Mathematical Programming 45 (1989) 547-566
North-Holland

A TOLERANT ALGORITHM FOR LINEARLY CONSTRAINED OPTIMIZATION CALCULATIONS

M.J.D. POWELL

Department of Applied Mathematics and Theoretical Physics, University of Cambridge, Silver Street, Cambridge CB3 9EW, England

Two extreme techniques when choosing a search direction in a linearly constrained optimization calculation are to take account of all the constraints or to use an active set method that satisfies selected constraints as equations, the remaining constraints being ignored. We prefer an intermediate method that treats all inequality constraints with "small" residuals as inequalities with zero right hand sides and that disregards the other inequality conditions. Thus the step along the search direction is not restricted by any constraints with small residuals, which can help efficiency greatly, particularly when some constraints are nearly degenerate. We study the implementation, convergence properties and performance of an algorithm that employs this idea. The implementation considerations include the choice and automatic adjustment of the tolerance that defines the "small" residuals, the calculation of the search directions, and the updating of second derivative approximations. The main convergence theorem imposes no conditions on the constraints except for boundedness of the feasible region. The numerical results indicate that a Fortran implementation of our algorithm is much more reliable than the software that was tested by Hock and Schittkowski (1981). Therefore the algorithm seems to be very suitable for general use, and it is particularly appropriate for semi-infinite programming calculations that have many linear constraints that come from discretizations of continua.

Key words: Convergence theory, degeneracies, linear constraints, matrix factorizations, nonlinear optimization, semi-infinite programming.

1. An outline of the algorithm

This paper considers an algorithm for minimizing a general differentiable objective function $\{F(x) | x \in \Omega \subset \mathbb{R}^n\}$ subject to the linear constraints

$$\begin{aligned} &a_j^T x = b_j, \quad j = 1, 2, \ldots, m', \\ &a_j^T x \leq b_j, \quad j = m'+1, \ldots, m, \\ &l_i \leq [x]_i \leq u_i, \quad i = 1, 2, \ldots, n, \end{aligned} \tag{1.1}$$

where Ω is the set of points that satisfy the constraints, and where the constraint gradients $\{a_j\}$, the right hand sides $\{b_j\}$ and the bounds $\{l_i\}$ and $\{u_i\}$ are given, $[x]_i$ being the ith component of x. It is assumed that $F(x)$ and its first derivative vector $\nabla F(x)$ can be calculated for any $x \in \Omega$. Some Fortran software that implements and that helped to create the given procedure was written by the author for IMSL, so particular attention was given to reliability and robustness.

The following preferences of the author are included in the algorithm: (1) the preservation of feasibility, apart from the effects of computer rounding errors, after a vector of variables has been found that satisfies all the constraints, (2) the use of line searches to force convergence, (3) the avoidance of very small changes to the variables due to near or actual constraint degeneracies when the variables are far from their optimal values, (4) the employment of the matrix factorizations of Goldfarb and Idnani (1983), and (5) the approximation of the second derivatives of $F(\cdot)$ by positive definite matrices that are updated by the BFGS formula. This section presents a brief discussion of these features that provides an outline of the method.

Several optimization algorithms preserve feasibility when all the constraints are linear (see, for instance, Fletcher, 1988; Gill, Murray and Wright, 1981), but the user may provide an initial vector of variables, $x_1 \in \mathbb{R}^n$ say, that violates some constraints. In this case our algorithm applies a standard technique, described in Section 2, that either replaces x_1 by a feasible vector or recognises that the constraints are inconsistent. Feasibility is important in software for general use, partly because the purpose of some constraints may be to keep x away from regions of $\mathbb{R}^n$ where $F(x)$ is not defined.

Line search procedures are iterative, and each change to the variables made by an iteration has the form

$$x_{k+1} = x_k + \alpha_k d_k, \tag{1.2}$$

where k is the iteration number, α_k is a positive step-length and d_k is a search direction in $\mathbb{R}^n$. Our algorithm sets d_k to the value of d that minimizes the quadratic approximation

$$Q_k(d) = F(x_k) + d^{\mathrm{T}} \nabla F(x_k) + \tfrac{1}{2} d^{\mathrm{T}} B_k d, \quad d \in \mathbb{R}^n, \tag{1.3}$$

to $F(x_k + d)$ subject to some linear constraints on d that are derived from expression (1.1), B_k being a positive definite matrix that should be viewed as an approximation to the second derivative matrix $\nabla^2 F(x_k)$ when $F(\cdot)$ is twice differentiable. Thus, in the usual case when $d_k \neq 0$, the vector of variables $x = x_k + \alpha d_k$ is feasible for all sufficiently small positive α, and we have the downhill condition

$$d_k^{\mathrm{T}} \nabla F(x_k) < 0. \tag{1.4}$$

Having generated $d_k \neq 0$, our algorithm picks a step-length α_k that makes x_{k+1} feasible and that gives a reduction $F(x_{k+1}) < F(x_k)$ in the objective function. Careful use of the flexibility in the choice of α_k is important to achieving convergence in many line search algorithms for unconstrained optimization (see, for instance, Wolfe, 1969), but now the feasibility of x_{k+1} imposes an upper bound on α_k. In particular, if the step from x_k to $x_k + d_k$ includes a move from the interior to the boundary of an inequality constraint, then we have the condition $\alpha_k \leq 1$. Because this restriction can cause severe inefficiencies, an important feature of our technique is that it never deliberately puts $x_k + d_k$ on the boundary of a constraint that is satisfied as a strict inequality at x_k.

In order to illustrate one purpose of this remark, we consider the minimization of the second component of x in $\mathbb{R}^2$ when the boundary of the feasible region is a regular polygon with 10^6 vertices on the unit circle $\{x \mid \|x\|_2 = 1\}$. If this calculation were solved by the simplex method for linear programming, then each iteration would move the current vector of variables from a vertex of this polygon to the adjacent vertex that gives a smaller value of the objective function, so very small steps would occur even when x_k is far from the solution. This inefficiency is exactly the one that is mentioned in the previous paragraph if d_k is a step from one vertex to the next along an edge of the polygon.

Therefore in our algorithm the linear constraints on the search direction include the condition that the step from x_k to $x_k + d_k$ must not be a move *towards* the boundary of any inequality constraint that has a small residual at x_k. The meaning of "small" in practice is made precise in Section 2: it depends on a relative tolerance parameter that is reduced automatically from a moderate value to one that is near the precision of the computer arithmetic. We let J_k be the set of indices of inequality constraints with small residuals at x_k, and, in order to allow bounds in this set, we write the last line of expression (1.1) in the form

$$\begin{aligned} &a_j^T x \leqslant -l_{j-m}, \quad j = m+1, \ldots, m+n, \\ &a_j^T x \leqslant u_{j-m-n}, \quad j = m+n+1, \ldots, m+2n, \end{aligned} \tag{1.5}$$

where $a_j = -e_{j-m}$ and $a_j = e_{j-m-n}$ for $m+1 \leqslant j \leqslant m+n$ and $m+n+1 \leqslant j \leqslant m+2n$ respectively, e_i being the ith coordinate vector of $\mathbb{R}^n$. Then the constraints on the search direction of the kth iteration are the conditions

$$\begin{aligned} &a_j^T d = 0, \quad j = 1, 2, \ldots, m', \\ &a_j^T d \leqslant 0, \quad j \in J_k, \end{aligned} \tag{1.6}$$

all the right hand sides being zero. The idea of employing such constraints is suggested by Polak (1971), but he does not report any numerical experiments on the usefulness of the idea in practice. Our calculation includes a shift of the variables onto the active constraint boundaries when the tolerance parameter reaches its least value, but there is no other refinement of the active constraint residuals.

The most important consequence of the conditions (1.6) is that, if any of the constraints (1.1) restricts the step-length α_k, then its index is not in J_k. Therefore the requirement that x_{k+1} be feasible allows relatively large changes to the variables, even if $\|d_k\|$ is very small due to an unsuitable matrix B_k in the quadratic approximation (1.3). It can happen that the number of indices in J_k exceeds the number of variables n, which distinguishes our algorithm from many other procedures for linearly constrained optimization.

We see that the calculation of the search direction is a strictly convex quadratic programming problem, namely the minimization of expression (1.3) subject to the conditions (1.6). The zero right hand sides of the constraints often cause this problem to be degenerate. We employ the quadratic programming algorithm of Goldfarb

and Idnani (1983) because it not only copes with the degeneracies very well but also can take advantage of the zero right hand sides. Details are given in Section 2 and some particular points are noted now. In difficult cases most of the work of this calculation is spent on identifying an *active set* of constraint indices $I_k \subset (\{1, 2, \ldots, m'\} \cup J_k)$ such that the search direction minimizes expression (1.3) subject to the equations

$$a_i^T d = 0, \quad i \in I_k, \tag{1.7}$$

where the active constraint gradients $\{a_i \mid i \in I_k\}$ are linearly independent. An estimate of I_k is adjusted automatically if necessary. We let A_k denote the matrix whose columns are the gradients of the constraints of the current estimate of I_k, and m_k denotes the number of columns of A_k. The Goldfarb-Idnani procedure stores and updates two matrices, namely an $n \times n$ full matrix Z_k and an $n \times m_k$ upper triangular matrix R_k that satisfy the equations

$$Z_k^T A_k = R_k, \qquad Z_k Z_k^T = B_k^{-1}. \tag{1.8}$$

When A_k is altered, the upper triangularity of $Z_k^T A_k$ is recovered by premultiplying Z_k^T by a suitable $n \times n$ orthogonal matrix, because this operation preserves the identity $Z_k Z_k^T = B_k^{-1}$. One small change from the Goldfarb-Idnani procedure in our algorithm is that we store and update $\hat{U}_k$ instead of R_k, where $\hat{U}_k = \hat{R}_k^{-1}$ and $\hat{R}_k$ is the $m_k \times m_k$ upper triangular matrix that is formed by deleting the zero rows of R_k.

The second derivative matrix B_k of expression (1.3) is not stored because Z_k defines B_k in a convenient way. Therefore the statement that B_k is revised by the BFGS formula means that Z_{k+1} is calculated to satisfy the condition that $(Z_{k+1} Z_{k+1}^T)^{-1}$ is the matrix that would be obtained if the BFGS formula were applied to $(Z_k Z_k^T)^{-1}$. Fortunately there is a stable way of generating Z_{k+1} that also gives the equation $Z_{k+1}^T A_k = R_k$, so no further work is needed to regain the first of the equations (1.8) after revising the second derivative approximation. No other updating formula in the Broyden linear family possesses this property (Powell, 1988).

Our algorithm is suitable for calculations with a moderate number of variables and very many constraints. The restriction on n is due to the full matrix Z_k, but large values of m can be treated efficiently because the search direction takes account of all constraints with small residuals. Therefore the given procedure is recommended for approximations to semi-infinite programming problems with an infinite number of constraints. Such problems occur, for example, in control calculations when, for each vector of variables, a time-dependent function is generated that must satisfy certain bounds. Here it is not unusual for an iteration to change all the elements of an active set, and then much second derivative information can be lost by those algorithms that employ reduced second derivative matrices. Therefore the ease of updating the Goldfarb-Idnani factorization of an approximation to $\nabla^2 F(\cdot)$ is an important feature of our work.

The details that are given in Section 2 include the initialization of the algorithm, the application and adjustment of the tolerance that is used to identify small

constraint residuals, the termination condition, some remarks on the line search procedure, and further attention to the quadratic programming calculation of the search direction. More information can be found in a report that presents a Fortran implementation of the procedure (Powell, 1989). Some convergence questions are studied in Section 3. Unfortunately, unless some convexity assumptions are made, it is not yet known whether the BFGS method for unconstrained optimization gives $\|\nabla F(x_k)\| \to 0$ when the objective function is twice continuously differentiable with bounded level sets, so our theory has to be incomplete. We suppose that the second derivative approximations $\{B_k \mid k = 1, 2, 3, \ldots\}$ and their inverses are uniformly bounded, and we prove convergence to one or more Kuhn-Tucker points in this case, even if the constraints are highly degenerate, without assuming specifically that the BFGS formula is employed to define each B_{k+1}. Finally, Section 4 gives some numerical results that show the efficacy of the algorithm when m is large and that provide comparisons with some other procedures for constrained optimization calculations.

2. Several details of the algorithm

It is usual for descriptions of optimization algorithms to ignore computer rounding errors, and mostly we follow this practice, although careful attention to such details is essential to the development of robust computer software. Therefore some important features of the implementation of Powell (1989) are not considered in this section. There are even some differences between the given algorithm and the Fortran software that are pointed out explicitly. These inconsistencies allow clearer explanations of some salient points, and they yield some interesting convergence questions. In the implementation, however, convergence questions are trivial, because the iterative procedure is stopped automatically when it is no longer possible to decrease the objective function. Thus the termination of the execution of the Fortran program is an immediate consequence of the fact that the number of different values of the objective function in any computer calculation is finite.

The algorithm begins by setting a constant called ZTEST to about 100 times the relative precision of the computer arithmetic. It is used frequently in the implementation to assess whether quantities that are small due to cancellation might be zero. For example, if there are equality constraints, the Fortran program employs ZTEST to decide whether to treat the vectors $\{a_j \mid j = 1, 2, \ldots, m'\}$ as linearly dependent, and if so whether the right hand sides $\{b_j \mid j = 1, 2, \ldots, m'\}$ are consistent. Of course the algorithm terminates if the equalities are inconsistent, and otherwise any redundant constraints are dropped in order that the m' equations of expression (1.6) are independent. Then the starting point x_1 is revised if necessary by an orthogonal projection so that $\{a_j^T x_1 = b_j \mid j = 1, 2, \ldots, m'\}$.

This projection gives some importance to the Euclidean metric initially, and also we let the initial second derivative approximation be the unit matrix. Therefore we

choose Z_1 to be orthogonal, and, if there are any equality constraints, then as in equation (1.8) we require $Z_1^T A_1 = R_1$, where initially the columns of A_1 are $\{a_j | j = 1, 2, \ldots, m'\}$ and where R_1 is upper triangular. Further, we let τ be the tolerance parameter that distinguishes small from large constraint residuals. Its initial value is $\tau = 0.01$, and it is reduced automatically on certain iterations until $\tau = \text{ZTEST}$.

After taking account of any equalities, the algorithm revises x_1 if necessary to meet the bounds $\{l_i \leq [x_1]_i \leq u_i | i = 1, 2, \ldots, n\}$. We view this problem as minimizing the sum of violations of the bounds subject to preserving all the equality constraints and bounds that hold already. These constraints are treated in the way that is mentioned in Section 1, so we employ τ to pick out the ones with small residuals at the current x_1 for inclusion in expression (1.6). Then a search direction is generated as before, after changing $\nabla F(x_k)$ in expression (1.3) to the gradient of the sum of bound violations, B_1 being the unit matrix. If the resultant d is zero and $\tau > \text{ZTEST}$, then τ is reduced and d is recalculated, but otherwise a zero d causes an error return because the equality constraints and bounds seem to be inconsistent. When d is nonzero, x_1 is replaced by $x_1 + \alpha d$, where the step-length α minimizes the new sum of bound violations, which is a piecewise linear function, subject to the satisfied equality and bound constraints. This procedure is repeated until either there is an error return or x_1 is within all the bounds. In the latter case we turn to the remaining constraints $\{a_j^T x_1 \leq b_j | j = m' + 1, \ldots, m\}$, adjusting x_1 if necessary in the way that is analogous to the given procedure for the bounds. Therefore from now on we assume that every x_k is feasible.

The test for small residuals depends not only on τ but also on nonnegative numbers $\{X_i | i = 1, 2, \ldots, n\}$ that usually reflect the magnitudes of the components of x. Initially the algorithm sets the values

$$X_i = |[x_1]_i|, \quad i = 1, 2, \ldots, n, \tag{2.1}$$

where x_1 is the starting vector that is provided by the user. Then, after each acceptable change to the variables, including changes to x_1 to achieve feasibility, X_i is replaced by $\max[X_i, |[x]_i|]$. It is deliberate that X_i might be zero initially, because we dislike the alternative of choosing an arbitrary positive number that would introduce an unnecessary dependence on scaling into the algorithm. The constraint $a_j^T x \leq b_j$ is deemed to have a small residual at x if and only if the inequality

$$|a_j^T x - b_j| \leq \tau \left\{ \sum_{i=1}^{n} X_i |[a_j]_i| + |b_j| \right\} \tag{2.2}$$

is satisfied, and a similar test is applied to bounds. For example, the bound $l_i \leq [x]_i$ is defined to have a small residual when $|[x]_i - l_i| \leq \tau\{X_i + |l_i|\}$.

After calculating each search direction d_k, the algorithm decides either to perform a line search or to reduce τ. This decision depends on some Lagrange multipliers and on the change $Q_k(0) - Q_k(d_k)$ in the quadratic objective function (1.3). Each adjustment to τ is by a factor of at least 10 subject to the lower bound $\tau \geq \text{ZTEST}$, so the number of reductions is finite. They can occur in the main iterations and during the procedure that has been mentioned already for achieving feasibility.

The required Lagrange multipliers are the ones at the solution of the quadratic programming problem that defines d_k. Here the current Z_k and R_k satisfy equation (1.8), the columns of A_k being $\{a_j | j \in I_k\}$, where as before the active set I_k is a subset of the constraint indices that occur in the conditions (1.6). The nontrivial multipliers are the components of the vector $\lambda_k \in \mathbb{R}^{m_k}$ that is defined by the equations

$$\nabla F(x_k) + B_k d_k = A_k \lambda_k, \qquad A_k^{\mathrm{T}} d_k = 0. \tag{2.3}$$

Writing $d_k = Z_k \mu_k$, which is used in the calculation of d_k, it follows from the first part of expression (1.8) and $A_k^{\mathrm{T}}(Z_k \mu_k) = 0$ that the first m_k components of μ_k are zero. Further, multiplying the first part of equation (2.3) by Z_k^{T} and using $Z_k^{\mathrm{T}} B_k Z_k = I$, we find the identity

$$Z_k^{\mathrm{T}} \nabla F(x_k) + \mu_k = R_k \lambda_k. \tag{2.4}$$

Hence the last $(n - m_k)$ components of μ_k are those of $-Z_k^{\mathrm{T}} \nabla F(x_k)$ and λ_k satisfies the $m_k \times m_k$ triangular system

$$\hat{R}_k \lambda_k = \hat{Z}_k^{\mathrm{T}} \nabla F(x_k), \tag{2.5}$$

where $\hat{R}_k$ and $\hat{Z}_k$ are formed by deleting the last $n - m_k$ rows from R_k and the last $n - m_k$ columns from Z_k respectively. Therefore, remembering that we store the upper triangular matrix $\hat{U}_k = \hat{R}_k^{-1}$ instead of R_k, the algorithm employs the multipliers

$$\lambda_k = \hat{U}_k \hat{Z}_k^{\mathrm{T}} \nabla F(x_k). \tag{2.6}$$

It is important to the next paragraph that, due to the Kuhn–Tucker conditions at the solution of the quadratic programming problem, the components of this vector that correspond to inequality constraints are nonpositive. The change in $Q_k(\cdot)$ is also easy to calculate, because, substituting $d_k = Z_k \mu_k$ in equation (1.3), we find that the form of μ_k implies the value

$$Q_k(0) - Q_k(d_k) = \tfrac{1}{2} \| \mu_k \|_2^2. \tag{2.7}$$

The main criterion for reducing τ depends on the remark that usually the least value of $Q_k(d)$ subject to the condition that $x_k + d$ shall be feasible is less than $Q_k(d_k)$, because the constraints on this new d are less restrictive than the constraints on d_k. Specifically, if $j \in J_k$ is the index of a constraint whose residual $a_j^{\mathrm{T}} x_k - b_j$ is negative, then the right hand side of the constraint $a_j^{\mathrm{T}} d \leq 0$ in expression (1.6) can be increased to the modulus of this residual. Further, letting r_k be the vector of residuals at x_k of the constraints whose indices are in I_k, a straightforward calculation shows that, if the inequality constraints $A_k^{\mathrm{T}} d \leq 0$ are altered to $A_k^{\mathrm{T}} d \leq -r_k$, then, instead of expression (2.7), we can achieve the difference

$$Q_k(0) - Q_k(d_k) = \tfrac{1}{2} \| \mu_k \|_2^2 + \lambda_k^{\mathrm{T}} r_k + O(\| r_k \|^2). \tag{2.8}$$

Because it seems unsuitable to continue with the nonzero residuals due to τ when the term $\lambda_k^{\mathrm{T}} r_k$ is much larger than $\frac{1}{2} \| \mu_k \|_2^2$, the conditions for revising τ are as follows.

The algorithm reduces τ if the current value was used on the previous iteration, if τ is not already at its lower bound, and if the inequality

$$\lambda_k^T r_k \geq 10 \max[\tfrac{1}{2}\|\mu_k\|_2^2, F(x_{k-1}) - F(x_k)] \tag{2.9}$$

is satisfied. The factor 10 is a consequence of numerical experiments, and the term $F(x_{k-1}) - F(x_k)$ is present to prevent too early a decrease in τ when the iterations are choosing long step-lengths.

There is another situation that can cause a reduction in τ which depends on an accuracy parameter, namely ACC, that is set by the user. The main purpose of this parameter is that the entire calculation finishes if the condition

$$\|\nabla F(x_k) - A_k \lambda_k\|_2 \leq \text{ACC} \tag{2.10}$$

is satisfied and $\tau = \text{ZTEST}$. This termination criterion is appropriate because we have $\nabla F(x_k) = A_k \lambda_k$ at every Kuhn-Tucker point. The only suitable action by the algorithm when condition (2.10) holds but τ is not minimal is to reduce the value of τ. This action covers the possibility $d_k = 0$ because then equation (2.3) implies that the left hand side of the test (2.10) is zero. We assume in the algorithm that inequality (2.10) can be satisfied, but in practice the user may set ACC to such a tiny number that rounding errors prevent the attainment of the termination condition. Therefore the implementation includes a test that gives an error return when the requested accuracy cannot be achieved.

The new value of τ is calculated as follows. In order to allow a large reduction when there are no near-degeneracies in the constraints, the algorithm determines the least value of τ, $\hat{\tau}$ say, such that condition (2.2) remains satisfied by all the constraints in the current active set I_k. Thus the reason for reducing τ would still be valid if τ were replaced by $\hat{\tau}$. Then τ is decreased to the number

$$\tau = \begin{cases} 0.1\hat{\tau}, & \hat{\tau} > 20\,\text{ZTEST}, \\ \text{ZTEST}, & \hat{\tau} \leq 20\,\text{ZTEST}, \end{cases} \tag{2.11}$$

so the reduction is by at least the factor 10 that has been mentioned already, except that, when achieving the lower bound $\tau = \text{ZTEST}$, the factor can be close to 2. Further, when τ is revised for the last time, we reinitialize the numbers $\{X_i \mid i = 1, 2, \ldots, n\}$ of inequality (2.2) to the moduli of the components of the current x_k. The reason for this precaution is that during the iterations the magnitudes of some of the variables may have decreased greatly.

Usually the new τ and the current x_k do not allow I_k to remain as the active set, because some of the active constraint residuals are no longer "small". Therefore the inequality constraints whose residuals become large are removed from I_k, except when τ reaches its lower bound, namely ZTEST. In this case it follows from equation (2.11) that, for the old values of $\{X_i \mid i = 1, 2, \ldots, n\}$, the relative residuals of these constraints are at most 20 ZTEST, and we apply the refinement procedure that is mentioned soon after equation (1.6) instead of reducing the active set. Specifically,

x_k is moved onto the active constraint boundaries by the formula

$$x_k \leftarrow x_k - \hat{Z}_k \hat{U}_k^{\mathrm{T}} r_k, \tag{2.12}$$

where r_k is the vector of active constraint residuals as before. When $\tau_{\mathrm{new}} > \mathrm{ZTEST}$, however, then x_k is not altered unless an accumulation of rounding errors has caused one or more active constraints to be violated by more than the relative accuracy ZTEST. An earlier version of the algorithm used formula (2.12) whenever τ was reduced, subject to some safeguards to preserve feasibility, but this technique was abandoned because in difficult calculations the larger changes to x_k sometimes caused substantial increases in the objective function, due to contributions from second and higher order terms of the Taylor series expansion of $F(x)$ about x_k.

On each iteration of the algorithm, the choice of step-length is based on the Wolfe (1969) conditions

$$F(x_k + \alpha_k d_k) \leqslant F(x_k) + 0.1\alpha_k d_k^{\mathrm{T}} \nabla F(x_k) \tag{2.13}$$

and

$$d_k^{\mathrm{T}} \nabla F(x_k + \alpha_k d_k) \geqslant 0.7 d_k^{\mathrm{T}} \nabla F(x_k), \tag{2.14}$$

the factors 0.1 and 0.7 being successful values of free parameters. These inequalities are consistent for a range of positive values of α_k, whenever $F(\cdot)$ is a continuously differentiable function that is bounded below and d_k satisfies the descent condition (1.4). When there are no constraints, they ensure that α_k is both small enough and large enough to provide good global convergence properties for suitable search directions, but, in the linearly constrained case, the feasibility of $x_k + \alpha_k d_k$ may rule out the attainment of inequality (2.14). Therefore the algorithm requires α_k to give the reduction (2.13) in the objective function and to satisfy either $\alpha_k = \bar{\alpha}_k$, or condition (2.14), where $\bar{\alpha}_k$ is the longest step-length that is allowed by feasibility. Another important feature, which helps the final rate of convergence, is that the initial trial step of each line search has the value $\min[1, \bar{\alpha}_k]$. In the Fortran implementation, however, there are some safeguards in case rounding errors prevent the attainment of the line search conditions of the algorithm.

A disadvantage of the $\alpha_k = \bar{\alpha}_k$ case is that it may not be possible or sensible to update the second derivative approximation by the BFGS formula, because we cannot satisfy the equation $Z_{k+1} Z_{k+1}^{\mathrm{T}} = B_{k+1}^{-1}$ unless B_{k+1} is positive definite, which demands the condition

$$d_k^{\mathrm{T}} \{\nabla F(x_k + \alpha_k d_k) - \nabla F(x_k)\} > 0. \tag{2.15}$$

Further, a very small value of this scalar product can cause severe ill-conditioning. Therefore each iteration includes BFGS updating if and only if the left hand side of inequality (2.15) is at least $0.1|d_k^{\mathrm{T}} \nabla F(x_k)|$. Otherwise the second derivative approximation is not revised. Moreover, the updating procedure includes the column scaling technique of Powell (1987), because it can give a substantial reduction in the number of iterations when $\|B_1\| = \|(Z_1 Z_1^{\mathrm{T}})^{-1}\|$ is much larger than the norm of a typical second derivative matrix.

We end this section by considering the calculation of the search direction, which is the vector that minimizes expression (1.3) subject to the homogeneous linear constraints (1.6). If the current guess of the active set I_k is correct, then d_k is defined by the linear system (2.3). Further, it follows from the Kuhn-Tucker conditions of this quadratic programming problem that, if for any guess of I_k we define d and λ by the equations

$$\nabla F(x_k) + B_k d = A_k \lambda, \qquad A_k^{\mathrm{T}} d = 0, \tag{2.16}$$

then we have the correct active set if and only if d satisfies the constraints (1.6) and the components of λ that correspond to inequality constraints are all nonpositive. For each estimate of I_k the matrices Z_k and $\hat{U}_k = \hat{R}_k^{-1}$ of the factorization (1.8) are available, and they are updated by an orthogonal transformation whenever the estimate is revised. Therefore, as in the paragraph that includes expressions (2.3)-(2.7), the system (2.16) has the solution

$$d = -\check{Z}_k \check{Z}_k^{\mathrm{T}} \nabla F(x_k) \tag{2.17}$$

and

$$\lambda = \hat{U}_k \hat{Z}_k^{\mathrm{T}} \nabla F(x_k), \tag{2.18}$$

where $\check{Z}_k$ is the $n \times (n - m_k)$ matrix whose columns are the last $n - m_k$ columns of Z_k.

We determine I_k by the dual algorithm of Goldfarb and Idnani (1983), which gives higher priority to the conditions on λ than to the conditions on d. The initial estimate of I_k is the active set of the previous iteration in order that we can make use of the current Z_k and $\hat{U}_k$. We begin by calculating the vector (2.18), and, if we find any positive multipliers of inequality constraints, then a recursive procedure makes deletions from the current active set until all the components of λ have acceptable signs. The stage when λ is acceptable may be reached several times during the sequence of estimates of I_k, but there is no cycling because the corresponding values of $Q_k(d)$ increase strictly monotonically.

At this stage we calculate the vector (2.17), defining it to be zero if $m_k = n$. If the conditions (1.6) are satisfied then $d = d_k$ as required. Otherwise we pick the most violated constraint, $a_l^{\mathrm{T}} d \leq 0$ say, and we add l to the active set. In theory a_l is not in the column space of A_k, because if it were then $A_k^{\mathrm{T}} d = 0$ would imply $a_l^{\mathrm{T}} d = 0$. Therefore this addition preserves the independence of the active constraint normals. The Fortran implementation (Powell, 1989) is less simple, however, because extensive use is made of ZTEST to decide, for example, whether constraint violations are negligible.

Having added l to the active set, and having updated m_k, A_k, Z_k and $\hat{U}_k$, which requires no change to the first $m_k - 1$ columns of these matrices, we calculate the new multiplier vector (2.18). We see that it has the value

$$\lambda = \tilde{\lambda} + \phi \hat{U}_k(\cdot, m_k), \tag{2.19}$$

where $\tilde{\lambda}$ is the old multiplier vector augmented by a zero component and where ϕ is the last component of $\hat{Z}_k^{\mathrm{T}} \nabla F(x_k)$. If the new multipliers have acceptable signs,

there is a branch to the part of the algorithm that is described in the previous paragraph, but otherwise we proceed as follows.

We know that the vector (2.19) has acceptable signs if $\phi = 0$ but not if ϕ has the required value. We take the view that ϕ is adjusted continuously from zero, and we let $\hat{\phi}$ be the first value at which a component of λ, $[\lambda]_q$ say, becomes unacceptable. The qth element of the active set is dropped, and m_k, A_k, Z_k and $\hat{U}_k$ are updated so that l remains the last element of the active set. Again we express λ in the form (2.19), $\tilde{\lambda}$ being the contribution to expression (2.18) from the first $m_k - 1$ columns of the new $\hat{U}_k$, and ϕ being the new value of the last component of $\hat{Z}_k^{\mathrm{T}} \nabla F(x_k)$. This new ϕ has the same sign as before and its modulus exceeds $|\hat{\phi}|$. If λ is now acceptable the algorithm branches to the testing of the conditions (1.6) that has been described already. Otherwise we apply the procedure of this paragraph recursively, dropping one constraint at a time, until λ becomes acceptable, except that we must qualify the statement that the vector (2.19) has acceptable signs if $\phi = 0$. The situation now is that some unacceptable signs may occur when $\phi = 0$, but, as ϕ is adjusted continuously to the required value through the old $\hat{\phi}$, all signs become acceptable no later than the old $\hat{\phi}$ and no multipliers of inequality constraints switch from negative to positive until ϕ reaches the old $\hat{\phi}$. Therefore the qualification does not disturb the obvious method for finding q.

The implementation of this procedure for calculating the search direction is straightforward. When the qth constraint is dropped from the active set, the factorization (1.8) is updated by applying Givens rotations that exchange adjacent columns of the old A_k until the gradient of the constraint to be dropped is the last column of A_k. Then the actual deletion of the constraint from the estimate of I_k only requires m_k to be replaced by $m_k - 1$. We prefer to work with the upper triangular matrix $\hat{U}_k$ instead of R_k, because it is helpful to our use of the form (2.19) to have the last column of $\hat{U}_k$ available explicitly.

3. Convergence questions

In this section we study some convergence properties of the algorithm when $\nabla F(\cdot)$ exists and satisfies the Lipschitz condition

$$\|\nabla F(x) - \nabla F(y)\|_2 \leq L \|x - y\|_2 \tag{3.1}$$

for all feasible x and y, where L is a constant. Further, as mentioned in the introduction, we assume that the positive definite matrices $\{B_k \mid k = 1, 2, 3, \ldots\}$ and their inverses are uniformly bounded. We write these conditions in the form

$$w\|d\|_2^2 \leq d^{\mathrm{T}} B_k d \leq W\|d\|_2^2, \quad d \in \mathbb{R}^n, \quad k = 1, 2, 3, \ldots, \tag{3.2}$$

where w and W are positive constants. Because no attention is given to the BFGS formula, our theory is valid for all matrices $\{B_k \mid k = 1, 2, 3, \ldots\}$ that satisfy the bounds (3.2). Therefore no superlinear convergence questions are considered. The

final assumption of this section is that computer rounding errors are negligible, except that they provide the positive lower bound ZTEST on τ.

The most important convergence question is answered in the following theorem.

Theorem. *The given conditions on* $F(\cdot)$ *imply that the algorithm terminates if the value of* ACC *in inequality* (2.10) *is set to any positive constant.*

Proof. Because the number of different values of τ is finite, and because the attainment of condition (2.10) causes either a reduction in τ or termination, it is sufficient to prove that this condition holds eventually if enough iterations are performed with any fixed positive value of τ. When the step-length of the line search satisfies inequality (2.14), then we can deduce a useful lower bound on α_k from expressions (2.3), (2.14), (3.1) and (3.2). Specifically, because equation (2.3) implies the identity

$$d_k^{\mathrm{T}}\nabla F(x_k) = -d_k^{\mathrm{T}}B_k d_k, \tag{3.3}$$

expressions (2.14) and (3.1) give the relation

$$\begin{aligned} 0.3 d_k^{\mathrm{T}}B_k d_k &= -0.3 d_k^{\mathrm{T}}\nabla F(x_k) \\ &\leqslant d_k^{\mathrm{T}}[\nabla F(x_k+\alpha_k d_k)-\nabla F(x_k)] \\ &\leqslant \alpha_k L\|d_k\|_2^2, \end{aligned} \tag{3.4}$$

so it follows from condition (3.2) that we have the bound $\alpha_k \geqslant 0.3w/L$.

Alternatively, it has been noted that the step-length may satisfy $\alpha_k = \bar{\alpha}_k$ instead of inequality (2.14). In this case, letting $l \notin J_k$ be the index of the constraint that restricts the step-length, the algorithm gives the conditions

$$a_l^{\mathrm{T}}(x_k+\alpha_k d_k) = b_l \tag{3.5}$$

and

$$|a_l^{\mathrm{T}}x_k - b_l| > \tau\left\{\sum_{i=1}^{n} X_i|[a_l]_i| + |b_l|\right\}, \tag{3.6}$$

where the right hand side b_l is taken from expression (1.5) if $m+1 \leqslant l \leqslant m+2n$. We note that, for each l, the term in braces on the right hand side of inequality (3.6) is a nondecreasing function of the iteration number, so its least positive value over all iterations, v_l say, is well defined. We let v be the least of the numbers $v_l/\|a_l\|_2$ as l ranges over all constraint indices that restrict the step-length of any iteration, which is a positive constant because the number of constraints is finite. Hence, in view of the fact that positivity of the left hand side of condition (3.6) implies positivity of the right hand side, this condition implies the relation

$$|a_l^{\mathrm{T}}x_k - b_l| > \tau v\|a_l\|_2. \tag{3.7}$$

Moreover, the identity

$$|a_l^{\mathrm{T}}x_k - b_l| = \alpha_k|a_l^{\mathrm{T}}d_k| \tag{3.8}$$

is an elementary consequence of equation (3.5), so the Cauchy-Schwarz inequality gives $\alpha_k \|d_k\|_2 > \tau v$. Therefore the step-length of every iteration of the algorithm has the lower bound

$$\alpha_k \geqslant \min[0.3w/L, \tau v/\|d_k\|_2]. \tag{3.9}$$

We combine this result with the remark that expressions (2.13), (3.3) and (3.2) provide the relation

$$F(x_{k+1}) \leqslant F(x_k) - 0.1\alpha_k d_k^{\mathrm{T}} B_k d_k \leqslant F(x_k) - 0.1w\alpha_k \|d_k\|_2^2. \tag{3.10}$$

Of course we also make use of the observation that the differences $\{F(x_k) - F(x_{k+1}) \mid k = 1, 2, 3, \ldots\}$ tend to zero because $\{F(x_k) \mid k = 1, 2, 3, \ldots\}$ is a monotonically decreasing sequence that is bounded below, due to the continuity of $F(\cdot)$ and to the boundedness of the feasible region. Thus, substituting inequality (3.9) in expression (3.10), we deduce the limit

$$\|d_k\|_2 \to 0. \tag{3.11}$$

We have not yet used the second part of condition (3.2), and we note now that, due to the symmetry of B_k, it implies the relation $\|B_k d_k\|_2 \leqslant W\|d_k\|_2$. Therefore the first part of equation (2.3) gives the bound

$$\|\nabla F(x_k) - A_k\lambda_k\|_2 \leqslant W\|d_k\|_2. \tag{3.12}$$

It follows from the limit (3.11) that the left hand side of the test (2.10) tends to zero as $k \to \infty$. Therefore the algorithm terminates as required. □

This proof of convergence does not depend on inequality (2.9), so in theory this condition could be dropped from the algorithm, but we will explain that it is useful. Specifically, it allows reductions in τ before the test (2.10) is satisfied, which helps to avoid unnecessary calculation when ACC is tiny and the tolerances on constraint residuals are relatively large. Inequality (2.9) would be unsuitable, however, if it could hold when τ is small at a point x_k that fails to come close to satisfying the Kuhn-Tucker conditions. The following argument addresses this question.

Let inequality (2.9) be satisfied with $\tau = \text{ZTEST}$. Then condition (2.2) shows that the residuals of the active constraints are near to the least value that is usually attainable from the computer arithmetic. Further, the left hand side of inequality (2.9) is at most a constant multiple of ZTEST, this bound being inherited from r_k due to the uniform boundedness of the Lagrange multipliers $\{\lambda_k \mid k = 1, 2, 3, \ldots\}$, which is a consequence of the independence of active constraint normals, the finiteness of the number of different active sets, equation (2.3) and the uniform boundedness of $\{\nabla F(x_k) + B_k d_k \mid k = 1, 2, 3, \ldots\}$. Therefore condition (2.9) implies that $\|\mu_k\|_2^2$ is bounded above by a multiple of ZTEST. It follows from the identity $d_k = Z_k\mu_k$, the boundedness of the matrices $\{Z_k \mid k = 1, 2, 3, \ldots\}$ and inequality (3.12) that the length of the Kuhn-Tucker residual vector $\nabla F(x_k) - A_k\lambda_k$ is at most a constant multiple of the square root of ZTEST. This is the best kind of accuracy

that one can reasonably expect from an algorithm that does not allow the *computed* value of the objective function to increase in acceptance tests for changes to the variables, because of the flatness of the objective function along directions from the optimum that preserve the zero residuals of the active constraints.

The final consideration of this section is the performance of the algorithm when the user sets ACC = 0 and the computer arithmetic is so precise that it is worthwhile to study the case when there is no positive lower bound on τ. If $\tau \to 0$ occurs, then inequality (2.9) holds an infinite number of times, and one can deduce from the argument of the previous paragraph that a subsequence of $\{x_k \mid k = 1, 2, 3, \ldots\}$ converges to a Kuhn-Tucker point, by restricting attention to the iterations that satisfy condition (2.9). It would be more usual, however, for τ to remain fixed after a finite number of iterations, because $\|r_k\|_2$ is zero for all sufficiently large k. Therefore we address the convergence of the algorithm when ACC = 0 and τ does not change. We have shown already that conditions (3.11) and (3.12) are satisfied, and now we extend our work to prove that every limit point of the sequence $\{x_k \mid k = 1, 2, 3, \ldots\}$ is a Kuhn-Tucker point. This analysis does not presume the limit $\|r_k\|_2 \to 0$.

Let x_* be any limit point of $\{x_k \mid k = 1, 2, 3, \ldots\}$, so we have the relation

$$\lim_{j \to \infty} x_{k(j)} = x_*, \tag{3.13}$$

where $\{k(j) \mid j = 1, 2, 3, \ldots\}$ is a strictly increasing sequence of positive integers. Because the number of different active sets is finite and because we can choose a subsequence of $\{k(j)\}$ if necessary, we assume without loss of generality that the sets $\{I_{k(j)} \mid j = 1, 2, 3, \ldots\}$ are all the same active set, I_* say. We have noted already that the gradients of active constraints are linearly independent, so it follows from expressions (2.3), (3.11) and (3.13) and from the boundedness of B_k that the Lagrange multiplier vectors $\{\lambda_{k(j)} \mid j = 1, 2, 3, \ldots\}$ converge to the limit λ_* that is defined by the equation

$$\nabla F(x_*) = A_* \lambda_*, \tag{3.14}$$

where A_* is the matrix whose columns are $\{a_i \mid i \in I_*\}$. Further, the algorithm ensures that the Lagrange multipliers of inequality constraints are all nonpositive. Therefore x_* is a Kuhn-Tucker point as required if the constraints whose indices are in I_* all have zero residuals at x_*, but it is possible that some nonzero residuals occur.

Let r_* be the vector whose components are the residuals at x_* of the constraints whose indices are in I_*. By feasibility, the conponents that correspond to any equality constraints are zero and the remaining components are all nonpositive. Therefore the scalar product $\lambda_*^{\mathrm{T}} r_*$ is a sum of nonnegative terms. Moreover, because τ remains constant, inequality (2.9) fails for all sufficiently large k, and the right hand side tends to zero due to the convergence of the sequence $\{F(x_k) \mid k = 1, 2, 3, \ldots\}$, the limit (3.11) and the bound $\|\mu_k\|_2^2 \leq W \|d_k\|_2^2$, which is derived from condition (3.2) and the identities $\|\mu_k\|_2^2 = \|Z_k^{-1} d_k\|_2^2 = d_k^{\mathrm{T}} B_k d_k$. Thus we deduce from inequality (2.9) that $\lambda_*^{\mathrm{T}} r_* = 0$. These remarks imply that, if any component of r_* is negative, then the corresponding component of λ_* is zero. Hence equation (3.14)

remains true if we delete from A_* and λ_* any columns and components that correspond to active constraints with nonzero residuals at x_*. Therefore we have proved the assertion that the Kuhn-Tucker conditions are satisfied at x_*.

We see that the analysis of this section does not impose any conditions on the constraints, such as strict complementarity and nondegeneracy. Therefore it suggests that our algorithm has some very useful advantages over those active set procedures that ignore inequality constraints with small nonzero residuals when calculating search directions. This suggestion is confirmed by some numerical results in the next section.

4. Numerical results and discussion

We draw our conclusions in this section from three of the test problems for optimization subject to linear constraints that are given by Hock and Schittkowski (1981), H&S say, and from three cases of a one-sided rational approximation calculation. We consider two versions of the BFGS updating formula. We test the advantages of setting $\tau = 0.01$ initially instead of a much smaller value. We investigate the safeguards of a Fortran implementation of the algorithm (Powell, 1989) that allows ACC = 0 in condition (2.10). We try the option of using single precision arithmetic. Finally some brief remarks are made on comparisons with other algorithms.

The H&S problems are the ones that are numbered 105, 112 and 118 in their book. Problem 105 has 8 variables and 1 inequality constraint in addition to bounds, but none of the constraints is active at the solution. Its main feature is a highly nonquadratic objective function, due to logarithmic, exponential and rational terms. There is a misprint in the specification of $F(\cdot)$ in H&S, which was discovered by consulting the source of the problem (Bracken and McCormick, 1968), namely that the coefficients y_{234} and y_{235} should be set to 260 instead of 250. Problem 112 has 10 variables and 3 equality constraints. Our algorithm solved it easily, but H&S report that poor numerical results were obtained by each of the six optimization procedures that they applied to this calculation. Problem 118 is a quadratic programming exercise in 15 variables and the solution is at a vertex of the feasible region. It seems straightforward, but, according to H&S, four out of six algorithms failed due to overflow or excessive computation time, and only one of the two remaining procedures gave an accurate result. It should be noted, however, that some of the implementations of algorithms that were used by H&S are less successful than implementations that have been tried by the author.

The one-sided rational approximation calculations are discretizations of the minimization of the integral

$$\int_0^5 [\mathrm{e}^t - p(t)/q(t)]^2 \, \mathrm{e}^{-2t} \, \mathrm{d}t \tag{4.1}$$

subject to the constraints

$$p(t)/q(t) \geq e^t, \quad 0 \leq t \leq 5, \tag{4.2}$$

the variables $x \in \mathbb{R}^5$ being the coefficients $\{[x]_i \mid i = 1, 2, \ldots, 5\}$ of the 2-2 rational function

$$\frac{p(t)}{q(t)} = \frac{[x]_1 + [x]_2 t + [x]_3 t^2}{1 + [x]_4 (t-5) + [x]_5 (t-5)^2}. \tag{4.3}$$

This form of $q(\cdot)$ reduces cancellation errors because, due to the growth of $\{e^t \mid 0 \leq t \leq 5\}$, $q(t)$ has to be relatively small at the right hand end of the interval $[0, 5]$. The discretization is the approximation of this interval by the point set $\{t_1, t_2, \ldots, t_r\}$ where $t_j = 5(j-1)/(r-1)$. Our three test problems, namely Fit 11, Fit 51 and Fit 251, are given by the values $r = 11$, $r = 51$ and $r = 251$ respectively. Specifically, the objective function is the sum

$$F(x) = \sum_{j=1}^{r} [1 - e^{-t_j} p(t_j)/q(t_j)]^2, \quad x \in \mathbb{R}^5, \tag{4.4}$$

there are $2r$ linear inequality constraints

$$p(t_j) \geq e^{t_j} q(t_j), \quad q(t_j) \geq 10^{-5}, \quad j = 1, 2, \ldots, r, \tag{4.5}$$

on the components of x, and for completeness we add the superfluous bounds

$$-10^{20} \leq [x]_i \leq 10^{20}, \quad i = 1, 2, \ldots, 5. \tag{4.6}$$

In all cases of this fitting problem we let the starting point of the iterative procedure be the feasible point $x_1 = (1 \ 1 \ 6 \ 0 \ 0)^T$.

The calculations were done by a Sun 3/50 workstation. First a double precision version of the Fortran implementation of our algorithm (Powell, 1989) was applied to the six test problems that have been mentioned, the accuracy parameter being set to ACC $= 10^{-6}$. The second column of Table 1 shows the numbers of function evaluations and iterations that were required. The subroutine that calculates $F(x)$

Table 1
Numbers of function values/iterations

Problem	Version of software				
	Standard version	Full BFGS	Small τ	ACC = 0 Real*8	ACC = 0 Real*4
Fit 11	35/25	36/29	30/23	36/26	33/23
Fit 51	36/30	36/29	49/46	37/31	42/31
Fit 251	30/22	36/27	159/142	39/25	23/15
H&S 105	67/49	72/51	69/53	77/51	76/52
H&S 112	38/30	36/26	35/29	69/34	44/25
H&S 118	39/23	39/23	33/18	39/23	34/18

also provides the gradient $\nabla F(x)$ whenever it is called. In all these trials the termination condition (2.10) was achieved. The constraints were satisfied too to the accuracy of the computer arithmetic, not only in these calculations but also in all the experiments that are reported later in this section. We are going to compare columns 3–6 of Table 1 to the second column.

We experimented with two different versions of the BFGS formula, because a Fortran implementation of our algorithm was given to IMSL in April, 1988 that applies only a part of the BFGS correction to the second derivative approximation. In this partial correction the first m_k columns of Z_{k+1} are the same as the first m_k columns of Z_k, and the last $n-m_k$ columns of Z_{k+1} are in the linear space that is spanned by the last $n-m_k$ columns of Z_k, in order that the equation $Z_{k+1}^{\mathrm{T}}A_k = R_k$ is inherited from $Z_k^{\mathrm{T}}A_k = R_k$, where m_k is still the number of active constraints. However, if the update has the property that $(Z_{k+1}Z_{k+1}^{\mathrm{T}})^{-1}$ is the matrix that is defined by the BFGS formula when $\delta_k = x_{k+1} - x_k$ and $\gamma_k = \nabla F(x_{k+1}) - \nabla F(x_k)$, then we have already chosen the last $n-m_k$ columns of Z_{k+1} correctly, but the first m_k columns of Z_{k+1} should have the values

$$Z_{k+1}(\cdot, i) = Z_k(\cdot, i) - \{\gamma_k^{\mathrm{T}} Z_k(\cdot, i)/\gamma_k^{\mathrm{T}}\delta_k\}\delta_k, \quad i = 1, 2, \ldots, m_k \tag{4.7}$$

(Powell, 1987). We see that the new Z_{k+1} also inherits the equation $Z_{k+1}^{\mathrm{T}}A_k = R_k$ from $Z_k^{\mathrm{T}}A_k = R_k$, because δ_k is orthogonal to the active constraint gradients, but the author overlooked this fact until after he had provided the IMSL software. Therefore the question arises whether to replace the updating formula of the original *standard version* of the Fortran implementation by a *full BFGS* update that includes equation (4.7). Columns 2 and 3 of Table 1 compare these alternatives. Since there does not seem to be a strong case for departing from the standard version, it is going to be retained by the author unless future experiments prove to be clearly in favour of the full update. The differences between the standard version and full BFGS are much less severe than the differences between reduced second derivative approximations and the full update, because the standard version does not delete any second derivative information when a constraint is added to the active set.

We recall that, in contrast to our method, it is usual for the search directions of active set algorithms to be independent of constraints with nonzero residuals, so many small step-lengths can occur due to early encounters with constraint boundaries. Therefore, as stated in Section 1, our search directions are calculated to move no closer to the boundaries of all constraints with small residuals, where "small" is measured by the test (2.2). Thus our procedure becomes a traditional active set algorithm if τ is chosen so that inequality (2.2) holds if and only if $a_j^{\mathrm{T}}x - b_j$ would be zero in exact arithmetic. We simulated this situation by running the Fortran program with $\tau = 10^{-6}$ initially instead of the standard initial choice $\tau = 10^{-2}$. Thus we obtained the results that are displayed in the "small τ" column of Table 1.

Of course the one-sided approximation test problems were chosen to demonstrate the ability of our algorithm to handle small constraint residuals efficiently. When one of the constraints (4.5) holds as an equation and when the spacing $t_{j+1} - t_j$ of

the discretization is small, then there are always constraints with small nonzero residuals too. Further, a move along the boundary of any one of the constraints (4.5) can soon lose feasibility because the constraints form two envelopes of smooth surfaces. Thus feasibility forced many line searches to give small step-lengths in problems Fit 51 and Fit 251 when the Fortran program was run with $\tau = 10^{-6}$ initially. In the Fit 51 calculation, for example, α_k was restricted by feasibility on 43 out of 46 iterations, but this restriction occurred on only 9 of the 30 iterations of the standard program. Further, the analysis of Section 3 suggests that the standard version would remain efficient for very large values of r in expressions (4.4) and (4.5), but many of the traditional active set algorithms would suffer severe difficulties from the near-degeneracies of the constraints.

An alternative view of the advantages of the standard version on problems Fit 51 and Fit 251 comes from considering the changes that need to be made to the active set. For example, suppose that the constraint of the first part of expression (4.5) with $j = j_1$ is in the current active set, but that, in order to reach the solution, the constraint index has to be increased to $j = j_2$. When $j_2 - j_1$ is large, which is not uncommon for a fine discretization, it would be highly inefficient if the index were increased in steps of one, because then at least $j_2 - j_1$ iterations would be performed. Unfortunately, this tends to happen in the traditional active set procedures, partly because each part of expression (4.5) gives a sequence of tangent planes to a smooth function. For example, the active sets of iterations 4–14 of the "small τ" minimization of Fit 51 are $\{2, 51\}$, $\{3, 51\}$, $\{3, 50\}$, $\{3, 49\}$, $\{3, 48\}$, $\{4, 48\}$, $\{4, 47\}$, $\{5, 47\}$, $\{5, 46\}$, $\{5, 45\}$ and $\{6, 45\}$, while the "standard version" gives the same vectors of variables $\{x_1, x_2, x_3, x_4\}$, but the active sets of iterations 4–6 are $\{3, 51\}$, $\{5, 48\}$ and $\{7, 42\}$. It is very helpful to the standard version that, if a constraint is present in I_{k+1} because it restricts the step-length α_k, then the residual of this constraint at x_k is not "small".

Sometimes the user wishes to achieve good accuracy without any fine tuning of the parameter ACC of the termination condition (2.10). It is therefore convenient if the software includes enough safeguards to provide termination when ACC = 0 without much unproductive calculation. Column 5 of Table 1 shows the performance of the standard version in this case, the value ACC = 10^{-6} being set in all the earlier experiments. Because these calculations were done in double precision arithmetic, a single precision implementation of the standard version with ACC = 0 was tried too. Its results are displayed in the last column of Table 1. It seems that the cost of setting ACC = 0 is not exorbitant, but line searches tend to require more function values than usual near the limit of attainable accuracy, because of disparities between the actual changes in computed values of $F(\cdot)$ and the changes that are predicted by the use of first derivatives.

Since the feasibility conditions are satisfied well, we measure the accuracy of our calculations by the final values of $F(\cdot)$. Each of the four computer programs that produced columns 2–5 of Table 1 yields all the digits in the Real*8 column of Table 2, which is good. Of course we expect errors in eight decimal numbers from

Table 2
Final values of $F(\cdot)$

Problem	Real*8	Real*4	H&S
Fit 11	0.000568306	0.000568303	—
Fit 51	0.002509683	0.002509985	—
Fit 251	0.011651287	0.011651262	—
H&S 105	1138.4107	1138.4105	1138.416240
H&S 112	−47.761091	−47.761101	−47.76109026
H&S 118	664.82045	664.82050	664.8204500

single precision arithmetic, and the Real*4 column of Table 2 was calculated by the software of the last column of Table 1. In view of the cancellation in the three fitting problems, the single precision software gives good accuracy too. Four of the entries in the Real*4 column of Table 2 demonstrate that rounding errors can cause a computed value of $F(\cdot)$ to be less than the least value of $F(\cdot)$ in exact arithmetic, due partly to the small constraint violations that are a usual consequence of rounding. The last column of Table 2 displays the final function values that are stated on pages 114, 120 and 126 of Hock and Schittkowski (1981). The liberty has been taken of replacing the H&S 112 number by the value that they give for their problem 111, because these two problems have equivalent objective functions, but the H&S entry for the H&S 112 calculation is too high. Further, it seems that H&S did not apply any software that finds a close approximation to the solution of H&S 105.

The numerical results on pages 154–156 of Hock and Schittkowski (1981) allow our software to be compared to six computer programs for minimization subject to *nonlinear* constraints. These results for problems H&S 105, H&S 112 and H&S 118 indicate that our algorithm is much more accurate and relaible than the other procedures. Indeed, it has just been mentioned that all the Real*8 versions of our software provide a definite improvement in the final value of the objective function of H&S 105, and earlier we noted that only one of the six older procedures finds a good solution to the quadratic programming calculation H&S 118. When applied to the H&S 112 problem, none of the older procedures reduces the norm of the Kuhn-Tucker residual vector below 10^{-3}, so, excluding a case where there is a large constraint violation, their values of the objective function remain above -47.72550 although Table 2 shows that $F(x) = -47.76109$ is attainable. Because these calculations seem to be among the more difficult test problems of H&S, they are likely to expose any weaknesses in *implementations* of optimization procedures. Therefore these comparisons do not necessarily show major defects in the *algorithms* that are behind the implementations.

Tables 1 and 2 provide some examples of the efficiency of our method, if one measures efficiency by the numbers of function and gradient evaluations and by the accuracies of the computed solutions. In particular, the three fitting problems show clearly the advantages of allowing the search directions to depend on constraints

with small residuals. These demonstrations and the theory of Section 3 suggest that the given procedure is highly suitable for the solution of optimization calculations with many linear constraints that may be nearly degenerate. Further, we have found that the matrix factorizations that are employed by Goldfarb and Idnani (1983) for quadratic programming extend conveniently to general differentiable objective functions.

Acknowledgements

IMSL (International Mathematical and Statistical Libraries, Houston, Texas) provided the motivation for the development of the algorithm that is described. The author is very grateful to IMSL, not only for its interest and support, but also for its willingness to agree to unlimited publication of the results, including the promulgation of the report (Powell, 1989) that includes a Fortran listing of an implementation of the algorithm. Further, he thanks M.D. Buhmann for several valuable suggestions on a preliminary version of this paper and two referees for their comments on a later version.

References

J. Bracken and G.P. McCormick, *Selected Applications of Nonlinear Programming* (Wiley, New York, 1968).

R. Fletcher, *Practical Methods of Optimization* (Wiley, Chichester, 1988).

P.E. Gill, W. Murray and M.H. Wright, *Practical Optimization* (Academic Press, New York, 1981).

D. Goldfarb and A. Idnani, "A numerically stable dual method for solving strictly convex quadratic programs," *Mathematical Programming* 27 (1983) 1-33.

W. Hock and K. Schittkowski, *Test Examples for Nonlinear Programming Codes, Lecture Notes in Economics and Mathematical Systems, Vol. 187* (Springer, Berlin, 1981).

E. Polak, *Computational Methods in Optimization: A Unified Approach* (Academic Press, New York, 1971).

M.J.D. Powell, "Updating conjugate directions by the BFGS formula," *Mathematical Programming* 38 (1987) 29-46.

M.J.D. Powell, "On a matrix factorization for linearly constrained optimization problems," Report DAMTP/1988/NA9, University of Cambridge (Cambridge, UK, 1988).

M.J.D. Powell, "TOLMIN: a Fortran package for linearly constrained optimization calculations," Report DAMTP/1989/NA2, University of Cambridge (Cambridge, UK, 1989).

P. Wolfe, "Convergence conditions for accent methods," *SIAM Review* 11 (1969) 226-235.