SIAM J. OPTIM.
Vol. 24, No. 3, pp. 1122–1137

ON WOLFE'S METHOD FOR RESOLVING DEGENERACY IN LINEARLY CONSTRAINED OPTIMIZATION*

ROGER FLETCHER†

Abstract. Wolfe [*J. Soc. Indust. Appl. Math.*, 11 (1963), pp. 205–211] describes a novel and very useful method for resolving degeneracy in the Simplex method for Linear Programming (LP). The simplicity and reliability of the method makes it an excellent choice in this author's opinion. The method is described here in the context of an active set method (ASM) format for LP. The method solves recursively generated subproblems that are smaller than the original, which contributes to efficiency. Data structures are described that enable the recursive aspect of the method to be implemented very easily with minimal storage overhead. The method is extended to solve a general form of linearly constrained optimization problem that includes quadratic programming (QP) and allows simple bounds on the variables and both equation and inequality general linear constraints. Issues of round-off error are discussed and heuristics are described that have proved to be very reliable in practice. The method is further extended to QP problems in which general linear constraints are handled as L_1 terms in the objective function.

Key words. degeneracy, Wolfe's method, linear programming, quadratic programming, linearly constrained optimization, L1QP

AMS subject classifications. 90-08, 90C05, 90C20, 90C26

DOI. 10.1137/130930522

1. Introduction.

The context of this paper is the design of an active set method (ASM) for the solution of linearly constrained optimization problems. It is now well understood that degeneracy is both a theoretical and a practical possibility for such problems. The simple proof of finite termination of the Simplex method for linear programming (LP) relies on strict improvement of the objective function at each iteration, but degeneracy can prevent this from happening. An example was given at an early date by Beale [1], showing that the Simplex method could cycle and fail to solve a simple LP problem unless special attention was taken to resolve degenerate basic feasible solutions. Hall and McKinnon [14] identify the smallest possible class of LP problems for which cycling can occur and give further simple examples.

Many techniques have been suggested for resolving degeneracy in LP. These include *lexicographic ordering* (Dantzig, Orden, and Wolfe [5]); *primal-dual algorithms* (Balinski and Gomory [2], Fletcher [8]); the *least index rule* (Bland [3]); and there is a generalization of [8] for quadratic programming (QP) (Fletcher [9]). These methods are provably convergent in exact arithmetic. However, an important issue for these methods in the presence of round-off error is the need to recognize whether or not a data value is an exact zero corrupted by round-off. A wrong decision here can have a serious effect on the performance of the method, and can easily lead to failure. There are also methods in which the data is perturbed in order to remove degeneracy, such as a very early method of Charnes [4] and the EXPAND procedure of Gill et al. [12]. Such methods are also not immune to difficulties caused by round-off; see, for example, [14].

*Received by the editors July 24, 2013; accepted for publication (in revised form) April 9, 2014; published electronically August 5, 2014.
http://www.siam.org/journals/siopt/24-3/93052.html
†Department of Mathematics, University of Dundee, Dundee, Scotland DD1 4HN, UK (fletcher@maths.dundee.ac.uk).

However, there is a method due to Wolfe [16], different in concept from all the above, that is extremely simple to implement, is provably convergent in exact arithmetic, and can be made very robust in the presence of round-off. The method was originally presented (in 1963) in the context of the Simplex method for LP, but is readily adapted for use with an ASM. In section 2, Wolfe's method is described in the context of a very basic ASM format for LP. The method solves recursively generated subproblems that are smaller than the original, which contributes to efficiency. Data structures are described that enable the recursive aspect of the method to be implemented very easily with minimal storage overhead.

In section 3, the method is extended to a completely general form of linearly constrained optimization problem that includes QP and also the case in which the objective function is smooth but possibly nonquadratic, which we refer to here as linear constraint programming (LCP). The format allows simple bounds on the variables, and both equation and inequality general linear constraints. Section 5 discusses the issue of round-off error and introduces heuristics that have proved to be very reliable in practice. Section 6 extends the method still further to QP problems in which general linear constraints are handled as L_1 terms in the objective function. A simple modification to the data structure enables this to be done with no significant complication. Illustrative examples are presented in sections 4 and 7.

2. Wolfe's method for LP. Wolfe [16] describes a very elegant recursive method for resolving degeneracy in the Simplex method for LP. Wolfe describes his method in the context of the tableau form of the Simplex method for solving an LP in the form

$$\begin{array}{ll} \text{minimize} & \mathbf{c}^T\mathbf{x} \\ \text{subject to} & \mathbf{x} \geq \mathbf{0}, \\ & A\mathbf{x} = \mathbf{b} \end{array} \tag{1}$$

in which slacks have been added to remove any general inequality constraints. However, it is well known (e.g., Fletcher [7]) that the Simplex method is entirely equivalent to an ASM for solving an LP in the form

$$\begin{array}{ll} \text{minimize} & \mathbf{c}^T\mathbf{x} \\ \text{subject to} & A^T\mathbf{x} \geq \mathbf{b}, \end{array} \tag{2}$$

where A is an $n \times m$ matrix, $m > n$, and we denote column i of A by $\mathbf{a}_i$. It is in this format that Wolfe's method will be explained. In the ASM formulation for LP, the current iterate $\mathbf{x}^c$ is a *vertex* of the feasible region. A constraint i for which $\mathbf{a}_i^T\mathbf{x}^c = b_i$ is said to be *active*, and the index set $\mathcal{A}^c$ collects the indices of active constraints at $\mathbf{x}^c$. An initial vertex $\mathbf{x}^1$ can be determined (or the problem shown to be infeasible) by a so-called *Phase* 1 calculation, also based on the ASM formulation, and using Wolfe's method to resolve degeneracy. A vertex is determined by the intersection of n active constraints whose coefficient vectors $\mathbf{a}_i$ are linearly independent. These constraints are said to form the *working set*, and the set $\mathcal{W}^c$ collects their indices. The columns $\mathbf{a}_i$ with $i \in \mathcal{W}^c$ form a nonsingular matrix B.

An ASM iteration starts with the computation of Lagrange multipliers $\boldsymbol{\lambda}$ from

$$B\boldsymbol{\lambda} = \mathbf{c}, \tag{3}$$

and if $\boldsymbol{\lambda} \geq \mathbf{0}$, then $\mathbf{x}^c$ is optimal. Otherwise a working set constraint, p say, with negative multiplier is chosen to be relaxed. A feasible descent direction of search $\mathbf{s}^c$ is calculated by solving the system

$$B^T\mathbf{s}^c = \mathbf{e}_p, \tag{4}$$

where $\mathbf{e}_p$ is the unit vector corresponding to the column of B in which $\mathbf{a}_p$ is located. Hence $\mathbf{s}^c$ is the corresponding column of B^{-T} and provides a direction along which constraint p is relaxed, whilst other working set constraints remain active. A search is made along the line $\mathbf{x}(\alpha) = \mathbf{x}^c + \alpha\mathbf{s}^c$, and α^c is chosen to be the largest value of $\alpha \geq 0$ subject to retaining feasibility with respect to constraints $\mathcal{N}^c = \{i : i \notin \mathcal{W}^c\}$ not in the working set.

The calculation of α^c is conveniently carried out by storing *residuals* $r_i = \mathbf{a}_i^T\mathbf{x}^c - b_i$ and calculating *denominators* $w_i = -\mathbf{a}_i^T\mathbf{s}^c$ for $i \in \mathcal{N}^c$. Then the so-called *ratio test*

$$\alpha^c = \min_{\substack{i\in\mathcal{N}^c \\ w_i>0}} (r_i/w_i) \tag{5}$$

determines the value of α^c. One possible outcome is that if there are no denominators $w_i > 0$, then $\mathbf{c}^T\mathbf{x}$ can be reduced without bound along the line, and the LP is said to be *unbounded*.

Otherwise α^c determines a new vertex $\mathbf{x}^{c+1}$, and residuals are updated by $r_i := r_i - \alpha^c w_i$. The newly active constraint, say, q, replaces the relaxed constraint p in $\mathcal{W}^c$ to give $\mathcal{W}^{c+1}$ and $\mathbf{a}_q$ replaces $\mathbf{a}_p$ in B. It can be shown that B remains nonsingular. Finally, some suitable invertible representation of B is updated (the *pivot* step). The calculations are entirely equivalent to Simplex iterations if slack variables are added to the inequality constraints; see again [7].

A vertex $\mathbf{x}^c$ is said to be *degenerate* when the set of *degenerate constraints* $\mathcal{A}^c \backslash \mathcal{W}^c$ is nonempty. A *degeneracy block* is said to occur if $\alpha^c = 0$ is chosen in the ratio test, so that $\mathbf{x}^{c+1} = \mathbf{x}^c$, and no progress is made. Although a degenerate constraint can be pivoted into the working set, it is possible that the method can *cycle* by returning to a previous working set after a number of iterations. Thus the algorithm never terminates, even though $\mathbf{x}^c$ may not be a solution of the LP. In the absence of a degeneracy block, each iteration reduces $\mathbf{c}^T\mathbf{x}$ and the algorithm terminates after a finite number of iterations, either by recognizing an unbounded LP, or by finding a solution.

Wolfe's method uses *recursion* to resolve degeneracy, and we shall refer to (2) as the *level* 1 *LP*. If a degeneracy block occurs in the level 1 LP, a new (level 2) LP is set up and solved by

a. ignoring all inactive constraints,
b. making arbitrary positive perturbations to the stored (zero) residuals of the degenerate constraints,
c. performing a sequence of LP-like ASM iterations on the modified problem, but without changing $\mathbf{x}^c$, until either optimality or unboundedness is recognized, in which case the code returns to level 1 and the residuals of level 2 inactive constraints are reset to zero.

We note that multiplying the perturbations by some fixed $\delta > 0$ has no effect on the resulting sequence of pivots. Thus, in the limit $\delta \searrow 0$, if a solution of the level 2 LP is found, then this is a solution of the level 1 LP. Alternatively, if the level 2 LP is unbounded, then the degenerate constraints at level 1 are no longer blocked, and ASM iterations can continued at level 1. We note, as for level 1, the level 2 LP must be solved in a finite number of iterations if there is no degeneracy block at level 2.

A further possibility, however, is that a degeneracy block does occur during the sequence of ASM iterations at level 2. This can be resolved recursively by setting up and solving a new level 3 LP in a similar way, and so on. If a solution is located at any level $l > 1$, the original (level 1) LP has been solved. If the level l LP is unbounded, the degeneracy block at level $l - 1$ is removed, and a return to level $l - 1$ is made.

We now show that the level 1 LP will always be solved (including the possibility of recognizing an unbounded LP) in a finite number of iterations. For a nondegenerate iteration at any level, α^c is strictly positive and becomes the new value of r_p, the residual of the constraint being relaxed (this follows from the definition of $\mathbf{s}^c$). Thus there is always at least one strictly inactive constraint at the start of any iteration (for $l > 1$), and hence at least one inactive constraint is ignored when going from level l to level $l+1$. Consequently, there is an upper limit, say, L, to the number of levels of recursion. Thus any LP at level L is solved finitely without a degeneracy block occurring. It now follows that any LP at level $L-1$ is solved finitely. Hence, by induction, the level 1 LP is solved finitely.

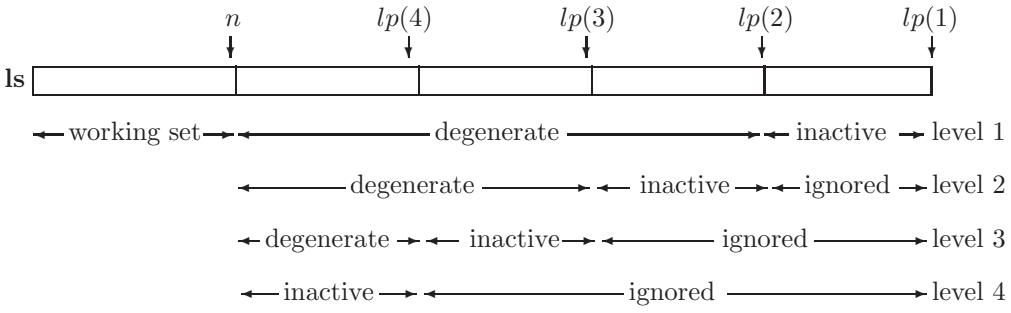

FIG. 1. *The status of* **ls** *and* **lp** *at level* 4 *of recursion.*

Although the concept of recursion is often regarded as being somewhat complex, in fact the implementation of Wolfe's method is surprisingly simple and elegant in a common programming language such as Fortran or C. The method can be carried out with no extra storage, other than a vector of pointers, which we refer to as **lp**. We assign the value $lp(1) = m$. An integer vector **ls** of length m is maintained that contains a permutation of the indices $i = 1, 2, \ldots, m$. Entries $ls(j)$ for $j = 1, 2, \ldots, n$ contain the elements of $\mathcal{W}^c$ and entries $ls(j)$ for $j = n+1, \ldots, m$ contain the elements of $\mathcal{N}^c$. When a degeneracy block occurs, the indices of degenerate constraints are permuted to the front of the partition of **ls** containing the $\mathcal{N}^c$ constraints, and a pointer $lp(2)$ marks the position of the rightmost degenerate constraint. After the residuals of degenerate constraints are perturbed, the level 2 LP can be solved with the same piece of code as for level 1, but with $lp(2)$ marking the extent of constraint set. This has the effect of ignoring the level 1 inactive constraints. If a degeneracy block is detected at level 2, then a similar rearrangement is made and a new pointer $lp(3)$ is set up, and so on. A schematic view of the vector **ls** is shown in Figure 1. The length of the vector **lp** is the value L referred to above, but this value is not known a priori, although it is bounded above by $m - n$. In practice, even for very large LPs, a value of say 50 has proved plenty large enough. The details of how **ls** and **lp** are integrated into an ASM code for solving (2) are shown in Figure 2. The residuals and denominators are stored in arrays **r** and **w**, respectively, indexed in natural order.

A feature of the code in Figure 2 is that if a degeneracy block is recognized, the arbitrary perturbations to **r** are simply $r_i = 1$. It has been suggested elsewhere that a random choice of perturbation might be made. This choice is likely (but is

```
Compute a feasible vertex x (Phase 1)
Set up ls, B, r, and set level = 1 and lp(1) = m
1:  Solve Bλ = c
    if λ ≥ 0 (Optimality test) then
        if level > 1, set r(ls(j)) = 0,  j = n+1, ..., lp(level)
        exit (x is a solution)
    endif
    Choose a constraint p with λ_p < 0 to be relaxed
    Compute s and set w(ls(j)) = -s^T a_{ls(j)},  j = n+1, ..., lp(level)
2:  Ratio test
    if w(ls(j)) ≤ 0,  j = n+1, ..., lp(level) then
        if level == 1, exit (LP is unbounded)
        set r(ls(j)) = 0,  j = n+1, ..., lp(level)
        set w(ls(j)) = -s^T a_{ls(j)},  j = lp(level)+1, ..., lp(level-1)
        set level = level - 1
        goto 2 (degeneracy block is removed)
    endif
    set q = arg min_{i: w(i)>0} r(i)/w(i),  i = ls(j), j = n+1, ..., lp(level)
    set alpha = r(q)/w(q)
    if alpha == 0 (degeneracy block) then
        set plev = n
        for j = n+1, ..., lp(level)
            set i = ls(j)
            if r(i) == 0 then
                set plev = plev + 1
                exchange ls(j) and ls(plev)
                set r(i) = 1
            endif
        endfor
        set level = level + 1
        set lp(level) = plev
        goto 2
    endif
    update r and (if level == 1) x
    set r(p) = alpha
    exchange the locations in ls containing p and q
    replace a_p by a_q in B
    goto 1
```

FIG. 2. *Wolfe's method for a simple LP.*

not guaranteed) to remove the need for recursion. However, the implementation of recursion is very straightforward and is not to be feared. Also each level of recursion reduces the size of the LP being solved, which is a small advantage. Choosing the value of 1 can actually promote further recursion in some LP problems—for example, those having integer coefficients. Also choosing random perturbations makes the outcome of the method unpredictable and difficult to replicate. A further point is that if there

is only one degenerate constraint ($plev = n + 1$ in the code), it is not necessary to invoke recursion, because it can be shown that the working set exchange removes the degeneracy block. Another important issue when implementing Wolfe's method, and indeed any method for LP and for degeneracy, is how to control the effects of round-off error. This is discussed in section 5.

Here it is convenient to introduce another item of terminology. In a nondegenerate ASM iteration the vector $\mathbf{s}^c$ is the direction of an *edge* of the feasible region, emanating from $\mathbf{x}^c$. Any multiplier $\lambda_i < 0$ determines an edge $\mathbf{s}_i$ that is a feasible descent direction, by solving $B^T\mathbf{s}_i = \mathbf{e}_i$ in an analogous way to (4). In early days the most negative multiplier was the preferred choice to determine p. More recently, the alternative possibility of choosing the *steepest edge direction* has been considered. Goldfarb and Reid [13] show how this can be done in the context of (1) and similar ideas hold for (2). The technique requires the updating of *edge weights* $w_i = ||\mathbf{s}_i||_2$ when B is updated, which requires a single extra solve with B. There is anecdotal evidence that the idea is more effective in the context of the dual of (1) (which has the same structure as (2)), rather than with (1) itself. My experience with the technique has been favorable, leading to significantly fewer iterations and an overall reduction in computing time. However, Wolfe's method operates independently of how the edge direction is chosen.

3. Wolfe's method for QP and LCP. QP and LCP differ from LP in that solutions often have fewer than n active constraints. Moreover, the simple formulation of LP (2) is inconvenient for practical use, so the remarks of this section are also relevant to LP. In particular, it is important to provide explicitly for equality constraints and simple bound variables. A widely used formulation that allows this in a convenient way is

$$\begin{aligned} &\text{minimize} && f(\mathbf{x}), \qquad \mathbf{x} \in \mathbb{R}^n, \\ &\text{subject to} && \mathbf{l} \le \begin{bmatrix} \mathbf{x} \\ A^T\mathbf{x} \end{bmatrix} \le \mathbf{u}, \end{aligned} \tag{6}$$

where A is an $n \times m$ matrix, $m \ge 0$, and $\mathbf{l}$ and $\mathbf{u}$ ($\mathbf{l} \le \mathbf{u}$) are vectors in $\mathbb{R}^{n+m}$ of lower and upper bounds, respectively. Thus we have two-sided bounds on both $\mathbf{x}$ and $A^T\mathbf{x}$. An equality constraint is specified simply by setting $l_i = u_i$. The objective function $f(\mathbf{x})$ is assumed to be continuously differentiable, and the gradient vector $\nabla f(\mathbf{x})$ is denoted by $\mathbf{g}(\mathbf{x})$. We assign an index number to each individual constraint in the order given above; the simple bounds are indexed by 1 through n and the general constraints by $n + 1$ through $n + m$. We now use $\mathbf{a}_i,\ i = 1, \ldots, n + m$, to denote columns of the matrix $[I\ A]$.

In an ASM for QP or LCP the current feasible point $\mathbf{x}^c$ may no longer be a vertex, and the working set $\mathcal{W}^c$ may contain fewer than n elements, say $\bar{n}$. A convenient way to proceed is to introduce $k = n - \bar{n}$ so-called *free variables* x_i, $i \in \mathcal{F}^c$, that are strictly feasible at $\mathbf{x}^c$ and are used to parametrize the null space $\{\mathbf{s} \mid \mathbf{a}_i^T\mathbf{s} = 0,\ i \in \mathcal{W}^c\}$ defined by the working set constraints. Free variables are also useful to handle unbounded variables ($u_i = -l_i = \infty$). As in the previous section, we maintain a nonsingular matrix B whose columns are the vectors $\mathbf{a}_i$, $i \in \mathcal{W}^c$, together with the vectors $\mathbf{a}_i$, $i \in \mathcal{F}^c$, and we denote $\mathcal{N}^c = \{i : i \notin \mathcal{W}^c \cup \mathcal{F}^c\}$. Also the constraints in $\mathcal{W}^c$ may now be either inequality constraints or equality constraints. It is convenient to divide the first n locations of the vector $\mathbf{ls}$ into three partitions: for equations, inequalities, and free variables, respectively.

We also need to be able to specify whether it is the lower or upper bound of a working set inequality constraint that is currently active. This requires one additional bit of information per constraint. This has been implemented by adding a sign to the elements of **ls**, with "+" indicating the lower bound and "−" the upper bound. It is also important to calculate residuals of inactive constraints from the nearest bound, for otherwise there can be round-off problems, particularly when one of the bounds is $\pm\infty$. Again the sign bit is used to indicate which bound is relevant. The choice of sign bit for an equality constraint or free variable is arbitrary. For simplicity we assume that only the lower bounds are relevant in this section.

At the start of an ASM iteration from $\mathbf{x}^c$, the vector $\boldsymbol{\lambda}$ defined by

$$B\boldsymbol{\lambda} = \mathbf{g}^c \tag{7}$$

is calculated, where $\mathbf{g}^c$ refers to the current gradient vector. Components of $\boldsymbol{\lambda}$ corresponding to free variables are the *reduced gradients*, and components corresponding to $\mathcal{W}^c$ constraints are Lagrange multiplier estimates. If the reduced gradients are zero (to within some tolerance) and the Lagrange multipliers of inequality constraints are nonnegative, then $\mathbf{x}^c$ is a KKT point and the ASM iterations terminate (see, e.g., [7]).

Otherwise, on iteration c a feasible descent direction of search $\mathbf{s}^c$ is calculated. There are various possibilities here. As in (4), an *edge direction* may be chosen, obtained by relaxing one of the inequality constraints, p say, in the working set whose multiplier is negative (if one exists that is). Alternatively, a *null space direction* might be chosen (if free variables exist) in which only the free variables are changed. There are many ways in which a null space direction might be specified. In strictly convex QP it might be the direction to the minimizer in the null space, and in LCP it might be a reduced quasi-Newton direction. Another possibility for QP or LCP is to use a reduced conjugate gradient method, or a reduced steepest descent method, based on the limited memory Ritz value method of Fletcher [10]. A stable way of deciding whether to choose an edge direction or a null space direction is to compare the slope of the steepest edge direction and the (negative) Euclidean length of the reduced gradient in the null space, and choose the most negative.

A search is then made along the line $\mathbf{x}(\alpha) = \mathbf{x}^c + \alpha\mathbf{s}^c$, and α^c is chosen to be some value of $\alpha \geq 0$ that retains feasibility. Denominators $w_i = -\mathbf{a}_i^T\mathbf{s}^c$ are calculated and used to update the residuals r_i, as for LP. There are then various possible outcomes. If an edge direction is chosen, either a new constraint q may become active as in the LP case, and $\mathbf{a}_q$ replaces $\mathbf{a}_p$ in B for the next iteration, or a shorter step might be chosen in order, for example, to ensure a reduction in $f(\mathbf{x})$. In this case the column $\mathbf{a}_q$ that replaces $\mathbf{a}_p$ in B is the one corresponding to a suitably chosen new free variable.

If $\mathbf{s}^c$ is a null space direction, then either a new constraint q becomes active, in which case a free variable p is chosen to be removed from $\mathcal{F}^c$, or $\mathbf{x}^{c+1}$ is reached without an inactive constraint becoming active, in which case no change to B is required.

A degeneracy block occurs if $\alpha^c = 0$ occurs, on account of a degenerate constraint q being chosen by the line search. The action to be taken is the same for both QP and LCP contexts, as follows:

a. If constraint q is a degenerate equality constraint, it is brought into the equation partition of **ls** and ASM iterations continue at level 1. Because constraints are never removed from this partition, this situation can only occur finitely and so cannot cause cycling.
b. If $\mathbf{s}^c$ is a null space direction, then a free variable will be removed from $\mathcal{F}^c$. As

there are only finitely many free variables, again ASM iterations can continue at level 1 without causing cycling.

c. Otherwise, $\mathbf{s}^c$ is an edge direction and constraint q is an inequality. In this case Wolfe's recursive method is invoked. No changes to the free variables are made, and all directions are edge directions. Eventually, the iteration returns to level 1. A return to level 1 is also made if a above occurs at levels $l > 1$.

If there are free variables present, the outcome of c may not be sufficient to remove the degeneracy block, if, subsequently, a null space direction is chosen on return to level 1. However, as in b above, the degeneracy block will eventually be resolved when all free variables have been removed, if not before. A simple example is given in the next section that illustrates this point and the use of recursion.

In effect, the only difference between the procedure for resolving degeneracy outlined here, and that in section 2, is in regard to how free variables and equality constraints are handled. The procedure is exactly the same, whether for LP, QP, or LCP. Just as for LP, $\mathbf{x}^c$ remains unchanged during recursion, and the gradient $\mathbf{g}^c$ likewise remains unchanged.

4. An illustrative QP example. In this section a simple QP example with five variables and three general constraints is given, to illustrate a number of features of Wolfe's method. The objective function is $f(\mathbf{x}) = \mathbf{c}^T\mathbf{x} + \frac{1}{2}\mathbf{x}^T G\mathbf{x}$ and the data for (6) is

$$\mathbf{c} = \begin{pmatrix} 0 \\ -6 \\ -6 \\ -12 \\ -9 \end{pmatrix}, \qquad A = \begin{bmatrix} 2 & 5 & 0 \\ 0 & 0 & -1 \\ 0 & -3 & 0 \\ 0 & 0 & -3 \\ -1 & -1 & 0 \end{bmatrix},$$

and G is the unit matrix. The upper bounds $\mathbf{u}$ are all at infinity, and the lower bounds $\mathbf{l}$ are zero, except for the lower bounds on x_1 and x_2, which are at $-\infty$. The initial working set is $\mathcal{W}^c = \{3, 4, 5\}$ and the free variables are $\mathcal{F}^c = \{1, 2\}$. The initial matrix B is the unit matrix, and the initial point is $\mathbf{x}^1 = \mathbf{0}$. The progress of Wolfe's method is shown in Table 1.

The optimality test at the initial point shows that the norm of the reduced gradient is 6 and (minus) the slope of the steepest edge direction is 12. Because the latter is larger, contraint 4 is chosen to be relaxed, and denominators are calculated. The first row shows degeneracy in r_6, r_7, and r_8, and the search is blocked because $w_8 > 0$. Hence recursion to level 2 takes place and the stored zeros in r_6, r_7, and r_8 are changed to ones. There are no inactive constraints to be ignored in this example.

At level 2, constraint 4 is again chosen in the optimality test and a regular LP iteration takes place. The steplength is $\alpha^c = \frac{1}{3}$, constraint 8 becomes active, and is exchanged with constraint 4 in B. Two further LP iterations follow. (Note that in the last of these, constraint 7 is degenerate. However, because there is only *one* degenerate constraint, there is no need to invoke recursion to level 3.) Finally, there are no working set constraints with negative multiplier, the code returns to level 1, and the previously degenerate constraint residuals are reset to zero.

If there were no free variables present, this would be flagged as an optimal solution. In this case, however, there are free variables with nonzero reduced gradients, and the optimality test chooses a null space step (the slope of the steepest edge step being regarded as zero). In this example a reduced steepest descent direction is calculated and used to calculate denominators. However, the search is still blocked

TABLE 1

Progress of Wolfe's method on a simple QP: The first column shows the level of recursion, and columns 2 through 6 each show the index of a working set constraint or free variable, followed by the corresponding multiplier or reduced gradient value. Indices of free variables are rendered in bold face. Columns 7, 8, and 9 of the table each show the index of a degenerate or inactive constraint, followed by its residual value r_i, measured from its lower bound. Rows of the table with entries only in columns 7, 8, and 9 contain the denominators w_i relating to the row above.

Level	Working set constraints / free variables										$\mathcal{N}^c$ constraints					
1	3	−6.	4	−12.	5	−9.	**1**	0.	**2**	−6.	6	0.	7	0.	8	0.
												0.		0.		3.
2	3	−6.	4	−12.	5	−9.	**1**	0.	**2**	−6.	6	1.	7	1.	8	1.
												0.		0.		3.
2	3	−6.	8	4.	5	−9.	**1**	0.	**2**	−2.	6	1.	7	1.	4	$\frac{1}{3}$
												1.		1.		0.
2	3	−6.	8	4.	6	9.	**1**	−18.	**2**	−2.	5	1.	7	0.	4	$\frac{1}{3}$
												0.		3.		0.
2	7	2.	8	4.	6	7.	**1**	−24.	**2**	−2.	5	1.	3	0.	4	$\frac{1}{3}$
1	7	2.	8	4.	6	7.	**1**	−24.	**2**	−2.	5	0.	3	0.	4	0.
												−24.		−48.		$\frac{2}{3}$
1	7	2.	8	6.	6	7.	4	6.	**1**	−24.	5	0.	3	0.	2	∞
												−24.		−48.		0.
1	7	$\frac{2}{3}$	8	1.	6	$\frac{1}{3}$	4	6.	**1**	0.	5	8.	3	4.	2	∞

by $r_4 = 0$ because $w_4 > 0$. A free variable is now chosen (2 in this case) to exchange with constraint 4. Because the number of free variables has been reduced, iterations continue at level 1.

A null space direction is again chosen on the next iteration and although there is still degeneracy in r_3 and r_5, this time the search is not blocked. A Cauchy step locates the minimizer along the line, and the resulting point is optimal. Only on this last iteration is the vector $\mathbf{x}^c$ updated, the solution being $\mathbf{x}^* = \begin{pmatrix} 4 & 0 & 8 & 0 & 4 \end{pmatrix}^T$. From the last line in the table, the optimal multipliers are $\boldsymbol{\lambda} = \begin{pmatrix} 0 & 0 & 0 & 6 & 0 & \frac{1}{3} & \frac{2}{3} & 1 \end{pmatrix}^T$.

Note that the table does not show values of $\mathbf{x}^c$ at any time. For any free variable, x_i say, we need both the value of x_i and the reduced gradient (which we store in r_i) to be available. Thus the value of x_i has to be stored separately. It is also important to do this when x_i is inactive and its bounds are $u_i = -l_i = \infty$, as in the last line of the table.

An alternative strategy that can be followed in the optimality test at levels 2 and above is to compare (minus) the slope of the steepest edge direction with the norm of the reduced gradient vector. If the latter is greater, then the code can immediately return to level 1. This could happen in the above example (when $r_1 = -18$), but the overall outcome is much the same. I have actually implemented the strategy described in the table.

5. Rounding error in linearly constrained optimization. The description of Wolfe's method in the previous sections has assumed that calculations are carried out in exact arithmetic. This is rarely possible in practice. When rounding errors are made, it is very unlikely that degeneracy (α^c is exactly zero) will be recognized in practice. Thus we might expect to see cycling as a cause of failure to solve an LP. A more difficult situation to address is when there are positive values of r_q and w_q that are both instances of zeros having been corrupted by round-off error. Because the true value of w_q is zero, the matrix B may become structurally singular or at least pathologically ill-conditioned.

With regard to errors in the r_i, my preference is to fix a tolerance, say, τ, and truncate to zero any residuals for which $|r_i| \leq \tau$. The user is able to control the scaling of the constraints, so a suitable tolerance is relatively easy to determine. I have found a default value of 10^{-12} works well. The effect of this truncation is to create degenerate constraints, but this situation can readily be resolved via Wolfe's method. It has often been observed that degeneracy is a feature of many LP and QP problems, in which case the decision to truncate small values of r_i to zero is probably correct.

With regard to the w_i, a possible way to proceed is again to truncate small values to zero. But in this case a suitable value for the tolerance is not easy to determine. I have found a very effective approach is to use a "thick pencil line search" (or ratio test). The idea is, I believe, due to Harris [15] in the paper describing the Devex LP code. The idea is to smear out the boundary of the feasible region, as if by drawing it with a thick pencil. Then any candidates that fall into this thick pencil region are considered for choice in the ratio test.

My interpretation of this in the context of an ASM is to use the same tolerance τ from Wolfe's method to define the "thick pencil" region. Thus

$$q = \arg \min_{\substack{i \in \mathcal{N}^c \\ w_i > 0}} \ ((r_i + \tau)/w_i) \tag{8}$$

defines the choice of constraint. Then $\alpha = r_q/w_q$ is chosen as before. Including τ in the test biases the choice to favor indices with large w_i. A possible consequence is that when the residuals are updated, some may become negative, but it is easily shown that these updated values are greater than $-\tau$. Since these negative residuals are subsequently replaced by zero, no spurious errors greater than τ in absolute value are introduced.

My experience with this idea has been entirely positive. To a large extent, it solves the round-off difficulties due to near-zero denominators, and to degeneracy, in one simple process.

6. An ASM for L1QP problems. Another interesting generic optimization problem with linear constraints is the L1QP problem. This refers to a situation derived from a QP (6) in which the general constraints are included in the objective function as a penalty term (an L_1 sum of infeasibilities), rather than being enforced explicitly. A weighting parameter ν is present to control the relative weight of the quadratic function and the penalty term. An important application of the L1QP problem is as a subproblem in the SL1QP method for NLP (Fletcher [6]). This is an L_∞ trust region method in which a trust region constraint imposes simple lower and upper bounds on the variables. Therefore, it is important for the L1QP formulation to allow the user to impose simple bounds on the variables explicitly, rather than through the L_1 penalty. Thus the formulation that we choose to solve is

$$\begin{aligned} &\text{minimize} && \phi(\mathbf{x}) = \nu q(\mathbf{x}) + \sum_{i=n+1}^{n+m} \big(\max(0, \mathbf{a}_i^T\mathbf{x} - u_i) + \max(0, l_i - \mathbf{a}_i^T\mathbf{x})\big) \\ &\text{subject to} && l_i \leq x_i \leq u_i, \qquad i = 1, 2, \ldots, n, \end{aligned} \tag{9}$$

with the same notation as (6), and where $q(\mathbf{x})$ is a quadratic function. We refer to the constraints in the L_1 term as *soft constraints*.

First-order KKT conditions for a solution to (9) (e.g., [7]) are

$$
\begin{aligned}
&\nu\nabla q(\mathbf{x}) = \begin{bmatrix} I & A \end{bmatrix}\boldsymbol{\lambda},\\
&l_i \le x_i \le u_i, \qquad i = 1, 2, \ldots, n,\\
&\left.\begin{array}{ll} \lambda_i \ge 0 & \text{if } x_i = l_i < u_i\\ \lambda_i \le 0 & \text{if } x_i = u_i > l_i\\ \lambda_i = 0 & \text{if } l_i < x_i < u_i \end{array}\right\} \quad i = 1, 2, \ldots, n,\\
&-1 \le \lambda_i \le 1, \qquad i = n+1, n+2, \ldots, n+m,\\
&\left.\begin{array}{ll} \lambda_i = 1 & \text{if } \mathbf{a}_i^T\mathbf{x} < l_i\\ \lambda_i = -1 & \text{if } \mathbf{a}_i^T\mathbf{x} > u_i\\ \lambda_i = 0 & \text{if } l_i < \mathbf{a}_i^T\mathbf{x}_i < u_i \end{array}\right\} \quad i = n+1, n+2, \ldots, n+m.
\end{aligned} \tag{10}
$$

We now consider the development of an ASM for the L1QP problem. We assume that any current point $\mathbf{x}^c$ is feasible with respect to the simple bounds. This is a trivial matter for a cold start, but in a warm start a Phase 1 calculation might be required. However, degeneracy is not a problem here because the simple bounds are linearly independent. At $\mathbf{x}^c$ we define sets $\mathcal{A}^c$, $\mathcal{W}^c$, $\mathcal{F}^c$, and $\mathcal{N}^c$ exactly as before (a soft constraint is active if its residual is zero). The working set $\mathcal{W}^c$ may include simple bound equations, which correspond to fixed variables that play no part in the calculation but may be useful for modeling. Otherwise, constraints in $\mathcal{W}^c$ are either simple bounds from inequalities, or soft constraints from either equations or inequalities. Degeneracy is said to hold at $\mathbf{x}^c$ if the set $\mathcal{A}^c\backslash\mathcal{W}^c$ is nonempty. However, we shall first describe the algorithm under the assumption that degeneracy does not arise. We define residuals r_i, $i \in \mathcal{N}^c$ as before, but choose the sign such that $r_i > 0$ if $l_i < \mathbf{a}_i^T\mathbf{x}_i < u_i$ (feasible) and $r_i < 0$ if either $\mathbf{a}_i^T\mathbf{x} < l_i$ or $\mathbf{a}_i^T\mathbf{x} > u_i$ (infeasible). If a soft constraint in $\mathcal{N}^c$ is infeasible it is assigned a multiplier ± 1 as appropriate (see (10)), and the sign of its entry in **ls** is used to indicate which option applies. If a soft constraint in $\mathcal{N}^c$ is feasible, its multiplier is zero. By virtue of nondegeneracy, $\phi(\mathbf{x})$ is differentiable at $\mathbf{x}^c$ and its gradient is

$$\mathbf{g}^c = \nu\nabla q(\mathbf{x}^c) - \sum_{i\in\mathcal{N}^c} \lambda_i \mathbf{a}_i. \tag{11}$$

At the start of an ASM iteration from $\mathbf{x}^c$, the vector $\boldsymbol{\lambda}$ defined by

$$B\boldsymbol{\lambda} = \mathbf{g}^c \tag{12}$$

is calculated. As for QP, components of $\boldsymbol{\lambda}$ corresponding to free variables are reduced gradients, and components corresponding to $\mathcal{W}^c$ constraints are Lagrange multiplier estimates. If the reduced gradients are zero (to within some tolerance) and the Lagrange multipliers of working set constraints satisfy the KKT conditions (10), then $\mathbf{x}^c$ is a KKT point and the ASM iterations terminate.

Otherwise, as for QP, either an edge direction or a null space direction may be chosen. Table 2 shows conditions under which an edge descent direction exists for a constraint $i \in \mathcal{W}^c$. For example, if i refers to a lower bound inequality constraint (hard or soft), and if $\lambda_i < 0$, then the direction $\mathbf{s}_i$ (the solution of $B^T\mathbf{s}_i = \mathbf{e}_i$) is a feasible descent direction, just as for QP. But if i refers to a soft constraint, then we are allowed to make the constraint infeasible if ϕ can be reduced. Thus if $\lambda_i > 1$, the direction

$-\mathbf{s}_i$ is a descent direction because the slope in this direction is $-\mathbf{s}_i^T\mathbf{g}(\mathbf{x}) = 1 - \lambda_i < 0$. That is, the contribution $-\lambda_i$ from the current gradient $\mathbf{g}^c$ outweighs the increase in slope of 1 due to constraint i becoming infeasible along $-\mathbf{s}_i$. As for QP, true slopes are calculated using edge weights, and the slope of the steepest edge direction and the (negative) length of the reduced gradient in the null space are compared. The most negative is then chosen.

TABLE 2
Conditions on a Lagrange multiplier λ_i that allow an edge descent direction.

	Inequality		Equation
	lower bound	upper bound	
$i \le n$	< 0	> 0	none
$i > n$	< 0 or > 1	> 0 or < -1	> 1 or < -1

A search is now made along the line $\mathbf{x}(\alpha) = \mathbf{x}^c + \alpha\mathbf{s}^c$ and α^c is chosen to be some value of $\alpha \ge 0$ that retains feasibility with respect to the simple bounds. The function $\phi(\mathbf{x}(\alpha))$ is a piecewise quadratic function, and the values of α at which the slope is discontinuous are referred to as *knots*. Possible strategies (subject to feasibility) are to choose α^c as the first knot, or the best knot, or the value determined by the unconstrained minimization method. Then $\mathbf{x}$, $\mathbf{g}$, and the residuals are updated, and the computation of $\mathbf{g}^{c+1}$ must allow for changed multipliers of the $\mathcal{N}^{c+1}$ constraints. Finally, the exchange of columns in B takes place as for the QP case.

We now address the issue of resolving a degeneracy block in this ASM. As for QP, this arises when the set $\mathcal{A}^c \backslash \mathcal{W}^c$ is nonempty, $\mathbf{s}^c$ is an edge direction, and a step $\alpha^c = 0$ is chosen in the line search. The possibility of adapting Wolfe's method has not previously been considered to the best of my knowledge, and raises some nontrivial issues.

An important issue is the extent to which degenerate soft constraints in $\mathcal{N}^c$ contribute to the last term in (11). For example, if we consider a soft inequality constraint $i \in \mathcal{N}^c$ that is close to its lower bound, then on the infeasible side of the discontinuity it contributes a term $-\mathbf{a}_i$ to the gradient $\mathbf{g}^c$, whereas on the feasible side the contribution is zero. If the residual is exactly zero, then the KKT conditions allow the multiplier to take any value between 0 and 1, but we need only allow one of the extreme values, either 0 or 1, to be taken. Similarly, for a soft equation, the contribution to $\mathbf{g}^c$ is $+\mathbf{a}_i$ on one side of the discontinuity, and $-\mathbf{a}_i$ on the other. If the soft equation is degenerate, we require the multiplier to take either of the extreme values $+1$ or -1.

To keep track of which degenerate soft constraints contribute a $\pm\mathbf{a}_i$ term to $\mathbf{g}^c$, without significantly complicating the data structures, we introduce an extremely small number $\epsilon > 0$, in the sense that it will be rounded off to zero if it takes part in any actual computation. Then the residual r_i of a degenerate inequality constraint in $\mathcal{N}^c$ takes the value $-\epsilon$ if it contributes to $\mathbf{g}^c$ and 0 if not. The residual of a degenerate equation always takes the value $-\epsilon$, and the sign of its entry in **ls** indicates which alternative applies. Essentially, we have made a negligible perturbation to the data so that for any $i \in \mathcal{N}^c$ we have $r_i < 0$ if and only if there is a contribution $\pm\mathbf{a}_i$ to $\mathbf{g}^c$. If a soft constraint with $r_q < 0$ becomes active, the contribution to $\mathbf{g}^c$ is removed when q enters the working set. We continue truncating near-zero updated values of r_i to either 0 or $-\epsilon$ as appropriate. Of course, because $\tau \gg \epsilon$, we are still able to distinguish between degenerate and nondegenerate constraints.

As for QP we only use recursion when $\mathbf{s}^c$ is an edge direction and a constraint q has become active in the line search. We proceed in a similar way to Figure 1 by ignoring inactive constraints and setting the pointer $lp(2)$. The degenerate constraint residuals are perturbed to $+1$ if zero, and to -1 if $-\epsilon$, thus removing the degeneracy block. We now solve the level 2 problem by much the same method as for level 1, possibly involving further recursion. Any return to lower levels of recursion involves resetting negative residuals back to $-\epsilon$ and positive residuals back to zero. If the problem at a level $l > 1$ is solved, then the code returns to level 1. If the level l problem is unbounded, then the code returns to level $l-1$ where a nonzero step can now be taken. The optimality test at higher levels is the same as for level 1. If, as in the QP case, the option of choosing a null space direction is allowed, then the code returns to level 1 in this case.

7. An illustrative L_1 example. In this section an example of a pure L_1 problem with three variables and six general constraints is given to illustrate additional features of Wolfe's method in the context of the previous section. The objective function $q(\mathbf{x})$ for (9) is zero and the coefficient matrix is

$$A = \begin{bmatrix} 0 & 1 & 0 & -2 & 1 & -1 \\ -1 & 2 & -1 & 1 & 2 & 1 \\ -1 & -1 & -2 & -2 & 0 & -2 \end{bmatrix}.$$

The upper bounds $\mathbf{u}$ are all at infinity, and the lower bounds are $l_1 = l_2 = l_3 = 0$ and $l_{3+i} = b_i$ for $i = 1, 2, \ldots, 6$ where

$$\mathbf{b}^T = \begin{pmatrix} 4 & 1 & 4 & -2 & 3 & -1 \end{pmatrix}.$$

Thus the simple bounds $\mathbf{x} \geq \mathbf{0}$ are hard constraints, and $A^T\mathbf{x} \geq \mathbf{b}$ are soft constraints. As in section 3 we use $\mathbf{a}_i,\ i = 1, \ldots, 9$ to denote columns of the matrix $[I\ A]$. The calculations are carried out by choosing the first knot in the line search.

The progress of the method is set out in Table 3, in the same format as in Table 1. The initial point is $\mathbf{x}^1 = \mathbf{0}$, $\mathcal{W}^1 = \{1, 2, 3\}$, and B is the 3×3 unit matrix. Initially, $\mathbf{r} = -\mathbf{b}$ and we see as in line 1 of Table 3 that constraints 4, 5, 6, and 8 are infeasible. Hence the initial gradient vector in (11) is

$$\mathbf{g}^1 = -\mathbf{a}_4 - \mathbf{a}_5 - \mathbf{a}_6 - \mathbf{a}_8 = (-2, -2, 4)^T.$$

Thus constraint 1 is chosen to be relaxed, and the resulting denominators are shown in Table 3. The initial steplength is $\alpha^1 = 1$ and constraints 5, 7, and 9 all become zero. Because of the way the thick pencil line search works, constraint 7 is chosen to be exchanged with constraint 1 in the working set. Because constraint 5 is initially infeasible, and contributes to $\mathbf{g}^1$, its zero residual is stored as $-\epsilon$.

The next iteration chooses constraint 2 to be relaxed, and a degeneracy block in r_5 is seen to arise. Thus recursion to level 2 takes place, and r_5 and r_9 are reset to -1 and 1, respectively. A regular iteration now takes place with $\alpha^c = 0.4$, and constraint 5 is exchanged with constraint 2. Because constraint 5 is initially infeasible, its contribution to $\mathbf{g}^c$ is removed. In the next iteration, at level 2 we see that the problem is *unbounded*. Thus a return to level 1 is made, r_2 and r_9 are reset to zero, and the inactive residuals are brought back into play. No changes to $\mathbf{x}^c$ have been made on the level 2 iterations.

TABLE 3
Progress of Wolfe's method on an L_1 problem.

Level	Working set constraints						Degenerate or inactive constraints									
1	1	−2.	2	−2.	3	4.	4	−4.	5	−1.	6	−4.	7	2.	8	−3.	9	1.
								0.		−1.		0.		2.		−1.		1.
1	7	1.	2	−3.	3	6.	4	−4.	5	$-\epsilon$	6	−4.	1	1.	8	−2.	9	0.
								1.		−2.5		1.		−0.5		−2.5		−0.5
2	7	1.	2	−3.	3	7.	5	−1.	9	1.								
								−2.5		−0.5								
2	7	0.4	5	−0.2	3	3.6	2	0.4	9	1.2								
								−0.4		−0.2								
1	7	0.4	5	−0.2	3	3.6	2	0.	9	0.	6	−4.	1	1.	8	−2.	4	−4.
								−0.4		−0.2		0.4		-0.2		−1.		0.4
1	7	0.4	8	0.8	3	3.8	2	0.8	9	0.4	6	−4.8	1	1.4	5	2.	4	−4.8

Now a regular iteration at level 1 takes place, $\alpha^c = 2$, and constraint 8 becomes active and is exchanged with constraint 5. Because constraint 8 is initially infeasible, its contribution to $\mathbf{g}^c$ is removed. The calculated multipliers are now seen to be optimal, and the solution is $\mathbf{x} = (1.4, 0.8, 0)^T$. The vector of multipliers, including those of the inactive constraints, is

$$\boldsymbol{\lambda} = (0 \quad 0 \quad 3.8 \quad 1 \quad 0 \quad 1 \quad 0.4 \quad 0.8 \quad 0)^T,$$

and the first-order conditions (10) are satisfied.

If $x_3 \geq 0$ were a soft constraint and not a hard constraint, its multiplier value of 3.8 would not be optimal, and it would be possible to reduce the L_1 sum of infeasibilities by allowing x_3 to become negative.

8. Conclusions. It has been shown how Wolfe's method may be adapted to resolve degeneracy in active set methods for a variety of linearly constrained optimization problems. The methods have been implemented in Fortran 77 codes `bqpd` and `qlcpd` for quadratic programming, `glcpd` for general linear constraint programming, and `ql1lcpd` for L1QP problems. `qlcpd` and `glcpd` are available through the COIN-OR depository.

Both referees, whose contributions have been very much appreciated, ask about numerical evidence vis-a-vis other established methods. Now the housekeeping costs of implementing a degeneracy scheme are negligible relative to other computations taking place in an ASM, so there is nothing to say here. One might ask whether or not Wolfe's method provides a better sequence of pivots than other methods. But this issue is affected by other factors that have a much greater effect, such as whether or not steepest edge pivots are used, and what type of null space steps are involved. (Although, since Wolfe's method is based on using virtual perturbations, it is likely to behave very similarly to methods based on actual perturbations.) Thus the main question at issue (in regard to degeneracy) is not one of speed of convergence or number of iterations, but of reliability. Therefore, we must ask: does Wolfe's method succeed in avoiding cycling and also slow convergence due to *stalling*, which in LP I take to mean returning to a previous active set, but with the cycle being broken due to round-off effects?

Clearly, Wolfe's method is proven to succeed in exact arithmetic, and I have argued that the heuristics described in section 5 are simple to implement and are

effective in controlling rounding error. I think that the strategy of *creating* degeneracy by truncating very small residuals to zero, and then using Wolfe's method, is easier to make reliable than one of introducing actual perturbations to remove degeneracy. The `bqpd` code has been in regular use for twenty years or more and no difficulties due to degeneracy have ever become apparent. It has also worked well in situations where degeneracy is a common occurrence, such as in branch-and-bound SQP-based codes for MINLP. The `glcpd` code is called numerous times as a subprogram in the NLP solver [11] and again no difficulties due to degeneracy have been experienced. Of course, other established codes can also claim to be successful in regard to how well they control the effects of degeneracy and rounding error, and are backed up by many years of experience. I do not dispute this, nor do I wish to suggest that the use of Wolfe's method automatically makes a code more reliable.

What I would say for Wolfe's method is that in my opinion it provides a very simple framework within which to resolve degeneracy, is easy to implement, and there are no awkward heuristics to decide. I think that it deserves to be better known, particularly the applicability to QP and L1QP. I would like to think that anyone developing a new active set solver would give it serious consideration.

With regard to the number of levels of recursion, most problems tend not to go beyond level 2 or 3. The largest level I have actually monitored by tracing every iteration is an LP with 1632 variables and 912 general constraints arising from the relaxation of a QAP problem, for which a level of 11 was recorded. On one occasion when `glcpd` was used as a subroutine in the solution of large NLP problems, there was one instance in which an upper bound of 20 on L was insufficient. I know of no occasions where an upper bound of 50 has been insufficient.

REFERENCES

[1] E. M. L. Beale, *Cycling in the dual simplex algorithm*, Naval Res. Logist. Quart., 2 (1955), pp. 269–275.

[2] M. L. Balinski and R. E. Gomory, *A mutual primal-dual simplex method*, in Recent Advances in Mathematical Programming, R. L. Graves and P. Wolfe, eds., McGraw-Hill, New York, 1963, pp. 17–26.

[3] R. G. Bland, *New finite pivoting rules for the simplex method*, Math. Oper. Res., 2 (1977), pp. 103–107.

[4] A. Charnes, *Optimality and degeneracy in linear programming*, Econometrica, 20 (1952), pp. 160–170.

[5] G. B. Dantzig, A. Orden, and P. Wolfe, *The generalized simplex method for minimizing a linear form under linear inequality restraints*, Pacific J. Math., 5 (1955), pp. 183–195.

[6] R. Fletcher, *Numerical experiments with an L_1 exact penalty function*, in Nonlinear Programming 4, O. L. Mangasarian, R. R. Meyer, and S. M. Robinson, eds., Academic Press, New York, 1981, pp. 99–129.

[7] R. Fletcher, *Practical Methods of Optimization*, 2nd ed., John Wiley & Sons, Chichester, UK, 1987.

[8] R. Fletcher, *Degeneracy in the presence of roundoff errors*, Linear Algebra Appl., 106 (1988), pp. 149–183.

[9] R. Fletcher, *Resolving degeneracy in quadratic programming*, Ann. Oper. Res., 47 (1993), pp. 307–334.

[10] R. Fletcher, *A limited memory steepest descent method*, Math. Program., 135 (2012), pp. 413–436.

[11] R. Fletcher, *A sequential linear constraint programming algorithm for NLP*, SIAM J. Optim., 2012, pp. 772–794.

[12] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright, *A practical anti-cycling procedure for linearly constrained optimization*, Math. Programming, 45 (1989), pp. 437–474.

[13] D. Goldfarb and J. K. Reid, *A practicable steepest-edge simplex algorithm*, Math. Programming, 12 (1977), pp. 361–371.
[14] J. A. J. Hall and K. I. M. McKinnon, *The simplest examples where the simplex method cycles and conditions where EXPAND fails to prevent cycling*, Math. Program., 100 (2004), pp. 133–150.
[15] P. M. J. Harris, *Pivot selection methods of the Devex LP code*, Math. Programming, 5 (1973), pp. 1–28.
[16] P. Wolfe, *A technique for resolving degeneracy in linear programming*, J. Soc. Indust. Appl. Math., 11 (1963), pp. 205–211.