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

A Strictly Improving Phase I Algorithm Using Least-Squares Subproblems

by
S. A. Leichner, G. B. Dantzig and J. W. Davis

TECHNICAL REPORT SOL 92-1

April 1992

Research and reproduction of this report were partially supported by the National Science Foundation Grants ECS-8906260, DMS-8913089; the Department of Energy Grant DE-FG03-92ER25116; Office of Naval Research Grant N00014-89-J-1659; the Electric Power Research Institute Contracts RP 8010-09 at Stanford University, and by Hewlett Packard Laboratories.

Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do NOT necessarily reflect the views of the above sponsors.

A Strictly Improving Phase I Algorithm Using Least-Squares Subproblems

S. A. Leichner G. B. Dantzig J. W. Davis

3 April 1992

Abstract

Although the simplex method's performance in solving linear programming problems is usually quite good, it does not guarantee strict improvement at each iteration on degenerate problems. Instead of trying to recognize and avoid degenerate steps in the simplex method (as some variants do), we have developed a new Phase I algorithm that is completely impervious to degeneracy, with strict improvement attained at each iteration. It is also noted that the new Phase I algorithm is closely related to a number of existing algorithms.

When tested on the 30 smallest *NETLIB* linear programming test problems, the computational results for the new Phase I algorithm were almost 3.5 times faster than the simplex method; on some problems, it was over 10 times faster.

1 Introduction

On highly degenerate problems, the simplex method often "stalls", performing a number of iterations at a degenerate point before producing any improvement in the objective value. Examples have been constructed by Hoffman [11] and Beale [1] to show that it is theoretically possible that the iterative steps can repeat and thus cycle forever, although this phenomenon is quite rare in practice. Instead of trying to make the simplex method more efficient by trying to avoid stalls due to degeneracy (or near degeneracy), we develop a new Phase I algorithm that is completely impervious to degeneracy.

This new method involves the use of least-squares subproblems in column selection, and is shown to have the property of strict improvement at each iteration, even if every basic solution (in the simplex method sense) is degenerate. Like the simplex method, we will show that the new Phase I algorithm terminates in a finite number of steps, and that in practice, this number is often quite low compared to the simplex method.

Our algorithm is quite similar to a number of existing algorithms, including one to solve the bounded least-squares problem (found in [3]), and another to

solve the non-negative least-squares problem (found in [12]). In addition, our algorithm is also closely related the algorithm developed by Dantzig [5] and by Van de Panne & Whinston [14]. R.W. Cottle noted that although this algorithm was designed to solve quadratic programs and thus has a different goal in mind, when applied to an alternate formulation of the Phase I feasibility problem, it is similar to a variant of our least-squares Phase I algorithm.

Section 2 develops most of the theory used by the algorithms presented. It is concerned with finding a feasible solution to a linear program, and after some preliminary results, begins to look at two-variable least-squares problems. The two-variable problem and its positive least-squares solution is used to select the incoming column, and thus build an improved "basic" solution. Conditions are derived under which the solutions to these least-squares problems are strictly positive. Although this is analogous in many ways to the simplex method, it is proved that strict improvement can be guaranteed at each iteration, even in the presence of degeneracy. Next, the detection of infeasibility is discussed, followed by a detailed description of the newly developed least-squares Phase I algorithm, and discussion of a number of variations. Finally, equivalences between our algorithms and the other related algorithms are discussed.

Section 3 presents computational results for the algorithm developed in Section 2. As noted, the least-squares Phase I algorithm has excellent performance, with run times 3.5 times faster than LSSOL's implementation of the simplex method [8]. On some problems, the least-squares Phase I algorithm was over 10 times faster.

Section 4 summarizes the work presented here and attempts to draw some conclusions. Suggestions for future work appear at the end of this section.

2 Finding a Feasible Solution

2.1 The Problem

The problem to be addressed is that of the Phase I problem solved by the simplex method. That is, find a vector x such that

$$\begin{aligned} Ax &= b, \\ x &\geq 0, \end{aligned} \tag{2-1}$$

where $x \in \Re^n$, $b \in \Re^m$, and $A \in \Re^{m \times n}$. Denoting the jth column of A by A_j, we assume $A_j \neq 0 \ \forall \ j$, and that $b \neq 0$, or else $x = 0$ is a trivial solution. Without loss of generality, we also assume that b has been rescaled so that $\| b \|_2 = 1$, and similarly the columns of A have been rescaled so that $\| A_j \|_2 = 1 \ \forall \ j$.

Fact 2.1
Let B be an independent subset of the columns of A, and let x_B satisfy

$$\begin{aligned} Bx_B &= b, \\ x_B &> 0. \end{aligned} \tag{2-2}$$

Then by setting $x_j = 0$ for $A_j \notin B$, it is a trivial matter to construct an x^* from x_B such that x^* is a solution to (2-1). (See [6] for a more detailed treatment of the relationship between (2-1) and (2-2).)

The main objective of the least-squares Phase I algorithm to be presented here is to find such a matrix B (if it exists), and positive weightings x_B, such that B and x_B comprise a solution to (2-2). In the remainder of this discussion, we will refer to this matrix B as a *basis*. Note that this notion of a basis differs from that of the simplex method, as our basis B may contain less than m columns and that these columns may not be sufficient in themselves to span the column space of A.

2.2 Preliminaries

We need to consider least-squares problems with positive solutions before we can present the algorithm. But before we can consider such problems, we need a few more preliminary facts and results. We will be making use of the Euclidean norm, which we will denote by $\| \cdot \|$.

Fact 2.2
Let $b, v, p \in \Re^m$, $b \neq 0$, $p \neq 0$, $v \neq 0$, and $v \neq b$. Let $u = b - v$. If $1 \cdot v$ is closer to b than any other scalar multiple λ, then we can conclude that

(a) $v^T v = b^T v > 0$;

(b) $u^T v = 0$;

(c) $u^T u = b^T u > 0$;

(d) $b^T b = u^T u + v^T v$;

(e) *If $p^T b \leq 0$, then $1 \cdot v$ is closer to b than any scalar multiple μp of p, $\mu \geq 0$;*

(f) *If $p^T b > 0$, then $1 \cdot v$ is closer to b than μp for all $\mu \geq 0$ if and only if $\| v \| > p^T b / \| p \|$;*

(g) *If $p \neq 0$ and $v \neq \mu p$, then $v^T v \, p^T p - (v^T p)^2 > 0$.*

Proof:

(a) The hypotheses imply that $\lambda = 1$ is the solution to

$$\min_{\lambda} \| b - \lambda v \|^2.$$

Therefore

$$\frac{d}{d\lambda}(b-\lambda v)^T(b-\lambda v)=0 \quad \text{for } \lambda=1,$$

and $v^Tv=b^Tv$ follows. We also have $b^Tv>0$ since $v\neq 0$.

(b) Since $u=b-v$ by definition, $u^Tv=(b-v)^Tv=0$ by **(a)**.

(c) It follows that $u^Tu=u^T(b-v)=u^Tb$.

(d) We have $b^Tb=b^T(u+v)=u^Tu+v^Tv$ by **(a)** and **(c)**.

(e) The unconstrained minimum of the problem

$$\min_{\mu}\|b-\mu p\|$$

occurs at $\mu^*=p^Tb/p^Tp$, and if $p^Tb\leq 0$, we have $\mu^*\leq 0$. Because $\|b-\mu p\|^2$ is a convex function of μ with a minimum at μ^*, we can conclude if $p^Tb\leq 0$,

$$\min_{\mu\geq 0}\|b-\mu p\|$$

must occur at $\mu=0$. Thus,

$$\min_{\mu\geq 0}\|b-\mu p\|=\|b\|.$$

Now consider

$$\|b-v\|=b^Tb-v^Tv<\|b\|.$$

If $p^Tb\leq 0$, we have

$$\|b-v\|<\|b\|=\min_{\mu\geq 0}\|b-\mu p\|.$$

(f) From part **(e)**, we know that the minimum of the problem

$$\min_{\mu}\|b-\mu p\|^2$$

occurs at $\mu^*=p^Tb/p^Tp$, and we see that if $p^Tb>0$, we have $\mu^*>0$. Thus,

$$\min_{\mu}\|b-\mu p\|^2=\min_{\mu\geq 0}\|b-\mu p\|^2.$$

If v is closer to b than is μp for all $\mu\geq 0$, then we have

$$\begin{aligned}
\|b-v\|^2 &< \min_{\mu\geq 0}\|b-\mu p\|^2\\
b^Tb-2v^Tb+v^Tv &< \min_{\mu\geq 0}(b^Tb-2\mu p^Tb+\mu^2p^Tp)\\
-2v^Tb+v^Tv &< -2\left(\frac{p^Tb}{p^Tp}\right)p^Tb+\left(\frac{p^Tb}{p^Tp}\right)^2p^Tp\\
-v^Tv &< -\frac{(p^Tb)^2}{p^Tp} \qquad \text{by Fact 2.2, part (a).}
\end{aligned}$$

Therefore, given $p^T b > 0$, we can say that

$$\| v \| > \frac{p^T b}{\| p \|}.$$

Now if instead of assuming $p^T b > 0$, we assume that

$$\| v \| > \frac{p^T b}{\| p \|},$$

we see that the argument reverses exactly. The result follows.

(g) This is a version of the Cauchy-Schwartz inequality. □

Fact 2.3
Given matrix $A \in \Re^{m \times n}$ $(A \neq 0)$, then $A^T A$ has full rank if and only if A has full column rank.

Fact 2.4
Let $\{A_1, A_2, \ldots, A_k\}$ be a set of linearly independent columns. A unique least-squares weighting vector x exists, yielding the unconditional minimum of $F = \| b - Ax \|$. The unique minimum is found by solving the normal equations $A^T A x = A^T b$.

Fact 2.5
Let $\{A_1, A_2, \ldots, A_k\}$ be a set of linearly independent columns. Let x^0 be the unique least-squares solution to $\min \| b - \sum_i A_i x_i \|$, and let $v^0 = \sum_i A_i x_i^0$. Let $x^1 \neq x^0$ be any other weighting of the columns A_i, and let $v^1 = \sum_i A_i x_i^1$. Then $\| b - (1-\lambda)v^1 - \lambda v^0 \|^2$ is monotonically increasing from $\lambda = 1$ to $\lambda = 0$.

Proof:

$$\| b - (1-\lambda)v^1 - \lambda v^0 \|^2 = \| b - v^1 \|^2 - 2\lambda(b - v^1)^T(v^0 - v^1) + \lambda^2 \| v^1 - v^0 \|^2$$

is a strictly convex function, and its minimum occurs at $\lambda = 1$. Thus this quadratic function of λ strictly increases from $\lambda = 1$ to $\lambda = 0$. □

Corollary 2.6
Let $v = (1-\lambda)v^1 + \lambda v^0$. If v^0 is closer to b than is v^1, then $u^T u < u_1^T u_1$, where $u_1 = b - v^1$, and $u = b - v$ for all $0 < \lambda < 1$.

Proof: Let $u_0 = b - v^0$. We know that $u_0 \neq u_1$ because $u_0^T u_0 < u_1^T u_1$. Therefore

$$\| u_1 - u_0 \|^2 = u_1^T u_1 + u_0^T u_0 - 2u_1^T u_0 > 0,$$

and $2u_1^T u_0 < u_1^T u_1 + u_0^T u_0$. Hence,

$$\begin{aligned} u^T u &= \Big(\lambda u_0 + (1-\lambda)u_1\Big)^T \Big(\lambda u_0 + (1-\lambda)u_1\Big) \\ &= \lambda^2 u_0^T u_0 + (1-\lambda)^2 u_1^T u_1 + 2\lambda(1-\lambda)u_1^T u_0 \\ &< \lambda^2 u_0^T u_0 + (1-\lambda)^2 u_1^T u_1 + \lambda(1-\lambda)(u_1^T u_1 + u_0^T u_0) \\ &= (1-\lambda)u_1^T u_1 + \lambda u_0^T u_0 \\ &< (1-\lambda)u_1^T u_1 + \lambda u_1^T u_1 \qquad \text{(because } u_0^T u_0 < u_1^T u_1 \text{)} \\ &= u_1^T u_1. \end{aligned}$$

Thus any nontrivial convex combination of two vectors v^0 and v^1 as shown above is strictly closer to b than whichever vector v^0 or v^1 is farthest away from b. □

Corollary 2.7
Let $A \in \Re^{m\times n}$ be made up of linearly independent columns, and let x^0 be the unique least-squares solution to $\min \| b - Ax \|$. Let $v^0 = Ax^0$. Assume $v^0 \not\geq 0$. If $v^1 > 0$, where v^1 is otherwise arbitrary, then a convex combination v of v^0 and v^1 can be found to generate a $v \geq 0$ closer to b than v^1 for some λ, $0 \leq \lambda < 1$.

Proof: From Fact 2.5, we know that $\| b - (1-\lambda)v^1 - \lambda v^0 \|^2$ is monotonically increasing from $\lambda = 1$ to $\lambda = 0$. Thus any convex combination, $v = (1-\lambda)v^1 + \lambda v^0$, has the property that

$$\| b - v^1 \|^2 > \| b - v \|^2 > \| b - v^0 \|^2 \quad (\lambda \neq 0, \lambda \neq 1)$$

If we want $v \geq 0$ with v closer to b than v^1, start with $\lambda = 1$ and decrease λ until all components of v are ≥ 0. This is possible because $v^1 > 0$. □

Fact 2.8
Given $x > 0$, $y \not\geq 0$, the minimum λ required to make the vector $\lambda x + (1-\lambda)y \geq 0$ is

$$\lambda^* = \max_{y_i < 0} \frac{-y_i}{x_i - y_i}.$$

This concludes the necessary preliminary results.

2.3 Least Squares with Positive Solutions

In this section, we will consider the conditions under which the solution to the (unconstrained) two-variable least-squares problem

$$\min \; F(\alpha, \beta) = \min_{\alpha,\beta} \| b - \alpha v - \beta p \|^2 \tag{2-3}$$

is unique, and the implications of a non-positive solution. Then we will derive conditions under which the solution is strictly positive.

Theorem 2.9
Let $p \neq 0$, $v \neq \mu p$ and $u = b - v$. Let $\min F = F_0$. Then,

(a) *the values of (α_0, β_0) yielding F_0 are unique;*

(b) *$F_0 = u^T u$ if and only if $(\alpha_0, \beta_0) = (1, 0)$.*

Proof:

(a) Noting that the condition $v \neq \mu p$ implies v and p are linearly independent, this follows directly from Fact 2.4.

(b) If $(\alpha_0, \beta_0) = (1, 0)$, then we see that $F_0 = u^T u$. By part **(a)**, we know that the values of (α_0, β_0) yielding F_0 are unique. Therefore, the converse also holds; that is, $F_0 = u^T u = \| b - 1 \cdot v - 0 \cdot p \|^2$ implies $(\alpha_0, \beta_0) = (1, 0)$. □

Corollary 2.10
Let $1 \cdot v$ be closer to b than any other λv. If $F_0 \neq u^T u$, then $F_0 < u^T u$ and $\beta_0 \neq 0$. In addition, if $\min\{\alpha_0, \beta_0\} < 0$, then $F_0 < u^T u$.

Proof: If $F_0 \neq u^T u$, we can conclude that $F_0 < u^T u$, as $F = u^T u$ is obtainable at $(\alpha, \beta) = (1, 0)$.

Now assume that $F_0 < u^T u$ and $\beta_0 = 0$. Thus, $\alpha_0 v + \beta_0 p = \alpha_0 v$. Since v is closer to b than is any λv $(\lambda \neq 1)$, $\alpha_0 v$ cannot be as close to b as is v for $\alpha_0 \neq 1$. Therefore if $\beta_0 = 0$, then $\alpha_0 = 1$. From Theorem 2.9, we know that if $(\alpha_0, \beta_0) = (1, 0)$, then $F = u^T u$, which is a contradiction.

In Theorem 2.9, we showed that $F_0 = u^T u$ if and only if $(\alpha_0, \beta_0) = (1, 0)$. We just showed that if $F_0 \neq u^T u$, then $F_0 < u^T u$. If $\min\{\alpha_0, \beta_0\} < 0$, then $(\alpha_0, \beta_0) \neq (1, 0)$. Thus we can conclude that $F_0 \neq u^T u$, which in turn implies that $F_0 < u^T u$. □

Theorem 2.11
Let $1 \cdot v$ be closer to b than any other λv. Also let $v \neq \mu p$ and let v be closer to b than any μp, $\mu \geq 0$. Let $F_0 = \min F$, and let (α_0, β_0) be the unique solution corresponding to F_0. Let $v_0 = \alpha_0 v + \beta_0 p$. If $\min\{\alpha_0, \beta_0\} < 0$, then there exist no $(\alpha_1, \beta_1) \geq 0$ such that $v_1 = \alpha_1 v + \beta_1 p$ is closer to b than is v.

Proof: Assume $\exists\, (\alpha_1, \beta_1) \geq 0$ such that v_1 is closer to b than is v, i.e., $\| b - v \| > \| b - v_1 \|$.

Case 1: $\alpha_0 < 0$, $\beta_0 \geq 0$.

Let $\tilde{v}$ be a convex combination of v_0 and v_1:

$$\begin{aligned} \tilde{v} &= \lambda v_0 + (1 - \lambda) v_1 \\ &= \Big(\lambda \alpha_0 + (1 - \lambda)\alpha_1\Big) v + \Big(\lambda \beta_0 + (1 - \lambda)\beta_1\Big) p \\ &= \tilde{\alpha} v + \tilde{\beta} p, \end{aligned}$$

where $\tilde{\alpha} = \lambda\alpha_0 + (1-\lambda)\alpha_1$ and $\tilde{\beta} = \lambda\beta_0 + (1-\lambda)\beta_1$.

Since v_0 is formed from the unique solution to $\min F$, we know that v_0 is closer to b than is v_1. Thus by Corollary 2.6, we can say that any convex combination of v_0 and v_1 is at least as close to b as is v_1. That is, $\| b - \tilde{v} \| \leq \| b - v_1 \|$. At some point between v_0 and v_1, $\tilde{\alpha} = 0$, since $\alpha_0 < 0$ and $\alpha_1 \geq 0$. At such a point, we have $\tilde{v} = \tilde{\beta}p$. However, we know that v is closer to b than any βp for $\beta \geq 0$, so v is closer to b than is $\tilde{v}$. That is, $\| b - v \| < \| b - \tilde{v} \|$. We also know that $\tilde{v}$ is at least as close to b as is v_1, i.e., $\| b - \tilde{v} \| \leq \| b - v_1 \|$. Thus we have $\| b - v \| < \| b - v_1 \|$, which contradicts the assumption.

Case 2: $\beta_0 < 0$, no conditions on α_0.

As in Case 1, let $\tilde{v}$ be a convex combination of v_0 and v_1:

$$\begin{aligned} \tilde{v} &= \lambda v_0 + (1-\lambda)v_1 \\ &= \tilde{\alpha}v + \tilde{\beta}p, \end{aligned}$$

where $\tilde{\alpha}$ and $\tilde{\beta}$ are as defined above in Case 1. Also as in Case 1, we can conclude that *any* convex combination of v_0 and v_1 is at least as close to b as is v_1. That is, $\| b - \tilde{v} \| \leq \| b - v_1 \|$. At some point between v_0 and v_1, we have $\tilde{\beta} = 0$. At such a point, $\tilde{v} = \tilde{\alpha}v$. However, we know that v is closer to b than any αv for $\alpha \neq 1$. So v is at least as close to b as is $\tilde{v}$, i.e., $\| b - v \| \leq \| b - \tilde{v} \|$. We also know that $\| b - \tilde{v} \| \leq \| b - v_1 \|$. Thus we have $\| b - v \| \leq \| b - v_1 \|$, which again contradicts the assumption. □

Now we will derive conditions under which the solution (α_0, β_0) to

$$\min F = \min_{\alpha,\beta} \| b - \alpha v - \beta p \|^2$$

is strictly positive.

Theorem 2.12
Let $v \neq \mu p$, $p \neq 0$. Let v be closer to b than any λv, $\lambda \neq 1$ and let $u = b - v$. Let F and (α_0, β_0) be as defined in Theorem 2.11. Then $\beta_0 > 0$ if and only if $p^T u > 0$.

Proof: Solving for β_0 at the minimum of F, we get

$$\beta_0 = \frac{b^T p \, v^T v - b^T v \, p^T v}{p^T p \, v^T v - (p^T v)^2}.$$

Consider the sign of β_0. From Fact 2.2, we know that $p^T p \, v^T v - (p^T v)^2 > 0$, so we need only consider the numerator:

$$\begin{aligned} p^T b \, v^T v - b^T v \, p^T v &= p^T b \, v^T v - p^T v \, v^T v \qquad \text{(by Fact 2.2, part (a))} \\ &= v^T v \, p^T u. \end{aligned}$$

Since $v \neq 0$, we know that $v^T v > 0$. Thus the sign of $p^T u$ determines the sign of the numerator and hence the sign of β_0. The result follows. □

Theorem 2.13
Given the same conditions as in Theorem 2.12 with the addition that v be closer to b than is μp for $\mu \geq 0$, then $\alpha_0 > 0$ if $p^T u > 0$.

Proof: Solving for α_0 at the minimum of F, we get

$$\alpha_0 = \frac{v^T v\, p^T p - (p^T v)^2 - p^T u\, p^T v}{p^T p\, v^T v - (p^T v)^2}.$$

From Fact 2.2, we know that $p^T p\, v^T v - (p^T v)^2 > 0$, so the sign of α_0 is the same as the sign of the numerator $v^T v\, p^T p - (p^T v)^2 - p^T u\, p^T v$.

Case 1: $p^T b > 0$:
From Fact 2.2 we know that $v^T v\, p^T p > (p^T b)^2$. Therefore,

$$\begin{aligned} v^T v\, p^T p - (p^T v)^2 - p^T u\, p^T v &> (p^T b)^2 - (p^T v)^2 - p^T v\, p^T u \\ &= (p^T b)^2 - (p^T v)^2 - p^T v\, p^T b + (p^T v)^2 \\ &= p^T b(p^T b - p^T v) \\ &= p^T b\, p^T u > 0, \end{aligned}$$

since $p^T b > 0$ and $p^T u > 0$. Thus in Case 1, $\alpha_0 > 0$ if $p^T u > 0$.

Case 2: $p^T b \leq 0$:
We know $p^T u > 0$ and thus $p^T b - p^T v > 0$. But $p^T b \leq 0$, so $p^T v$ must be negative. Again we consider the sign of the numerator of α_0. By Fact 2.2, we know $v^T v\, p^T p - (p^T v)^2 > 0$, so consider $-p^T u\, p^T v$. However, we know that $p^T u > 0$ and $p^T v < 0$. Thus $-p^T u\, p^T v > 0$. Thus in Case 2, $\alpha_0 > 0$ if $p^T u > 0$. □

Corollary 2.14
Given the conditions of Theorem 2.13, the unconditional minimum of F is at $(\alpha_0, \beta_0) > 0$ if $u^T p > 0$.

Proof: This follows directly from Theorems 2.12 and 2.13. □

2.4 Column Selection Criteria

We have considered the problem (2-3), and conditions under which $(\alpha_0, \beta_0) > 0$. So far, we have held both v and p fixed. We now consider holding only v fixed, but allowing p to be chosen as one of the columns of the matrix A from (2-1). In particular, we select that column A_s whose nonnegative combination with v brings us closer to b than any other A_j, $j \neq s$. This is done by solving the problem

$$\min\ G = \min_j \left(\min_{\alpha,\beta} \| b - \alpha v - \beta A_j \|^2 \right). \qquad (2\text{-}4)$$

Before finding the minimum of G in the general case, we will first consider the simpler problem of finding $A_j = A_s$ that yields the minimum of G when $v = 0$. The solution $\beta = \beta_0$ will be used to set $v = \beta_0 A_s$ as the initial approximation to b.

Theorem 2.15
If $b^T A_j > 0$ for some j, then the solution to

$$\min_j \left(\inf_\beta \| b - \beta A_j \|^2 \right) \tag{2-5}$$

is attained at $\beta_0 = b^T A_s$ where

$$s = \underset{j}{\text{argmax}}\ b^T A_j. \tag{2-6}$$

Otherwise, if $b^T A_j \leq 0$ for all j, then no matter which j is chosen, the inf *of (2-5) is attained at $\beta = 0$. In the latter case, the closest nonnegative approximation to b is given by $x = 0$.*

Proof: First consider

$$\min\ F = \min_\beta \| b - \beta A_j \|^2.$$

Because the columns of A are normalized, the minimum of F occurs at

$$\beta_0 = \frac{b^T A_j}{A_j^T A_j} = b^T A_j.$$

Evaluating F at β_0, we get

$$\begin{aligned} F\Big|_{\beta=\beta_0} &= (b - \beta_0 A_j)^T (b - \beta_0 A_j) \\ &= b^T b - (b^T A_j)^2. \end{aligned}$$

In choosing a particular A_j, we want to minimize this value subject to $\beta_0 > 0$. This corresponds to maximizing $(b^T A_j)^2$ over all positive $b^T A_j$. If $b^T A_j \leq 0$ for all j, then the inf of problem (2-5) is clearly attained at $\beta = 0$, regardless of the choice of j. □

We set $v = b^T A_s A_s$ as the initial approximation to b (where s is defined by (2-6)), and we proceed to seek to improve this approximation by considering the problem (2-4). For convenience of discussion, we assume that s defined by (2-6) is unique, although this is not a necessary condition.

Theorem 2.16
Let $1 \cdot v$ be closer to b than any scalar multiple λv for $\lambda \neq 1$. Let $\mu A_j \neq v \ \forall j$, where the columns A_j are normalized as in (2-1). Let v be closer to b than μA_j for $\mu \geq 0$ and all j, and let $u = b - v$. Under these conditions, the solution to

$$\min_j \left(\inf_{(\alpha,\beta)>0} \| b - \alpha v - \beta A_j \|^2 \right) \tag{2-7}$$

is attained at $j = s$ where

$$s = \underset{j}{\text{argmax}} \frac{A_j^T u}{\left(v^T v - (A_j^T v)^2 \right)^{1/2}}, \tag{2-8}$$

provided that $\exists\ j : A_j^T u > 0$. If $\exists$ no j such that $A_j^T u > 0$, then any $(\alpha, \beta) \geq 0$ will produce a value of (2-7) that is greater than or equal to $\| b - v \|^2$.

Proof: First consider

$$\min \ F_j = \min_{\alpha,\beta} \| b - \alpha v - \beta A_j \|^2. \tag{2-9}$$

From the proofs of Theorem 2.12 and Theorem 2.13, we know that the unconditional minimum of F_j occurs at (α_j, β_j) where

$$(\alpha_j, \beta_j) = \left(\frac{A_j^T A_j \ v^T v - A_j^T v \ A_j^T b}{A_j^T A_j \ v^T v - (A_j^T v)^2}, \ \ \frac{v^T v \ A_j^T u}{A_j^T A_j \ v^T v - (A_j^T v)^2} \right).$$

Since $A_j^T A_j = 1$, this can be rewritten as

$$(\alpha_j, \beta_j) = \left(\frac{v^T v - A_j^T v \ A_j^T b}{v^T v - (A_j^T v)^2}, \ \ \frac{v^T v \ A_j^T u}{v^T v - (A_j^T v)^2} \right).$$

Evaluating F_j at the minimum (α_j, β_j), we get

$$\begin{aligned}
F_j \Big|_{\substack{\alpha=\alpha_j \\ \beta=\beta_j}} &= (b - \alpha_j v - \beta_j A_j)^T (b - \alpha_j v - \beta_j A_j) \\
&= b^T (b - \alpha_j v - \beta_j A_j) - \alpha_j v^T (b - \alpha_j v - \beta_j A_j) - \beta_j A_j^T (b - \alpha_j v - \beta_j A_j) \\
&= b^T (b - \alpha_j v - \beta_j A_j) \\
&= b^T b - \alpha_j \ v^T v - \beta_j \ b^T A_j \qquad \text{(from Fact 2.2, part (a))} \\
&= b^T b - \left(\frac{(v^T v)^2 - v^T v \ b^T A_j \ v^T A_j}{v^T v - (v^T A_j)^2} \right) - \left(\frac{v^T v \ A_j^T u \ b^T A_j}{v^T v - (v^T A_j)^2} \right) \\
&= b^T b - v^T v \left(\frac{v^T v - 2 b^T A_j \ v^T A_j + (A_j^T b)^2}{v^T v - (v^T A_j)^2} \right).
\end{aligned}$$

We choose A_s so as to minimize F_j. This corresponds to maximizing the following over A_j:

$$\left(\frac{v^Tv - 2b^TA_j\, v^TA_j + (b^TA_j)^2}{v^Tv - (v^TA_j)^2}\right) = \frac{\left(A_j^T(b-v)\right)^2}{v^Tv - (A_j^Tv)^2} + 1,$$

which is the same as choosing $j = s$ so as to maximize

$$\frac{\left(A_j^T(b-v)\right)^2}{v^Tv - (A_j^Tv)^2} = \frac{(A_j^Tu)^2}{v^Tv - (A_j^Tv)^2}.$$

Let us consider the solution (α_0, β_0) to (2-9). Let $\min F = F(\alpha_0, \beta_0) = F_0$. Note that by Theorem 2.9, the solution (α_0, β_0) is unique, and any other solution $(\alpha_1, \beta_1) \neq (\alpha_0, \beta_0)$ must have the property that $F(\alpha_1, \beta_1) > F(\alpha_0, \beta_0)$.

Case 1: $u^TA_j = 0$

From the proof of Theorem 2.12 and Theorem 2.13, we know that $(\alpha_0, \beta_0) = (1, 0)$, and $F(\alpha_0, \beta_0) = \| b - v \|^2$. If we were to insist that we minimize over strictly positive (α, β), we would get an $(\alpha_1, \beta_1) \geq (\epsilon_1, \epsilon_2) > 0$ for some (ϵ_1, ϵ_2), and $(\alpha_1, \beta_1) \neq (\alpha_0, \beta_0)$. Thus from Theorem 2.9 and Corollary 2.10, $F(\alpha_1, \beta_1) > F(\alpha_0, \beta_0)$, or in other words, $F(\alpha_1, \beta_1) > \| b - v \|^2$.

Case 2: $u^TA_j < 0$

From the proof of Theorem 2.12, we see that $\beta_0 < 0$. From Theorem 2.11, we know that there is no $(\alpha_1, \beta_1) \geq 0$ such that $F(\alpha_1, \beta_1) < \| b - v \|^2$. Therefore if we were to insist that we minimize over strictly positive (α, β), we would get an $(\alpha_1, \beta_1) \geq (\epsilon_1, \epsilon_2) > 0$ such that $F(\alpha_1, \beta_1) \geq \| b - v \|^2$.

Case 3: $u^TA_j > 0$

From Corollary 2.14, we know that $(\alpha_0, \beta_0) > 0$. We also know that $F = \| b - v \|^2$ is attainable with $(\alpha, \beta) = (1, 0)$. Because the solution is unique, we can conclude that $F(1, 0) > F(\alpha_0, \beta_0)$, or in other words, $F(\alpha_0, \beta_0) < \| b - v \|^2$.

Only in Case 3 are we able to find an $(\alpha, \beta) > 0$ such that

$$\min_{(\alpha,\beta)>0} \| b - \alpha v - \beta A_j \|^2 < \| b - v \|^2.$$

Therefore, the only columns to consider to improve the approximation v are those such that $u^TA_j > 0$, provided that any such columns exist. Thus from the solution we computed for problem (2-9), we see that the solution to to (2-7) is given by

$$s = \underset{j:A_j^Tu>0}{\operatorname{argmax}} \frac{A_j^Tu}{\left(v^Tv - (A_j^Tv)^2\right)^{1/2}},$$

provided that $\exists\, j : A_j^Tu > 0$. We also showed that if $A^Tu \leq 0$, then any solution with $(\alpha, \beta) \geq 0$ will produce a minimum of (2-7) that is $\geq \| b - v \|^2$ (and hence $(\alpha, \beta) = (1, 0)$ is as good a solution as any possible). □

We will eventually consider more carefully the implications of the situation when $A^T u \leq 0$, but we need more results first.

2.5 Basic Weightings

A set of linearly independent columns of A, $\{A_1, A_2, \ldots, A_k\}$, will be denoted by B, and be called a *basis*; the set of indices $\{1, 2, \ldots, k\}$ will be referred to as *basic indices*. A nonnegative weighting of a basis is any $v = Bx_B$ with $x_B \geq 0$. It is a positive weighting if $x_B > 0$. A positive weighting such that v is closer to b than is any other positive weighting will be referred to as a *basic weighting*.

Given a basis, there may or may not exist a positive weighting that is closest to b. That is to say, the closest nonnegative weighting may have some weights $(x_B)_i = 0$, in which case it is a basic weighting for the subset of columns of B where $(x_B)_i > 0$.

Now consider the least-squares problem

$$\min \| b - Bx_B \|^2$$

Let x_B^0 be the solution and let $v^0 = Bx_B^0$. We will refer to such a v^0 as the *least-squares weighting* of B. Notice that this implies that a positive weighting of a basis B is a basic weighting if and only if it is the least-squares weighting of the columns of B.

Fact 2.17
Any basic weighting v of a basis B is unique.

Theorem 2.18
Let $v = Bx_B$ be a positive weighting of the columns of B. Let $u = b - v$. Then a necessary and sufficient condition that v be a basic weighting is $u^T B = 0$.

Proof: We know that if v is a positive weighting, then v is a basic weighting if and only if it is a least-squares weighting. Thus it is equivalent to prove that a necessary and sufficient condition that v be a least-squares weighting is $u^T B = 0$. The normal equations, which are necessary and sufficient conditions for v to be the least-squares weighting, are

$$B^T(Bx_B - b) = 0$$

Noting $Bx_B - b = v - b = u$, the result follows. □

Corollary 2.19
Given v, a basic weighting corresponding to B, and $u^T A_s \neq 0$ where $u = b - v$, then $\{B, A_s\}$ is a set of linearly independent columns.

Proof: By Theorem 2.18, $u^T B = 0$ because v is a basic weighting. Since $u^T A_s \neq 0$, we know that A_s is not contained in the space spanned by the columns of B. (Otherwise A_s would be a linear combination of the columns of B, and we would have $u^T A_s = 0$.) □

Corollary 2.20
If v is a basic weighting corresponding to B, and $u = b - v$, then if we select A_s by the criterion (2-8), where $A_s^T u > 0$, then A_s is linearly independent of the columns in B.

Proof: This follows directly from Corollary 2.19, as $A_s^T u \neq 0$. □

Corollary 2.21
If A_s is chosen as in Corollary 2.20, then the least-squares problem

$$\min \| b - Bx_B - A_s x_s \|^2 \tag{2-10}$$

has a unique solution.

Proof: B has linearly independent columns. By Corollary 2.20, $\{B, A_s\}$ is also a set of linearly independent columns. Therefore by Fact 2.4, the least-squares problem above has a unique solution. □

2.6 Strict Improvement

Theorem 2.22
Let $v = Bx_B$ be a basic weighting corresponding to basis B and let $u = b - v$. Let v be closer to b than μA_s for $\mu \geq 0$ and let A_s be linearly independent of the columns of B. Let $v_1 = Bx_B^1 + A_s x_s^1$ be the unique least-squares weighting of the problem (2-10), and let $u_1 = b - v_1$. If $u^T A_s > 0$, then the least-squares weighting satisfies $x_s^1 > 0$.

Proof: We know v is closer to b than is λv for $\lambda \neq 1$ because v is a basic weighting. We also know $A_s \neq \mu v$ because $u^T v = 0$ by Fact 2.2, and we know that $u^T A_s > 0$. Let $F = \| b - \alpha v - x_s A_s \|^2$. Let $\tilde{\alpha}$ and $\tilde{x}_s$ be the values of α and x_s at the minimum of F. From Corollary 2.14, we can conclude that both $\tilde{\alpha}$ and $\tilde{x}_s$ are positive. Finally, let $\tilde{v} = \tilde{x}_s A_s + \tilde{\alpha} v$. Since v^1 is the least-squares weighting of $\{B, A_s\}$, we have $\| b - v^1 \|^2 \leq \| b - \tilde{v} \|^2$.

We need to show that $\| b - \tilde{v} \|^2 \leq \| b - v \|^2$. We know

$$\begin{aligned} \| b - v \|^2 &= b^T b - 2v^T v + v^T v \quad \text{(from Fact 2.2)} \\ &= b^T b - v^T v. \end{aligned}$$

From the proof of Theorem 2.16, we know that

$$\tilde{x}_s = \frac{v^T v \; A_s^T u}{v^T v - (A_s^T v)^2} \quad \text{and} \quad \tilde{\alpha} = \frac{v^T v - A_s^T v \; A_s^T b}{v^T v - (A_s^T v)^2},$$

and that

$$\| b - \tilde{v} \|^2 = b^T b - v^T v \left(\frac{(A_s^T u)^2}{v^T v - (A_s^T v)^2} + 1 \right).$$

We know that $A_s^T u \neq 0$, so $(A_s^T u)^2 > 0$. We also know that $v^T v - (v^T A_s)^2 > 0$ from Fact 2.2. Thus

$$\frac{(A_s^T u)^2}{v^T v - (v^T A_s)^2} > 0,$$

so that

$$v^T v \left(\frac{(A_s^T u)^2}{v^T v - (v^T A_s)^2} + 1 \right) > v^T v.$$

Thus

$$\begin{aligned} \| b - \tilde{v} \|^2 &= b^T b - v^T v \left(\frac{(A_s^T u)^2}{v^T v - (v^T A_s)^2} + 1 \right) \\ &< b^T b - v^T v \\ &= \| b - v \|^2. \end{aligned}$$

So we have

$$\| b - v^1 \|^2 \leq \| b - \tilde{v} \|^2 < \| b - v \|^2. \tag{2-11}$$

Assume on the contrary that x_s^1 is nonpositive. From Fact 2.5, we know that any convex combination of $\tilde{v}$ and v^1 is at least as close to b as is $\tilde{v}$. The coefficient of A_s is 0 at some combination, since $\tilde{x}_s > 0$ and we are assuming that $x_s^1 \leq 0$. However, $\tilde{v}$ is strictly closer to b than is v_0, and if the coefficient of A_s is 0, this convex combination can be no closer to b than v_0. Thus, somewhere between v_1 and $\tilde{v}$ we have a solution that is further away from b than is v_0. This is a contradiction. □

Corollary 2.23
Given the same conditions as in Theorem 2.22, v^1 is strictly closer to b than is v.

Proof: This follows directly from (2-11). □

Now consider the following situation. As before, assume we have a basis B, a basic weighting $v = Bx_B$, and $u = b - v$. Now let A_s be another column from A, where A_s is linearly independent of the columns in B, $u^T A_s > 0$, and v is closer to b than is μA_s for $\mu \geq 0$. The linearly independent set of columns $\{B, A_s\}$, and the solution $\{y_B^*, y_s^*\}$ to

$$\min \| b - By_B - A_s y_s \|^2$$

has the property that $y_s^* > 0$ by Theorem 2.22. Note however, that we cannot say that $y_B^* > 0$.

Theorem 2.24
Given the situation described above, we can find a new basis $\tilde{B}$ where $\tilde{B}$ is formed from a subset of the columns of $(B \quad A_s)$, and the solution $\tilde{x}_B > 0$ has the property that

$$\min \| b - \tilde{B}\tilde{x}_B \|^2 < \min \| b - Bx_B \|^2.$$

In addition, A_s will be one of the columns in $\tilde{B}$.

Proof: Let $y^* = (y_B^* \quad y_s^*)^T$ be the solution to

$$\min \| b - (B \quad A_s)\, y \|^2.$$

Because $u^T A_s > 0$, we can use Theorem 2.22 to assert that $y_s^* > 0$. From Corollary 2.23, we also have the fact that

$$\| b - Bx_B \|^2 > \| b - (B \quad A_s)\, y^* \|^2.$$

In order to continue, we need some additional notation. We will be forming $\tilde{B}$ from $(B \quad A_s)$, and we may need to iteratively delete a number of the columns of B. We will denote the subset of the columns remaining at some iteration by B^y (indicating that B^y corresponds to the current y_B^*). Initially, this means that $B = B^y$.

Case 1: $y^* > 0$

Let $\tilde{B} = (B^y \quad A_s)$ and let $\tilde{x}_B = y^*$. This assignment is valid, since y^* is a basic weighting of the columns of $(B^y \quad A_s)$, and A_s is one of the columns of $\tilde{B}$.

Case 2: $y^* \geq 0$

Let $\hat{y}^*$ be the positive elements of y^*. Remember that $y_s^* > 0$, so $y_s^* \in \hat{y}^*$. Let $\hat{B}^y$ be the columns of B^y corresponding to the elements of $\hat{y}^*$. Let $\tilde{B} = (\hat{B}^y \quad A_s)$ and let $\tilde{x}_B = \hat{y}^*$. Again, this is valid because $\hat{y}^*$ is a basic weighting of the columns of $(\hat{B}^y \quad A_s)$, and A_s is one of the columns of $\tilde{B}$.

Case 3: We do not have $y^* \geq 0$.

In this case, we know that at least one element of y_B^* is negative, and we have a significantly more complicated situation. We need to drop one or more columns from $(B^y \quad A_s)$ such that there exists a positive least-squares solution for the remaining columns. In order to do this, we must redefine B^y and y^*, and then go back and check the sign of y^* against Cases 1–3 again. This means that we could perform the steps (to be described) in Case 3 more than once.

The first time through Case 3, we define $x^c = x_B$, and we have $B^y = B$. (Note that neither equation will be true in subsequent times through Case 3.)

We now form the convex combination

$$c = \begin{pmatrix} c_B \\ c_s \end{pmatrix} = \lambda \begin{pmatrix} x^c \\ 0 \end{pmatrix} + (1-\lambda) \begin{pmatrix} y_B^* \\ y_s^* \end{pmatrix}, \; x^c > 0, \; y_s^* > 0$$

and seek the smalled $\lambda < 1$ such that $c \geq 0$. This λ is determined by

$$\lambda = \max_{y_j^* < 0} \frac{-y_j^*}{x_j^c - y_j^*}.$$

It is clear for $\lambda \in (0,1)$ that $c_s > 0$. We will drop all columns of $(\, B^y \quad A_s \,)$ corresponding to zero elements of c.

Each pass through Case 3, one convex combination is formed. In the first pass, we have $B^y = B$, and we have proved that A_s is retained the first time a convex combination is formed during the column-dropping process. We need to prove that no matter how many convex combinations (and corresponding passes through Case 3) are necessary, A_s is not dropped. In order to prove this, we must redefine x^c, B^y and y^* as follows.

Let $\hat{c}$ be the positive elements of c. Let $\hat{B}^y$ be the columns of B^y corresponding to the elements of $\hat{c}$. Let $\hat{y}^*$ be the solution to

$$\min \| b - (\, \hat{B}^y \quad A_s \,) \hat{y} \|^2.$$

In order to redefine x^c to be $\hat{c}$, we need

$$\| b - Bx_B \|^2 > \| b - (\, \hat{B}^y \quad A_s \,) \hat{c} \|^2 \geq \| b - (\, \hat{B}^y \quad A_s \,) \hat{y}^* \|^2. \tag{2-12}$$

In addition, in order to redefine y^* to be $\hat{y}^*$ and B^y to be $\hat{B}^y$, we need

$$\hat{y}_s^* > 0. \tag{2-13}$$

We have

$$\| b - Bx_B \|^2 > \| b - (\, B^y \quad A_s \,) c \|^2 > \| b - (\, B^y \quad A_s \,) y^* \|^2$$

from Fact 2.5. We also know that

$$\| b - (\, B^y \quad A_s \,) c \|^2 = \| b - (\, \hat{A}^y \quad A_s \,) \hat{c} \|^2$$

from the definition of $\hat{c}$ and $\hat{A}^y$, and since $\hat{y}^*$ is a least-squares solution, we have

$$\| b - (\, \hat{B}^y \quad A_s \,) \hat{c} \|^2 \geq \| b - (\, \hat{B}^y \quad A_s \,) \hat{y}^* \|^2.$$

Finally, both $\hat{y}^*$ and y^* are least-squares solutions, but $\hat{B}^y$ is a proper subset of B^y, so

$$\| b - (\, \hat{B}^y \quad A_s \,) \hat{y}^* \|^2 \geq \| b - (\, B^y \quad A_s \,) y^* \|^2.$$

Altogether, this gives us

$$\| b - Bx_B \|^2 > \| b - (\, B^y \quad A_s \,) \hat{c} \|^2 \geq \| b - (\, \hat{B}^y \quad A_s \,) \hat{y}^* \|^2 \geq \| b - (\, B^y \quad A_s \,) y^* \|^2,$$

which proves (2-12).

Now we need to consider (2-13), the sign of $\hat{y}_s^*$. We will prove that $\hat{y}_s^* > 0$ by contradiction, so assume that $\hat{y}_s^* \leq 0$. We will form a convex combination of y^* and $\hat{y}^*$ as follows. Let z be defined as

$$z_j = \begin{cases} \hat{y}_j^*, & \text{if } B_j^y \in \hat{B}^y; \\ 0, & \text{if } B_j^y \notin \hat{B}^y. \end{cases}$$

This gives us

$$\| b - (\hat{B}^y \quad A_s)\hat{y}^* \|^2 = \left\| b - (B^y \quad A_s)\begin{pmatrix} z \\ \hat{y}_s^* \end{pmatrix} \right\|^2 \geq \| b - (B^y \quad A_s) y^* \|^2.$$

Now form the convex combination

$$c^z = \begin{pmatrix} c_B^z \\ c_s^z \end{pmatrix} = \lambda \begin{pmatrix} z \\ \hat{y}_s^* \end{pmatrix} + (1-\lambda)\begin{pmatrix} y_B^* \\ y_s^* \end{pmatrix}.$$

Because $\hat{y}_s^* \leq 0$ and $y_s^* > 0$, we can pick $\lambda \in (0, 1]$ such that $c_s^z = 0$. Fact 2.5 says that

$$\| b - (\hat{B}^y \quad A_s)\hat{y}^* \|^2 \geq \| b - (\hat{B}^y \quad A_s) c^z \|^2 \geq \| b - (B^y \quad A_s) y^* \|^2.$$

However, this would imply that

$$\| b - Bx_B \|^2 > \| b - (\hat{B}^y \quad A_s) c^z \|^2,$$

which when $c_s^z = 0$, can be rewritten as

$$\| b - Bx_B \|^2 > \left\| b - (\hat{B}^y \quad A_s)\begin{pmatrix} c_B^z \\ c_s^z \end{pmatrix} \right\|^2 = \left\| b - (\hat{B}^y \quad A_s)\begin{pmatrix} c_B^z \\ 0 \end{pmatrix} \right\|^2,$$

and thus

$$\| b - Bx_B \|^2 > \| b - \hat{B}^y c_B^z \|^2.$$

However, $\hat{B}^y$ is a proper subset of the columns of B, and x_B is the least-squares solution to

$$\min \| b - Bx_B \|^2,$$

so that we know

$$\| b - Bx_B \|^2 \leq \| b - \hat{B}^y c_B^z \|^2.$$

This is our contradiction. Thus we must have $\hat{y}_s^* > 0$, and (2-13) has been shown to be true. We can therefore set $x^c = \hat{c}$, $B^y = \hat{B}^y$, $y^* = \hat{y}^*$, and go back and check this new y^* against Cases 1–3. Note that if Case 3 again applies and additional columns must be dropped, we will have the following upon entry:

$$\| b - Bx_B \|^2 > \| b - (B^y \quad A_s) x^c \|^2 \geq \| b - (B^y \quad A_s) y^* \|^2,$$

and when a new convex combination c is formed, we will find

$$\| b - Bx_B \|^2 > \| b - (B^y \quad A_s)\, x^c \|^2 > \| b - (B^y \quad A_s)\, c \|^2 > \| b - (B^y \quad A_s)\, y^* \|^2$$

by Fact 2.5. Thus (2-12) will still hold in subsequent passes through Case 3. We repeat this process of checking the sign of y^* and dropping columns until Case 1 or Case 2 is satisfied.

We will now show that y^* will eventually satisfy Case 1 or Case 2. We know that $y_s^* > 0$, no matter how many convex combinations are formed, and no matter how many times y^* and B^y are consequently redefined. Therefore column A_s is always retained during this process. Thus we can conclude that in the worst case, we will find $y^* > 0$ when $B^y = \emptyset$ (and A_s is the only remaining column in $\tilde{B}$), at which time Case 1 is satisfied. □

Note that Theorem 2.24 did not disallow the possibility that all columns be dropped from A^y before we can find a $y^* > 0$. That is, it implies that it could be possible that $\{A_k\} = A_s$ and $x^1 = y_s^*$. We will prove that this is actually *not* possible.

Corollary 2.25
If the first column ever added to B is A_r where r is determined by (2-6) in Theorem 2.15, then y^* will satisfy Case 1 or 2 before all columns other than A_s are dropped from the basis.

Proof: Assume that y^* never satisfies Case 1 or Case 2 until only column A_s is remaining (and $y^* = y_s^* > 0$). From Theorem 2.24, we know that

$$\| b - Bx_B \|^2 > \| b - \tilde{B}\tilde{x}_B \|^2,$$

which (if all columns other than A_s are dropped) can be rewritten as

$$\| b - Bx_B \|^2 > \| b - A_s y_s^* \|^2.$$

Since A_r was the first column placed in B, we know from Theorem 2.24 that

$$\| b - A_r y_r^* \|^2 > \| b - Bx_B \|^2,$$

where $y_r^* > 0$ is the solution to

$$\min \| b - A_r y_r \|^2.$$

Thus

$$\| b - A_r y_r^* \|^2 > \| b - Bx_B \|^2 > \| b - A_s y_s^* \|^2,$$

which implies

$$\| b - A_r y_r^* \|^2 > \| b - A_s y_s^* \|^2.$$

However, A_r was chosen to be the solution to (2-5) (see Theorem 2.15). Thus we have a contradiction. □

2.7 Infeasibility

Until now, we have not considered the feasibility or infeasibility of (2-1). This section discusses the issue of infeasibility and how it affects the solution to the least-squares problems considered in Theorems 2.15 and 2.16, as well as other related issues.

Fact 2.26
If $A^T b \leq 0$, we can conclude that (2-1) is infeasible.

Proof: Given that $b \neq 0$, this follows directly from Farkas' Lemma. □

Fact 2.26 implies that if in the process of finding the solution to (2-5) (see Theorem 2.15), we find that $A_j^T b \leq 0 \ \forall \ j$, then we have established that (2-1) is infeasible.

Fact 2.27
Let A be as in Fact 2.26. Let $v \neq 0$ be closer to b than any λv, $\lambda \neq 1$, and let $u = b - v$. If $u \neq 0$ and $u^T A \leq 0$, then the original problem (2-1) is infeasible.

Proof: We know from Fact 2.2 that $b^T u = u^T u$, and we know that $u \neq 0$. Therefore this follows directly from Farkas' Lemma. □

Corollary 2.28
Let v be a basic weighting corresponding to basis B and let $u = b - v$. If $u^T A_j \leq 0 \ \ \forall \ A_j \notin B$, then (2-1) is infeasible.

Proof: We know that v is a basic weighting of B, so we can conclude from Theorem 2.18 that $u^T A_j = 0 \ \forall \ A_j \in B$. Thus we have $u^T A_j \leq 0 \ \forall \ A_j$. We can see by Fact 2.27 that (2-1) is thus infeasible. □

Theorem 2.29
Let $v = Bx_B$ be a basic weighting corresponding to the basis B and let $u = b - v$. If infeasibility is detected (i.e., if $u^T A_j \leq 0 \ \ \forall \ A_j \notin B$), then v is at least as close to b as is any other nonnegative weighting of the columns of A.

Proof: Let BI be the set of indices such that $j \in BI$ if and only if $A_j \in B$. We can write v as

$$v = \sum_{j \in BI} A_j (x_B)_j ,$$

where $(x_B)_j$ is the component of x_B corresponding to A_j.

Since we have discovered infeasibility, we know that $u \neq 0$. We also know that v is closer to b than is any λv_B, $\lambda \neq 1$ because x_B is the least-squares solution to

$$\min \left\| b - \sum_{j \in BI} A_j (x_B)_j \right\|^2 .$$

We therefore know:

$$\begin{aligned} u^T b &= u^T u \quad \text{by Fact 2.2} \\ u^T A_j &= 0 \ \ \forall\, j \in BI \quad \text{by Theorem 2.18} \\ u^T A_j &\leq 0 \ \ \forall\, j \notin BI \quad \text{because infeasibility has been detected.} \end{aligned}$$

Let

$$\tilde{v} = \sum_{j=1}^{n} A_j \tilde{x}_j$$

be any other nonnegative weighting of the columns of A. Therefore we have $\tilde{x}_j \geq 0 \ \forall\, j$. Let $\tilde{u} = b - \tilde{v}$. Consider

$$\begin{aligned} u^T \tilde{u}_B &= u^T b - u^T \sum_{j=1}^{n} A_j \tilde{x}_j \\ &= u^T u - \sum_{j \in BI} u^T A_j \tilde{x}_j - \sum_{j \notin BI} u^T A_j \tilde{x}_j \\ &= u^T u - \sum_{j \notin BI} u^T A_j \tilde{x}_j \geq u^T u. \end{aligned}$$

Now consider

$$u^T u = \| u \| \| u \|; \quad u^T \tilde{u} = \| u \| \| \tilde{u} \| \cos\theta,$$

where θ is the angle between u and $\tilde{u}$. We have

$$\| u \| \| \tilde{u} \| \cos\theta \geq \| u \| \| u \| \Rightarrow \| \tilde{u} \| \| \tilde{u} \| \cos\theta \geq \| u \| \| \tilde{u} \|,$$

which can be extended to

$$\| \tilde{u} \| \| \tilde{u} \| \geq \| \tilde{u} \| \| \tilde{u} \| \cos\theta \geq \| u \| \| \tilde{u} \| \geq \| u \| \| \tilde{u} \| \cos\theta.$$

Therefore $\tilde{u}^T \tilde{u} \geq u^T \tilde{u} \geq u^T u$. So we can conclude that v is at least as close to b as is any other nonnegative weighting of the columns of A. □

Corollary 2.30
Given B, v and u as in Theorem 2.29. If v is closer to b than is any other nonnegative weighting of the columns of A, then infeasibility will be detected (that is, we will find $u^T A_j \leq 0 \ \forall\, j \notin B$).

Proof: Assume that $u^T A_s > 0$ for some $A_s \notin B$. If we add A_s to B and drop columns as described in Section 2.6, then we will find a $\tilde{B}$ and $\tilde{x}_B > 0$ such that $\tilde{v} = \tilde{B}\tilde{x}_B$ is closer to b than is v. This contradicts the assumption that v is closer to b than is any other nonnegative weighting of the columns of A. □

2.8 The Least-Squares Phase I Algorithm

As stated in Section 2.1, the main objective of the Phase I algorithm to be presented here is to build a basis matrix, made up of selected columns of A, and to find positive weightings of these columns such that the basis and the associated positive weightings are a solution to (2-2). Remember that we have rescaled (2-1) such that $\| A_j \| = 1 \; \forall \; j$ and $\| b \| = 1$.

At the start of an iteration, we are given a basis B, composed of a linearly independent subset of the columns of A, with the property that the least-squares weighting of these columns which yields the closest approximation to the right-hand side b is a positive weighting. The iterative step tests whether the current approximation is the closest approximation with a positive weighting, and if not, selects an incoming column with the property that a positive combination of the previous approximation and the incoming column is a strictly better approximation to the right-hand side. There exists, however, a (possibly proper) subset of the columns of the augmented basis such that the least-squares weighting is positive, and is also a better approximation to the right-hand side than the simple positive combination. This subset of the augmented set of basic columns can be used to start the next iteration.

Algorithm Pseudocode

Notation and Variables:

A: The original $m \times n$ matrix of (2-1).

b: The original right-hand side of (2-1).

nbi: The number of currently basic columns (the number of columns in B).

B: The matrix of currently basic columns. $m \times nbi$ of B is used, and nbi can be as large as m.

BI: An m-vector containing indices of the columns of B relating them to the columns of A. For example, if $BI[i] = j$, the ith column of B contains the jth column of A. $BI[i] = 0$ means no column corresponds to the ith component of BI, which implies there must be less than i basic columns.

x_B: The current positive weightings of the columns of B. The first nbi components of x_B (corresponding to the nbi columns in B) are used, and nbi can be as large as m.

v: The current approximation Bx_B.

u: The current residual $b - v$.

y: The current least-squares solution of $\min \| b - By \|^2$. The first nbi components of y are used, and nbi can be as large as m.

X: The vector to hold the solution to (2-1). Note that x_B holds the solution to (2-2).

0. Initialization {no columns initially in basis}
   $B := \emptyset$ {no columns in basis}
   $x_B := \emptyset$ {no current basic solution}
   $BI := \emptyset$ {no basic indices}
   $nbi := 0$ {set number of basic columns to 0}
   $u := b$ {set residual to b}
   $v := 0$ {set current approximation $v = Bx_B = 0$}
   $X := 0$ {set final solution to 0}
1. Startup {find the first column to place in B}
   If $b^T A_j \leq 0 \ \forall \ j = 1, \ldots, n$ then
      Go to 4. {the problem is infeasible}
   $s := \text{argmax}_{j:b^T A_j > 0} \, b^T A_j$
   Place A_s in the first column of B.
      $BI[1] := s$ {record first column}
      $y[1] := b^T A_s$ {record basic solution}
      $nbi := 1$ {set number of basic columns}
2. Main Loop {Add columns to B until $u = 0$ or infeas. is discovered.}
   $x_B := y$
   $v := Bx_B$ {set current approximation}
   $u := b - v$ {set current residual}
   If $u = 0$ then
      Go to 4. {solution found}
   If $u^T A_j \leq 0 \ \forall \ j \notin BI$ then
      Go to 4. {infeasibility discovered}
   $nbi := nbi + 1$ {increment number of basic columns}

$$s = \underset{\substack{j \notin BI \\ j: A_j^T u > 0}}{\text{argmax}} \frac{A_j^T u}{\left(v^T v - (A_j^T v)^2\right)^{1/2}}$$

   Place A_s in column nbi of B.
   $BI[nbi] := s$ {record position of incoming column}
   Set $c = \begin{pmatrix} x_B \\ 0 \end{pmatrix}$
3. Least-Squares Loop
   Solve $\| By - b \|^2$ for y
   If $y > 0$ then
      Go to 2. {new B and x_B found; repeat Main Loop}
   If $y \geq 0$ then

Remove all columns from B and elements from x_B that correspond to zero components of y. Update BI accordingly.
Go to 2. {new B and x_B found; repeat Main Loop}

$\lambda := \max_{y_i<0} -y_i/(c_i - y_i)$

$c := \lambda c + (1-\lambda)y$ {form convex combination of c and y }
Remove all columns from B and elements from x_B and c that correspond to zero components of c. Update BI accordingly.
Go to 3. {repeat Least-Squares loop}

4. Done {clean up}
for $i := 1$ to nbi do {form the answer to the original problem}
$X_{BI[i]} := x_B[i]$
Report on feasibility and output the solution found.

We now make the following observations:

1. The approximation $v = Bx_B$ is strictly closer to b at each iteration (see Theorem 2.24). This means that degeneracy does not cause problems; cycling cannot occur, and the process terminates in a finite number of steps because given B, x_B is uniquely determined, and there are a finite number of bases B.

2. If infeasibility is detected, then the current approximation $v = Bx_B$ is closer to b than is any other nonnegative weighting of the columns of A (see Theorem 2.29). Similarly, if $u \neq 0$ and $v = Bx_B$ is closer to b than is any other nonnegative weighting of the columns of A, then infeasibility will be detected when trying to add a new column (see Corollary 2.30).

3. Whenever a new column is chosen from A to enter B, it is linearly independent of the columns currently in B (see Corollary 2.19). Also due to the linear independence of entering columns, our least-squares solution (the solution to $\min \| b - \tilde{B}y \|$) is always unique, so numerical difficulties aside, rank deficient least-squares problems are never encountered.

2.9 Variations

Free Variables
Consider the system

$$\begin{array}{lcll} Ax & = & b & \\ x_i & \geq & 0 & \text{for some set of } i \\ x_j & & \text{free} & \text{for some set of } j \end{array} \qquad (2\text{-}14)$$

This system could always be converted to the form of (2-1) by replacing each free variable x_j with two new nonnegative variables x'_j and x''_j where the original x_j

is now represented by $x'_j - x''_j$. However, let us consider how to solve a feasibility problem with free variables without actually adding any more variables.

In the pure algorithm described in Section 2.8, the entering columns is A_s where s is determined by (2-8). This assumes that all variables are nonnegative. If free variables are present, we can write the rule for selecting the incoming column s as

$$s = \text{argmax} \left\{ \max_{\substack{i:x_i \geq 0 \\ i:A_i^T u > 0}} \frac{A_i^T u}{\left(v^T v - (A_i^T v)^2\right)^{1/2}}, \max_{\substack{j:x_j \text{ free} \\ j:|A_j^T u| > 0}} \frac{|A_j^T u|}{\left(v^T v - (A_j^T v)^2\right)^{1/2}} \right\}.$$

Therefore we see that we can select an incoming column when free variables are present without actually adding new variables and columns. Infeasibility is detected when $A_i^T u \leq 0 \quad \forall\, i : x_i \geq 0$, and $A_j^T u = 0 \quad \forall\, j : x_j$ is free.

The only other change in the algorithm concerns testing the sign of y in the Least-Squares Loop part of the algorithm. Elements of y corresponding to free variables may take on any nonzero value. If such an element of y is found to be zero, it is dropped just as any other zero element of y would be. The only components of y that must be strictly positive are those corresponding to variables restricted to be nonnegative. Thus, when λ is formed, the only components $y_i < 0$ that are considered are those corresponding to nonnegative variables.

All the results previously discussed (strict improvement, infeasibility detection, etc.) still hold, as these modifications to the pure algorithm simply implement the result of splitting free variables without actually doing the work of adding variables and columns.

No Denominator

Consider the selection of the incoming column A_s. We must form $A_j^T u$ for all columns not currently in the basis, and (2-8) for all columns such that $A_j^T u > 0$. As the denominator is never directly used in any of the proofs relating to infeasibility, strict improvement, etc., if we replace the denominator of (2-8) with 1, we produce a perfectly valid column-selection rule that requires less computation per candidate column.

It would be expected that this modified rule might not perform as well as the pure rule, since we may not be selecting the A_s that minimizes (2-7). However, we can still guarantee $(\alpha, \beta) > 0$ because $A_s^T u > 0$.

As it turns out, the results produced (on test problems to be described later) show very little difference in iteration count between either of these two variations ("no denominator" and "free variable") and the pure algorithm. Apparently, it is the direction of the candidate incoming column that is important, not the particular scaling factor used in the column-selection rule.

Crash Basis

Until now, we have built up the basis B starting from scratch—that is, initially B contains no columns. If we could pick out a starting set of linearly independent columns without much effort, we could start with them already in B.

Consider multiplying by -1 all rows of A (and elements of b) corresponding to negative elements of b. This gives us a positive right-hand side. Now consider selecting all columns of this modified A that are columns of the identity matrix, i.e., columns of the form e_j, where e is a vector of zeros with a 1 in the jth position. (Note that most columns of this form will correspond to slack variables.) If we place all such columns in B, and eliminate duplicate columns, we will get the problem $\min \| b - Bx \|$, and the solution x_B will consist of the elements of b corresponding to the nonzero elements of the identity columns.

All of the theory for the pure algorithm holds for this method as well, with the exception of Corollary 2.25. Since we do not know that the column A_s satisfying (2-5) was the first column selected to enter the basis, we can no longer guarantee that all columns (other than the incoming column) will not be dropped. In our computational experience, such dropping of all columns has not been observed, and is thus presumed to be rare in practice.

2.10 Discussion

We have proved in Theorem 2.29 and Corollary 2.30 that if (2-1) is infeasible, we will detect such infeasibility when our current solution $x_B > 0$ is as close as possible to the right-hand side b. That is, $v = Bx_B$ is at least as close to b as is any other nonnegative weighting of the columns of A. Thus we see that the least-squares algorithm actually solves the problem

$$\begin{aligned} \min \quad & \frac{1}{2}\| b - Ax \|^2 \\ \text{s.t.} \quad & x \geq 0. \end{aligned} \tag{2-15}$$

If the least-squares algorithm finds a feasible solution to (2-1), then the optimal objective value of (2-15) is 0. Otherwise, the optimal objective of (2-15) is positive, we have an optimal solution x_B, and no feasible solution exists to (2-1).

Problem (2-15) is often called the "non-negative least-squares" problem (NNLS), and can also be considered a degenerate case of the "bounded least-squares" problem (BLS), in which x can have both upper and lower bounds. Lawson and Hanson [12] present an algorithm for NNLS that is very closely related to our algorithm. The only differences between the algorithms is that first, a denominator of "1" is used in the column selection rule of NNLS. Secondly, after an incoming column A_s has been successfully introduced into the basis (and all necessary dropping of other basic columns is complete), our algorithm takes all basic variables with a zero value and makes them nonbasic, while Lawson and Hanson's NNLS algorithm does not.

We implemented both the NNLS algorithm and the "denominator-of-1" variation of the least-squares algorithm, and tested the efficiency of the algorithms on the smallest 30 linear programming problems available from NETLIB. (See [7] and Section 3 for more details about these problems.) In practice, these two algorithms rarely chose different incoming columns. A substantial difference between the two algorithms was observed only once, on problem SCSD6, which is highly degenerate in the simplex method sense. It also appears to be the case (in practice) that the NNLS algorithm will occasionally have the same final basic index set as does the least-squares algorithm with an additional handful of basic variables. In our experiments, these additional basic variables were all at a very low level (on the order of 10^{-10} or less). Thus the least-squares algorithm occasionally found a "cleaner" solution than did the NNLS algorithm, apparently due to numerical error.

Björck [3] presents an algorithm for BLS, that when applied to NNLS, is even closer to our algorithm that is Lawson and Hanson's algorithm for NNLS. The only difference between the BLS algorithm and the least-squares algorithm is that a denominator of "1" is used in the column selection rule of NNLS. Therefore, the "denominator-of-1" version of the least-squares algorithm is equivalent to the BLS algorithm. Neither [3] nor [12] give any numerical results for this algorithm.

We also found the following interesting equivalence between the least-squares algorithm and the convex quadratic programming algorithm due to Dantzig [5] and Van de Panne & Whinston [14]. Before we can show this equivalence, we must first reformulate (2-15) as a quadratic programming problem.

The constrained least-squares (2-15) has the following necessary and sufficient conditions for optimality:

$$\begin{aligned} -A^T b + A^T A x &\geq 0 \\ x &\geq 0 \\ x^T(-A^T b + A^T A x) &= 0. \end{aligned}$$

These conditions can be formulated as the Linear Complementarity Problem (LCP)

$$\begin{aligned} q + Mx &\geq 0 \\ x &\geq 0 \\ x^T(q + Mx) &= 0, \end{aligned} \tag{2-16}$$

where $q = -A^T b$ and $M = A^T A$. Because this M is symmetric and positive semi-definite, this LCP is equivalent to the quadratic program

$$\begin{aligned} \min \quad & q^T x + \frac{1}{2} x^T M x = c \\ \text{s.t.} \quad & x \geq 0. \end{aligned} \tag{2-17}$$

(This problem could, of course, be derived directly from (2-15).)

Accordingly, any convex quadratic programming algorithm would be applicable to (2-17). Although we are going to demonstrate an equivalence between the quadratic programming algorithm developed by Dantzig and by Van de Panne & Whinston, the very same algorithmic ideas are developed by Goldfarb & Idnani [10] from an active-set point of view. (The following description of this algorithm is adapted from the book by Cottle, Pang and Stone [4].)

The Quadratic Programming Algorithm

The algorithm is based on principal pivotal transforms of the system $y = q + Mx$, which can be written in tableau form as

	1	x
$2c$	0	q^T
y	q	M

where c is as in (2-17). At any point in the algorithm, we will have two index sets α and β, corresponding to basic and nonbasic x variables, respectively. Thus if we partition and relabel the original tableau as

	1	x_β	y_α
$2c$	0	q_β^T	q_α^T
y_β	q_β	$M_{\beta\beta}$	$M_{\beta\alpha}$
x_α	q_α	$M_{\alpha\beta}$	$M_{\alpha\alpha}$

then we can write a general tableau during the algorithm in terms of the original tableau as

	1	x_β	y_α
$2c$	$-q_\alpha^T M_{\alpha\alpha}^{-1} q_\alpha$	$q_\beta^T - q_\alpha^T M_{\alpha\alpha}^{-1} M_{\alpha\beta}$	$q_\alpha^T M_{\alpha\alpha}^{-1}$
y_β	$q_\beta - M_{\beta\alpha} M_{\alpha\alpha}^{-1} q_\alpha$	$M_{\beta\beta} - M_{\beta\alpha} M_{\alpha\alpha}^{-1} M_{\alpha\beta}$	$M_{\beta\alpha} M_{\alpha\alpha}^{-1}$
x_α	$-M_{\alpha\alpha}^{-1} q_\alpha$	$-M_{\alpha\alpha}^{-1} M_{\alpha\beta}$	$M_{\alpha\alpha}^{-1}$

From this tableau, we can see that when the nonbasic variables x_β and y_α are set to 0, the basic variables x_α and y_β can be expressed as

$$\begin{aligned} x_\alpha &= -M_{\alpha\alpha}^{-1} q_\alpha \\ y_\beta &= q_\beta - M_{\beta\alpha} M_{\alpha\alpha}^{-1} q_\alpha. \end{aligned}$$

We will rewrite the general tableau (after k principal pivots) as

	1	x_β^k	y_α^k
$2c$	θ^k	$(q_\beta^k)^T$	$(q_\alpha^k)^k$
y_β^k	q_β^k	$M_{\beta\beta}^k$	$M_{\beta\alpha}^k$
x_α^k	q_α^k	$M_{\alpha\beta}^k$	$M_{\alpha\alpha}^k$

in order to ease the notation when examining the algorithm in more detail. We now describe the quadratic programming algorithm as applied to (2-17) in pseudocode.

0. Initialization.
   $(q^0, M^0) := (q, M)$
   $k := 0$
   $\alpha := \emptyset$
   $\beta := \{1, \ldots, n\}$
1. Check stopping conditions.
   $s := \text{argmin}\{q_i^k : i \in \beta\}$
   If $q_s^k \geq 0$, the solution is $x_\alpha = q_\alpha^k$, $x_\beta = 0$.
   If $q_s^k < 0$, choose x_s^k as the incoming variable.
   Note that we do not need to check for infeasibility here, as (2-17) always has a solution.
2. Determine outgoing (blocking) basic x variables, if any.
   Let

$$\gamma = \min_{i:\left((M_{\alpha\beta}^k)_s\right)_i < 0} \frac{(q_\alpha^k)_i}{-\left((M_{\alpha\beta}^k)_s\right)_i}$$

   where $(M_{\alpha\beta}^k)_s$ is the sth column of $M_{\alpha\beta}^k$,
   $((M_{\alpha\beta}^k)_s)_i$ is the ith component of that column, and
   similarly $(q_\alpha^k)_i$ is the ith component of q_α^k.
3. Perform a pivot.
   If

$$\gamma \geq \frac{(q_\beta^k)_i}{-\left((M_{\beta\beta}^k)_s\right)_i},$$

   then there is no outgoing basic x variable. In this case,
   Move index s from index set β to α.
   Set $k = k + 1$.
   Go to Step 1.
   Otherwise, let r be the index corresponding to the minimum ratio γ
   Move index r from index set α to β.
   Set $k = k + 1$.
   Go to Step 2.

Further discussion of this algorithm can be found in Cottle, Pang & Stone [4].

We now demonstrate the equivalences between this algorithm and the no-denominator version of the least-squares algorithm. Let us assume that at some point in both algorithms we have the same index sets α and β corresponding to basic and nonbasic x variables, respectively. We can rewrite the partitioned original tableau in terms of our least-squares algorithm as

	1	x_β	y_α
$2c$	0	$-b^TN$	$-b^TB$
y_β	$-N^Tb$	N^TN	N^TB
x_α	$-B^Tb$	B^TN	B^TB

Note that $q_\beta = -N^Tb$, and so on. Similarly, we rewrite the general tableau as

	1	x_β	y_α
$2c$	$-b^TB(B^TB)^{-1}B^Tb$	$-b^TN + b^TB(B^TB)^{-1}B^TN$	$-b^TB(B^TB)^{-1}$
y_β	$-N^Tb + N^TB(B^TB)^{-1}B^Tb$	$N^TN - N^TB(B^TB)^{-1}B^TN$	$N^TB(B^TB)^{-1}$
x_α	$(B^TB)^{-1}B^Tb$	$-(B^TB)^{-1}B^TN$	$(B^TB)^{-1}$

Thus we see that $x_\alpha = -M_{\alpha\alpha}^{-1}q_\alpha = (B^TB)^{-1}B^Tb = x_B$, and the two solutions are the same. Therefore, given the same index set of basic x variables, both the quadratic programming algorithm and the least-squares algorithm produce the same solution $x_\alpha = x_B$.

To further compare the two algorithms, assume that again both algorithms have the same index sets α and β corresponding to the basic and nonbasic x variables. We will consider which nonbasic x variable is chosen to become basic in both algorithms.

In the quadratic programming algorithm we select incoming variable x_s, where

$$\begin{aligned} s &= \operatorname*{argmin}_{i\in\beta}\{q_i^k\}, \\ &= \operatorname*{argmin}_{i\in\beta}\left\{(q_\beta - M_{\beta\alpha}M_{\alpha\alpha}^{-1}q_\alpha)_i\right\} \\ &= \operatorname*{argmin}_{i\in\beta}\left\{-N_i^Tb + N_i^TBx_B\right\} \\ &= \operatorname*{argmin}_{i\in\beta}\left\{-N_i^Tu\right\}. \end{aligned}$$

If we find that $q_s^k = -N_s^Tu \geq 0$, then in the quadratic programming algorithm, we know we have a solution to (2-17). Similarly, if $N_s^Tu \leq 0$, the least-squares algorithm has either found a solution to (2-1) (in which case $u = 0$), or has detected infeasibility of (2-1). In either case, the least-squares algorithm has

found a solution to (2-17). Thus $N_s^T u \leq 0$ is the termination condition for both algorithms. (Note that the quadratic programming algorithm never finds infeasibility, as (2-17) always has a solution.) On the other hand, if $q_s^k = -N_s^T u < 0$, then x_s is the incoming variable for both problems. Thus the two algorithms have the same termination conditions and the same column-selection rules.

Given that we start an iteration with the same index sets, and that the same incoming variable x_s is selected, we will now consider how the two algorithms proceed. In the least-squares algorithm, x_s simply becomes basic, even though this could cause other basic variables to become negative. However, the quadratic programming algorithm checks for and pivots out all (if any) variables which would become negative, only then making x_s basic. If there is no outgoing basic variable, then both algorithms add x_s to the set of basic variables, all other basic variables remain nonnegative, and the algorithms again share the same index sets.

Assume now that bringing x_s into the set of basic x variables will cause at least one of the other basic variables to become negative. In the case of the least-squares algorithm, this situation means that at least one of the previously basic x variables is nonnegative, and at least one of these negative basic variables must be dropped from the basis. Similarly, the quadratic programming algorithm must find and eliminate from the basis at least one outgoing (blocking) variable. We will now demonstrate that given the same index sets, and the same incoming variable x_s, if a column must be dropped from the basis in order to maintain nonnegativity of the basic x variables, both algorithms will select the same basic x variable to be removed from the basis.

The quadratic programming algorithm will select x_r to become nonbasic, where

$$\begin{aligned} r &= \operatorname*{argmin}_{((M_{\alpha\beta}^k)_s)_i<0} \frac{(q_\alpha^k)_i}{-\left((M_{\alpha\beta}^k)_s\right)_i}, \\ &= \operatorname*{argmin}_{-\left((B^TB)^{-1}B^TN_s\right)_i<0} \frac{\left((B^TB)^{-1}\right)_i}{\left((B^TB)^{-1}B^TN_s\right)_i} \\ &= \operatorname*{argmin}_{(d_D)_i<0} \frac{(x_B)_i}{-(d_D)_i}, \end{aligned}$$

where $d_D = -(B^TB)^{-1}B^TN_s$. On the other hand, the least-squares algorithm will select x_r to become nonbasic, where

$$r = \operatorname*{argmax}_{y_i<0} \frac{-y_i}{(x_B)_i - y_i} = \operatorname*{argmin}_{y_i<0} \frac{(x_B)_i}{-y_i}$$

$$y = \left(\begin{pmatrix} B^T \\ N_s^T \end{pmatrix} (B \quad N_s) \right)^{-1} \begin{pmatrix} B^T \\ N_s^T \end{pmatrix} b.$$

From Theorem 2.24 we know that the last component of y is strictly positive, and is thus not considered in the minimum-ratio test. After much algebraic manipulation, we can again rewrite the expression for this minimum-ratio test as

$$r = \underset{(x_b - \gamma d_D)_i < 0}{\operatorname{argmin}} \frac{(x_B)_i}{-(x_B - \gamma d_D)_i},$$

where

$$\gamma = \frac{c}{d}; \qquad c = N_s^T u; \qquad d = N_s^T B (B^T B)^{-1} B^T N_s - N_s^T N_s.$$

In order to demonstrate that both minimum-ratio tests must select the same index r, we will derive conditions that must be met in order for a *different* index to be chosen. This will produce a contradiction.

Assume that for some indices j and k, we have

$$\frac{(x_B)_j}{-(d_D)_j} > \frac{(x_B)_k}{-(d_D)_k} \quad \text{and} \quad \frac{(x_B)_j}{-(x_B - \gamma d_D)_j} < \frac{(x_B)_k}{-(x_B - \gamma d_D)_k},$$

where

$$(d_D)_j < 0; \qquad (d_D)_k < 0; \qquad (x_B - \gamma d_D)_j < 0; \qquad (x_B - \gamma d_D)_k < 0.$$

This can only be true if we have both

$$\frac{-(d_D)_j}{(x_B)_j} < \frac{-(d_D)_k}{(x_B)_k} \quad \text{and} \quad \frac{-\gamma(d_D)_j}{(x_B)_j} < \frac{-\gamma(d_D)_k}{(x_B)_k},$$

which can only be true if $\gamma > 0$ and

$$(d_D)_j < 0 \quad \text{and} \quad (d_D)_j > \frac{1}{\gamma}(x_B)_j$$
$$(d_D)_k < 0 \quad \text{and} \quad (d_D)_k > \frac{1}{\gamma}(x_B)_k.$$

We know that that all of these conditions cannot be satisfied simultaneously if $\gamma > 0$, as we know $x_B > 0$. Therefore, both algorithms select the same index r, thus choosing the same basic x variable to become nonbasic.

At this point, the basic variable x_r is dropped from the basis in both algorithms, moving index r from the basic to the nonbasic index set. In the quadratic programming algorithm, the next outgoing (blocking) basic variable must be determined, as x_s has not yet been brought into the basis. In the least-squares algorithm, things are slightly different because the incoming variable x_s is already basic. However, once x_r is dropped from the basis and a new value for

y is computed by the least-squares algorithm, we are back in the same situation as described earlier. This time we have a smaller set of basic variables, and by the same argument given above, both algorithms will again select the same basic x variable to leave the basis. Thus we may conclude that given the same initial index sets α and β, the same incoming column is selected, and the same series of variables is dropped from the basis. Once all necessary basic columns have been dropped, the quadratic programming algorithm finally brings x_s into the basis, and both algorithms again share the same sets of basic and nonbasic indices.

Therefore we see that the only difference between the two algorithms is the fact that after x_s has been successfully introduced into the basis (and all dropping of basic x variables is complete), the least-squares algorithm takes all basic variables with a zero value and makes them nonbasic, while the quadratic programming algorithm does not. In this degenerate situation, the two algorithms can develop different index sets, and can thus take a different path to find a different final solution.

Notice that we have also just proved that this quadratic programming algorithm is equivalent to the NNLS algorithm in Lawson & Hanson [12].

3 Computational Results

In this section, we consider the actual performance of the algorithm developed in Section 2. All computational results have been obtained using the 30 smallest linear programming test problems available from *NETLIB* [7]. These test problems were converted to the form of (2-1), with the addition that free variables were allowed (although nonpositive variables were not). This produced a set of equivalent problems with modified dimensions; see Table 3-1 for the original and modified dimensions. For bitmaps of the nonzero patterns of many of the original problems, see [13].

The test environment consisted of the following hardware and software:

Computer:	HP 9000/835
	32M main memory
	100M virtual memory
Operating System:	HP-UX 7.0
Language:	HP FORTRAN, 64-bit real numbers

The run times reported here include only computation time, as the amount of main memory used virtually eliminated swapping, and I/O time was negligible.

3.1 The Least-Squares Phase I Algorithm

This section considers the computational results of the least-squares Phase I algorithm and its variations, as developed in Section 2. See Sections 2.8 and 2.9 for pseudo code of the algorithm, and a discussion of the variations. We do not

	Original		Converted	
Problem(Name)	Rows	Columns	Rows	Columns
1(AFIRO)	28	32	28	51
2(SC50A)	51	48	51	78
3(SC50B)	51	48	51	78
4(ADLITTLE)	57	97	57	138
5(BLEND)	75	83	75	114
6(SHARE2B)	97	79	97	162
7(SC105)	106	103	106	163
8(STOCFOR1)	118	111	118	165
9(RECIPE)	92	180	212	298
10(SCAGR7)	130	140	130	185
11(BOEING2)	167	143	244	382
12(ISRAEL)	175	142	175	316
13(SHARE1B)	118	225	118	253
14(VTP.BASE)	199	203	346	475
15(SC205)	206	203	206	317
16(GROW7)	141	301	421	581
17(BEACONFD)	174	262	174	295
18(BRANDY)	221	249	221	303
19(SCSD1)	78	760	78	760
20(E226)	224	282	224	472
21(FORPLAN)	162	421	187	514
22(BORE3D)	234	315	247	346
23(AGG)	489	163	489	615
24(CAPRI)	272	353	419	613
25(SCORPION)	389	358	389	466
26(BANDM)	306	472	306	472
27(SCTAP1)	301	480	301	660
28(SCFXM1)	331	457	331	600
29(STAIR)	357	467	445	620
30(SCSD6)	148	1350	148	1350

Table 3-1: Size of Test Problems

consider the "pure" algorithm at all, as the variation allowing free variables is always more efficient when they are present. Thus we will consider the following four least-squares algorithms:

P1F: The least-squares Phase I algorithm, in which free variables have not been explicitly converted to pairs of nonnegative variables.

P1D1F: The same as P1F, with the exception that the denominator of the column selection rule is replaced by "1".

CBP1F: The same as P1F, using a crash basis as described in Section 2.9.

CBP1D1F: The same as P1D1F, using a crash basis as described in Section 2.9.

All of these least-squares Phase I algorithms were implemented using dense matrix methods. QR factorization was used to solve the embedded least-squares subproblems, and this factorization was updated using Givens rotations as columns were added and dropped from the basis. See for example [9] for details on using the QR factorization to solve least-squares problems, and for details on Givens rotations (also known as plane rotations).

The least-squares algorithms are compared with LSSOL, an established package. LSSOL is written in Fortran 77 using dense matrix techniques, and solves a class of linearly constrained quadratic programming problems. See [8] for a more detailed description of LSSOL. In order to compare our Phase I algorithm to LSSOL, we use the "FS" option. This causes LSSOL to find a feasible solution to the problem submitted, using its implementation of Phase I of the simplex method. LSSOL (FS option) will be denoted by FS in the following result tables, and by LSSOL:FS in the text.

When looking at the least-squares Phase I algorithms, we see that there is no clear-cut definition for an iteration, because the loop in which columns are dropped from the basis could be executed a large number of times between column selections. We decided to define an iteration to occur whenever a least-squares problem is solved. This means that adding a single column to the basis is considered to be an iteration, as is dropping a column by forming a convex combination (see Step 3 of the algorithm). In addition, consider the situation in which some elements of the solution are 0, and are consequently dropped. Although no least-squares problem is solved in this case, enough work is performed in matrix updates, etc., that this is also considered to be an iteration.

Table 3-2 displays the iteration counts of the four least-squares Phase I algorithms and LSSOL:FS. Note that the iteration counts for some of the problems are the same for both the non-crash-basis and crash-basis versions of the least-squares Phase I algorithm (e.g. STOCFOR1). This is due to the fact that not all problems contain columns of the form we look for when building the crash basis. Also notice that some of the iteration counts are 0. This indicates the situation when the initial crash basis provides a feasible solution, and no iterations are necessary.

We see from Table 3-2 that LSSOL:FS takes a number of iterations comparable to that taken by the least-squares Phase I algorithms, with the total lying between P1D1F and CBP1F. As stated earlier, the notion of an iteration in the least-squares Phase I algorithms is somewhat nonstandard. For this reason, actual run times are probably a more accurate measure of the least-squares algorithms' performance than their iteration counts.

Table 3-3 displays the run times of the four least-squares Phase I algorithms and LSSOL:FS, reported in CPU-seconds. Note that a number of runtimes are reported as "0". This indicates that the CPU-times recorded for these problems round down to less than one CPU-second. We see from Table 3-3 that the least-squares algorithms run in substantially less time than does LSSOL:FS;

Problem(Name)	P1F	P1D1F	CBP1F	CBP1D1F	FS
1(AFIRO)	7	7	0	0	8
2(SC50A)	20	20	0	0	3
3(SC50B)	5	5	0	0	0
4(ADLITTLE)	56	56	27	27	70
5(BLEND)	8	8	0	0	49
6(SHARE2B)	248	248	229	217	74
7(SC105)	34	34	0	0	2
8(STOCFOR1)	98	98	98	98	44
9(RECIPE)	106	106	37	37	46
10(SCAGR7)	169	169	145	145	22
11(BOEING2)	206	204	163	161	181
12(ISRAEL)	171	171	8	8	338
13(SHARE1B)	189	189	176	176	150
14(VTP.BASE)	770	561	488	507	1059
15(SC205)	61	61	0	0	2
16(GROW7)	280	280	0	0	203
17(BEACONFD)	122	122	89	89	52
18(BRANDY)	365	365	340	340	300
19(SCSD1)	10	8	10	8	16
20(E226)	203	203	154	154	209
21(FORPLAN)	378	378	347	347	438
22(BORE3D)	175	175	164	164	126
23(AGG)	543	539	134	136	149
24(CAPRI)	586	590	440	444	451
25(SCORPION)	321	321	293	293	27
26(BANDM)	484	485	419	419	369
27(SCTAP1)	440	411	322	349	430
28(SCFXM1)	350	343	300	293	271
29(STAIR)	452	452	464	464	173
30(SCSD6)	207	68	207	68	61
TOTAL	7064	6677	5054	4944	5323

Table 3-2: Least-Squares Phase I Algorithm Iteration Counts

the LSSOL:FS total time is almost 3.5 times longer than the total of the best least-squares Phase I algorithm (CBP1D1F). In fact, LSSOL:FS solved only one problem (SCAGR7) faster than did CBP1D1F.

Looking at the least-squares algorithms more closely, the "denominator-of-1" versions (P1D1F and CBP1D1F) are quite similar to their "full denominator" counterparts (P1F and CBP1F). However, the crash basis versions (CBP1F and CBP1D1F) are noticeably faster than the "from scratch" versions (P1F and P1D1F).

So far, we have discussed iteration count and run times. We will now consider in some sense the path taken by the least-squares Phase I algorithms. In particular, we will consider the norm of the residual (error vector) as a function of iteration count. For the least-squares Phase I algorithms, this is $\| u \|$, where u is as defined in the pseudocode algorithm found in Section 2.8

Problem(Name)	P1F	P1D1F	CBP1F	CBP1D1F	FS
1(AFIRO)	0	0	0	0	1
2(SC50A)	0	0	0	0	1
3(SC50B)	0	0	0	0	1
4(ADLITTLE)	2	2	1	1	5
5(BLEND)	0	1	0	0	6
6(SHARE2B)	12	12	11	11	15
7(SC105)	2	2	1	1	5
8(STOCFOR1)	7	7	7	7	16
9(RECIPE)	21	21	10	10	61
10(SCAGR7)	15	15	15	15	14
11(BOEING2)	61	59	48	47	209
12(ISRAEL)	32	30	5	4	227
13(SHARE1B)	18	17	17	18	51
14(VTP.BASE)	416	311	271	277	2026
15(SC205)	11	11	2	2	34
16(GROW7)	207	209	11	11	801
17(BEACONFD)	20	21	18	17	42
18(BRANDY)	79	78	80	75	208
19(SCSD1)	3	3	4	3	7
20(E226)	63	61	51	51	196
21(FORPLAN)	98	95	90	90	398
22(BORE3D)	48	47	47	46	153
23(AGG)	676	686	182	181	889
24(CAPRI)	545	549	472	478	1438
25(SCORPION)	182	182	172	173	277
26(BANDM)	244	242	224	223	571
27(SCTAP1)	274	263	221	229	695
28(SCFXM1)	223	222	202	199	530
29(STAIR)	545	544	586	589	816
30(SCSD6)	130	41	128	41	45
TOTAL	3934	3731	2876	2799	9738

Table 3-3: Least-Squares Phase I Algorithm Run Times

Figure 3-1 illustrates the typical behavior of $\| u \|$ in P1F. We chose to display the behavior of only one of the four least-squares Phase I algorithms because in this aspect, they were virtually identical. Notice that P1F finds a solution "close" to feasibility very quickly, spending the vast majority of its iterations reducing $\| u \|$ from 0.1 to 0.

The last comparison we will make between the least-squares Phase I algorithms and LSSOL:FS will be based on the accuracy of the final solution found. When each algorithm found an $\hat{x}$ that it claimed to be a feasible solution (or as close as possible to such a solution), the error quantity $\| b - A\hat{x} \|$ was formed and reported. These errors in the final solution are displayed in Table 3-4.

In most problems, there is very little difference in accuracy between the least-squares Phase I algorithms and LSSOL:FS. However, in the case of FORPLAN, the error of the least-squares algorithms is quite large relative to the error of

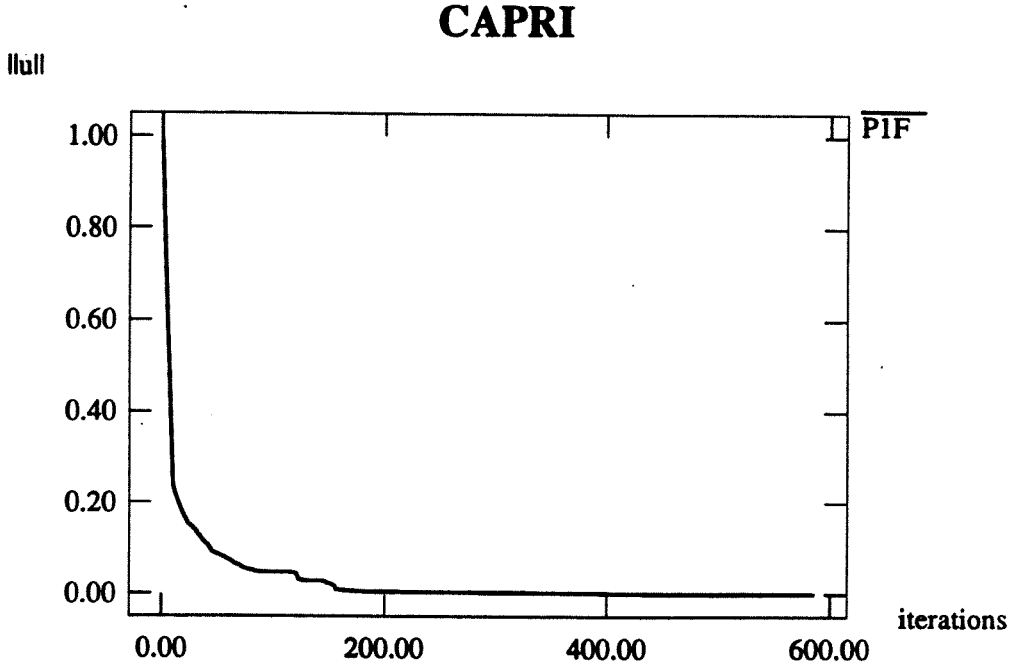

Figure 3-1: Residual Behavior of P1F on CAPRI

LSSOL:FS. This is not surprising, as the error in the computed solution $\hat{x}$ in the least-squares algorithms depends on the square of the condition number of the matrix of currently basic columns (except when B is square). (See, for example [3].) This inaccuracy can occasionally cause least-squares Phase I algorithms to report infeasibility incorrectly.

It may be possible to handle this lack of accuracy by adding iterative refinement to the end of the least-squares Phase I algorithms. That is, we take the (previously final) solution $\hat{x}$ and let $x^0 = \hat{x}$. Let B be the matrix of currently basic columns. We then perform some (small) number of iterations

$$\begin{aligned} u^i &= b - Bx^i \\ B\delta x^i &= u^i \\ x^{i+1} &= x^i + \delta x^i. \end{aligned}$$

In practice, we have the QR factorization of B as opposed to B, and B is generally rectangular, so we would actually perform iterative refinement as follows:

$$\begin{aligned} u^i &= b - Q\begin{pmatrix} R \\ 0 \end{pmatrix} x^i \\ \begin{pmatrix} R \\ 0 \end{pmatrix} \delta x^i &= Q^T u^i \end{aligned}$$

Problem(Name)	P1F	P1D1F	CBP1F	CBP1D1F	FS
1(AFIRO)	0	0	0	0	5e-11
2(SC50A)	7e-14	7e-14	0	0	5e-13
3(SC50B)	0	0	0	0	4e-14
4(ADLITTLE)	3e-13	3e-13	2e-13	2e-13	7e-12
5(BLEND)	0	0	0	0	9e-12
6(SHARE2B)	2e-12	1e-12	1e-12	2e-12	6e-11
7(SC105)	4e-13	4e-13	0	0	2e-12
8(STOCFOR1)	4e-14	4e-14	4e-14	4e-14	2e-10
9(RECIPE)	1e-14	1e-14	1e-14	1e-14	3e-13
10(SCAGR7)	5e-12	8e-12	6e-12	7e-12	2e-10
j1(BOEING2)	1e-11	3e-11	4e-11	9e-12	1e-10
12(ISRAEL)	3e-10	5e-10	1e-10	1e-10	5e-09
13(SHARE1B)	3e-10	3e-10	2e-10	2e-10	9e-09
14(VTP.BASE)	7e-10	8e-10	1e-09	2e-09	9e-09
15(SC205)	1e-12	1e-12	0	0	3e-11
16(GROW7)	0	0	0	0	1e-08
17(BEACONFD)	6e-11	6e-11	6e-11	6e-11	6e-10
18(BRANDY)	3e-12	3e-12	1e-10	1e-10	5e-11
19(SCSD1)	3e-16	2e-16	3e-16	2e-16	2e-15
20(E226)	5e-13	2e-13	7e-14	7e-14	2e-11
21(FORPLAN)	8e-01	8e-01	8e-01	8e-01	9e-10
22(BORE3D)	7e-12	7e-12	7e-12	7e-12	3e-09
23(AGG)	1e-07	1e-07	1e-09	1e-09	1e-08
24(CAPRI)	6e-11	6e-11	1e-10	1e-10	3e-09
25(SCORPION)	2e-15	2e-15	2e-15	2e-15	2e-14
26(BANDM)	1e-11	6e-11	2e-11	1e-11	4e-11
27(SCTAP1)	6e-12	3e-12	4e-12	7e-12	5e-12
28(SCFXM1)	3e-12	2e-11	2e-11	2e-11	9e-10
29(STAIR)	7e-13	6e-13	2e-12	2e-12	3e-09
30(SCSD6)	1e-15	2e-15	1e-15	2e-15	2e-14

Table 3-4: Least-Squares Phase I Algorithm Accuracy

$$x^{i+1} = x^i + \delta x^i.$$

As long as Q is stored, this would not be too much work, especially if only a few iterations were necessary. Note that this scheme for iterative refinement depends on $Bx = b$ being nearly feasible. This may not be the case if we have accumulated a significant amount of error. See [2] for more details, as well as for the description of a method of iterative refinement that is not dependent on the feasibility of $Bx = b$.

To summarize, the four least-squares Phase I algorithms have very favorable run times relative to the simplex method as implemented by LSSOL, with the "crash basis" versions performing the best. They also find a solution "close" to feasibility very quickly. However, due to the least-squares subproblems embedded in the four Phase I algorithms, accuracy can be a problem on larger, more ill-conditioned problems. It is possible that iterative refinement of the final

Problem(Name)	P1F	P1D1F	CBP1F	CBP1D1F	LS
1(AFIRO)	7	7	0	0	22
2(SC50A)	20	20	0	0	38
3(SC50B)	5	5	0	0	26
4(ADLITTLE)	56	56	27	27	148
5(BLEND)	8	8	0	0	17
6(SHARE2B)	248	248	229	217	243
7(SC105)	34	34	0	0	88
8(STOCFOR1)	98	98	98	98	149
9(RECIPE)	106	106	37	37	204
10(SCAGR7)	169	169	145	145	251
11(BOEING2)	206	204	163	161	925
12(ISRAEL)	171	171	8	8	836
13(SHARE1B)	189	189	176	176	409
14(VTP.BASE)	770	561	488	507	2499
15(SC205)	61	61	0	0	218
16(GROW7)	280	280	0	0	620
17(BEACONFD)	122	122	89	89	214
18(BRANDY)	365	365	340	340	598
19(SCSD1)	10	8	10	8	35
20(E226)	203	203	154	154	399
21(FORPLAN)	378	378	347	347	436
22(BORE3D)	175	175	164	164	385
23(AGG)	543	539	134	136	608
24(CAPRI)	586	590	440	444	778
25(SCORPION)	321	321	293	293	320
26(BANDM)	484	485	419	419	874
27(SCTAP1)	440	411	322	349	1210
28(SCFXM1)	350	343	300	293	520
29(STAIR)	452	452	464	464	679
30(SCSD6)	207	68	207	68	178
TOTAL	7064	6677	5054	4944	13927

Table 3-5: Phase I Algorithm and LSSOL:LS Iteration Counts

solution could alleviate such difficulties, but this variation was not implemented.

3.2 A Constrained Least Squares Problem

So far we have considered computation results on problem (2-1). However, as we noted in Section 2.10, our least-squares Phase I algorithm actually solves (2-15), a constrained least-squares problem. Thus we decided to compare our Phase I algorithm with that of LSSOL when applied to (2-15) (by using the "LS" option). LSSOL (LS option) will be denoted by LS in the following result tables, and by LSSOL:LS in the text.

Table 3-5 displays the iteration counts of the least-squares Phase I algorithms considered in Section 3.1 and LSSOL:LS, and Table 3-6 displays the run times. Again, the least-squares Phase I algorithms do well, with the best, CBP1D1F,

Problem(Name)	P1F	P1D1F	CBP1F	CBP1D1F	LS
1(AFIRO)	0	0	0	0	1
2(SC50A)	0	0	0	0	1
3(SC50B)	0	0	0	0	1
4(ADLITTLE)	2	2	1	1	5
5(BLEND)	0	1	0	0	2
6(SHARE2B)	12	12	11	11	11
7(SC105)	2	2	1	1	7
8(STOCFOR1)	7	7	7	7	10
9(RECIPE)	21	21	10	10	49
10(SCAGR7)	15	15	15	15	19
11(BOEING2)	61	59	48	47	214
12(ISRAEL)	32	30	5	4	109
13(SHARE1B)	18	17	17	18	31
14(VTP.BASE)	416	311	271	277	812
15(SC205)	11	11	2	2	51
16(GROW7)	207	209	11	11	409
17(BEACONFD)	20	21	18	17	55
18(BRANDY)	79	78	80	75	109
19(SCSD1)	3	3	4	3	47
20(E226)	63	61	51	51	151
21(FORPLAN)	98	95	90	90	174
22(BORE3D)	48	47	47	46	120
23(AGG)	676	686	182	181	608
24(CAPRI)	545	549	472	478	529
25(SCORPION)	182	182	172	173	217
26(BANDM)	244	242	224	223	409
27(SCTAP1)	274	263	221	229	711
28(SCFXM1)	223	222	202	199	400
29(STAIR)	545	544	586	589	573
30(SCSD6)	130	41	128	41	342
TOTAL	3934	3731	2876	2799	6177

Table 3-6: Phase I Algorithm and LSSOL:LS Run Times

running in 45% of the total time taken by LSSOL:LS. Apparently, our least-squares Phase I algorithms are also efficient in solving this particular kind of constrained least-squares problem.

4 Summary and Conclusions

Section 2 developed a Phase I algorithm using least-squares subproblems. Subproblems are solved to provide both an approximation to the right-hand side that is as close as possible in the 2-norm (given the current basic columns), and to select the next incoming column. The incoming column-selection rule is myopic, selecting that column whose nonnegative combination with the current approximation brings us closest to the right-hand side. This Phase I algorithm

has the property of strict improvement at each iteration, even in the presence of degeneracy (in the simplex method sense). Finally, equivalences between the least-squares Phase I algorithm and other algorithms were discussed.

Four variations of this least-squares Phase I algorithm were developed, and the computational results were excellent. The best least-squares Phase I algorithm ran almost 3.5 times faster than LSSOL's implementation of Phase I of the simplex method. In addition, the same algorithm ran 2.2 times faster than LSSOL's constrained LS solver, when applied to problems of the form (2-15).

Future Work

We have demonstrated the initial proof-of-concept of the Phase I algorithm developed in Section 2. In order for this algorithm to be commercially competitive, it must be fine-tuned, just as the simplex method has been since its discovery. In particular, the problem of inaccuracy on unstable problems must be addressed, perhaps with the inclusion of iterative refinement of the solutions to the embedded least-squares subproblems. In addition, these algorithms should be implemented using sparse matrix methods to see how they compare to sparse matrix implementations of the simplex method.

References

[1] **E.M.L. Beale (1955)**, "Cycling in the Dual Simplex Algorithm", *Naval Res. Logist. Quart.*, 2(4), 1955, pp.269–276.

[2] **Å. Björck (1967/68)**, "Iterative Refinement of Linear Least Squares Solutions", *BIT* 7, pp. 257–278 and *BIT* 8, pp. 8–30.

[3] **Å. Björck (1987)**, "Least Square Methods", Working Paper, Department of Mathematics, Linköping University, S-581 83 Linköping, Sweden, 1987.

[4] **R.W. Cottle, J.S.Pang & R.E. Stone**, *The Linear Complementarity Problem*, Academic Press, forthcoming.

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

[6] **D. Gale (1960)**, *Theory of Linear Economic Models*, McGraw-Hill, New York.

[7] **D.M. Gay (1985)**, "Electronic Mail Distribution of Linear Programming Test Problems", *Mathematical Programming Society COAL Newsletter* 13, pp.10–12.

[8] **P.E. Gill, S.J. Hammarling, W. Murray, M.A. Saunders & M.H. Wright (1986)**, "User's Guide for LSSOL", Technical Report SOL 86–1, Department of Operations Research, Stanford University.

[9] **P.E. Gill, W. Murray & M.H. Wright (1981)**, *Practical Optimization*, Academic Press, San Francisco.

[10] **Goldfarb & Idnani(1983)**, "A Numerically Stable Dual Method for Solving Strictly Convex Quadratic Programs", *Math. Prog.* 27, pp.1–33.

[11] **A.J. Hoffman (1953)**, "Cycling in the Simplex Algorithm", National Bureau of standards, Report No. 2974, December 16, 1953, 7 pp.

[12] **C.L. Lawson & R.J. Hanson (1974)**, *Solving Least-Squares Problems*, Prentice-Hall, Englewood Cliffs, New Jersey.

[13] **I.J. Lustig (1987)**, "An Analysis of an Available Set of Linear Programming Test Problems", Technical Report SOL 87-11, August 1987, Department of Operations Research, Stanford University.

[14] **C. Van de Panne & A. Whinston (1969)**, "The Symmetric Foundation of the Simplex Method for Quadratic Programming", *Econometrica*, Vol. 37, pp.507–527.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	April 1992	Technical Report

4. TITLE AND SUBTITLE

A Strictly Improving Phase I Algorithm Using Least-Squares Subproblems

5. FUNDING NUMBERS

N00014-89-J-1659
DE-FG03-92ER25116

6. AUTHOR(S)

S. A. Leichner, G. B. Dantzig and J. W. Davis

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Department of Operations Research - SOL
Stanford University
Stanford, CA 94305-4022

8. PERFORMING ORGANIZATION REPORT NUMBER

1111MA

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Office of Naval Research - Department of the Navy
800 N. Quincy Street
Arlington, VA 22217

Office of Energy Research
U.S. Department of Energy
Washington, DC 20585

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

SOL 92-1

12a. DISTRIBUTION/AVAILABILITY STATEMENT

UNLIMITED

12b. DISTRIBUTION CODE

UL

13. ABSTRACT (Maximum 200 words)

Although the simplex method's performance in solving linear programming problems is usually quite good, it does not guarantee strict improvement at each iteration on degenerate problems. Instead of trying to recognize and avoid degenerate steps in the simplex method (as some variants do), we have developed a new Phase I algorithm that is completely impervious to degeneracy, with a strict improvement attained at each iteration. It is also noted that the new Phase I Algorithm is closely related to a number of existing algorithms.

When tested on the 30 smallest *NETLIB* linear progragming test problems, the computational results for the new Phase I algorithm were almost 3.5 times faster than the simplex method; on some problems, it was over 10 times faster.

14. SUBJECT TERMS

Linear programming; least squares; strict improvement; Phase I.

15. NUMBER OF PAGES

43 pages

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED			SAR