & J^t \\ J & 0 \end{bmatrix} \begin{bmatrix} dx \\ dy \end{bmatrix} = \begin{bmatrix} v \\ rg \end{bmatrix}. \tag{9}$$

In the above, Dw and Dz are diagonal matrices, with diagonal elements either 0 or of the form (w/su) or (z/sl) . These and the right hand side of (9) are defined in [18]. The matrix H is either the true Lagrangian Hessian $\nabla_x^2 L$ or a positive definite approximation of this

matrix. The coefficient matrix of (9) is symmetric but indefinite. Note that, if x_i is at a bound in an optimal solution (su_i or $sl_i \rightarrow 0$) and the corresponding multiplier component in w or z is positive, then the elements of Dw and Dz tend to become very large as the optimum is approached. Their magnitude is limited by imposing a small positive lower limit on the bound slacks su and sl .

Line search

The line search objective used here is the L1 exact penalty function associated with the barrier subproblem (6)–(8). This choice was inspired by the widespread use of this function in SQP algorithms, and was originally suggested by Han in [16] and later by Powell [22]. The function is

$$P(x; pw, \mu) = f(x) - \mu \left(\sum_{i \in U} \ln(u_i - x_i) + \sum_{i \in L} \ln(x_i - l_i) \right) + \sum_{i=1}^m pw_i (\text{abs}(g_i(x) - b_i)) \quad (10)$$

If each weight pw_i is larger than the absolute value of the i th optimal Lagrange multiplier for the barrier subproblem, then any local solution of that subproblem is a constrained stationary point of P (see Luenberger [19]). Hence it is a suitable linesearch merit function for an algorithm if that algorithm generates descent directions for P . P is differentiable at all points x where $\text{abs}(g_i(x) - b_i) \neq 0$ for all i .

Define

$$\phi(\alpha_x) = P(x + \alpha_x dx; pw, \mu) \quad (11)$$

and

$$\phi'(\alpha_x) = \nabla_x P^T(x + \alpha_x dx; pw, \mu) dx \quad (12)$$

It is known (see [22]) that the dx component of the Newton direction obtained by solving the linear system (9) is a descent direction for P under the following conditions:

- (a) $pw_i \geq \text{abs}(y_i^+)$ where y_i^+ is the updated multiplier estimate, $y_i^+ = y_i + dy_i$.
- (b) The matrix in the upper left partition of (9), $H + Dw + Dz$, is positive definite on the null space of J .

Condition (a) is satisfied by updating the penalty weights using a slight modification of Powell's update formula [21]:

$$pw_i^+ = \max(1.5 \text{abs}(y_i^+), (pw_i + \text{abs}(y_i^+))/2) \quad (13)$$

where the factor of 1.5 assures that pw_i^+ slightly exceeds the current multiplier estimate. Condition (b) is satisfied if H is a positive definite approximation of the Lagrangian Hessian,

but it may not be satisfied if H is the true Lagrangian Hessian (or a finite difference approximation to it). Our extensive computational experiments show that dx is often a descent direction for P when H is the true Hessian, probably because the nonnegative diagonal matrices Dw and Dz tend to overcome lack of positive definiteness in H . However, dx fails to be a descent direction for P in a small to moderate fraction of iterations. Table 3 shows dx to be uphill about 20% of the time for problems posed in Fortran, where there is considerable error in second derivative estimates, but only 10% of the time in GAMS problems, where this error is much less.

A natural strategy for dealing with uphill directions is to add a positive multiple of the identity matrix, say λI , to $H + Dw + Dz$, and increase λ until dx is a descent direction. The resulting linear system (9) is the Kuhn-Tucker system for the following quadratic program:

$$\text{minimize } -v^T dx + 1/2 dx^T (H + Dw + Dz) dx \quad (14)$$

$$\text{subject to } J dx = rg \quad (15)$$

$$\text{and } dx^T dx \leq r \quad (16)$$

where λ is the Lagrange multiplier for the trust region constraint (16), and $r > 0$ is the square of the trust region radius. This requires that the matrix in (9) be refactorized.

The line search used is a standard backtracking algorithm using safeguarded quadratic interpolation. The steps are:

- (a) $\alpha_x \leftarrow \alpha_x^{\max}$.
- (b) $x \leftarrow x + \alpha_x dx$. Evaluate $f(x)$, $g(x)$ and $\phi(\alpha_x)$ in (11).
- (c) If $\phi(\alpha_x) \leq \phi(0) + 10^{-6} \alpha_x \phi'(0)$ stop, return α_x .
- (d) $\alpha_x^+ \leftarrow \max(0.1\alpha_x, quad)$ where $quad$ is the minimizer of a quadratic fit through the data $\phi(0)$, $\phi'(0)$, $\phi(\alpha_x)$. If $\alpha_x^+ > \alpha_x$, $\alpha_x \leftarrow \alpha_x/2$, else $\alpha_x \leftarrow \alpha_x^+$.
- (e) go to step (b).

The Armijo constant of 10^{-6} in step (b) is quite small, but algorithm performance was not sensitive to this factor in a limited set of experiments. We plan to revisit this issue in the future.

The algorithm

This safeguarded interior point algorithm requires initial values for the barrier parameter μ , the variables (x, y, w, z) , the initial Lagrangian Hessian estimate if the quasi-Newton option is used, and various tolerances and parameters. Initial x values are user-supplied, but if any are too close to or violate a bound, they must be perturbed to be interior to the bound. In the description that follows, e is a vector of all ones of appropriate dimension,

$$\bar{l}_i = l_i + 0.1(1 + \text{abs}(l_i)), \bar{u}_i = u_i - 0.1(1 + \text{abs}(u_i)),$$

and η is the step size reduction factor, insuring that x , w , and z do not quite reach their bounds.

0. Initialize: $\mu \leftarrow \mu_0, y \leftarrow e, w \leftarrow 10e, z \leftarrow 10e, x \leftarrow x_0$ (user supplied) $\eta \leftarrow .995$. If $x_i \leq \bar{l}_i$, then $x_i = \bar{l}_i$ and if $x_i \geq \bar{u}_i$, then $x_i = \bar{u}_i$. If the approximate Hessian option is chosen, $H \leftarrow I$. Set the inner and outer iteration counters, inner and t respectively, to 0.
 1. Compute f, g , and ∇g at the current point. Evaluate the infinity norm of the Kuhn-Tucker error for the original problem, $ktnorm$, and for the barrier problem, $bktnorm$.
 2. Check convergence criteria and stop if any are met. See below for details.
 3. (Inner loop convergence test for updating μ). If $bktnorm < 1000\mu$ or $inner \geq nsub$ (default value 5) then $\mu \leftarrow \min(0.95\mu, 0.01^*(0.95)^t * ktnorm)$ and $inner \leftarrow 0$.
 4. If the true Hessian option is used, compute the Hessians of f and each g_i and form $\nabla_x^2 L$. If the quasi-Newton option is used, update the approximate Lagrangian Hessian. Form the coefficient matrix of the Newton system, (9).
 5. Solve the Newton system (9) for (dx, dy) and find (dw, dz) from the remaining Newton equations (see [18]).
 6. Replace y by $y^+ = y + dy$.
 7. If $t = 1, pw \leftarrow \text{abs}(y^+)$. Otherwise, update the penalty weights pw using Eq. (13).
 8. Test for descent direction:
If $\phi'(0) < -eps$ where eps is a small positive number, go to step 9.
Select an initial value for λ or, if λ is already initialized, multiply it by 256 (this factor is large to reduce the number of refactorizations). Add λI to $H + Dw + Dz$ in (9) and return to step 5. If this step is repeated more than five times, do step 9 below and go to step 11, i.e., force a constrained Newton step.
 9. Compute $\bar{\alpha}_x, \bar{\alpha}_w, \bar{\alpha}_z$ as the largest steps along dx, dw, dz which maintain x, w, z within their bounds, and define maximum allowable step sizes as $\alpha_x^{\max} = \eta\bar{\alpha}_x, \alpha_w^{\max} = \eta\bar{\alpha}_w, \alpha_z^{\max} = \eta\bar{\alpha}_z$. Compute new estimates for w and z by: $w \leftarrow w + \alpha_w^{\max} dw, z \leftarrow z + \alpha_z^{\max} dz$.
 10. Linesearch in x : call the linesearch procedure outlined in steps (a)–(e) above to compute a new estimate for x .
 11. Increase $inner$ and t by one and go to step 1.

The convergence criteria in step 2 have become more sophisticated as this implementation has evolved. Initially they were as shown below, applying only at feasible points:

2. If any of the following convergence criteria are met, stop:
 - 2.1 Each of the equality constraints (2) is satisfied to within a relative feasibility tolerance ε_f (default value .0001), i.e.,
 - 2.2 $|g_i(x) - b_i| \leq \varepsilon_f(1 + |b_i|), i = 1, \dots, m$, and either
 - 2.3 $ktnorm < \varepsilon_0$, where ε_0 is the optimality tolerance (default value .0001), or
 - 2.4 The fractional change in the objective is less than ε_0 for $nstall$ consecutive iterations (default value 5)

$$|f(x_{t+1}) - f(x_t)| \leq \varepsilon_0(1 + |f(x_t)|), \text{ or}$$

- 2.5 The iteration limit is reached, i.e., $t \geq itlim$ (default value 300).

However, some problems never attained feasibility, and these would run until the iteration limit was reached, with values changing very slowly at the end. Total run times for the 89 GAMS problems were reduced by about a factor of 3 by adding stop criteria which apply at both feasible and infeasible points to those described above. At each iteration an “incumbent” point (x_i, y_i, w_i, z_i) is updated conditionally. Starting at the initial values described in step 0 above, it is replaced by the current point if the incumbent is infeasible and (a1) the current point has a smaller number of infeasibilities, or (a2) it has the same number of infeasibilities and a largest infeasibility lower than the incumbents. If the incumbent is feasible it is replaced if the current point is feasible and (b1) its objective value is better than the incumbents, or (b2) the objective values are nearly the same but $ktnorm$ has improved. A logical variable “sufficient” is set to true if (a1) or ((a2) or (b1) or (b2) and the relative improvement exceeds ε_0). A counter, *noprogress*, is counted up when sufficient is false and is reset to 0 if sufficient is true. If *noprogress* exceeds a limit (default value 35) the algorithm stops. The final point returned is always the incumbent, so in cases where feasibility is attained and then lost, the best feasible point found is returned.

The approximate Hessian update in step 4 uses the BFGS updating formula as modified by Powell in [22] and also described in [18]. Since the approximate Hessian using this update is dense, this option is probably not efficient when the dimension of H is more than a few hundred, because of the large amount of memory and time required to store and manipulate H . This is confirmed by the results in Table 3. A large dense H will also make the linear system (9) much harder to solve. However, the approximate Hessian need not have dimension equal to the total number of variables, n , in problems where some variables appear linearly in each problem function (e.g., slacks). In such problems, the true Lagrangian Hessian has a row and column of zeros corresponding to each linear variable. Since nonzero rows and columns correspond to variables that appear nonlinearly in at least one function (called nonlinear variables), only the portion corresponding to the nonlinear variables need be approximated and updated. For this reason, our quasi-Newton implementation maintains an approximate Hessian whose dimension equals the number of nonlinear variables. Hence in problems with less than, say, 100 nonlinear variables, the approximate Hessian will be small enough and can be handled easily. As a consequence, H will not be positive definite, only positive semidefinite, when there are linear variables. The computational results of Section 3 show no adverse affects of this consequence.

The formula for updating μ in step 3 is similar to that used in the original algorithm in [18]. The basic idea is to reduce μ as $ktnorm$ decreases, keeping μ smaller than $ktnorm$ via the factor 0.01. The 0.95μ term guarantees that μ decreases, while $(0.95)^t$ in the second term guarantees superlinear convergence for LP's (see [24]). This aspect of the algorithm is purely empirical because strong duality theory, which provides a more solid foundation in linear programming, is not applicable to nonconvex NLP's.

2. Implementation

The algorithm described above has been implemented in a Fortran system called INTOPT. This system is designed to solve large sparse NLP's, using sparse data structures to store and access first and second derivatives of all problem functions and the coefficient matrix of the

linear Newton equations (9) (henceforth called A). The computational results of Section 3 show that, for problems with more than a few thousand nonzero Jacobian elements, over half the computation time is spent solving this system, so both its sparsity and symmetry must be exploited. We have chosen to solve these equations using the Harwell MA47 sparse linear equation solver for symmetric indefinite systems from the Harwell library, because it is known to be both efficient and reliable. It uses multifrontal sparse Gaussian elimination, and is described by Duff and Reid in [9]. This routine has been both faster and more reliable than the general purpose MA28 Harwell linear solver used in the previous implementation of this algorithm described in [18].

INTOPT has interfaces which accommodate problems posed in two forms: (a) as “black box” Fortran subroutines, where all problem functions are evaluated simultaneously and first and second derivatives are approximated by differencing and (b) as models coded in the GAMS modeling language [3]. GAMS provides analytic first derivatives and identifies those which are constant. Second derivatives are approximated by differencing the non-constant first derivatives.

With Fortran models, INTOPT must estimate numerically which Jacobian elements are identically zero, nonzero but constant, and non-constant. This classification phase approximates first partials of all functions with respect to each structural variable by either forward or central differencing, at each of K selected points (default $K = 4$). A derivative is assumed to be nonzero if at least one of these estimates has absolute value larger than a tolerance (default 1.D-8), and is non-constant if at least one of the pairwise differences between the values at the K points has absolute value greater than a relative tolerance (default 1.D-6). Variables are classified linear if all derivatives are constant, and these variables are skipped when derivatives are reevaluated at other points. Second partials are classified by a similar procedure, using a differencing formula requiring 4 function evaluations, as described in [13]. This derivative classification can require substantial time, and of course can classify incorrectly. Classification times are discussed in Section 3.

With GAMS models, a similar numerical procedure is required to classify second derivatives arising from the non-constant first derivatives. Much less time is required than in the Fortran case, but it can still be significant for large problems. An ideal environment for INTOPT (or any other NLP algorithm requiring second derivatives) is an interface to a modeling language which efficiently supplies analytic first and second derivatives in sparse form, classifying derivatives quickly and exactly, and re-computing only those which are non-constant. The AMPL modeling language [12], which now provides second derivatives [14], is such an environment.

Creating, storing and scaling the coefficient matrix of the Newton equations

The coefficient matrix of (9), A , is stored in packed column form, one triangle only. When the Jacobian, J , is acquired by rows, the upper triangle is stored because it contains J^T , whose columns are available. The lower triangle is stored when J is acquired by columns. Each time non-constant first and second derivatives are re-computed, A is updated. Related Jacobian pointers allow storage in A of its J and J^T submatrices. A is symmetrically scaled prior to being passed to MA47, using scale factors which are integral powers of the machine

base, b (default $b = 16$). The powers are chosen with the goal of dividing row and column i of A by the integral power of b closest to the maximum absolute element in that row and column. A lower bound on A 's condition number is dramatically reduced by this scaling.

3. Computational results

In this Section we describe the results of applying this INTOPT implementation to test problems coded both in Fortran and GAMS, using both true Hessian and Quasi-Newton modes. All computational tests were performed on a DEC Alpha computer running under the UNIX operating system OSF/1. All INTOPT routines were written in Fortran and all except the Harwell MA47 linear equation solver were compiled using the Digital Fortran 77 compiler at optimization level 5 (the highest level). The Harwell routines also used level 5 but with loop rearranging prohibited. Execution times are given in CPU seconds, obtained by the Fortran function call *etime*, which has a resolution of 1/60 sec. Important algorithm parameters are as specified in the algorithm description of Section 1 and the initial μ is 1.0 unless otherwise indicated.

Problem characteristics

Of the 34 Fortran test problems, 31 come from the well known Himmelblau [17] and Dembo [5] sets. The number of variables and constraints, and A matrix size, non-zeros and density for these problems are in Lasdon, Plummer, and Yu [18]. All but 2 have 24 or fewer variables, all have 22 or fewer constraints, and all but 3 have small, dense A matrices with 312 or fewer nonzeros. The remaining 3 problems are all instances of a small multiperiod economic planning model described in Murtagh and Saunders [20] with 25, 50, and 75 time periods. Ranked by A matrix nonzeros, the largest of these problems is econ75, which has 375 primal and 552 dual variables, 150 constraints, and an A matrix of size 525 with 2431 nonzeros and a density of 0.88%.

There are 61 GAMS models with 89 SOLVE statements. Fifty five models come from the library included with GAMS, and 6 others (somewhat more difficult) were provided by the GAMS Development company. Detailed data for all GAMS problems are provided in Gajulapalli [13]. We have sorted them by A nonzeros in increasing order, and then divided them into 4 subsets of nearly equal size (23 problems in sets 1–3, 20 in set 4). Summary statistics for the 4 sets are shown in Table 1 below.

While none of these problems are large by LP standards, set 4 is large enough to require a sparsity-exploiting solver: A averages over 6000 nonzeros, with a maximum of 24,309. Both A and the Lagrangian Hessian, H , become quite sparse as size increases, with H densities which are gratifyingly small, e.g., an average density of .01% for the 22 largest problems in set 4. In addition, many of the constant nonzero elements of A have identical values, partially due to its symmetry and to the presence of many constraint slacks, which contribute only +1's and -1's to J and all zeros to H . For set 4, H averages only 6.5% unique elements, and J 11.9%. This is exploited by the "super-sparse" data structures used in the code.

Table 1. GAMS problem sets-summary characteristics.

Problem set	Constraints	Variables (incl. slacks)	Jacobian nonzeros	A nonzeros	Hessian density (%)	A density (%)
1,max	10	21	38	106	30	72
1,avg.	5.5	8.0	22.3	59.6	7.7	37.9
2,max	27	63	147	353	2.3	47.7
2,avg.	17.3	28.6	87.8	220.9	1.3	12.0
3,max	102	153	437	1146	0.634	12.9
3,avg.	51.4	78.1	227.2	639.3	0.2	4.4
4,max	1970	1207	9777	24309	0.273	8.6
4,avg.	328	518.2	2318.9	6051	0.01	1.5

Table 2. INTOPT solution outcomes using true Hessian and quasi-Newton.

	Success	Fail	KTC	Feas FC	Infeas FC	Solv
Fortran, true	32	2	23	10	1	0
Fortran, QN	32	2	29	4	1	0
GAMS, true	80	9	64	20	4	1
GAMS,QN	76	13	44	36	9	0

Algorithm performance-base case results

Solution outcomes for the safeguarded algorithm described in Section 1 on both problem sets, using the true Hessian and Quasi-Newton (QN) options, are shown in Table 2. A run of the algorithm is deemed a success if the point returned is feasible and its objective value deviates from one of a set of predefined locally optimal values by no more than 0.5%, or is an improvement over the best of these values. Several problems have more than one distinct local optima, and all optimal objective values known to us are included. The heading *KTC* means the Kuhn-Tucker conditions are satisfied to within the optimality tolerance. *Feas FC* indicates that the final point was feasible and either the fractional objective change was less than the optimality tolerance for *nstall* consecutive iterations or the incumbent had not been updated for *nprog* consecutive iterations. *Infeas FC* means that the final point was infeasible and the fractional change in the largest infeasibility was less than the optimality tolerance for *nstall* consecutive iterations or the incumbent had not been updated for *nprog* consecutive iterations. *Solv* means failure of the Harwell MA47 symmetric indefinite solver when attempting to solve the Newton system (9). Fortran runs use *nstall* = 12, *nprog* = 50, while GAMS runs use *nstall* = 5, *nprog* = 35. We believe the GAMS runs can use the smaller limits with only a few premature terminations because derivatives are available to higher accuracy.

The outcomes for the Fortran models are very satisfactory, especially since two of these models are nonsmooth and one failure occurs there. We did not expect the QN option

Table 3. INTOPT performance measures using true Hessian and quasi-Newton.

	Outer Itns	Inner Itns	Newt steps	Uphill dirs	LS fails	Diag corr.	Alg. time (cpu sec)	Setup time
Fortran,true	17.6	33.1	14.5	6.6	6.2	11.4	2.1	1.8
Fortran,QN	20.2	28.0	13.9	0.4	1.9	0	0.5	1.1
GAMS,true	13.4	24.3	6.5	2.3	0.5	2.6	8.0	0.9
GAMS,QN	19.1	31.2	10.1	1.6	0.6	1.3	67.6	0.6

to have more KTC terminations than the true Hessian, and attribute this to numerical differencing error in the second derivatives, as these are correct only to about 4 significant digits. The GAMS outcomes support this conclusion, since there the true Hessian option has a much larger fraction of KTC terminations, and second derivatives have approximately 8 digits of accuracy. The number of GAMS failures is also quite small, and some of these can be solved with different INTOPT parameters or options, as we discuss shortly.

Table 3 shows performance measures for the runs of Table 2. All are averages over the 34 Fortran and 89 GAMS problems.

For the Fortran problems, true Hessian, the average number of inner iterations, 33.1, is somewhat above the 25.2 averaged by the unsafeguarded algorithm in [18] on the 31 Himmelblau and Dembo problems. However, the unsafeguarded method failed on 3 of these problems, all within the more difficult Dembo set, and it used an iteration limit of 100 rather than the 300 used here. The safeguarded algorithm has roughly 2 inner iterations per outer iteration, and 14.5 pure Newton steps per run. The Newton direction is uphill for the linesearch objective an average of 6.6 times per problem, and there are 6.2 iterations per problem where the linesearch cannot find an improved point. The Diagonal corrections of step 8 are invoked an average of 11.4 times (both when the linesearch fails and when the initial search direction is uphill), and are almost always successful. Hence the diagonal corrections occur, on average, in about 20% ($6.6/33.1$) of the inner iterations, and about 1.72 ($11.4/6.6$) corrections are required to achieve a sufficiently downhill direction. Values for the Quasi-Newton option are similar, except that uphill directions are rarely produced. Average solution time for the QN option, 0.5 seconds, is much shorter than for the true Hessian, 2.1 seconds. This is because second derivatives take a long time to estimate using finite differencing, and these are all small problems, where the dense approximate Hessian is not a liability.

For the GAMS problems, true Hessian option, the iteration counts are significantly reduced over those for the Fortran models, despite the fact that some GAMS models are much larger. The GAMS runs also have far fewer uphill directions and linesearch failures. We believe that this is caused by our use of smaller values for the iteration limits $nprog$ and $nstall$ for the GAMS models, and by the increased precision of first and second derivatives within GAMS. As with the Fortran models, the Diagonal corrections almost always produce downhill directions. The QN option solves 4 fewer of these problems than the true Hessian option, and takes about 8 times longer on average, due to the long times required by MA47 to solve linear systems with so many nonzeros. The QN option with dense approximate

Hessian is clearly limited to problems with at most a few hundred variables, but use of a sparse approximation could extend this considerably.

For the Fortran problems, average setup time (time needed to estimate which first and second derivatives are nonzero and/or nonconstant) is nearly equal to algorithm time for the true Hessian option and more than twice this value for the QN option. However, relative setup time is much smaller for the GAMS runs, since GAMS specifies the nonzero/nonconstant status of first derivatives. Breaking down average algorithm time for the 12 largest GAMS problems, 58% is spent in the MA47 linear equation solver and 21% estimating second derivatives. The linear algebra portion should increase with problem size.

Comparisons using the same implementation but changing the barrier parameter at each iteration and omitting the linesearch (i.e., the unsafeguarded version of this algorithm described in [18]) favor the safeguarded version. For the Fortran problems, the unsafeguarded algorithm fails on 3 of the 34 problems using the true Hessian and 4 of 34 in QN mode, and has significantly fewer KTC terminations, 13 for both true and QN modes respectively. For the GAMS problems, the unsafeguarded version fails on 4 more problems than the safeguarded algorithm in true Hessian mode and on 9 more problems in QN mode. Again there are far fewer KTC terminations, 52 with the true Hessian and 20 in QN mode.

If the inner iteration limit $nsub$ in step 3 is set to 1, this effectively disables the inner loop, forcing a reduction in μ at each iteration. Algorithm performance on the GAMS models changes very little. There is one more failure, one fewer KTC termination, and iteration averages and time remain about the same. However, 3 of the 9 problems where the algorithm failed previously are now solved. Since 2 of these failures terminated infeasible, this suggests that, when the algorithm terminates infeasible, setting $nsub$ to 1, resetting μ to 1, and restarting from the current point may be an effective strategy. One additional base case failure terminates with a feasible solution which is within 0.7% of a known optimal value (our success criterion is 0.5%), and 3 more can be solved by changing the shrink parameter η or the lower limit on the bound slacks sl and su . Hence the true Hessian option can solve 87 of the 89 problems with some tuning.

The widely used GAMS solvers CONOPT [6] and MINOS [20] were also applied to these problems, also using their default parameters and options. MINOS failed on 8 of the 89 models, terminating with an infeasible solution in all cases. Using its default parameters, CONOPT solved all but 3 (2 reached iteration limits, one had too large Jacobian elements). Two of the 3 CONOPT failures are solved by INTOPT, as are 4 of the 8 MINOS failures. This indicates that INTOPT may be an effective complement to CONOPT and/or MINOS, but the sample is far too small to draw reliable conclusions. MINOS required 102 cpu seconds to process all 89 problems and CONOPT needed 219 cpu seconds versus 714 for INTOPT. INTOPT spends 87% of its time processing the 10 longest problems, and similarly for MINOS and CONOPT. When the problems are sorted by A matrix nonzeros, INTOPT reports lower times on almost all the smaller problems, winning on 42 of the 89 versus CONOPT and 38 versus MINOS. Regarding CONOPT, we believe that INTOPT's shorter times are due to time spent by CONOPT in problem preprocessing (see Drud, [7]), since most CONOPT times are about 0.15 seconds for these smaller problems. These comparisons are confounded by the fact that CONOPT's preprocessing includes reformulating the objective equation (GAMS adds an extra objective variable and sets it equal to the true objective;

CONOPT removes this variable and equation), and INTOPT does not. The formulation with the extra row and column is harder to solve for almost all NLP codes.

4. Conclusions

This paper is the first to report computational experience with a nonlinear interior point algorithm on large sparse NLP's. This experience is sufficiently favorable to warrant further development efforts. First, there are several promising algorithmic options to investigate. These include the line search objective defined in Argaez and Tapia [1] together with their weakened centrality measure. These may make the inner loop of this algorithm more effective by making the inner subproblem easier to solve. Another option is to treat the bound slacks su and sl as independent, allowing their defining equations (2)–(3) to be violated during the iterations. The primal variables x can then be allowed to exactly equal a bound, while the bound slacks are maintained positive throughout. The modified barrier functions defined by Polyak [21] and investigated by Breitfeld and Shanno [2] may be used in place of the classical logarithmic barrier function, avoiding its inherent ill conditioning. Finally, use of a sparse approximate Hessian update within the quasi-Newton option is certainly worth investigating. It would enable this option to address problems with over a thousand nonlinear variables, but its efficiency and reliability are open questions.

Planned software enhancements include adding an automatic row and column scaling option, and perhaps some presolve tactics. These have been very effective in linear programming, and have been examined by Drud [7] for nonlinear problems. Further tests and tuning of INTOPT's algorithmic options and parameters are also envisioned.

Heading the computational list is testing INTOPT in environments that provide analytic second derivatives. These have recently become available within the AMPL modeling language [14]. INTOPT should be more reliable and faster when differencing errors in its Hessian evaluations are eliminated and Hessian evaluation time is reduced. The quasi-Newton option using a dense approximate Hessian is also competitive with available SQP algorithms for small problems, and may be superior for problems with hundreds of nonlinear variables. Comparisons with SQP codes are planned.

References

1. M. Argaez and R.A. Tapia, "On the global convergence of a modified augmented Lagrangian linesearch interior point Newton method for nonlinear programming," Technical Report TR95-29, Dept. of Computational and Applied Mathematics, Rice University, 1995.
2. M. Breitfeld and D. Shanno, "Computational experience with modified log-barrier methods for nonlinear programming," Report RRR17-93, RUTCOR, Rutgers University, 1993.
3. A. Brooke, D. Kendrick, and A. Meeraus, GAMS-A User's Guide, Boyd and Fraser Publishing Co.: Danvers, MA, 1988.
4. R. Byrd, J.C. Gilbert, and J. Nocedal, "A trust region method based on interior point techniques for nonlinear programming," Technical Report OTC 96/02, Optimization Technology Center, Argonne National Laboratory and Northwestern University, 1996.
5. R.S. Dembo, "A set of geometric programming test problems and their solutions," Mathematical Programming, vol. 10, pp. 192–213, 1976.
6. A. Drud, "CONOPT-A large-scale GRG code," ORSA Journal on Computing, vol. 6, no. 2, pp. 207–216, 1994.

7. A. Drud, "A general pre-processor for GAMS models," Talk given at INFORMS San Diego Meeting, 5/4–5/7, 1997, Session MC33.1.
8. A. Drud, "A general pre-processor for GAMS Models," Talk Presented at INFORMS San Diego Meeting, 5/4–5/7, 1997, Session MC33.1.
9. I.S. Duff and J.K. Reid, "A fortran code for direct solution of indefinite sparse symmetric linear systems," Rutherford Appleton Laboratory Report RAL-95-001, Oxfordshire, UK, Jan. 1995.
10. A.S. El-Bakry, R.A. Tapia, T. Tsuchiya, and Y. Zhang, "On the formulation and theory of the Newton interior-point method for nonlinear programming," *Journal of Optimization Theory and Applications*, vol. 89, no. 3, pp. 507–541, 1996.
11. A.V. Fiacco and G.P. McCormick, *Nonlinear Programming: Sequential Unconstrained Minimization Techniques*, John Wiley and Sons: New York, NY, 1968.
12. R. Fourer, D. Gay, and B. Kernighan, *AMPL, A Modeling Language for Mathematical Programming*, Boyd and Fraser Publishing Co.: Danvers, MA, 1993.
13. R.S. Gajulapalli, "INTOPT: An interior point algorithm for large scale nonlinear optimization," Doctoral Dissertation, University of Texas at Austin, 1995.
14. D. Gay, "More AD of nonlinear AMPL models: Computing Hessian information and exploiting partial separability," in *Computational Differentiation*, M. Berz, C. Bischof, G. Corliss, and A. Griewank (Eds.), SIAM, 1996.
15. S. Granville, "Optimal reactive dispatch through interior point methods," Internal Report, CEPTEL, Av. Hum Q. 5-Cidade Universitária, Ilha do Fundão, CEP 21941, Rio de Janeiro, R.J., Brazil, 1991.
16. S. P. Han, "Superlinearly convergent variable metric algorithms for general nonlinear programming problems," *Mathematical Programming*, vol. 11, pp. 263–282, 1976.
17. D.M. Himmelblau, *Applied Nonlinear Programming*, McGraw-Hill: New York, 1972.
18. L.S. Lasdon, J. Plummer, and G. Yu, "Primal-dual and primal interior point algorithms for general nonlinear programs," *ORSA Journal on Computing*, vol. 7, no. 3, pp. 321–332, 1995.
19. D. Luenberger, *Linear and Nonlinear Programming*, Addison-Wesley Publishing Co.: Reading Mass, 1984.
20. B.A. Murtagh and M.A. Saunders, "A projected Lagrangian algorithm and its implementation for sparse nonlinear constraints," *Mathematical Programming Study*, vol. 16, pp. 84–117, 1982.
21. R. Polyak, "Modified barrier functions," *Mathematical Programming*, vol. 54, pp. 177–222, 1992.
22. M.J.D. Powell, "A fast algorithm for nonlinearly constrained optimization calculations," in *Numerical Analysis*, Dundee 1977, G.A. Watson (Ed.), Springer-Verlag: Berlin, 1977.
23. Y. Wu, A. Debs, and R. Marsten, "A direct nonlinear predictor-corrector primal-dual interior point algorithm for optimal power flows," *IEEE Transactions on Power Systems*, vol. 9, no. 2, pp. 876–883, 1994.
24. Y. Zhang and R.A. Tapia, "Superlinear and quadratic convergence of primal-dual interior-point methods for linear programming revisited," Report TR91-27, Department of Mathematical Sciences, Rice University, 1991.