4

Systems
Optimization
Laboratory

A Practical Anti-Cycling Procedure for Linear and Nonlinear Programming

by
Philip E. Gill, Walter Murray
Michael A. Saunders and Margaret H. Wright

TECHNICAL REPORT SOL 88-4

July 1988

Department of Operations Research
Stanford University
Stanford, CA 94305

4

SYSTEMS OPTIMIZATION LABORATORY
DEPARTMENT OF OPERATIONS RESEARCH
STANFORD UNIVERSITY
STANFORD, CALIFORNIA 94305-4022

A Practical Anti-Cycling Procedure for Linear and Nonlinear Programming

by

Philip E. Gill, Walter Murray
Michael A. Saunders and Margaret H. Wright

TECHNICAL REPORT SOL 88-4

July 1988

Research and reproduction of this report were partially supported by the U.S. Department of Energy Grant DE-FG03-87ER25030; National Science Foundation Grants CCR-8413211 and ECS-8715153; Office of Naval Research Contract N00014-87-K-0142, and Air Force Office of Scientific Research Grant AFOSR 87-0196.

A PRACTICAL ANTI-CYCLING PROCEDURE FOR LINEAR AND NONLINEAR PROGRAMMING

Philip E. GILL, Walter MURRAY,
Michael A. SAUNDERS and Margaret H. WRIGHT

Systems Optimization Laboratory, Department of Operations Research
Stanford University, Stanford, California 94305-4022, USA

Technical Report SOL 88-4[*]
July 1988

Abstract

A new method is given for preventing the simplex method from cycling. Key features are that a positive step is taken at every iteration, and nonbasic variables are allowed to be slightly infeasible. There is no additional work per iteration. Computational results are given for the first 53 test problems in *netlib*, indicating reliable performance in all cases.

The method may be applied to active-set methods for solving nonlinear programs with linear constraints. Keywords: [illegible] [illegible] -XPAND (Expanding Tolerance Into [illegible]) [illegible]

Keywords: Linear programming, simplex method, degeneracy, cycling)

1. Introduction

Degeneracy is often regarded as a discomforting but otherwise tolerable hindrance to the simplex method, and to other active-set algorithms for solving optimization problems involving linear constraints. Sequences of non-improving steps are known to occur (perhaps many times during a run), but such sequences are rarely observed to be infinite. The phenomenon of "stalling" is therefore recognized and accepted, but "cycling" is deemed very unlikely to occur.

In spite of such folklore, a rigorous anti-cycling procedure can provide welcome peace of mind to users and implementors alike, particularly if the cost is small. Such a procedure was given by Wolfe [Wol63], and the possible benefits have been demonstrated recently by Ryan and Osborne [RO86]. (See also Falkner and Ryan [FR87].) Our aim here is to present an alternative anti-cycling procedure that, like Wolfe's method, involves little overhead and has proved to be effective in practice. We also investigate the relationship with Wolfe's method.

[*]The material contained in this report is based upon research supported by the Air Force Office of Scientific Research Grant 87-01962; the U.S. Department of Energy Grant DE-FG03-87ER25030; National Science Foundation Grants CCR-8413211 and ECS-8715153; and the Office of Naval Research Contract N00014-87-K-0142.

1.1. The standard LP problem

Most of our discussion will be in terms of the simplex method [Dan63] and the primal linear programming problem

$$\begin{array}{lll} \text{minimize} & c^T x & \\ \text{subject to} & Ax = b, & l \le x \le u, \end{array} \tag{1}$$

where $A \in \Re^{m \times n}$ $(m \le n)$.[1] Let $Ax \equiv Bx_B + Nx_N$ denote the partitioning of A and x into basic and nonbasic variables, where $B \in \Re^{m \times m}$ is the usual basis matrix.

In typical implementations of the simplex method, the general constraints $Ax = b$ are satisfied throughout and infeasibility occurs only with respect to the bounds. For most iterations, factors of the current B are obtained by updating, but at the beginning of a run and periodically thereafter, B is factorized directly and the basic variables are recomputed to satisfy $Bx_B + Nx_N = b$. If the newly computed x_B does not satisfy its upper and lower bounds to within some *feasibility tolerance* $\delta > 0$, Phase 1 of the simplex method is invoked to move the infeasible variables towards their violated bounds. Phase 2 starts (or resumes) when the feasibility tolerance is satisfied.

A similar *optimality tolerance* is used to judge whether any reduced costs are sufficiently positive or negative to give an improved solution. Note that the feasibility and optimality tolerances are typically of order 10^{-6}, which is much larger than a typical machine precision ϵ $(\approx 10^{-16})$.

In practice this means that "optimal" solutions are in reality *feasible* and *near-optimal* solutions to the perturbed problem

$$\begin{array}{llll} \text{minimize} & c^T x & & \\ \text{subject to} & Ax = b, & l_B - \delta e \le x_B \le u_B + \delta e, & \\ & & l_N \le x_N \le u_N, & \end{array} \tag{2}$$

where e is a vector of ones, and B and N relate to the final basis obtained.

For the anti-cycling procedure developed here, we find it convenient to allow *nonbasic* variables to be infeasible in a similar way. Thus, in place of (1) and (2), we aim to find a feasible and near-optimal solution to the perturbed problem

$$\begin{array}{lll} \text{minimize} & c^T x & \\ \text{subject to} & Ax = b, & l - \delta e \le x \le u + \delta e. \end{array} \tag{3}$$

Since the final B-N partition is somewhat unpredictable, we anticipate that practitioners accustomed to (2) will find problem (3) essentially equivalent. In practice, few (if any) nonbasic variables will terminate infeasible.

In terms of conventional error analysis, the constraints of (3) can always be satisfied numerically, as long as δ is sufficiently large compared to machine precision.

[1]Implicitly, A and x are of the form $A = (\;\bar{A}\;\; I\;)$, $x = (\;\bar{x}\;\; s\;)$, where s is a set of slack variables with appropriate bounds in l and u. However, we never need to distinguish between $\bar{x}$ and s.

(This is true even if (1) has no feasible solution.) Note however that we don't rigorously "use up" all the freedom allowed by δ to optimize the objective function; i.e., if a solution appears to be optimal we don't try to improve the objective at the expense of making additional variables slightly infeasible. Instead, we are content to terminate knowing that *some* basic or nonbasic variables may lie up to δ outside their true bounds.

1.2. Key aspects

Two main features are involved in the new anti-cycling procedure:

- The feasibility tolerance δ is increased slightly at the start of every iteration.
- Numerical values are stored for all components of x.

The first feature allows a positive step to be taken at every iteration. The second allows slight infeasibilities to be recorded correctly when basic variables become nonbasic. We shall speak of the "EXPAND" procedure[2] when referring to this particular refinement of the simplex method.

The EXPAND procedure is a practical *row-selection* method, in the sense that it chooses a pivot row in the simplex method. We are able to retain the "maximum pivot" property of the row-selection method due to Harris [Har73]. In addition, we are able to remove an unforeseen weakness in previous implementations of the Harris procedure.

1.3. Other anti-cycling methods

Wolfe's method [Wol63] and the lexicographic rule [DOW55,Dan63] are both row-selection procedures. As with the EXPAND procedure, these methods can be used with any column-selection ("pricing") strategy. In particular, they allow "partial pricing"—an important advantage when $n \gg m$.

Recently, a new *column*-selection method has been given by Dantzig [Dan88] and further developed by Klotz [Klo88]. It may be used with any row-selection method. An additional "pricing" vector is required and the method is not directly amenable to partial pricing. However, it appears to be promising for highly degenerate problems.

The first anti-cycling method of Bland [Bla77] prescribes both the pivot column and the pivot row, so again does not allow partial pricing.

In contrast, Benichou *et al.* [BGHR77] perturb the vector b and then apply a "normal" simplex procedure. If the perturbation is chosen randomly, the probability of cycling is zero. The EXPAND procedure is somewhat akin to this approach, particularly when the perturbation is removed (Section 4.4). However, we do not rely on random numbers, and we recommend a considerably smaller perturbation than the $O(10^{-3})$ referred to in [BGHR77].

The primal-dual methods of Balinski and Gomory [BG63], Graves [Gra65] and Fletcher [Fle85] involve a nested sequence of subproblems similar to those arising in Wolfe's method.

[2]EXPanding-tolerance ANti-Degeneracy procedure

2. Nonbasic Solutions

Historically, the simplex method has been implemented in such a way that numerical values are recorded for the *basic* variables only. The value of each nonbasic variable is typically *implied* (by a status indicator) to be the variable's lower or upper bound. Often the bounds are implicitly zero and infinity ($0 \leq x_j \leq \infty$ for all j) and nonbasic variables are implicitly zero.

In more general implementations, some or all variables x_j are allowed to have arbitrary bounds l_j and u_j, and the value of a nonbasic variable is normally defined to be one of the bounds; thus, $x_j = l_j$ or u_j according to a status indicator. A complication arises with "free variables" (satisfying $-\infty \leq x_j \leq +\infty$), since provision must be made for them to be nonbasic even though they are likely to be basic at a solution. Typically, a nonbasic free variable is defined to have the value zero, thereby avoiding the need to store any other value. (This approach was used in various versions of MINOS up to and including MINOS 5.0 [MS83].)

History aside, many implementors have recognized that certain benefits arise if an arbitrary value can be stored for each and every variable. For example, in mathematical programming systems such as MPSX/370 and MPS III, the BASIC procedure is designed to input numerical values for any number of variables and produce from them a basic solution. Further examples are the "pseudo-constraints" of Fletcher and Jackson [FJ74], the "temporary constraints" of Gill and Murray [GM78], and the "pegged variables" of Nazareth [Naz86,Naz87]. The essential idea is that *nonbasic variables can be temporarily frozen at specified values.*

Thus in linear and nonlinear programming, while the term *nonbasic* is often taken to mean "equal to zero" or "equal to an upper or lower bound", it is more useful to define a nonbasic variable as one that is *currently fixed at a specified value.* The working-set strategy involved will eventually allow such a variable to move as if it were being released from a normal bound.

This definition was adopted in MINOS 5.1 [MS87]. Explicit values are stored for all variables, and at each iteration of the simplex and reduced-gradient algorithms, nonbasic variables are allowed to take any strictly feasible value:

$$l_j \leq x_j \leq u_j. \tag{4}$$

An advantage during cold starts is that variables can be initialized at the "safe" value of zero in cases where a user has specified deceptively large bounds, such as $l_j = -10^8$, $u_j = +10^8$. (There is no need to initialize x_j at one of its bounds.) Similar advantages arise when restarting modified problems and recovering from singular bases.

For the anti-cycling procedure of this paper, we have generalized (4) by allowing nonbasic variables to be slightly outside their bounds:

$$l_j - \delta \leq x_j \leq u_j + \delta, \tag{5}$$

where δ is the feasibility tolerance mentioned earlier. This also eliminates a difficulty with the Harris-type steplength procedure, as we now describe.

3. Steplength Procedures

In general, optimization algorithms proceed by generating a search direction p and then changing the variables according to $x \leftarrow x + \alpha p$ for some steplength α ($\alpha \geq 0$).

In the primal simplex method, x always satisfies the constraints $Ax = b$. The "pricing" or "column-selection" strategy chooses a nonbasic variable to be moved from its current value. (This variable usually enters the basis.) A search direction p is then determined, such that $A(x + \alpha p) = b$ for any step α.

The subsequent steplength computation is known as the "ratio test" or the "row-selection" procedure. The aim is to find which variable is the first to encounter a bound. (This variable usually leaves the basis.)

3.1. The standard ratio test

A "textbook" ratio test assumes that the current point x is feasible ($l \leq x \leq u$) and finds the largest step α that keeps the new point feasible: $l \leq x + \alpha p \leq u$. Some blocking variable x_r reaches one of its bounds exactly, so that $x_r + \alpha p_r = l_r$ or u_r depending on the sign of p_r. For each j, let α_j be the step that takes x_j to one of its bounds:

$$\alpha_j = \begin{cases} (l_j - x_j)/p_j & p_j < 0, \\ (u_j - x_j)/p_j & p_j > 0, \\ \infty & \text{otherwise.} \end{cases} \tag{6}$$

Further, let $\alpha_r = \min_j \alpha_j$. The maximum feasible step is then $\alpha = \alpha_r \geq 0$, and the blocking variable is x_r.

Given x, p, l and u, we see that the ratio test determines a steplength α and an index r. We shall refer to this procedure by writing

$$(\alpha, r) = \mathit{ratio\_test}(x, p, l, u).$$

A danger with the textbook ratio test is that the pivot element p_r could be very small if x_r is close to its bound. (If $p_r/\|p\|$ is small, the next basis matrix will be ill-conditioned.) To provide a nominal safeguard, we define a "cut-off" value below which small elements p_j are treated as zero in (6). (In MINOS, "small" means $|p_j| \leq \mathit{tolp}$ where $\mathit{tolp} = \epsilon^{2/3}$ for linear programs and $\epsilon^{2/3}\|p\|$ for nonlinear programs, with $\epsilon^{2/3} \approx 10^{-11}$. Some implications are discussed in Section 5.1.)

3.2. The Harris ratio test

In [Har73], Harris observed that some freedom in choosing r can often be introduced by using a *two-pass* procedure. The first pass determines a perturbed steplength $\alpha 1$ that is slightly too large to keep x feasible:

$$(\alpha 1, r1) = \mathit{ratio\_test}(x, p, l - \delta e, u + \delta e),$$

where δ is the usual feasibility tolerance ($\approx 10^{-6}$). It is important to note that x is assumed to satisfy the perturbed bounds $(l - \delta e, u + \delta e)$.[3] It follows that $\alpha 1 \geq 0$.

[3]In Phase 1 of the simplex method, some components of l and u are altered to $\pm\infty$ (perhaps implicitly) to make this true; see Section 7.

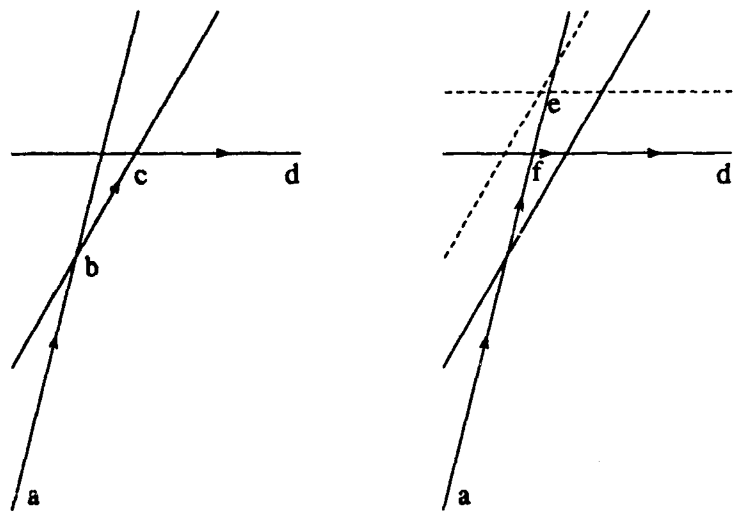

Figure 1: Paths followed with standard (a–b- c–d) and Harris (a–f–d) ratio tests.

The second pass then considers all unperturbed steps α_j (6) no larger than $\alpha 1$, choosing the index associated with the largest pivot:

$$|p_r| = \max_j |p_j| \quad \text{such that} \quad \alpha_j \le \alpha 1.$$

We define the corresponding steplength to be $\alpha 2 = \alpha_r$. The step $\alpha = \alpha 2$ is then acceptable as long as it is positive. In general, we define $\alpha = \max\{\alpha 2, 0\}$.

3.3. An example with $\alpha 2$ positive

Let $(l, u) = (0, \infty)$ and suppose the current feasible solution is $x = (0.0009, 1)^T$. The Harris ratio test with $\delta = 10^{-3}$ would then lead to a step $x \leftarrow x + \alpha p$ of the form

$$\begin{pmatrix} -0.0001 \\ 0 \end{pmatrix} \leftarrow \begin{pmatrix} 0.0009 \\ 1 \end{pmatrix} + \alpha \begin{pmatrix} -0.1 \\ -100 \end{pmatrix}.$$

The steplength is positive ($\alpha = \alpha 2 = 0.01$, $r = 2$). This is slightly too large to keep x feasible, but the larger pivot is successfully chosen.

Figure 1 illustrates a similar case. The standard ratio test would lead to the path (a–b–c–d); the solution would stay strictly feasible, but the constraint encountered at b corresponds to a rather small pivot element. By allowing this constraint to be slightly violated, the Harris ratio test would choose a less oblique constraint to be encountered at f, giving the shorter and numerically more reliable path (a–f–d).

3.4. An unexpected error

Note that $\alpha 2$ will be *negative* if the blocking variable lies slightly outside its bound and p is leading it further from the same bound. (For example, x_1 above is now -0.0001 and p_1 *could be negative at the next iteration.)*

The natural inclination is to set $\alpha = 0$ and interpret this as a zero or "degenerate" step. Note however that *the blocking variable becomes nonbasic.* In most implementations of the simplex method, this means that the blocking variable is actually *changed* (perhaps implicitly) to lie exactly on its bound.

In effect, if the blocking variable is slightly infeasible (say $l_r - \delta \le x_r < l_r$), then most implementations of the Harris procedure change the variables according to

$$x \leftarrow x + \alpha e_r, \tag{7}$$

where $|\alpha| \le \delta$. Such a change produces an *unintentional error in satisfying $Ax = b$.* The error may be of order δ, which is typically much larger than ϵ.

In practice, errors of this kind tend to be eliminated each time the basis is refactorized, since the basic variables are typically recomputed in order to satisfy $Ax = b$ accurately.[4] Provision is made to return to Phase 1 if the recomputed variables lie outside their bounds by more than δ. On well-behaved problems, few iterations (if any) are required to regain feasibility, but in runs lasting thousands of iterations, the risk of a few extra iterations every 50 (a typical factorization frequency) amounts to a nontrivial overhead. In the worst case, "few" can be more than 50 and an optimum may not be achieved.

For nonlinear problems, the perturbation to x in (7) can cause a discontinuity in the nonlinear functions and may lead to a failure in the linesearch procedure.

3.5. A simple cure

To avoid this difficulty with the Harris ratio test, it would be sufficient to implement a "zero" step literally. If a blocking variable is slightly infeasible, we should make it nonbasic and *retain its infeasible value rather than moving it onto its bound.* The variable should be temporarily frozen at that value (across basis factorizations if necessary) until the normal pricing strategy allows it to move. Provision should still be made to revert to Phase 1 after refactorization, but given a stable basis-handling package, the likelihood of losing feasibility will be greatly reduced.

An alternative is to allow a *negative step* whenever $\alpha 2 < 0$, giving the blocking variable a chance to move exactly onto its bound. This approach has been used in the quadratic programming and linear least-squares codes QPSOL 3.2 and LSSOL 1.0 [GMSW84,GHM*86]. However, *it is then necessary to perform a ratio test on* the reverse search direction $-p$, obtaining a possibly *different* blocking variable that again may be unable to reach its bound exactly. Since the objective value will move slightly in the wrong direction, there is also the possibility of cycling.

We propose a further alternative next.

[4]Solve $Bx_B = b - Nx_N$, or preferably solve $By = b - Ax$ and update $x_B \leftarrow x_B + y$.

4. An Anti-cycling Procedure

One way to prevent cycling of the classical kind is to ensure that zero or negative steps never occur. In the proposed procedure we insist that $\alpha > 0$, so that the objective function *always improves.*

Given a feasibility tolerance δ, suppose as before that the current x is feasible to within that tolerance:

$$l - \delta e \leq x \leq u + \delta e. \tag{8}$$

Now suppose that δ is changed to a slightly larger tolerance $\bar{\delta}$ as follows:

$$\bar{\delta} = \delta + \tau, \quad \text{where} \quad 0 < \tau \ll \delta. \tag{9}$$

Since $\bar{\delta} > \delta$, we have

$$l - \bar{\delta} e < x < u + \bar{\delta} e, \tag{10}$$

and it is clearly possible to take a positive step *in any direction p* before encountering a perturbed bound. To find an acceptable positive step, we apply the Harris ratio test in the normal way. If the resulting step $\alpha 2$ is negative or too small, we replace it by a certain step $\alpha_{\min}$ as follows.

EXPAND procedure:

1. (First pass) Define a "largest allowable step" $\alpha 1$ *using the increased feasibility tolerance* $\bar{\delta}$:
$$(\alpha 1, r1) = \mathit{ratio\_test}(x, p, l - \bar{\delta} e, u + \bar{\delta} e).$$

2. (Second pass) Find the step $\alpha 2$ and index r associated with the largest allowable pivot:
$$\alpha 2 = \alpha_r \quad \text{where} \quad |p_r| = \max_j |p_j| \quad \text{such that} \quad \alpha_j \leq \alpha 1.$$
(The quantities α_j are defined by equation (6).)

3. Define a "smallest allowable step" $\alpha_{\min} = \tau/|p_r|$.

4. If $\alpha 2 \geq \alpha_{\min}$, set $\alpha = \alpha 2$. (When x changes to $x + \alpha p$, this step allows the blocking variable x_r to reach its bound exactly.)

5. Otherwise, set $\alpha = \alpha_{\min}$. (In this case, the new value of x_r will "overshoot" its bound, but its infeasibility will be no greater than $\bar{\delta}$.)

The first pass gives a positive step $\alpha 1$, and from (8)–(10) it is clear that

$$\alpha 1 \geq \tau/|p_{r1}|.$$

Since the second pass maximizes the pivot element, we then have

$$0 < \alpha_{\min} \equiv \tau/|p_r| \leq \tau/|p_{r1}| \leq \alpha 1,$$

and it follows that $\alpha_{\min}$ is both positive and not too large. It is preferable to take the step $\alpha 2$ whenever possible (to allow x_r to reach its bound), but if $\alpha 2$ is negative or too small, the step $\alpha_{\min}$ is acceptable.

To summarize, we define $\alpha = \max\{\alpha 2, \alpha_{\min}\}$ and $\bar{x} = x + \alpha p$. We have shown that a positive step is taken ($\alpha_{\min} \leq \alpha \leq \alpha 1$) and that the new point satisfies the required bounds:

$$l - \bar{\delta} e \leq \bar{x} \leq u + \bar{\delta}. \tag{11}$$

Since (11) is analogous to (8), the process can be repeated once the feasibility tolerance is (again) increased as in (9).

4.1. Infeasible nonbasic values

Let Δ be the distance between the blocking variable and the corresponding bound: $\Delta = |x_r - l_r|$ or $|x_r - u_r|$.

When $\Delta \geq \tau$, the preferred step $\alpha = \alpha 2$ is taken and the blocking variable reaches its bound exactly. This is the normal "nondegenerate" case.

If $\Delta < \tau$, the step $\alpha = \alpha_{\min}$ is taken and the blocking variable moves a total distance of τ and terminates *infeasible*. The final value of $\bar{x}_r$ must be recorded when the blocking variable is made nonbasic.

Figure 2 illustrates a normal step and two examples of a degenerate step. We assume the blocking variable x_r is being constrained by its lower bound (so $p_r < 0$). The sloping lines plot the value of $x_r + \alpha p_r$ against α, with three possible starting values for x_r. The lower bound is at the horizontal α axis.

In the top case, x_r is relatively large initially and reaches its bound after a reasonably large step. We take the normal Harris step $\alpha = \alpha 2 > \alpha_{\min}$.

In the middle case, x_r starts out feasible but reaches its bound after a very small step. We insist on taking a larger step $\alpha = \alpha_{\min}$, and the variable becomes slightly infeasible. We count this as a "degenerate" step, even though a positive move is made.

In the third case, x_r is already infeasible and becomes even more infeasible. However, after a step $\alpha = \alpha_{\min}$ it still satisfies the required bound $x_r \geq l_r - \bar{\delta}$. Again we count this as a degenerate step.

The interpretation of "a degenerate step" is that "a slight infeasibility has just been created among the nonbasic variables. The objective function has improved in compensation". Since it is common for blocking variables to reenter the basis at some later iteration, the total number of nonbasic infeasibilities at any stage is generally less than the number of degenerate steps so far.

4.2. Typical parameter values

The preceding sections have discussed the steplength computation for one iteration of the simplex method. Various parameters are involved in a complete implementation, as listed below. We indicate specific values that might be used in practice on a machine with about 16 digits of precision.

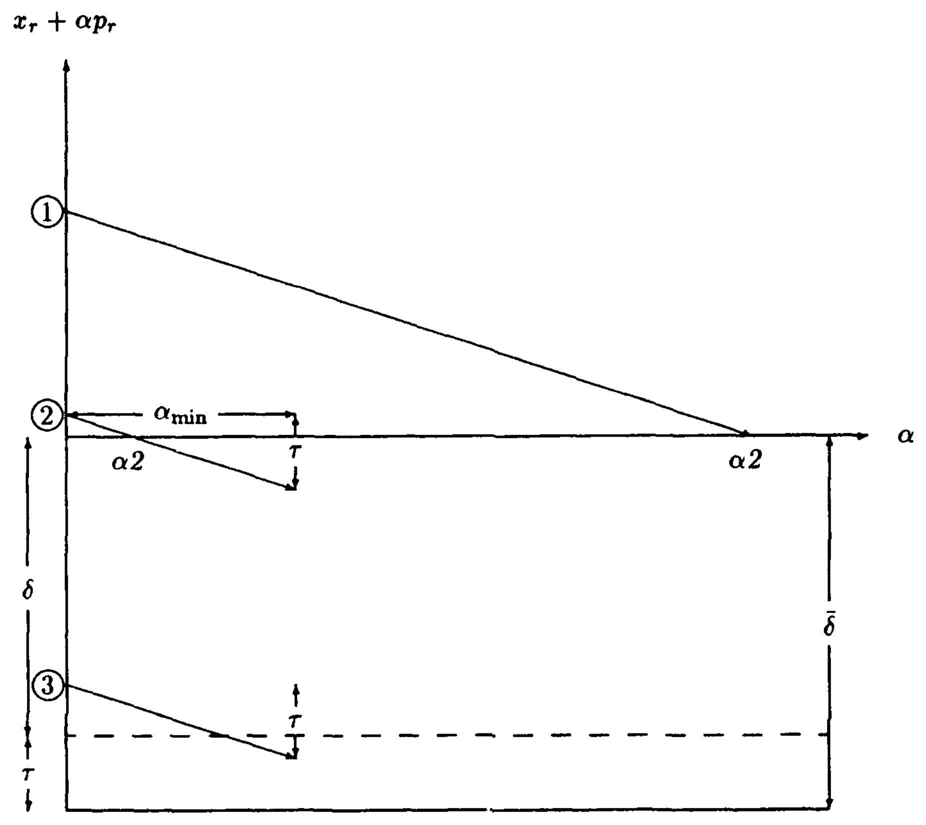

Figure 2: Three possible starting values for the blocking variable x_r.

$\delta =$ 10^{-6} is the "main" feasibility tolerance.

$\delta_0 =$ 0.5δ is the feasibility tolerance used at the start of a cycle of iterations.

$\delta_K =$ 0.99δ is the feasibility tolerance reached at the end of a cycle of iterations.

$K =$ 10000 is the number of iterations in a cycle.

$\tau =$ $(\delta_K - \delta_0)/K \approx \frac{1}{2}10^{-10}$ is the amount by which the feasibility tolerance is incremented each iteration.

$\delta_k =$ $\delta_{k-1} + \tau$ is the feasibility tolerance used at the k-th iteration of a cycle ($k = 1$ to K).

In the present implementation, the "main" tolerance δ may be set by default or specified by the user. Note that δ_0 and δ_K are both similar to δ. The aim is to keep the "current" feasibility tolerance δ_k much the same as the one intended by the user, but to increase it steadily from δ_0 to δ_K over a rather long cycle of consecutive iterations.

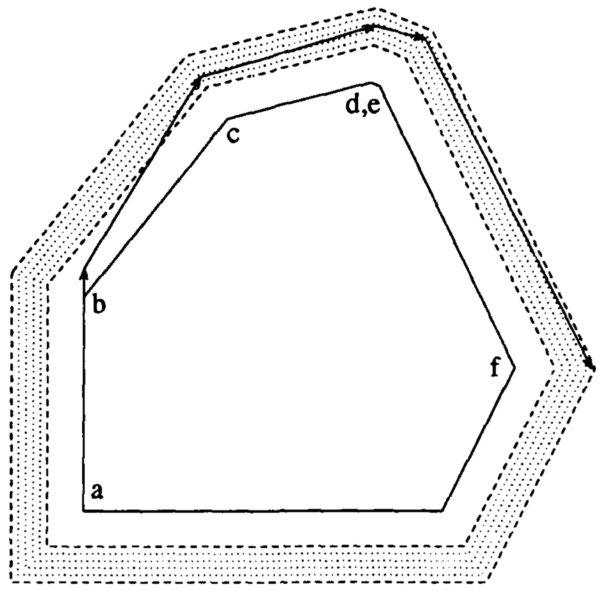

Figure 3: Expansion of feasible region as δ_k increases from δ_0 to δ_K.

4.3. Illustration

Figure 3 depicts a feasible region that expands between the two dashed lines as δ_k increases from δ_0 to δ_K ($K = 5$). The first feasible point is at vertex a, and the step towards vertex b will not go beyond the first dotted boundary.

If the feasibility tolerance were *zero*, the simplex method would follow the path (a–b–c–d–e–f). With a positive tolerance, the step (c–d) would be lengthened and a slight infeasibility would arise temporarily, as illustrated in Figure 1. Vertices b, c and f would be reached exactly.

The path indicated by arrows might be taken if the feasible region were defined by certain additional hyperplanes (not shown). These hyperplanes would be in higher dimensions and would have to cause near-degeneracy at each of the *expanded* vertices corresponding to b, c, d and e. The main idea is that, even if an iterate lies on or close to a degenerate vertex of the current feasible region (i.e., close to one of the dotted lines), a forward move will always be possible within the next feasible region (defined by the next dotted line).

4.4. A resetting procedure

At certain stages we require a "resetting procedure" to remove nonbasic infeasibilities. The main steps are as follows.

1. The values of nonbasic variables are scanned. Any that lie within δ of a bound are moved exactly onto the bound. (This will include variables that were

slightly infeasible when they were last removed from the basis.) A count is kept of the number of nontrivial adjustments made—say, those greater than 10^{-10}.

2. If the count is positive, the basic variables are recomputed from the remaining variables, thereby satisfying $Ax = b$ to (essentially) machine precision.

3. The current feasibility tolerance is reinitialized to δ_0.

If a problem requiies more than K iterations, we invoke the resetting procedure and continue with a new cycle of K iterations. (The decision to resume in Phase 1 or Phase 2 is based on δ_1.)

We also invoke the resetting procedure when the optimizer reaches an apparently optimal, infeasible or unbounded solution, unless this situation has already occurred R times, where R is a further parameter. If any nontrivial adjustments are made, iterations are continued. Typically, $R = 1$ would be sensible for apparent optimality, since in most practical cases the optimality test is satisfied (again) immediately after the reset. The final solution is then "conventional" in the sense that no nonbasic variables lie outside a bound. In badly conditioned cases, an arbitrary number of iterations may be needed to regain feasibility and optimality following a reset, and $R = 2$ may be preferable, since the second reset will normally adjust fewer nonbasics and there is still a chance of terminating at a "conventional" solution.

Note that the solution obtained after k iterations in a cycle, or k *iterations after* a reset, is feasible to within δ_k (assuming Phase 1 has terminated). We may regard *all preceding iterations* as a means of reaching such a point, and it is irrelevant that the feasibility tolerance has been adjusted during the process. Since $\delta_k \leq \delta$, it should be acceptable to terminate at such a point if the optimality test is satisfied. We can therefore advocate using $R = 0$ if there is concern over the arbitrary number of iterations that may be required following a reset. In other words, though we *must* reset every K iterations, there is no real need to reset once the optimality criteria are satisfied. The solution will satisfy the constraints of problem (3) with feasibility tolerance equal to the current δ_k.

A similar situation exists in the degeneracy-resolving procedure of Benichou *et al.* [BGHR77, pp. 292–294], in which a perturbation of order δ ($\delta = 10^{-2}$ or 10^{-3}) is added to the right-hand side vector b. Once the perturbed problem has been solved, the perturbation is removed and the dual simplex algorithm is applied (often requiring no furtiier iterations). If δ were reasonably small (say $\delta = 10^{-6}$), one could argue that the solution to the *perturbed* problem be accepted for all practical purposes.

4.5. Convergence

In ou. case, the only question of non-convergence arises with resetting every K iterations. If K were small, the potential loss of ground after each reset could conceivably lead to a classical cycle of period K. Our choice of $K = 10000$ is intended to make the probability of such a cycle negligible.

To emphasize the point, we note that previous implementations of the simplex method have been operating (in effect) with K set to the basis factorization frequency—typically 50 or less. Failures due to cycling have been rare (though not completely absent; for example, see Benichou *et al.* [BGHR77, pp. 292–294]). Various other implementation details were probably contributing factors.

To a large extent, the chance of failure due to resetting depends on cond(B), the condition number of a typical basis matrix. If cond(B) approaches $1/\epsilon \approx 10^{16}$ where ϵ is the machine precision, then *any* algorithm is likely to fail. However, there should be no risk of failure when cond(B) approaches $1/\delta \approx 10^6$. By choosing K large and retaining the values of slightly infeasible nonbasic variables across basis factorizations, we essentially remove all risk.

5. A Simplified Procedure

A preliminary implementation of the EXPAND procedure was used for the experiments conducted by Lustig [Lus87]. This version was simpler and potentially more efficient on nondegenerate problems; we therefore describe it briefly. As before, we assume that the feasibility tolerance has just been increased to $\bar{\delta} = \delta + \tau$.

Simplified EXPAND procedure:

1. (First pass) Obtain $(\alpha 1, r1) = \mathit{ratio\_test}(x, p, l, u)$.

2. Define a "smallest allowable step" $\alpha_{\min} = \tau/|p_{r1}|$.

3. If $\alpha 1 \geq \alpha_{\min}$, set $(\alpha, r) = (\alpha 1, r1)$ and exit.

4. (Second pass) Otherwise, set $(\alpha, r) = \mathit{ratio\_test}(x, p, l - \bar{\delta}e, u + \bar{\delta}e)$.

Ironically, this approach reverses the two passes in the Harris procedure. It has the advantage of terminating frequently after the first pass (which is just the classical ratio test applied to the true problem data). The blocking variable reaches its bound exactly.

If a second pass is required, the blocking variable must be made nonbasic at a slightly infeasible value ($l_r - \bar{\delta}$ or $u_r + \bar{\delta}$).

A possible disadvantage is that the pivot element $|p_r|$ is not maximized within a set of candidates. Nevertheless, the final step satisfies $\alpha \geq \tau/|p_r|$ whether one or two passes are required. This tends to prevent selection of a small pivot element, unless the feasible region is unbounded. We do not expect numerical instability if δ and τ have the recommended values (Section 4.2). No difficulties were encountered in the computational tests.

5.1. The effect of ignoring small elements of p

A crucial requirement of the EXPAND procedures is that all components of x be at least a distance τ inside the current perturbed bounds $(l - \bar{\delta}, u + \bar{\delta})$. If small elements of p are ignored during the computation of α (Section 3.1), there is a slight risk that the required property will not hold for the next iteration.

The risk exists if $\alpha|p_j| > \tau$ for any ignored elements p_j; i.e., if $\alpha > \tau/tolp$. For typical parameter values, this means if $\alpha > 5$. In such cases, once x has been updated to $x + \alpha p$, a suitable precaution would be to test if any components lie outside the bounds $(l - \bar{\delta}, u + \bar{\delta})$. Any that do could be moved onto those bounds.

To date we have not included such a precaution in our implementations. It would be more strongly recommended if the parameters ϵ, δ and τ were substantially different from those assumed here.

An alternative is to ignore fewer elements of p (by reducing *tolp*), since when $\tau \gg \epsilon$ there is essentially no danger of a small pivot being selected even if $tolp = 0$. However, it is common for many elements of p to be very small, and excluding such elements from the ratio test can give significant savings on large problems.

6. Relationship to Wolfe's Procedure

The EXPAND procedure, simplified or otherwise, may be interpreted as a modification of Wolfe's anti-cycling procedure [Wol63], as we will now show. We discuss the case with general lower bounds on x (but no upper bounds).

6.1. Wolfe's procedure

Let LP$_0$ denote the problem to be solved:

$$\text{LP}_0 \qquad \text{minimize} \quad c^T x \qquad \text{subject to} \quad Ax = b, \quad x \ge l.$$

Wolfe's "ad hoc" procedure takes effect when the simplex method encounters a degenerate feasible vertex (say $x = x_0$). The degeneracy structure of x_0 is used to define the following subsidiary linear program:

$$\text{LP}_1 \qquad \text{minimize} \quad c^T x \qquad \text{subject to} \quad Ax = b, \quad x_D \ge l_D - d, \quad x_N \ge l_N,$$

where d is a positive vector, D denotes basic variables that are currently on a bound, and N denotes variables that are currently nonbasic. (Problem LP$_1$ is the same as LP$_0$ except for the bounds on the basic variables. For degenerate variables the bounds have been relaxed, and for the other basic variables they have been removed.)[5]

Clearly, x_0 is a non-degenerate feasible point for the new problem. When the simplex method is applied to LP$_1$, the values obtained for x are not directly relevant to LP$_0$, but the bases generated (and the associated dual variables) have meaning for both problems. Three situations may arise:

[5]Wolfe's procedure can be described using alterations to x and b as well as l, but the concepts are the same.

1. A finite optimum is obtained for LP_1. The basis and dual variables are also optimal for LP_0, and the required solution is $x = x_0$. The procedure may be terminated after x and its bounds are restored to their original values (x_0 and l).

2. LP_1 is found to be unbounded when a certain nonbasic variable is considered for entry into the basis. The same basis and nonbasic variable will produce a feasible descent direction for LP_0. (This is the *direction of recession* described by Osborne [Osb85].) The variables and bounds are again restored to their original values, and iterations continue on the original problem.

3. A degenerate vertex arises (say $x = x_1$). We may again invoke the procedure of defining a subsidiary LP. Since the variable that just entered the basis must have moved away from its bound in order to cause the degeneracy, the degree of degeneracy must be less than before. The procedure can be invoked again, using x_1 to define a new subsidiary problem LP_2.

In general, the procedure may be applied recursively. Starting with $k = 0$, problem LP_k reaches case 1, 2 or 3 in a finite number of iterations (since the objective function for LP_k is monotonically decreasing). Case 3 leads to a new problem LP_{k+1} but can occur only a finite number of times (since the degree of degeneracy is monotonically decreasing).

6.2. Discussion

Wolfe's procedure is appealing for at least two reasons: it uses the simplex method itself to resolve degeneracy, and it can be implemented with a minimum of overhead (at least for the case $l = 0$, $u = \infty$), as shown by Ryan and Osborne [RO86].

Although a degenerate vertex is unlikely to be encountered in LP_k ($k > 0$), particularly if d is defined using random positive numbers, it remains necessary to cope with the possibility. This may be an inconvenience.

Another drawback is the need to decide that degeneracy is present and the need to define the precise set of degenerate constraints. Suppose we have a set of basic variables that are not quite on their bounds. This could include *all* the basic variables. If we do not include some of them in the definition of x_D, it is probable that only a very short step will be taken after the degeneracy has been "resolved", and we may need to invoke the procedure again.

6.3. A modification

Suppose we introduce a parameter δ into the Wolfe procedure, so that instead of $x_D \geq l - d$ we now have $x_D \geq l - \delta d$, where $\delta > 0$. We shall regard d as fixed, but δ remains to be specified.

Note that if there is a unique choice of blocking variable when $\delta = 1$, the same blocking variable will be chosen for *any* positive value of δ. Even if the choice is not unique, providing we use a consistent criterion for choosing among the set of blocking variables, the choice will still be independent of δ. (Either of the EXPAND procedures would be suitable for making the choice.)

We may extend the argument to show that the sequence of bases generated when solving LP_1 is independent of δ.

The significance of this observation is that δ may be chosen *extremely small* (assuming $\|d\|$ is of order 1). But then, if δ is sufficiently small there is no need to define LP_1; we can simply solve the original problem with a tiny modification to some of the bounds.

We emphasize that solving LP_0 with the modified bounds generates the same bases as solving LP_1, and if no further degenerate vertices are encountered, we either determine a feasible descent direction or confirm that x is the required solution.

6.4. A further modification

As in the original Wolfe procedure, the difficulty is that we cannot guarantee that a further degenerate vertex will not occur. We now show how to avoid such an eventuality by judicious choice of d. First note, however, that choosing the elements of d at random is *not* a good strategy from the perspective of preserving well-conditioned bases. Indeed, the best choice is $d = e$, the vector of ones (assuming the problem is well scaled). In this case, the blocking variable corresponds to the largest eligible pivot element in the search direction p.

Such a structured choice for d would seem to increase the probability of a degenerate vertex arising. Observe, however, that once the best possible choice of blocking variable has been defined (say x_r), only d_r need be defined. We are then free to increase d_i, $i \neq r$, since had this been our initial choice of d, the blocking variable would remain the same. Suppose we increase all elements d_i $(i \neq r)$ by τ, where τ is of order one. Such a choice prevents the current vertex from being degenerate in LP_1. Also, in the following iteration the value of the step to the blocking variable is relatively large (of order τ), and hence again results in a good choice for the blocking variable as regards the condition of the basis.

The second iteration fixes $d_{r'}$, say, but we are still free to alter d_i, $i \neq r$, r'. By proceeding in this manner we can prevent the occurrence of degenerate vertices. Note that although d_r has been fixed, if x_r ever becomes basic again, we are able to redefine its bound. This may appear to negate the argument that it would still be the first blocking variable. However, provided *all* d_i are being increased similarly, there is no contradiction.

With δ small and $d = \delta e$ used to modify all bounds (not just those of x_D), the approach just described is equivalent to the EXPAND procedure.

Note that a strict implementation of Wolfe's procedure would preserve x_0 and eventually determine a feasible descent direction p. In the modified procedure, we take a sequence of steps (to $\tilde{x}$ say) before p is determined. We then step along p from $\tilde{x}$ rather than x_0. However, if δ is sufficiently small, $\|x_0 - \tilde{x}\|$ is negligible and the infeasibility incurred (with respect to $x \geq l$) is strictly bounded.

6.5. Summary

We have shown that the EXPAND procedure is closely related to Wolfe's method. Some advantages are as follows:

- There is no need to judge whether or not degeneracy is present, or to specify a set of degenerate variables.
- There is no storage overhead or logical overhead. General bounds on x can be handled without complication.
- There is no numerical or logical information to be preserved across basis factorizations (other than x and the current δ).
- All iterations are "equal", in the sense that there is exactly one steplength determination per iteration. (In Wolfe's method, if LP_k is found to be degenerate, the steplength procedure is effectively repeated at the beginning and end of LP_{k+1}.)

7. Issues Arising in Phase 1

Broadly speaking, Phase 1 of the simplex method is implemented by applying a normal (Phase 2) procedure to a *modified problem*, whose bounds have been altered to make the current solution feasible.

First we need to summarize the main aspects of Phase 1. (See also Orchard-Hays [Orc68] and Beale [Bea70].) Some finer points can then be discussed.

7.1. The Phase-1 bounds and objective

Suppose a variable x_j has true bounds (l_j, u_j). If x_j lies above its upper bound by more than the current feasibility tolerance ($x_j - u_j > \delta_k$), its bounds are treated as (l_j, ∞). Similarly, if its lower bound is not satisfied ($l_j - x_j > \delta_k$), the bounds are taken to be $(-\infty, u_j)$. Otherwise, the true bounds on x_j are retained.

The Phase-1 objective function $\tilde{c}^T x$ is then defined as $\tilde{c}_j = 1$, -1 and 0 respectively. Note that $\tilde{c}$ is redefined every Phase-1 iteration. It is used to compute reduced costs and hence a search direction p in the normal way.

For most Phase-1 iterations, an appropriate step along p can be defined as usual by applying the EXPAND procedure to the data $(x, p, \tilde{l}, \tilde{u})$, where $\tilde{l}$ and $\tilde{u}$ are the modified bounds. We define this step to be α_F, since it keeps variables feasible with respect to any bounds that they currently satisfy.

For some iterations, a special step α_I must be taken; see Section 7.3. (This step allows one or more infeasible variables to become feasible.)

Thus, Phase 1 is essentially the same as Phase 2 except that $\tilde{c}$, $\tilde{l}$ and $\tilde{u}$ are redefined every iteration, and two possible steps are computed rather than one. (The step ultimately taken is $\alpha = \alpha_F$ or α_I, subject to safeguards discussed below.)

To see that progress is guaranteed, observe that the Phase-1 objective decreases as α increases from zero. If any infeasible variables become feasible as α continues to increase, the sum of infeasibilities will decrease at a lower rate (and could even start to increase), but the *number* of infeasibilities will be lower. Convergence is therefore assured.

(A more intricate steplength procedure can be designed to minimize the piecewise linear function $\tilde{c}^T(x + \alpha p)$, where $\tilde{c}$ is regarded as a function of α; for example, see Greenberg [Gre78], Fourer [Fou85]. However, we adopt the simpler approach as it is effective in practice. Both approaches have the desirable property that many infeasibilities can be removed in one iteration.)

7.2. Advantages of increasing δ_k

Note that the EXPAND procedure does allow α to increase from zero. Also, if the final steplength leaves the number of infeasibilities unaltered, the sum of infeasibilities at the start of the next Phase-1 iteration will be lower simply because the tolerance used to measure infeasibility has increased (from δ_k to $\delta_k + \tau$). Thus *for two separate reasons*, either the sum or the number of infeasibilities will decrease after each Phase-1 iteration.

Although τ is typically very small, it is intended to be significantly larger than machine precision ϵ, and preferably larger than the cut-off value *tolp* (Section 3.1). An important benefit is that it *helps mask the rounding error* that is inevitably present when x is updated to $x + \alpha p$. The set of infeasible variables (as measured by δ_k) is therefore guaranteed to stay the same or to diminish.[6] We believe that many "infinite loop" failures of simplex implementations have been due to an inadvertent *oscillation* in the number of infeasibilities when $\tilde{c}$ is redefined each Phase-1 iteration with a constant δ_k. (An example is described by Ogryczak [Ogr87]. Similar examples were encountered with MINOS prior to the present implementation.)

If a problem is still infeasible after K iterations, the feasibility tolerance is reduced from δ_K to δ_0 for the next cycle of iterations. An apparent disadvantage is that the number of infeasibilities may increase by some arbitrary number (say q), and the sum could increase by as much as $q\delta_K$. However, this is normally inconsequential even if q is nearly as large as m. (Here it is important that many infeasibilities can be removed in one Phase-1 iteration.) Similar comments apply when the resetting procedure is invoked at an apparently optimal solution.

7.3. The special Phase-1 step

By construction, the Phase-1 objective causes at least some of the infeasible variables to move towards the feasible region. Let $\tilde{\alpha}_j$ denote the step that allows such a variable to reach its nearest bound exactly (and hence become feasible):

$$\tilde{\alpha}_j = \begin{cases} (l_j - x_j)/p_j & p_j > 0, \\ (u_j - x_j)/p_j & p_j < 0, \\ \infty & \text{otherwise} \end{cases}$$

[6]Fletcher's method for resolving degeneracy also possesses favorable properties in the presence of rounding error; see [Fle85,Fle87].

(*cf.* (6)). The special Phase-1 step referred to above is then $\alpha_I = \tilde{\alpha}_j$ for some index $j \in I$ (where I temporarily denotes the set of infeasible variables).

In general it would be sensible to define

$$\alpha_I = \tilde{\alpha}_s = \max_{j \in I} \tilde{\alpha}_j, \qquad \alpha = \min\{\alpha_I, \alpha_F\}, \tag{12}$$

since if any infeasible variables become feasible as the steplength increases, α_I marks the point at which the *maximum* number become feasible. (Note that α should not exceed α_I because some of the infeasible variables could become *more* infeasible as α increases. Also note that α_F could be infinite.)

If $\alpha_I = \tilde{\alpha}_s$, a danger is that the corresponding pivot element p_s could be arbitrarily small.

Following the philosophy of Harris (Section 3.2), some freedom to maximize the pivot element can be obtained by using a two-pass procedure. The perturbed bounds $l - \delta e$ and $u + \delta e$ are used in the first pass as usual, but this gives a step $\tilde{\alpha}1$ that is slightly too *small.* The second pass then considers all unperturbed steps $\tilde{\alpha}_j$ no smaller than $\tilde{\alpha}1$:

$$|p_s| = \max_j |p_j| \quad \text{such that} \quad \tilde{\alpha}_j \geq \tilde{\alpha}1.$$

This was the method used in Lustig's experiments [Lus87] and in all preceding versions of MINOS. It appears to have performed reliably for many years.

Nevertheless it is evident that $|p_s|$ could still be as small as the cut-off value *tolp*. We have therefore adopted the following safer stategy. In the first pass, we find the largest relevant pivot element:

$$\phi = \max_{j \in I} |p_j|. \tag{13}$$

In the second pass we then find the largest step subject to the pivot element being reasonably close to ϕ:

$$\alpha_I = \tilde{\alpha}_s = \max_{j \in I} \tilde{\alpha}_j \quad \text{such that} \quad |p_j| \geq \gamma\phi \tag{14}$$

for some constant γ, where $0 < \gamma \leq 1$. Experience suggests that the step α_I should be taken whenever possible (to remove the associated infeasible variable from the basis). We therefore define

$$\alpha = \begin{cases} \alpha_I & \alpha_I \leq \alpha 1, \\ \alpha_F & \text{otherwise}, \end{cases} \tag{15}$$

where $\alpha 1$ (from the first pass of the Harris procedure) is slightly larger than α_F. Together, (13)–(15) define the steplength in place of (12). By observation on 53 test problems, the values $\gamma = 0.1$ and $\gamma = 0.01$ seem to impede Phase 1 relative to the unsafeguarded $\gamma = 0$. We have therefore settled on $\gamma = 0.001$.

A final comment: computation of the special step α_I does not depend critically on the feasibility tolerance, and is therefore compatible with the EXPAND procedure.

8. Nonlinear Programs with Linear Constraints

Here we assume that the problem to be solved is the same as in (1), except that the objective function $c^T x$ is replaced by a smooth general function $F(x)$. The problem could therefore be a quadratic program (QP) or a more general linearly constrained (LC) optimization problem.

The environment we have in mind is active-set methods for QP and LC problems (Fletcher [Fle81]; Gill, Murray and Wright [GMW81]).[7] It has been observed by Osborne [Osb85] that Wolfe's anti-cycling procedure generalizes to certain LC algorithms, including the reduced-gradient method of Wolfe [Wol62]. The same is true of the present approach. The following preliminary implementation has been developed for the reduced-gradient algorithm in MINOS.

8.1. A normal iteration

Conceptually, the EXPAND procedure may be applied directly. Thus, at each iteration we increase the feasibility tolerance slightly, and after obtaining a search direction p, we compute a positive step and a blocking index (α, r) as before. We rename this step $\alpha_{\max}$ since it may be preferable to take a shorter step. Any step in the interval $(0, \alpha_{\max}]$ will give a point that is acceptably feasible.

In general, we then perform a *linesearch* to find a step $\tilde{\alpha}$ that approximately minimizes the objective function over the specified interval:

$$\tilde{\alpha} \approx \operatorname{argmin} F(x + \alpha p), \quad \alpha \in (0, \alpha_{\max}].$$

Note that

$$\alpha_{\max} = \max\{\alpha_{\min}, \alpha 2\},$$

where $\alpha 2$ is the step that allows the blocking variable to reach its true bound (see Figure 2). In case 1 of Figure 2, the search would be performed over the "large" interval $(0, \alpha 2]$, while for cases 2 and 3 it would be performed over $(0, \alpha_{\min}]$.

If the linesearch is successful (the objective function is "sufficiently reduced"), there is no danger of cycling and the optimization algorithm proceeds normally. The current point is updated ($x \leftarrow x + \tilde{\alpha} p$), and if the maximum step was chosen, the blocking constraint is added to the working set (i.e., x_r becomes nonbasic).

8.2. Avoiding the linesearch

In practice, it may be inefficient or unwise to attempt a linesearch, since $\alpha_{\max}$ could be very small if degeneracy is present. Even if the linesearch returns the maximum step $\tilde{\alpha} = \alpha_{\max}$, the improvement in objective value may be very slight. More seriously, the "noise level" in $F(x + \alpha p)$ over the interval $(0, \alpha_{\max}]$ may be too great to allow identification of an improved point, and the linesearch will be obliged to "fail".

[7]The EXPAND procedure has been implemented in the 1988 versions of QPSOL and LSSOL.

We therefore make use of the step $\alpha 2$ that allows the blocking variable to reach its bound exactly. In Figure 2, this step is positive for the first two cases, but negative for the third (and therefore not shown).

In cases 1 and 2 ($\alpha 2 > 0$) we always perform the linesearch.

In case 3 ($\alpha 2 \leq 0$), degeneracy is present and we usually take a *zero step*. (We skip the linesearch and make the blocking variable nonbasic.)

The only exception is when case 3 would create a vertex of the feasible region; this is the most likely circumstance under which a zero step could lead to cycling. If $\alpha 2 \leq 0$ and the working set has only one degree of freedom, we attempt to find a positive step by performing a linesearch over the interval $(0, \alpha_{\min}]$.

8.3. Recovery from a linesearch failure

If the linesearch ever fails to find an improved point, we try to determine whether the failure was due to the search interval being too small.

If $\alpha 2 < \alpha_{\min}$ (cases 2 and 3), we force a step to the true bound ($\alpha = \alpha 2$) and update the working set.

Otherwise, we assume that a better search direction is required. We leave the working set unaltered and invoke a series of recovery procedures. (In MINOS, these include re-estimating any unknown components of the objective gradient using central differences, resetting the approximate reduced Hessian, deleting a constraint from the working set, and refactorizing the basis.)

8.4. Nonlinear constraints

Nonlinear programs involving nonlinear constraints are often treated by SQP and SLC methods, involving a sequence of linearly constrained subproblems to which the above anti-degeneracy procedures may be applied.

It is clearly *necessary* that cycling be avoided within the LC subproblems. Whether this is sufficient to prevent the overall NLC algorithm from cycling remains to be investigated.

9. Computational Results

In this section we compare three steplength procedures for the simplex method. For convenience we give them the following names:

SP1: The "textbook" ratio test of Section 3.1.

SP2: The simplified EXPAND procedure of Section 5.

SP3: The maximum-pivot EXPAND procedure of Section 4. (This includes Harris-type tie-breaking and is the preferred method of this paper.)

SP3 has been implemented in GAMS/MINOS (see [BKM88]) and in MINOS 5.3 (May 1988). All three procedures have been implemented in MINOS 5.3 and tested on the first 53 linear programs in the *netlib* collection [Gay85]. The problems were ordered according to the number of nonzero elements as in [Lus87]. The main run-time options specified were

```
PRINT LEVEL                  0
CRASH OPTION                 1
CRASH TOLERANCE            0.1
SCALE OPTION                 2
PARTIAL PRICE               10
EXPAND FREQUENCY         10000
FEASIBILITY TOLERANCE  1.0E-6
```

(the default options for linear problems in MINOS 5.3). The last two options define $K = 10000$ and $\delta = 10^{-6}$ for the EXPAND procedures. The limit on calls to the resetting procedure at an apparent optimum was set to $R = 2$ (Section 4.4).

The CRASH parameters cause MINOS to choose an approximately triangular basis from the columns of $A = (\,\bar{A}\;\; I\,)$. In most cases the scaling option has the effect of making $\|\tilde{x}^*\| = O(1)$, where $\tilde{x}^*$ is the scaled optimal solution.[8] This helps justify $\delta = 10^{-6}$ as a feasibility tolerance for the scaled problem.

Tables 1, 2 and 3 give results for the simplex method using SP1, SP2 and SP3 respectively. The "objective function" values indicate that the final objective was accurate to four or more digits (except for one problem that terminated early). The meaning of "degenerate steps" depends on the method; see below. Solution times are given in CPU seconds; they do not include time for data input or solution output.[9]

Figure 4 plots the "total iterations" in Tables 1 and 2 relative to the iterations in Table 3. Figure 5 compares CPU times similarly.

[8]Exceptions were problems GROW7, GROW15 and GROW22, for which $\|\tilde{x}^*\| = O(10^7)$, $\|x^*\| = O(10^6)$.

[9]Tests were run as batch jobs on a DEC VAXstation II. The operating system was VAX/VMS version 4.5. The compiler was VAX FORTRAN version 4.6 with default options, including code optimization and D_floating arithmetic (relative precision $\epsilon \approx 2.8 \times 10^{-17}$). The memory available kept paging to a minimum.

9.1. The textbook ratio test

SP1 was safeguarded by treating p_j as zero in equation (6), using $tolp = \epsilon^{2/3}$. Since rounding error can cause the steplength to be negative, a further precaution was to set $\alpha = 0$ if the ratio test gave $\alpha \leq 10^{-16}$. ("Degenerate steps" counts the number of times this occurred.) Following conventional practice, blocking variables were set exactly on their bounds when they became nonbasic.

Although it would be reasonably easy to break (near) ties in favor of large $|p_j|$, we chose not to tamper further with the classical procedure; methodical tie-breaking is the province of the Harris and EXPAND procedures.

In the test runs, small pivots slipped through the $\epsilon^{2/3}$ sieve several times on each of five problems (SCTAP2, SCTAP3, SCSD8, PILOTJA and PILOT). In general, these are detected as near-singularities when the *LU* factors of the basis are updated. Refactorization is invoked and some variable x_j is replaced by an appropriate slack variable. Since x_j retains its value when rejected from the basis, iterations continue without apparent interruption.

Only one real failure was encountered: problem SCSD8 was terminated at iteration 5804 after stalling for the final 1000 iterations (far short of optimality). There was no obvious cycle in progress, but small pivots were encountered frequently during the run, causing the basis to be ill-conditioned for many groups of iterations. Empirically, ill-conditioning can only aggravate stalling (particularly for a method that has no guarantee of terminating).

Figure 4 illustrates that on most problems, SP1 led to more simplex iterations than SP2 or SP3.

9.2. The simplified EXPAND procedure

In Table 2, "degenerate steps" means the number of times SP2 required two passes to determine a blocking variable.

No singularities were encountered during the tests, and all problems terminated successfully. The resetting procedure was invoked at 10000 iterations for the last two problems, with 372 and 423 nonbasic variables (respectively) being moved a distance in the range $(\epsilon^{0.8}, \delta_K)$ onto their bounds.

With δ as small as 10^{-6}, resets do not disturb x greatly. After resetting at an apparent optimum, most problems were confirmed optimal with no further iterations. On PILOT4, GANGES, PILOTJA and PILOT, 35, 190, 98 and 28 nonbasic variables were moved onto their bound and 4, 1, 1 and 15 additional iterations were needed to confirm optimality. In the case of GANGES, a second reset moved one nonbasic variable but led to no further iterations.

9.3. The EXPAND procedure

In Table 3, "degenerate steps" means the number of times α was forced to be as large as $\alpha_{\min}$. This is the number of times a blocking variable was made nonbasic at an infeasible value, rather than reaching its bound exactly.

To date, there has been no specific study of the effect of maximizing the pivot element within a steplength procedure—the Harris approach to tie-breaking. Folklore has it that "stability is improved and the number of simplex iterations is often reduced". However, such a statement is not especially meaningful without a precise definition of the procedures being compared.

Here, it is meaningful to compare both versions of the EXPAND procedure because it is understood that in each case the surrounding simplex algorithm increases the feasibility tolerance every iteration and deals correctly with infeasible blocking variables (by retaining their numerical values when they become nonbasic).

The results in Tables 2 and 3 essentially confirm the folklore. Figure 4 illustrates the trend more clearly: the ratio of SP2 to SP3 iterations is mostly greater than one. In Figure 5, the CPU-time ratios are shifted slightly downwards, reflecting the fact that SP2 usually requires only one pass, whereas SP3 always requires two.

Only two problems required additional iterations after resetting. On PILOTJA, 81 nonbasics were moved onto their bound at iteration 6070, and 63 iterations later a second reset moved 7 nonbasics. Termination occurred 8 iterations later (with no attempt to reset).

On PILOT, 127 nonbasics were moved onto their bounds at iteration 10000, and 43 at iteration 17698. Termination occurred after a further 18 iterations. As expected, the effect of resets was slightly less noticeable than with SP2.

9.4. Other parameter values

The 53 test problems have been solved many times, with and without scaling and partial pricing. One of the main parameters of interest is the feasibility tolerance. We have experimented with the values $\delta = 10^{-4}$, 10^{-5}, 10^{-6} and 10^{-7} (Harris recommended $\delta = 5 \times 10^{-4}$), but the sensitivity of the simplex method to minor algorithmic changes seems to have masked any useful trend. Significant improvements were certainly observed on some of the problems with $\delta = 10^{-4}$. The risk is a greater disturbance after resetting on problems that are somewhat ill-conditioned (notably PILOTJA and PILOT).

As a further test, we disabled the "expand" feature of SP3 by specifying $K = \infty$, $\tau = 0$. This has the effect of fixing the feasibility tolerance at $\delta_0 = 0.5 \times 10^{-6}$, and most closely resembles the Harris-type tie-breaking (without losing the feature of making infeasible blocking variables nonbasic at their correct value). No failures occurred on four runs with and without scaling and partial pricing, confirming that the probability of failure with the Harris procedure is indeed low, given the second safeguard. The iteration counts were much the same as when δ was allowed to expand.

Note that once the second safeguard is implemented, the assurance gained by allowing δ to expand comes at no cost. There is no reason to keep δ fixed.

Problem		Objective function		Total itns	Degen steps	Percent degen	Solve time VAX II secs
1	AFIRO	-4.6475314285714E+02		6	3	50.00	0.45
2	ADLITTLE	2.2549496316238E+05		113	20	17.70	5.90
3	SC205	-5.2202061211707E+01		118	14	11.86	13.03
4	SCAGR7	-2.3313897523795E+06		86	10	11.63	6.80
5	SHARE2B	-4.1573224074142E+02		124	48	38.71	8.31
6	RECIPE	-2.6661600000000E+02		33	3	9.09	2.12
7	VTPBASE	1.2983146246136E+05		64	30	46.88	6.39
8	SHARE1B	-7.6589318579186E+04		277	4	1.44	24.68
9	BORE3D	1.3730803942085E+03		207	155	74.88	28.25
10	SCORPION	1.8781248227381E+03		109	41	37.61	21.23
11	CAPRI	2.6900129137682E+03		246	50	20.33	32.86
12	SCAGR25	-1.4753433060769E+07		376	79	21.01	89.30
13	SCTAP1	1.4122500000000E+03		259	123	47.49	38.68
14	BRANDY	1.5185098964881E+03		331	74	22.36	53.65
15	ISRAEL	-8.9664482186305E+05		266	20	7.52	35.55
16	ETAMACRO	-7.5571521755166E+02		470	134	28.51	93.29
17	SCFXM1	1.8416759028349E+04		378	86	22.75	69.71
18	GROW7	-4.7787811814712E+07		179	65	36.31	37.90
19	BANDM	-1.5862801845012E+02		483	80	16.56	100.91
20	E226	-1.8751929066371E+01		580	202	34.83	85.34
21	STANDATA	1.2576995000000E+03		145	109	75.17	23.71
22	SCSD1	8.6666666743334E+00		1084	1000	92.25	89.14
23	GFRDPNC	6.9022359995488E+06		672	350	52.08	187.48
24	BEACONFD	3.3592485807200E+04		116	27	23.28	13.90
25	STAIR	-2.5126695119296E+02		469	63	13.43	231.65
26	SCRS8	9.0429998618888E+02		537	183	34.08	143.88
27	SEBA	1.5711600000000E+04		445	54	12.13	106.98
28	SHELL	1.2088253460000E+09		303	73	24.09	76.90
29	PILOT4	-2.5811392588836E+03		1613	229	14.20	706.74
30	SCFXM2	3.6660261564999E+04		917	201	21.92	326.56
31	SCSD6	5.0500000078262E+01		1476	1137	77.03	214.56
32	GROW15	-1.0687094129358E+08		397	148	37.28	150.31
33	SHIP04S	1.7987147004454E+06		152	34	22.37	36.23
34	FFFFF800	5.5567959102690E+05		938	388	41.36	287.07
35	GANGES	-1.0958596920679E+05		687	214	31.15	358.13
36	SCFXM3	5.4901254549751E+04		1359	303	22.30	724.11
37	SCTAP2	1.7248071428571E+03		785	530	67.52	453.01
38	GROW22	-1.6083433648256E+08		635	254	40.00	371.58
39	SHIP04L	1.7933245379704E+06		266	55	20.68	65.46
40	PILOTWE	-2.7200970057530E+06		5003	838	16.75	3540.04
41	SIERRA	1.5394460531792E+07		1340	825	61.57	704.48
42	SHIP08S	1.9200982105346E+06		242	64	26.45	102.41
43	SCTAP3	1.4240000000000E+03		1151	876	76.11	837.87
44	SHIP12S	1.4892361344061E+06		434	111	25.58	253.79
45	25FV47	5.5018458882865E+03		8687	1100	12.66	5794.72
46	SCSD8	1.318E+03	Stalled	5804	3566	61.44	2787.51
47	NESM	1.4076079386175E+07		3067	0	0.00	1338.22
48	CZPROB	2.1851966988566E+06		1519	151	9.94	788.61
49	PILOTJA	-6.1130625520046E+03		7086	504	7.11	6317.42
50	SHIP08L	1.9090552113891E+06		522	149	28.54	250.04
51	SHIP12L	1.4701879193293E+06		877	252	28.73	594.19
52	80BAU3B	9.8723001910483E+05		10896	1483	13.61	12674.42
53	PILOT	-5.5740380062649E+02		20720	1630	7.87	81173.13

Table 1: Results from MINOS 5.3 using textbook ratio test

	Problem	Objective function	Total itns	Degen steps	Percent degen	Solve time VAX II secs
1	AFIRO	-4.6475314285714E+02	6	3	50.00	0.47
2	ADLITTLE	2.2549496316238E+05	92	12	13.04	4.75
3	SC205	-5.2202061211707E+01	123	13	10.57	14.23
4	SCAGR7	-2.3313897523795E+06	86	10	11.63	6.73
5	SHARE2B	-4.1573224074142E+02	113	17	15.04	7.69
6	RECIPE	-2.6661600000000E+02	33	3	9.09	1.91
7	VTPBASE	1.2983146246136E+05	71	24	33.80	6.97
8	SHARE1B	-7.6589318579186E+04	242	3	1.24	21.56
9	BORE3D	1.3730803942085E+03	165	80	48.48	23.41
10	SCORPION	1.8781248227381E+03	105	41	39.05	20.10
11	CAPRI	2.6900129137682E+03	245	22	8.98	33.21
12	SCAGR25	-1.4753433060769E+07	361	68	18.84	87.47
13	SCTAP1	1.4122500000000E+03	291	70	24.05	42.55
14	BRANDY	1.5185098964881E+03	462	35	7.58	75.15
15	ISRAEL	-8.9664482186305E+05	225	29	12.89	30.13
16	ETAMACRO	-7.5571521718413E+02	570	135	23.68	119.26
17	SCFXM1	1.8416759028349E+04	396	61	15.40	72.28
18	GROW7	-4.7787811814712E+07	174	18	10.34	40.82
19	BANDM	-1.5862801845012E+02	456	25	5.48	98.82
20	E226	-1.8751929066371E+01	494	108	21.86	74.55
21	STANDATA	1.2576995000000E+03	106	46	43.40	18.80
22	SCSD1	8.6666666743334E+00	508	321	63.19	46.96
23	GFRDPNC	6.9022359995488E+06	687	280	40.76	193.26
24	BEACONFD	3.3592485807200E+04	116	21	18.10	13.60
25	STAIR	-2.5126695119296E+02	415	42	10.12	202.79
26	SCRS8	9.0429998618888E+02	609	171	28.08	167.78
27	SEBA	1.5711600000000E+04	463	52	11.23	118.27
28	SHELL	1.2088253460000E+09	304	54	17.76	77.81
29	PILOT4	-2.5811392616949E+03	1870	182	9.73	827.58
30	SCFXM2	3.6660261564999E+04	858	130	15.15	298.76
31	SCSD6	5.0500000078262E+01	1425	586	41.12	215.19
32	GROW15	-1.0687094129358E+08	435	50	11.49	179.69
33	SHIP04S	1.7987147004454E+06	151	27	17.88	36.40
34	FFFFF800	5.5567958085232E+05	1073	353	32.90	341.59
35	GANGES	-1.0958598988428E+05	703	234	33.29	384.19
36	SCFXM3	5.4901254549751E+04	1318	188	14.26	692.73
37	SCTAP2	1.7248071428571E+03	750	392	52.27	376.31
38	GROW22	-1.6083433648256E+08	638	72	11.29	397.37
39	SHIP04L	1.7933245379704E+06	277	47	16.97	67.91
40	PILOTWE	-2.7201032443839E+06	4982	572	11.48	3520.13
41	SIERRA	1.5394390923795E+07	1317	542	41.15	701.36
42	SHIP08S	1.9200982105346E+06	269	61	22.68	117.89
43	SCTAP3	1.4240000000000E+03	1096	624	56.93	736.06
44	SHIP12S	1.4892361344061E+06	431	75	17.40	259.39
45	25FV47	5.5018458882864E+03	7514	440	5.86	5058.45
46	SCSD8	9.0459999992546E+02	4281	1693	39.55	1722.09
47	NESM	1.4076079386175E+07	3067	0	0.00	1338.27
48	CZPROB	2.1851966988566E+06	1497	62	4.14	839.22
49	PILOTJA	-6.1130540560627E+03	7398	547	7.39	6466.02
50	SHIP08L	1.9090552113891E+06	463	69	14.90	236.16
51	SHIP12L	1.4701879193293E+06	852	164	19.25	589.06
52	80BAU3B	9.8722736149636E+05	10769	921	8.55	12652.68
53	PILOT	-5.5746058728842E+02	18441	4005	21.72	75072.32

Table 2: Results from MINOS 5.3 using simplified EXPAND procedure

	Problem	Objective function	Total itns	Degen steps	Percent degen	Solve time VAX II secs
1	AFIRO	-4.6475314285714E+02	6	3	50.00	0.49
2	ADLITTLE	2.2549496316238E+05	92	14	15.22	5.07
3	SC205	-5.2202061211707E+01	117	13	11.11	15.14
4	SCAGR7	-2.3313897523795E+06	86	10	11.63	7.32
5	SHARE2B	-4.1573224074142E+02	104	33	31.73	7.80
6	RECIPE	-2.6661600000000E+02	33	3	9.09	2.20
7	VTPBASE	1.2983146246136E+05	59	17	28.81	6.72
8	SHARE1B	-7.6589318579186E+04	269	3	1.12	25.28
9	BORE3D	1.3730803942085E+03	159	62	38.99	23.82
10	SCORPION	1.8781248227381E+03	103	38	36.89	19.87
11	CAPRI	2.6900129137682E+03	228	33	14.47	32.19
12	SCAGR25	-1.4753433060769E+07	361	74	20.50	91.79
13	SCTAP1	1.4122500000000E+03	242	65	26.86	37.33
14	BRANDY	1.5185098964881E+03	462	57	12.34	78.95
15	ISRAEL	-8.9664482186305E+05	261	17	6.51	38.20
16	ETAMACRO	-7.5571522106445E+02	501	115	22.95	106.96
17	SCFXM1	1.8416759028349E+04	386	66	17.10	72.68
18	GROW7	-4.7787811814712E+07	184	32	17.39	42.67
19	BANDM	-1.5862801845012E+02	487	47	9.65	107.71
20	E226	-1.8751929066371E+01	462	102	22.08	72.75
21	STANDATA	1.2576995000000E+03	97	52	53.61	17.47
22	SCSD1	8.6666666743334E+00	427	225	52.69	38.28
23	GFRDPNC	6.9022359995488E+06	717	337	47.00	206.55
24	BEACONFD	3.3592485807200E+04	116	22	18.97	14.10
25	STAIR	-2.5126695119296E+02	364	40	10.99	190.08
26	SCRS8	9.0429998618888E+02	625	130	20.80	177.86
27	SEBA	1.5711600000000E+04	417	44	10.55	106.56
28	SHELL	1.2088253460000E+09	310	54	17.42	78.57
29	PILOT4	-2.5811392623740E+03	1452	141	9.71	656.83
30	SCFXM2	3.6660261564999E+04	880	138	15.68	319.19
31	SCSD6	5.0500000078262E+01	1099	503	45.77	164.71
32	GROW15	-1.0687094129358E+08	446	65	14.57	194.65
33	SHIP04S	1.7987147004454E+06	149	25	16.78	35.20
34	FFFFF800	5.5567961145338E+05	866	293	33.83	281.97
35	GANGES	-1.0958635746320E+05	679	218	32.11	372.73
36	SCFXM3	5.4901254549751E+04	1184	180	15.20	632.04
37	SCTAP2	1.7248071428571E+03	680	386	56.76	342.76
38	GROW22	-1.6083433648256E+08	643	83	12.91	403.74
39	SHIP04L	1.7933245379704E+06	266	38	14.29	67.03
40	PILOTWE	-2.7201043693969E+06	5267	598	11.35	3850.05
41	SIERRA	1.5394362183632E+07	1266	568	44.87	700.02
42	SHIP08S	1.9200982105346E+06	254	59	23.23	113.50
43	SCTAP3	1.4240000000000E+03	840	502	59.76	570.60
44	SHIP12S	1.4892361344061E+06	445	87	19.55	274.72
45	25FV47	5.5018467790998E+03	8136	837	10.29	5722.41
46	SCSD8	9.0499999992546E+02	2857	1251	43.79	1174.23
47	NESM	1.4076087003981E+07	2853	34	1.19	1296.87
48	CZPROB	2.1851966988566E+06	1503	130	8.65	836.44
49	PILOTJA	-6.1131152948481E+03	6141	422	6.87	5496.13
50	SHIP08L	1.9090552113891E+06	467	67	14.35	244.25
51	SHIP12L	1.4701879193293E+06	891	222	24.92	621.37
52	80BAU3B	9.8723197704930E+05	9693	1068	11.02	11768.52
53	PILOT	-5.5740387782461E+02	17716	1624	9.17	74443.58

Table 3: Results from MINOS 5.3 using maximum-pivot EXPAND procedure

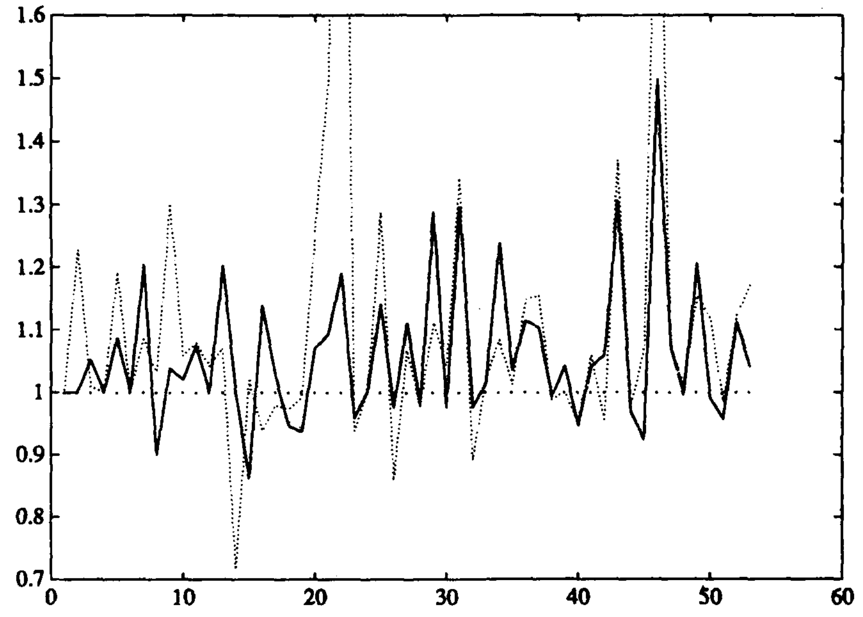

Figure 4: Comparison of iterations required by the simplex method with different steplength procedures. The ratios i_1/i_3 (...) and i_2/i_3 (—) are plotted for 53 test problems, where (i_1, i_2, i_3) are the iterations for the (textbook, simplified EXPAND, maximum-pivot EXPAND) procedures respectively.

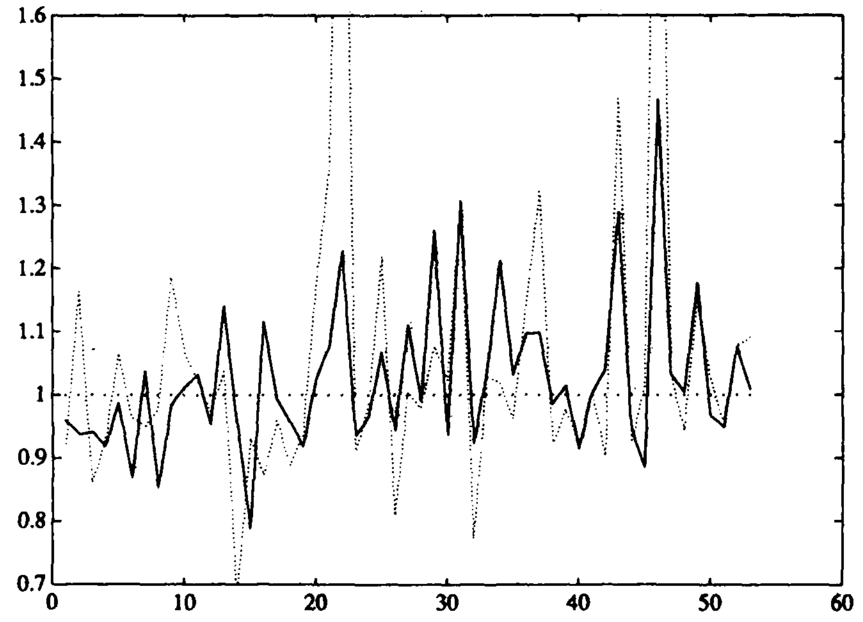

Figure 5: Similar comparison of times required by the simplex method with different steplength procedures.

10. Conclusions

The linear program (1) involves general constraints $Ax = b$ and bounds $l \leq x \leq u$. The simplex method aims to satisfy $Ax = b$ to machine precision, while working towards a solution that satisfies the bounds to some looser tolerance δ. (The opposite is true for certain other iterative methods.)

The EXPAND procedure was developed in response to sporadic failures that occurred during Lustig's experiments with MINOS 5.1 on the same 53 test problems used here [Lus87]. We have not experienced any failures since.

Perhaps the main advance has been in treating the infeasible blocking variables generated by a Harris-type ratio test. By retaining the infeasible values when such variables become nonbasic, we satisfy $Ax = b$ to machine precision throughout. Only *then* can we take correct advantage of satisfying bounds loosely in the manner pioneered by Harris. An important benefit is that there is virtually no reversion to Phase 1 after refactorization—a common occurrence previously on ill-conditioned problems.

The precaution of expanding δ every iteration provides added theoretical assurance of convergence (given the consequent similarity to Wolfe's anti-degeneracy procedure), as well as added practical assurance in the presence of rounding error.

Acknowledgements

The authors would like to express their appreciation to Irvin Lustig for his energetic experimentation during the summer of 1987. The present research, and the mechanism for making numerous batch runs on multiple test problems, are a direct result. We are also grateful to David Gay for making the test problems available through *netlib*.

References

[Bea70] E. M. L. Beale. Advanced algorithmic features for general mathematical programming systems. In J. Abadie, editor, *Integer and Nonlinear Programming*, pages 119–137, North-Holland, The Netherlands, 1970.

[BG63] M. L. Balinski and R. E. Gomory. A mutual primal-dual simplex method. In R. L. Graves and P. Wolfe, editors, *Recent Advances in Mathematical Programming*, McGraw-Hill, New York, 1963.

[BGHR77] M. Benichou, J. M. Gauthier, G. Hentges, and G. Ribière. The efficient solution of large-scale linear programming problems—some algorithmic techniques and computational results. *Mathematical Programming*, 13, 280–322, 1977.

[BKM88] A. Brooke, D. Kendrick, and A. Meeraus. *GAMS: A User's Guide*. The Scientific Press, Redwood City, California, 1988.

[Bla77] R. G. Bland. New finite pivoting rules for the simplex method. *Mathematics of Operations Research*, 2, 103–107, 1977.

[Dan63] G. B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, Princeton, New Jersey, 1963.

[Dan88] G. B. Dantzig. *Making progress during a stall in the simplex algorithm*. Report SOL 88-5, Department of Operations Research, Stanford University, 1988.

[DOW55] G. B. Dantzig, A. Orden, and P. Wolfe. The generalized simplex method for minimizing a linear form under linear inequality constraints. *Pacific J. of Mathematics*, 5, 183–195, 1955.

[FJ74] R. Fletcher and M. P. Jackson. Minimization of a quadratic function of many variables subject only to upper and lower bounds. *J. Institute of Mathematics and its Applications*, 14, 159–174, 1974.

[Fle81] R. Fletcher. *Practical Methods of Optimization. Volume 2: Constrained Optimization*, John Wiley and Sons, Chichester and New York, 1981.

[Fle85] R. Fletcher. *Degeneracy in the presence of round-off errors.* Technical Report NA/89, Department of Mathematical Sciences, University of Dundee, Dundee, Scotland, 1985.

[Fle87] R. Fletcher. Recent developments in linear and quadratic programming. In A. Iserles and M. J. D. Powell, editors, *The State of the Art in Numerical Analysis*, pages 213–243, Oxford University Press, Oxford and New York, 1987.

[Fou85] R. Fourer. A simplex algorithm for piecewise-linear programming I: derivation and proof. *Mathematical Programming*, 33, 204–233, 1985.

[FR8'‚ J. C. Falkner and D. M. Ryan. Aspects of bus crew scheduling using a set partitioning model. Presented at the Fourth International Workshop on Computer-Aided Scheduling of Public Transport, Hamburg, 1987.

[Gay85] D. M. Gay. Electronic mail distribution of linear programming test problems. *Mathematical Programming Society COAL Newsletter*, 13, 10–12, 1985.

[GHM*86] P. E. Gill, S. J. Hammarling, W. Murray, M. A. Saunders, and M. H. Wright. *User's Guide for LSSOL (Version 1.0): a Fortran package for constrained linear least-squares and convex quadratic programming.* Report SOL 86-1, Department of Operations Research, Stanford University, 1986.

[GM78] P. E. Gill and W. Murray. Numerically stable methods for quadratic programming. *Mathematical Programming*, 14, 349–372, 1978.

[GMSW84] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright. *User's Guide for SOL/QPSOL (revised).* Report SOL 84-6, Department of Operations Research, Stanford University, 1984.

[GMW81] P. E. Gill, W. Murray, and M. H. Wright. *Practical Optimization.* Academic Press, London and New York, 1981.

[Gra65] G. W. Graves. A complete constructive algorithm for the general mixed linear programming problem. *Naval Research Logistics Quarterly*, 12, 1–34, 1965.

[Gre78] H. J. Greenberg. Pivot selection tactics. In H. J. Greenberg, editor, *Design and Implementation of Optimization Software*, pages 143–174, Sijthoff and Noordhoff, The Netherlands, 1978.

[Har73] P. M. J. Harris. Pivot selection methods of the Devex LP code. *Mathematical Programming*, 5, 1–28, 1973. Reprinted in *Mathematical Programming Study*, 4, 30–57, 1975.

[Klo88] E. S. Klotz. *Dynamic pricing criteria in linear programming.* PhD thesis, Department of Operations Research, Stanford University, 1988.

[Lus87] I. J. Lustig. *An analysis of an available set of linear programming test problems.* Report SOL 87-11, Department of Operations Research, Stanford University, 1987. To appear in *Computers and Operations Research.*

[MS83] B. A. Murtagh and M. A. Saunders. *MINOS 5.0 User's Guide.* Report SOL 83-20, Department of Operations Research, Stanford University, 1983.

[MS87] B. A. Murtagh and M. A. Saunders. *MINOS 5.1 User's Guide.* Report SOL 83-20R, Department of Operations Research, Stanford University, 1987.

[Naz86] J. L. Nazareth. Implementation aids for the optimization algorithms that solve sequences of linear programs. *ACM Transactions on Mathematical Software*, 12, 307–323, 1986.

[Naz87] J. L. Nazareth. *Computer Solution of Linear Programs*. Oxford University Press, New York and Oxford, 1987.

[Ogr87] W. Ogryczak. On practical stopping rules for the simplex method. *Mathematical Programming Study*, 31, 167–174, 1987.

[Orc68] W. Orchard-Hays. *Advanced Linear-Programming Computing Techniques*. McGraw-Hill, New York, 1968.

[Osb85] M. R. Osborne. *Finite Algorithms in Optimization and Data Analysis*. John Wiley and Sons, Chichester and New York, 1985.

[RO86] D. M. Ryan and M. R. Osborne. On the solution of highly degenerate linear programs. 1986. To appear in *Mathematical Programming*.

[Wol62] P. Wolfe. The reduced-gradient method. 1962. Unpublished manuscript, the RAND Corporation.

[Wol63] P. Wolfe. A technique for resolving degeneracy in linear programming. *SIAM J. Appl. Math.*, 11, 205–211, 1963.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER Technical Report SOL 88-4	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A Practical Anti-Cycling Procedure for Linear and Nonlinear Programming		5. TYPE OF REPORT & PERIOD COVERED Technical Report
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Philip E. Gill, Walter Murray, Michael A. Saunders, and Margaret H. Wright.		8. CONTRACT OR GRANT NUMBER(s) N00014-87-K-0142 AFOSR-87-0196
9. PERFORMING ORGANIZATION NAME AND ADDRESS Department of Operations Research - SOL Stanford University Stanford, CA 94305-4022		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 1111MA
11. CONTROLLING OFFICE NAME AND ADDRESS Office of Naval Research - Dept. of the Navy 800 N. Quincy Street Arlington, VA 22217		12. REPORT DATE July 1988
		13. NUMBER OF PAGES 31 Pages
Air Force Office of Scientific Research/NM Building 410 Bolling Air Force Base Washington, DC 20332		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

This document has been approved for public release and sale; its distribution is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Linear Programming, Simplex Method, Degeneracy, Cycling.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

A new method is given for preventing the simplex method from cycling. Key features are that a positive step is taken at every iteration, and nonbasic variables are allowed to be slightly infeasible. There is no additional work per iteration. Computational results are given for the first 53 test problems in "netlib," indicating reliable performance in all cases.

The method may be applied to active-set methods for solving nonlinear programs with linear constraints.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)