To the Graduate Council:

I am submitting herewith a thesis written by Darren A. Smith entitled "Shadow Prices in Linear Programs." I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Management Science.

James K. Ho, Major Professor

We have read this thesis
and recommend its acceptance:

Kenneth C. Gilbert

Dale R. Fox

Accepted for the Council:

Vice Provost
and Dean of the Graduate School

STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a Master's degree at The University of Tennessee, Knoxville, I agree that the Library shall make it available to borrowers under rules of the Library. Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgment of the source is made.

Permission for extensive quotation from or reproduction of this thesis may be granted by my major professor, or in his absence, by the Head of Interlibrary Services when, in the opinion of either, the proposed use of the material is for scholarly purposes. Any copying or use of the material in this thesis for financial gain shall not be allowed without my written permission.

Signature Dawn A. Smith

Date 11-6-87

SHADOW PRICES IN LINEAR PROGRAMS

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Darren A. Smith

December 1987

ABSTRACT

It is well known that in linear programming, the optimal values of the dual variables can be interpreted as shadow prices (marginal values) of the right-hand-side coefficients. However, this is true only under nondegeneracy assumptions. Since real problems are often degenerate, the output from conventional LP software regarding such marginal information can be misleading. This thesis surveys and generalizes known results in this topic and demonstrates how true shadow prices can be computed with or without modification to existing software.

TABLE OF CONTENTS

CHAPTER	PAGE
1. INTRODUCTION	1
2. PREVIOUS RESULTS	2
3. THE GENERAL CASE	12
4. ALGORITHMS	19
5. A NUMERICAL EXAMPLE	22
6. COMPUTING WITH CONVENTIONAL LP CODES	25
7. COMPUTING WITH ENHANCED LP CODES	35
8. MISCELLANEOUS TOPICS	44
9. DISCUSSION	47
BIBLIOGRAPHY	48
APPENDIXES	51
APPENDIX A. PLI SUBROUTINES.	52
APPENDIX B. SAMPLE OUTPUT FROM OTHER EXAMPLES	62
VITA	66

LIST OF TABLES

TABLE	PAGE
1. Interpretation of Shadow Prices	13
2. Extreme Values in Y* for Example LP	23
3. Shadow Prices for the Example LP	24
4. LINDO Output for Example LP	26
5. LINDO PARA Output for First Constraint	27
6. LINDO PARA Output for Last Constraint	28
7. MPSX Row at Limit Level Range Output	31
8. Procedures in the Experimental LP Code	38
9. Output from Experimental LP Code	43

CHAPTER 1

INTRODUCTION

In most elementary treatments of linear programming, such as typically found in textbooks on Management Science and Operations Research, the dual variables of an LP are interpreted as marginal values of the right-hand-side coefficients. As the latter often represent resources of limited supply, such marginal values have come to be known as shadow prices. They indicate how much additional units of the corresponding resources are worth. However, this equivalence between dual variables and shadow prices holds only under the assumption of nondegeneracy. Their nonequivalence in general, while perhaps well known among specialists, is almost never discussed in textbooks for students and would be practitioners. (Exceptions are Murty[14] and Shapiro[19]). Even commercial software for LP fails to alert users of this warning. As a result, misleading outputs of LP models may have resulted in many inadvertent misuses of the approach. The purpose of this thesis is to summarize known results on this topic that have appeared in the literature, generalize them to handle any LP, examine how true shadow prices can be computed with existing software, and show extensions to LP software necessary for the automatic generation of such prices.

CHAPTER 2

PREVIOUS RESULTS

For conciseness, only theoretical results in the literature, well known or otherwise, that are essential to the computation of shadow prices are summarized. Related topics such as uniqueness of solutions and degeneracy of LP's can be found in other works in the bibliography (Greenberg[9],Mangasarian[13],Pérold[16]).

Consider the following primal-dual pair of linear programs:

$$
\begin{array}{lll}
 & \text{Maximize} & cx \\
(P) & \text{subject to} & Ax \leq b \\
 & & x \geq 0;
\end{array}
$$

$$
\begin{array}{lll}
 & \text{Minimize} & yb \\
(D) & \text{subject to} & yA \geq c \\
 & & y \geq 0;
\end{array}
$$

where A is m x n, $c:R^n, x:R^n, b:R^m, y:R^m$. (: means belongs to)

Definition 1. Denote the optimal objective value for (P), as a function of the right-hand-side, by

$$v(b) = \max\{cx \mid Ax \leq b; x \geq 0\};$$

the set of feasible solutions and the set of optimal solutions to (P) by

$$X = \{x:R^n \mid Ax \leq b; x \geq 0\};$$

$$X^* = \{x:X \mid cx = v(b)\} \text{ respectively; and}$$

the set of feasible solutions and the set of optimal solutions to (D) by

$$Y = \{y:R^m \mid yA >= c; y >= 0\},$$

$$Y^* = \{y:Y \mid yb = v(b)\}.$$

Proposition 1.

v(b) is a non-decreasing, piecewise linear concave function.

Proof. (See, e.g. Bradley et al.[5], Shapiro[19])

We need to show that

i) v(b) is non-decreasing,

ii) v(b) is piecewise linear,

iii) v(b) is concave.

For (i):

Let $\tilde{b}_i = b_i + d_i$ for $d:R^m$ and $d_i > 0$ for $i = 1$ to m. Since substituting $\tilde{b}$ in (P) relaxes the constraints, we have $v(\tilde{b}) >= v(b)$. Therefore v(b) is non-decreasing.

(ii):

Consider the following parametrization of (P):

P(d)

$$\begin{aligned} \max \quad & cx \\ \text{s.t.} \quad & Ax <= b_1 + db_2 \\ & x >= 0; \\ & \text{where } d:R. \end{aligned}$$

As d increases or decreases basis changes will take

place. On any successive interval where the basis is unchanged the value of P(d) is given by

$$y(b_1 + db_2). \qquad (1)$$

In (1) $y = c_B B^{-1}$, where B is the current optimal basis and c_B are the objective coefficients of the basic variables, is not a function of d. Rewriting (1) as $yb_1 + yb_2 d$ it is clear that P(d) is piecewise linear with slope yb_2 on a particular segment. Thus v(b) is piecewise linear.

For (iii):

Let b_1 and b_2 be any points in R^m.

If $v(b_1)$ or $v(b_2)$ = -infinity, then the result follows trivially. If $v(b_1)$ or $v(b_2)$ = infinity, then the dual has no feasible solution which implies v(b) = infinity for all b belonging to R^m.

Now let b_1 and b_2 satisfy $v(b_1) = cx_1$ and

$$v(b_2) = cx_2;$$

with x_1, x_2 satisfying $Ax <= b$, $x >= 0$.

For $0 <= t <= 1$,

$$A[tx_1 + (1 - t)x_2] <= tb_1 + (1 - t)b_2;$$

$$tx_1 + (1 - t)x_2 >= 0. \qquad (2)$$

From (2), $tx_1 + (1 - t)x_2$ is feasible for (P) when $b = tb_1 + (1 - t)b_2$ but it may not be optimal.

Therefore $v(tb_1 + (1 - t)b_2) >= c(tx_1 + (1 - t)x_2)$

$$= t(cx_1) + (1 - t)cx_2$$

$$= tv(b_1) + (1 - t)v(b_2).$$

Definition 2. The set of subgradients for a concave function $f: R^m \to R$ at b is defined as

$$Df(b) = \{y:R^m \mid f(b+u) \leq f(b) + yu, \text{ for all } u:R^m\}.$$

Proposition 2.

When v(b) is finite, $Dv(b) = Y^*$.

Proof. (See, e.g. Murty[14],Akgul[2])

We need to show that for $y:R^m$ and $b:R^m$,

i) if y is an optimal dual solution, then y is a subgradient of v(b); and

ii) if y is a subgradient vector for v(b), then y is an optimal dual solution.

For (i):

If y_1 is optimal for (D) then

$$v(b_1) = y_1 b_1.$$

Since y_1 is feasible but not necessarily optimal to (D) for some $b:R^m$ we have $v(b) \leq y_1 b$, which can be written as

$$v(b) \leq v(b_1) + y_1(b - b_1).$$

Thus y_1 is a subgradient vector of v(b) at the point b_1.

For (ii):

We need to show that

a) $v(b) = yb$;

b) $yA \geq c$;

c) $y \geq 0$.

By hypothesis,

$$v(b) \leq v(b_1) + y(b - b_1) \text{ for all } b:R^m. \qquad (3)$$

(a)

Since v(b) is finite we have v(0) = 0.

Also v(tb) = tv(b) for t >= 0.

Now let b = 0,

$$v(0) <= v(b_1) + y(0 - b_1) \quad (4)$$

$$0 <= v(b_1) - yb_1$$

$$yb_1 <= v(b_1). \quad (5)$$

Letting $b = 2b_1$,

$$v(2b_1) <= v(b_1) + y(2b_1 - b_1) \quad (6)$$

$$2v(b_1) <= v(b_1) + y(b_1)$$

$$v(b_1) <= yb_1 \quad (7)$$

Therefore, based on (5) and (7) we have

$$v(b_1) = yb_1. \quad (8)$$

(b)

Substituting (8) in the subgradient inequality,(3), we have

$$v(b) <= yb. \quad (9)$$

If $b = A_{.j}$ is substituted in (9) then

$$v(A_{.j}) <= yA_{.j}. \quad (10)$$

But $v(A_{.j}) >= c_j$, (11)

since $x = I_{.j}$ is a feasible but not necessarily optimal solution to (D) when $b = A_{.j}$.

Combining (10) and (11) gives

$$c_j <= yA_{.j} \quad \text{for all } j = 1 \text{ to } n, \text{ i.e. } yA >= c. \quad (12)$$

(c)

In (3), let $b_1 = b$ and $b = b + u$.

Then (3) becomes as stated in defintion 2:

$$v(b + u) <= v(b) + yu. \tag{13}$$

Let $u = e_i$.

Then (13) becomes

$$v(b + e_i) <= v(b) + y_i. \tag{14}$$

Note that $v(b + e_i) >= v(b)$ (15)

since we are relaxing the constraints.

Combining (14) and (15) we have:

$v(b) <= v(b + e_i) <= v(b) + y_i$ which implies

$$y_i >= 0 \text{ for all } i. \tag{16}$$

Thus based on (8), (12), and (16) y is also an optimal dual solution.

Definition 3. The directional derivative of a function $f:R^m \to R$ at b in the direction of u is defined as

$$D_u f(b) = \lim[t \to 0^+] \ (f(b+tu) - f(b)) / t.$$

Proposition 3.

$$D_u v(b) = \min \{yu \mid y:Y^*\}.$$

Proof. (see e.g. Gauvin[8])

$v(b + tu) = \max\{cx \mid Ax <= b + tu; x >= 0\}$

$= \min\{y(b + tu) \mid yA >= c; y >= 0\}$

$= \min\{yb \mid yA >= c; y >= 0\} + t \min\{yu \mid yA >= c; y >= 0\}$

$= v(b) + t \min\{yu \mid y:Y^*\}.$

Therefore:

$$D_u v(b) = \lim[t \to 0^+] \{(v(b + tu) - v(b)) / t\}$$

$$= \lim[t \to 0^+] \{(v(b) + t \min(yu \mid y:Y^*) - v(b)) / t\}$$

$$= \min\{yu \mid y:Y^*\}.$$

Definition 4. For (P), the buying (or positive) shadow price of constraint i is defined as

$$p_i^+ = D_{e(i)} v(b) \text{ and}$$

the selling (or negative) shadow price of constraint i is defined as

$$p_i^- = -D_{-e(i)} v(b)$$

where e(i) is the i^{th} unit vector.

In other words, the buying shadow price is the (instantaneous) rate of change in v(b) for an increase in b_i and the selling shadow price is the negative of the (instantaneous) rate of change in v(b) for a decrease in b_i.

Proposition 4.

$$p_i^+ = \min \{y_i \mid y:Y^*\}$$

$$p_i^- = \max \{y_i \mid y:Y^*\}.$$

Proof. From Proposition 3,

$$D_u v(b) = \min\{yu \mid y:Y^*\}.$$

Therefore $p_i^+ = D_{e(i)} v(b) = \min\{y_i \mid y:Y^*\}$;

$$p_i^- = -D_{-e(i)} v(b) = -\min\{-y_i \mid y:Y^*\}$$

$$= \max\{y_i \mid y:Y^*\}.$$

Definition 5. Denote the index set of the basic variables in an optimal tableau for (P) by B, the index set of the nonbasic variables by N, the index set of slack variables by S. Denote the row of reduced costs by **c**, the right-hand-side by **b**, the ith row and jth column of the tableau by $\hat{a}^i$ and $\hat{a}_j$ respectively. Let $\mathbf{c}_S$ and $\hat{a}_S$ be **c** and $\hat{a}^i$ restricted to S respectively. Let $T= \{i|\mathbf{b}_i = 0\}$ be the index set for the rows with degeneracy and y* be the current dual optimal solution, i.e. $y^* = \mathbf{c}_S$.

Proposition 5. (Akgul[1],Best[4])

Y^* is characterized by $y:Y^* \iff y = y^* + SUM\{t_k \hat{a}^k{}_S | k:T\}$ for some $t:R^{|T|}$ such that

$$\mathbf{c} + SUM\{t_k \hat{a}^k | k:T\} >= 0.$$

Proof. (Akgul[2])

Consider an optimal tableau for (P):

	B		N	**b**	
c	0	0	>= 0		
	I_1	0	$\mathbf{A}_1$	0	T
	0	I_2	$\mathbf{A}_2$	>0	

I)

Let $\tilde{c} = \mathbf{c} + SUM\{t_i \hat{a}^i \mid i:T\} >= 0.$

We need to show that the subvector $\tilde{c}_S = \tilde{y}$ is another optimal dual solution, i.e

i) $\tilde{y} >= 0$;

ii) $\tilde{y}A >= c$;

iii) $\tilde{y}b = y^*b$.

i) Follows from the definition of $\tilde{c}$.

ii) By definition $\mathbf{c} = -c + y^*A$. (17)

Also, $\tilde{c} = \mathbf{c} + t[I_1 | 0 | \mathbf{A}_1]$, (18)

thus $\tilde{c} = -c + y^*A + t[I_1 | 0 | \mathbf{A}_1]$. (19)

The subsystem of (19) indexed by S is

$\tilde{c}_S = -c_S + y^*A_S + t[I_1 | 0 | \mathbf{A}_1]_S$. (20)

Since $c_S = 0$ and $A_S = I$, (20) results in

$\tilde{c}_S = y^* + SUM\{t_i \hat{a}_S^i | i:T\} = \tilde{y}$.

Now since $\tilde{c} = -c + \tilde{y}A$ by definition and since $\tilde{c} >= 0$, $\tilde{y}A >= c$.

iii)

We need to show that $\tilde{y}b = y^*b$.

Let $\mathbf{y} = \tilde{y} - y^* = SUM\{t_i \hat{a}^i_S | i:T\}$.

Now it suffices to show that $\mathbf{y}b = 0$.

But $\mathbf{y} = tB^{-1}[A_1]_S$

$= [t,0]B^{-1}A_S = [t,0]B^{-1}$, since $A_S = I$.

Hence

$\mathbf{y}b = [t,0]B^{-1}b$

$= [t,0]\ [\mathbf{b}_T, \mathbf{b}_T']^T$

$= [t,0]\ [0,>0]^T = 0.$

Thus $\tilde{y}$ is another optimal dual solution.

II)

Let $\tilde{y}$ be another optimal dual solution.

Let Q be $[I|\mathbf{A}]$, i.e. Q is an optimal tableau for (P).

Since Q has full row rank, a right inverse exists, for example, $\hat{Q} = [I|0]'$. (21)

Therefore $v:R^m$ exist such that

$$\tilde{c} = c^* + v[I|\mathbf{A}] \tag{22}$$

Since $\tilde{c}$ is dual feasible, we have after partioning:

$$[\tilde{c}_B|\tilde{c}_N] = [c^*_B|c^*_N] + [v|v\mathbf{A}] >= 0. \tag{23}$$

Since $c^*_B = 0$ we have

$$v_i >= 0 \quad \text{for } i = 1 \text{ to } m. \tag{24}$$

Note that using the inverse in (21) gives $v = \tilde{c}_B$.

Taking the subsystem of (22) indexed by S and postmultiplying by b, we have:

$$\tilde{y}b = y^*b + vB^{-1}b. \tag{25}$$

But since $\tilde{y}$ is optimal we have $\tilde{y}b = y^*b$.

Therefore $vB^{-1}b = 0$. (26)

(24) and (26) imply that

$$v_i = 0 \quad \text{for } B^{-1}b \text{ nonzero.}$$

Let $t_k = v_k$ for k:T. Thus $\tilde{y}$ can be represented as

$$\tilde{y} = y^* + SUM\{t_k\hat{a}^k{}_S|k:T]$$

for some $t:R^{|T|}$ such that

$$\mathbf{c} + SUM\{t_k\hat{a}^k|k:T\} >= 0.$$

CHAPTER 3

THE GENERAL CASE

The above definition of shadow prices found in the literature (e.g Akgul [1],Gauvin[8]) assumes an LP in the canonical form (P) such that "prices" are always nonnegative quantities. To accomodate both maximization and minimization problems with any combination of inequality as well as equality constraints we need a more general definition that allows a consistent sign convention.

Definition 6a. For any LP, the incremental shadow price p_i^+ of constraint i is defined as the instantaneous rate of improvement in v(b) with an increase in b_i; the decremental shadow price p_i^- of constraint i is defined as the negative of the instantaneous rate of improvement in v(b) with a decrease in b_i.

In this sense, a negative incremental price or a positive decremental price is the rate of deterioration of the objective value as the right-hand-side is changed. Note that whether an increase or a decrease is actually involved depends on the direction of optimization. The possible cases are listed in Table 1.

Table 1. Interpretation of Shadow Prices.

Type of constraint	Increase in v(b)	Decrease in v(b)
Minimization:		
<=	$p_i^- < 0$	$p_i^+ > 0$
>=	$p_i^+ < 0$	$p_i^- > 0$
=	$p_i^+ < 0$ or $p_i^- < 0$	$p_i^+ > 0$ or $p_i^- > 0$
Maximization:		
<=	$p_i^+ > 0$	$p_i^- < 0$
>=	$p_i^- > 0$	$p_i^+ < 0$
=	$p_i^+ > 0$ or $p_i^- > 0$	$p_i^+ < 0$ or $p_i^- < 0$;

Definition 6b Suppose LP in general has the form

(LP)

maximize cx

subject to Lx <= p

Gx >= q

Ex = r

x >= 0.

Or, equivalently

$$
\begin{array}{lll}
 & \text{maximize} & cx \\
 & \text{subject to} & Lx \leq p \\
 & & -Gx \leq -q \\
(P') & & Ex \leq r \\
 & & -Ex \leq -r \\
 & & x \geq 0.
\end{array}
$$

The dual of (P') is:

$$
\begin{array}{lll}
 & \text{minimize} & p^T y_L - q^T y_G + r^T (y_{E+} - y_{E-}) \\
(D') & \text{subject to} & L^T y_L - G^T y_G + E^T (y_{E+} - y_{E-}) \geq c \\
 & & y_L, y_G, y_{E+}, y_{E-} \geq 0
\end{array}
$$

Then (P') and (D') have the form of (P) and (D) respectively. Finally let the equivalent problem to (D') be

$$
\begin{array}{lll}
 & \text{minimize} & p^T y_L - q^T y_G + r^T y_E \\
(DLP) & \text{subject to} & L^T y_L - G^T y_G + E^T y_E \geq c \\
 & & y_L \geq 0 \\
 & & y_G \leq 0 \\
 & & y_E \text{ unrestricted.}
\end{array}
$$

In the following proposition, Y and Y* for (LP) is defined using (DLP). This convention is standard practice in LP implementation (see, e.g. [10],[18]). Finally, let Y' and Y'* correspond to (D'). Note that y_G in (DLP) is $-y_G$ in (D') and y_E in (DLP) is $y_{E+} - y_{E-}$ in (D').

Proposition 6.

For any LP: $p_i^+ = \min\{y_i | y:Y^*\}$

$p_i^- = \max\{y_i | y:Y^*\}$.

Proof.

First consider the case of maximization. For less-than-or equal-to constraints, the results follow from Proposition 4. A greater-than-or-equal-to constraint $a^i x >= b_i$ can be written as $-a^i x <= -b_i$ as in (P'). Applying Proposition 4 to (P') and (D'):

p_i^+ = instantaneous rate of improvement with increase of b_i in (LP)

= instantaneous rate of improvement with decrease of b_i in (P')

= $-\max \{y_i | y:Y'^*\}$

= $\min \{-y_i | y:Y'^*\}$

= $\min \{y_i |\ y:Y^*\}$.

Expressing an equal-to-constraint as two less-than-or-equal-to constraints as in (P'), we have:

p_i^+ = instantaneous rate of improvement with increase of b_i in (LP)

= instantaneous rate of improvement with simultaneous increase of b_i and decrease of $-b_i$ in (P')

= $\min \{y_{iE+} | y:Y'^*\} - \max\{y_{iE-} | y:Y'^*\}$

= $\min \{y_{iE+} - y_{iE-} | y:Y'^*\}$

= $\min \{y_{iE} | y:Y^*\}$.

The proof for p_i^- is similar.

For minimization, the objective min cx can be written as -max(-cx). Both (D') and (DLP) are as before except -c replaces c. Since any rate of improvement in max(-cx) is the same as that in min(cx), the results follow.

Since y*, the current dual optimal solution given by the optimal basis, belongs to Y* some or all(e.g. when (P) is non-degenerate) of its components may already be true shadow prices. Therefore, we need to identify such cases and then proceed to find the remaining missing shadow prices. This can be done using right-hand-side ranging in LP sensitivity analysis.

For constraint i, let r_{i+} and r_{i-} be the allowable increase and allowable decrease given by the right-hand-side range analysis (See e.g. Bradley et. al.[5]).

Proposition 7

If $r_{i+} > 0$, then $p_i^+ = y^*_i$. If $r_{i-} > 0$, then $p_i^- = y^*_i$.

Proof. It suffices to show for the case of (P). The general case follows with appropriate sign manipulations.

If $r_{i+} > 0$, then for small enough $db_i > 0$, $v(b+e_i db_i) = c_B B^{-1}(b+e_i db_i)$ where B is the current optimal basis, c_B are the objective coefficients of the basic variables, and e_i is the i^{th} unit column vector. Therefore

$$p_i^+ = \lim[db_i \to 0^+] \{v(b+e_i db_i) - v(b)\} / db_i$$
$$= (c_B B^{-1} b + c_B B^{-1} e_i db_i - c_B B^{-1} b) / db_i$$
$$= c_B B^{-1} e_i$$
$$= y^*_i.$$

$$p_i^- = - \lim[db_i \to 0^-] \{v(b - e_i db_i) - v(b)\} / db_i$$
$$= - (c_B B^{-1} b - c_B B^{-1} e_i db_i - c_B B^{-1} b) / db_i$$
$$= c_B B^{-1} e_i$$
$$= y^*_i.$$

Next, we consider how many of the right-hand-side ranges might be zero. In an m x n LP, there are 2m r_i's to consider (m for increases, m for decreases). When one row is degenerate, at most 'm' of the r_i's will be zero. When two or more rows are degenerate, all of the r_i's could be zero. This can be seen by looking at the upper and lower bounds on r_i:

$$\text{Max}[i]\{-\mathbf{b}_i / B^{-1}{}_{ij} \mid B^{-1}{}_{ij} > 0\} <= r_j$$
$$<= \text{Min}[i]\{-\mathbf{b}_i / B^{-1}{}_{ij} \mid B^{-1}{}_{ij} < 0\}; \text{ for } j = 1 \text{ to } m.$$

If row i is degenerate then a positive element in row i of the basis inverse will result in $r_{j-} = 0$. If there is a negative element in row i of the basis inverse then $r_{j+} = 0$ occurs. Each degenerate row will have at least one nonzero element in B^{-1}. Therefore, even when a primal degenerate problem has $p^+ = p^-$, because of the definition of right-hand-side sensitivity analysis, some of the r_i's will be zero. As a result, further analysis will always be

necessary on a degenerate problem in order to find all the true shadow prices. In the case of multiple optimal dual solutions, each solution will supply some of the missing shadow price information since at each solution some components of y* will increase and some components of y* will decrease. However, it is possible that the Range output for a particular solution may incorrectly indicate that no components of y* are true shadow prices.

CHAPTER 4

ALGORITHMS

For any shadow price p_i^+ or p_i^- that is not given by y^*, two approaches can be taken for its computation.

I. Direct Search using:

a) parametrization;

b) perturbation;

II. Constrained Dual using:

c) dual simplex;

d) implicit Y^*.

a) Right-hand-side parametrization:

By varying the right-hand-side b_i parametrically, an adjacent basis (in the sense of Murty[14]) with a different objective value is sought. If found, the dual variable y_i for this new basis provides a rate of improvement leading into this basis, which will be the same as the rate of improvement leading out of the original optimal basis. If the objective value does not change, the rate of improvement is zero. In this case, either the direction of change is infeasible or any change in b_i in that direction will not affect the objective value.

b) Right-hand-side perturbation:

This is direct application of the definition of shadow prices. The right-hand-side coefficient is altered by a suitably small amount, the LP reoptimized and the rate of change computed. See Dantzig[7] for a more sophisticated version of this approach.

c) Constrained dual:

This follows from Proposition 6. Each missing shadow price requires the solution of (DLP) with yb = v(b) and y_i optimized. With conventional software, this method is too cumbersome to be practical. In enhanced software, the method below is more efficient.

d) Implicit Y*:

In this method, any missing shadow price can be solved by a subproblem implied by Propositions 5 and 6. For example, for p_i^+, we have

$$\begin{array}{lll} & \text{Minimize} & [y^* + \text{SUM}\{t_k \hat{a}^k_S \mid k:T\}]e_i \\ (S^+) & \text{subject to} & \text{SUM}\{t_k \hat{a}^k \mid k:T\} >= -c. \end{array}$$

where $t:R^{|T|}$. For finding p_i^-, a subproblem (S^-) is obtained from(S^+) by maximizing the same objective. Let (S) denote either a (S^+) or a (S^-) subproblem. (S) has decision variables associated with the degenerate constraints in (P) and constraints aimed at maintaining dual feasibility of

(P). The optimal objective function value of (S) will be the desired shadow price. In Chapter 7, efficient implementation of this approach will be discussed.

CHAPTER 5

A NUMERICAL EXAMPLE

Before discussing the implementation of methods to compute true shadow prices, it will help fix ideas by considering a numerical example. Consider the following LP adapted from Ho[10,Ch 3].

$$
\begin{aligned}
\text{Minimize} \quad & 74A + 40B + 50C + 10D \\
\text{Subject to} \quad & 3A + B + 2C \geq 8 \\
& 2A + 2B + D \geq 11 \\
& 4A + 3C \geq 10.6667 \\
& A,B,C,D \geq 0
\end{aligned}
$$

Let S1, S2, S3 be the slack variables and X, Y, Z, be the dual variables to the first, second and third constraints respectively. The optimal solution to the LP is

A = 2.667

B = 0.0

C = 0.0

D = 5.667

S1 = 0.0

S2 = 0.0

S3 = 0.0

with an objective value of 254.00. However, S1 is basic and the primal optimal solution is degenerate. Indeed, there are two alternative optimal dual (basic) solutions, namely,

X = 0.0

Y = -10.0

Z = -13.5

and

X = -18.0

Y = -10.0

Z = 0.0.

Note that the output of any commercially available LP software lists only one or the other dual optimal solutions which cannot be interpreted directly as marginal values. The minimum and maximum values for the optimal dual solutions are given in Table 2.

Table 2. Extreme Values in Y* for Example LP.

Constraint	min	max
First	-18.0	0.0
Second	-10.0	-10.0
Third	-13.5	0.0

The interpretation of these values as shadow prices are tabulated in Table 3.

Table 3. Shadow Prices for the Example LP.

Constraint	db_i	dv(b)	p_i^+	p_i^-
First	-0.1	0.0		0.0
	+0.1	+1.8	-18.0	
Second	-0.1	-0.1		-10.0
	+0.1	+1.0	-10.0	
Third	-0.1	0.0		0.0
	+0.1	+1.35	-13.5	

The db_i and dv(b) columns are obtained from actual perturbations of the LP. The results verify Proposition 6.

CHAPTER 6

COMPUTING WITH CONVENTIONAL LP CODES

Given that the outputs of conventional LP codes do not provide complete information on all the true shadow prices it is of interest to see if we can still obtain such answers using only the very same codes. For this purpose, two popular packages were examined: LINDO [18] on IBM PC's and MPSX/370 [12] on IBM mainframes.

Two different procedures are suggested for use with LINDO:

i) parametric right-hand-side;

ii) perturbation.

In the parametric approach, LINDO is used to solve the LP and do the Range (sensitivity) analysis. See Table 4 for the screen ouput for the example problem at this stage. For each right-hand-side that has an allowable increase of zero, the following procedure is performed. The PARA command is used with a new right-hand-side value that is large enough to force a change in the objective value if possible.

TABLE 4. LINDO Output for Example LP.

```
: MIN 74A + 40B + 50C + 10D

     ? ST
     ? 3A + B + 2C > 8
     ? 2A + 2B + D > 11
     ? 4A + 3C  > 10.66667
     ? END
     : GO
     LP OPTIMUM FOUND AT STEP    2

            OBJECTIVE FUNCTION
     1)          254.000000

     VARIABLE        VALUE                    REDUCED COST
           A       2.666667                     .000000
           B        .000000                   20.000000
           C        .000000                    9.500000
           D       5.666667                     .000000

         ROW    SLACK OR SURPLUS             DUAL PRICES
           2)       .000000                     .000000
           3)       .000000                  -10.000000
           4)       .000000                  -13.500000

     NO. ITERATIONS  =    2

     DO RANGE(SENSITIVITY) ANALYSIS?
     ?Y

     RANGES IN WHICH THE BASIS IS UNCHANGED:

                            OBJ COEFFICIENT RANGES
     VARIABLE        CURRENT         ALLOWABLE        ALLOWABLE
                      COEF           INCREASE         DECREASE
           A      74.000000        12.666670        54.000000
           B      40.000000        INFINITY         20.000000
           C      50.000000        INFINITY          9.500000
           D      10.000000        10.000000         6.333333

                            RIGHT-HAND-SIDE RANGES
          ROW       CURRENT         ALLOWABLE        ALLOWABLE
                     RHS            INCREASE         DECREASE
            2      8.000000          .000000          .000000
            3     11.000000        INFINITY          5.666667
            4     10.666670          .000000          .000000
```

Table 5 shows the output from PARA for the first constraint in the example. The dual variable (-18.00) corresponding to the first new objective value gives the incremental shadow price (p_i^+) sought. In cases where the objective value does not change, the last dual variable listed should be used.

TABLE 5. LINDO PARA Output for First Constraint.

```
:PARA
ROW:
2
NEW RHS VALUE =
8.001

     VAR       VAR      PIVOT      RHS       DUAL        OBJ
     OUT       IN        ROW      VAL     VARIABLE      VAL

  ART       SLK 4         2     8.0000     .00000     254.000
    D       SLK 3         3    16.5000   -18.0000     407.000
```

Similarly, for each right-hand-side that has an allowable decrease of zero, the PARA command is used with a decrement to the right-hand-side value to find the decremental shadow price. Table 6 shows the ouput from PARA for the third constraint in the example.

TABLE 6. LINDO PARA Output for Last Constraint.

```
: PARA
ROW:
4
NEW RHS VALUE =
10.6
```

VAR OUT	VAR IN	PIVOT ROW	RHS VAL	DUAL VARIABLE	OBJ VAL
SLK 2	SLK 4	2	10.6667	-13.5000	254.000
ART A	ART	0	10.6667	.0000	254.000

Note that whenever the objective changes from the original optimal value in one case of parametrization, the LP needs to be resolved before proceeding with another case. This is necessary because LINDO starts with the last available tableau to execute the PARA option. Therefore, performing consecutive parametric analysis will not lead to desired results. This is a definite drawback of this approach with LINDO as the LP may have to be resolved as many times as there are missing shadow prices.

The perturbation procedure involves using the ALTER command to change a right-hand-side value, again by an appropriately small amount. Then the LP is resolved and the output is examined to determine missing shadow prices using their definitions directly. The results of this procedure for the example are listed in Table 3.

With LINDO, this approach is not as inefficient as it

may seem because the last tableau available is used to start an altered LP. However additional computation must be performed to determine the rates of change. Also, the perturbations must be made independently, i.e, another ALTER must be used to erase the previous case.

An extension of the perturbation procedure would be to examine the Range analysis after each perturbation in order to find any new shadow prices. For example, from Table 4 $r_{1+} = 0$. If we change b_1 to 8.001 and resolve the LP we get the following:

$y_1 = -18.0$	$r_{1+} = 8.499$	$r_{1-} = 0.001$
$y_2 = -10.0$	$r_{2+} = \text{infinity}$	$r_{2-} = 5.666$
$y_3 = 0.0$	$r_{3+} = 0.001334$	$r_{3-} = \text{infinity}$

Based on Proposition 7, it appears that no shadow prices are missing since all $r_i > 0$. However, this is the Range analysis for the perturbed problem and from sample problems solved, it seems that for perturbations in b of 0.001 values of r_i smaller than 0.003 can be considered zero for applying Proposition 7. Now from Table 4 and the above output we have the following shadow prices for the example:

$p_1^+ = -18.0$	$p_1^- = ?$
$p_2^+ = -10.0$	$p_2^- = -10.0$
$p_3^+ = ?$	$p_3^- = 0.0$

It remains to find p_1^- and p_3^+. If we set $b_1 = 7.999$ and resolve the LP we have

$Y_1 = 0.0$	$r_{1+} = .001$	$r_{1-} = \text{infinity}$
$Y_2 = -10.0$	$r_{2+} = \text{infinity}$	$r_{2-} = 5.666667$
$Y_3 = -13.5$	$r_{3+} = 11.33333$	$r_{3-} = 0.001334$

Assuming r_{1+} and r_{3-} to be zero, we have the following complete list of shadow prices:

$p_1^+ = -18.0$	$p_1^- = 0.0$
$p_2^+ = -10.0$	$p_1^- = -10.0$
$p_3^+ = -13.5$	$p_3^- = 0.0.$

This approach points out an undesirable feature in using Range analysis to find missing shadow prices. Although the dual solution in Table 4 and that obtained from letting $b_1 = 7.999$ are identical, the Range data failed to provide all the shadow prices from the original solution, thus requiring extra work by the LP user.

As will be seen, when using the Implicit Y* approach, Range analysis will not be used. Instead, certain properties of (S) will be utilized to determine missing shadow price cases.

The above discussion about the use of LINDO is based on implementations on IBM compatible PC's and VAX mainframes. While the necessary procedures are not too tedious for small problems (say, where the range output fits on one screen), for large problems with 25 or more

constraints the approach becomes quite cumbersome. It is therefore not practical for routine applications. In Chapter 7 automated shadow price computations requiring no extra work by the LP user will be discussed.

Using MPSX/370, two procedures are suggested. First, solve the LP with the RANGE option and examine the ranges for rows at limit level for correct cases of shadow prices. Table 7 lists an example of the pertinant information. The

Table 7. MPSX Row at Limit Level Range Output.

row	lower activity	unit cost	upper activity	unit cost
Second	5.33333	-10.0	infinity	10.0
Third	10.66666	-13.5	22.0	13.5

interpretation of Table 7 is as follows. Since the lower activity for Second is $< b_2$ it states that a unit decrease in b_2 will increase the objective function value by -10.0. Since the example LP is a minimization problem from Definition 6a we have: $p_2^- = -10.0$. Likewise, since the upper activity of Second is $> b_2$, it states that a unit increase in b_2 will increase the objective function value by 10.0. Since the example is a minimization problem, we

have: p_2^+ = -10.0. The same is true for Third since the upper activity is > b_3. Therefore p_3^+ = -13.5. Note that since Third's lower activity equals b_3, p_3^- is a missing shadow price.

Unfortunately, the Range output and the sign convention for the marginal prices in the row at limit level data is not consistent with LINDO. Also, the row at intermediate level data can be misleading so this information will not be utilized when determining missing shadow prices.

For cases not found using the row at limit level output, use the PARARHS option. IN MPSX/370, this is based on incrementing the original right-hand-side by successive multiples of a change column until a maximum increment is reached. All three parameters: the change column, the multiple, and the maximum increment must be specified in the control program. The change column is the appropriate unit vector for finding an incremental shadow price and the negative unit vector for finding a decremental shadow price. The parameters should be chosen to reduce extraneous computation and output. In most cases using a value of 0.001 accomplishes this and also reduces roundoff error in the new objective function.

Note that in MPSX/370, each PARARHS command is based on the optimal tableau of the LP and not on the tableau for previous parametrization. Therefore, no redundant

resolution of the LP is necessary as is the case with LINDO.

Since MPSX/370 is not interactive, the above procedure can become cumbersome. For the same reason, the perturbation method is deemed impractical. However, the parametric approach can be automated at the expense of extraneous computation. This second procedure is carried out by setting up in a single control program all the PARARHS runs, regardless of whether they eventually become necessary or not.

In Chapter 4 it was stated that the Implicit Y* approach is more efficient for automating shadow price computations. After seeing how LINDO and MPSX find shadow prices using available routines, it is worthwhile to see what advantages the Implicit Y* procedure has over LINDO and MPSX.

First, since the size of the subproblem used to find shadow prices is much smaller than the original LP, computational complexity is reduced at each iteration. LINDO and MPSX both maintain the original problem and perform perturbations and parametrics directly from the current tableau. MPSX has an advantage over LINDO since it parameterizes from the original optimal tableau of the LP, thus MPSX has a smaller eta file than LINDO at each iteration.

Secondly, the data for the subproblem can be retrieved

easily as will be shown in Chapter 7. The importance of this is that the computational advantage due to the reduced size of the subproblem constraint matrix is not offset by the effort in retrieving the elements in the matrix.

Finally, as will be seen in Chapter 7, when solving for a particular shadow price the simplex method may revert to a Phase I just as it may when using the perturbation method or parametrics with LINDO or MPSX. But since the constraint matrix is considerably smaller in the subproblem, the penalty for bouncing into Phase I is negligible compared to LINDO and MPSX.

CHAPTER 7

COMPUTING WITH ENHANCED LP CODES

As shown above, although it is possible to reconstruct all true shadow prices for an LP by repeated application of available features in conventional software, it would be much more convenient to have such information generated automatically. This will of course involve the modification of existing codes. To gain insight into the complexities of such attempts, an experimental LP code has been extended to include the computation of true shadow prices using method (d) described in Chapter 4.

Consider subproblem (S^+) in Chapter 4. Each column in this LP is the transpose of a degenerate row in the optimal tableau of (P). There are n+m constraints, corresponding to the variables (including slacks) in (P). Note that a basic variable in the optimal tableau implies either a null constraint in (S^+) or a nonnegativity constraint on a t_k. This is because a basic column in the optimal tableau of (P) has a single element of unity. If this element occurs in a degenerate row, then a non-negativity constraint results. Otherwise it is a null constraint. Therefore, (S^+) effectively has n constraints in nonnegative variables t_k, k:T. Computationally, it is more efficient to solve its dual. Letting J be the index set of nonbasic variables in

the optimal tableau of (P), the dual to (S) can be written as follows.

$$
\begin{array}{lll}
 & \text{Maximize} & \text{SUM } -c_j w_j \\
(DS^+) & \text{subject to} & \text{SUM } \hat{a}^k{}_j w_j <= \hat{a}^k{}_{Si},\ k{:}T; \\
 & & w_j >= 0,\ j{:}J.
\end{array}
$$

Compared to (S), the efficiency of (DS) lies in the retrieval of data from the optimal tableau of (P) and in the iterative process between missing shadow price cases. Also, since the number of degenerate rows will typically be much less than the number of non-basic variables in (P), using (DS) will reduce the size of the basis matrix.

First, consider the retrieval process. The experimental code uses a sparse storage procedure where all the constraint coefficients are stored in an array called 'A'. Associated with this is an 'IA' array which contains the row indices of the elements in 'A' and an 'LA' array which signifies the beginning of a new column. Since each column in (DS) is a column in the optimal tableau of (P) restricted to the degenerate rows, only one FTRAN is needed to retrieve each column in (DS). Otherwise, if (S) is used the whole tableau must be retrieved at once and then each column of (S) would be constructed with the transpose of each degenerate row in the optimal tableau of (P).

(DS) is is also more efficient in the iterative

process needed to solve for each missing shadow price. Since the objective function coefficients are imbedded in the 'A' array, when using (S) each case would involve insertions and deletions in the 'A', 'IA', and 'LA' arrays. By solving the dual (DS), only the right-hand-side will be altered for each case.

Another advantage of using (DS) is that logical columns in (P) corresponding to equality constraints can be eliminated. Recall that the constraints in (S) preserve dual feasibility. That is, for a <= constraint in (P) the constraint in (S) will be >=. Likewise for a >= constraint in (P) the constraint in (S) is <=. But for an equality constraint in (P), the related constraint in (S) is non-binding. This would be represented by:

$a_i x + s_i = 0$ and

$-\text{infinity} <= s_i <= \text{infinity}$.

Based on the complementary slackness condition of $y_i * s_i = 0$, this implies that $y_i = 0$ in the dual for a non-binding constraint i. As a result when (P) has equality constraints the number of elements in (DS) will be smaller than (S).

Features of the experimental code.

The experimental code uses the revised simplex method with sparse storage and product form of the inverse. The

implementation of (DS) using the experimental code relies on the subroutines listed in Table 8.

Table 8. Procedures in the Experimental LP Code.

Procedure name	Action
INITIAL_BASIS	defines an all logical initial basis.
FTRAN	solves $B\hat{a}_j = a_j$ ==> $\hat{a}_j$
BTRAN	solves $yB = c_B$ ==> y.
FORMC	checks feasibility and defines Phase I or Phase II objective accordingly.
CHUZR	chooses the pivot row using the minimum ratio test.
UNPACK(j)	unpacks column j of the constraint matrix.
PRICE	prices out nonbasic columns for pivot column selection.
WRETA	adds a new eta to ETA file.
UPDATE	updates basis information and new right-hand-side.
INVERT	reinverts basis matrix after either a specified iteration or when the ETA file reaches its upper size limit.
PRIMAL	runs the revised simplex method using the above procedures.

Appendix A includes the PLI extension to the experimental code which generates true shadow prices for

the general case. Included also are the variable definitions and program comments.

The implementation process consists of two parts:

1) Retrieval

2) Computation

Retrieval

The following steps are necessary in the retrieval process:

i) Determine which rows are degenerate.

ii) Retrieve the optimal dual solution and reduced costs during the output routine and store in C_TEMP and store the associated column in IC_TEMP, except for portions of the dual solution corresponding to equality constraints. Store these in EC_TEMP and the associated column in EIC_TEMP.

iii) For each nonbasic column and logical column which does not correspond to an equality constraint, perform an FTRAN and load the values from degenerate rows into A_TEMP. Update IA_TEMP and LA_TEMP.

iv) For each logical column which corresponds to an equality constraint, perform an FTRAN and load the values from degenerate rows into EA_TEMP. Update EIA_TEMP and ELA_TEMP.

v) For data in (iii), merge A_TEMP with the proper elements in C_TEMP and load into 'A'. Update 'IA' and 'LA'. For each

logical column, put its position in OBJ_INDEX and its associated constraint index in COL_INDEX.

vi) For data in (iv), merge EA_TEMP with the proper elements in EC_TEMP and load in 'EA'. Update 'EIA'. Put the position of each column in EOBJ_INDEX and its corresponding constraint index in ECOL_INDEX.

Equality logicals are treated differently from less-than-or-equal and greater-than-or-equal logicals since the data for equality logicals is only needed for the right-hand-side in solving (DS). Only the constraint matrix is included in 'A'.

Computation

As mentioned in Chapter 6, the solution of (S) or (DS) does not utilize sensitivity analysis in order to reduce the number of missing shadow prices as does the LINDO and MPSX approach. Instead, the formulation of (S) and thus (DS) provides preliminary screening for some shadow prices.

From the formulation of (S), the following results:

(1) If all $\hat{a}_j$ associated with the i^{th} logical are >= 0 in the optimal tableau of (P) then

$$p_i^+ = e_i y^*.$$

(2) If all $\hat{a}_j$ associated with the i^{th} logical are <= 0 in the optimal tableau of (P) then

$$p_i^- = e_i y^*.$$

(3) If the i^{th} logical is nonbasic but $e_i y^* = 0$, then

$$p_i^+ = 0.$$

(4) If the i^{th} logical is basic in (P) and = 0, then

$$p_i^+ = 0.$$

These are true because the constraints in (S) can be written as

SUM $\{-t_k \hat{a}^k | k:T\}$ <= **c** and

since **c** is always nonnegative, the origin will always be in the feasible region of (S).

Also, from the complementary slackness condition of $y_i * s_i = 0$, we have:

(5) If the i^{th} logical is basic in (P) and > 0, then

$$p_i^+ = p_i^- = 0.$$

The experimental code solves (DS) as follows:

(I) First missing shadow price --

1a) if equality logical then retrieve the right-hand-side from 'EA',

1b) otherwise retrieve the right-hand-side from 'A';.

2) set cobj = b(1) and b(1) = 0;

3) call INITIAL_BASIS;

4) call PRIMAL;

5) if (DS) is feasible then the shadow price will be the optimal objective function value + cobj, otherwise the shadow price is zero.

II) For all other cases--

1a) if equality logical then retrieve the right-hand-side from 'EA',

1b) otherwise retrieve the right-hand-side from 'A';

2) set cobj = b(1) and b(1) = 0;

3) setting y(*) = b(*), call FTRAN;

4) if all y_i >= 0, then the missing shadow price = cobj + y(1), where y(1) is the element of $B^{-1}b$ corresponding to the row of the objective function;

5) if some y_i < 0, then call PRIMAL;

6) if (DS) is feasible then the shadow price will be the optimal objective function value + cobj, otherwise the shadow price is zero.

Step (2) is needed in both cases because the formulation of (S) includes a constant in the objective function. The variable cobj stores the constant and adds it back to the optimal objective function value to determine the shadow price. Table 9 lists the output from the experimental code for the example LP. Appendix B includes output from several more examples.

TABLE 9. Output from Experimental LP Code.

VARIABLE	VALUE	REDUCED COST
1	2.666666667E+00	0.000000000E+00
2	0.000000000E+00	2.000000000E+00
3	0.000000000E+00	1.400000000E+00
4	5.666666667E+00	0.000000000E+00

ROW	LOGICAL	DUAL VALUE
1	-2.540000000E+02	1.000000000E+00
2	0.000000000E+00	-1.800000000E+01
3	0.000000000E+00	-1.000000000E+01
4	6.357828776E-07	5.551115123E-17

SHADOW PRICES:

ROW	INCREMENTAL	DECREMENTAL
2	-1.800000000E+01	0.000000000E+00
3	-1.000000000E+00	-1.000000000E+01
4	-1.350000241E+01	0.000000000E+00

CHAPTER 8

MISCELLANEOUS TOPICS

Combination Shadow Prices

As noted by Akgul[1], Proposition 3, $D_{\mathbf{u}}v(b) = \min \{y\mathbf{u} | y:Y^*\}$ can be utilized to determine a shadow price for a combination of resources. For example, if we are interested in finding the change in the objective function for simultaneous changes in the right-hand-side, this change vector would be substituted for **u** and with the appropriate manipulations, (DS) could be used to determine the shadow prices. To determine the shadow price for a combination of resources (DS) only needs to be solved once. However, constructing the right-hand-side is more complex than when $\mathbf{u} = e_i$. Recall that in finding the shadow price for the i^{th} constraint, each right-hand-side is stored contiguously in 'A' or 'EA'. But for the combination case, elements in 'A' and 'EA' corresponding to logicals must be manipulated to construct the right-hand-side. The formulation of (DS) has not been modified to handle the combination shadow price case.

To find combination shadow prices using MPSX set the change column needed for PARARHS to the values of **u**. Ideally this vector will be known beforehand so the LP will not have to be solved twice. From the PARARHS output the shadow price for this combination of resources **u** would be

calculated directly from the change in the objective function value.

Finding combination shadow prices on LINDO is another matter since LINDO parametrics only allow one right-hand-side at a time to be changed. However, Schrage[18,Ch 7] points out that by adding a variable D in constraint i with coefficient $\mathbf{u}_i$, then parametrizing an added constraint D = 0, circumvents this limitation. Since this approach does not allow the flexibility to change **u** easily, the following steps are recommended:

1) solve the original problem;

2) alter the right-hand-side by adding **u**;.

3) resolve the problem (recall that LINDO uses the last available tableau).

Mixed Degenerate Case

It has been shown that if an LP is degenerate then multiple dual solutions may exist. But what about the mixed degenerate case where an LP has multiple optimal solutions of which some are degenerate? In this mixed case the dual solution will be unique. Based on Proposition 7, for a non-degenerate solution of the LP all the r_i's in the Range analysis will be nonzero implying the current optimal dual solution supplies complete shadow price information. Unfortunately, if the LP lists a degenerate solution as output, then more work will be required by the LP user to

determine the true shadow prices (since some r_i's = 0) even though the dual solution associated with the degenerate basis provides complete shadow price information.

CHAPTER 9

DISCUSSION

It is shown that conventional LP software does not provide complete shadow prices in general. Although marginal values can indeed be found using repeated applications of existing codes, it would be desirable to automate such computations in commercial packages. This can be done without extensive modifications. In view of the fact that dual variables are routinely interpreted (often incorrectly as seen above) as shadow prices by practitioners, it is important that only truly meaningful information is generated.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Akgul,M., A note on shadow prices in linear programming, J. Opl. Res. Soc. 35 (1984) 425-431.

[2] Akgul,M., Private communication, June 1986.

[3] Aucamp,D.C. and D.I. Steinberg, The computation of shadow prices in linear programming, J. Opl. Res. Soc. 33 (1982) 557-565.

[4] Best,M.J., A compact formulation of an elastoplastic analysis problem, J. Optim. Theory Appl. 37 (1982) 343-353.

[5] Bradley,S.P.,et al.,**Applied Mathematical Programming**, Addison-Wesley, Reading, Massachusetts, 1977.

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

[7] Dantzig, G.B., Are dual variables prices? If not, how to make them more so,TR SOL 78-6, Systems Optimization Laboratory, Stanford University, March 1978.

[8] Gauvin, J., Quelques précision sur les prix marginaux en programmation linéaire, INFOR 18(1980) 68-73.

[9] Greenberg, H.J., An analysis of degeneracy, Naval Research Logistics Quarterly, 33 (1986) 635-655.

[10] Ho, J.K., **Linear and Dynamic Programming with Lotus 1-2-3**, MIS Press, Portland, 1987.

[11] Ho,J.K. and D.Smith, Computing True Shadow Prices in Linear Programming, (to appear).

[12] IBM, MPSX/370 Program Reference Manual, 1979.

[13] Mangasarian,O.L., Uniqueness of solution in linear programming, Lin. Alg. & Appl. 25 (1979) 151-162.

[14] Murty,K.G., **Linear Programming**,Wiley, New York, 1983.

[15] Peterson,E.L., An economic interpretation of duality in linear programming, J. Math. Anal. Appl. 30 (1970) 172-196.

[16] Pêrold,A.F., Exploiting degeneracy in the Simplex Method, in G.B. Dantzig et al.(eds), **Large-Scale Linear Programming**, IIASA, Laxenburg, 1981.

[17] Rockafellar,T., **Convex Analysis**, Princeton: Princeton University Press, 1970.

[18] Schrage,L., **Linear, Integer, and Quadratic Programming with LINDO**, Third Edition, Palo Alto: Scientific Press, 1986.

[19] Shapiro,J.F., **Mathematical Programming: Structure and Algorithms**, New York: Wiley 1979.

[20] Williams,A.C., Marginal values in linear programming, SIAM 11 (1963) 82-94.

APPENDIXES

APPENDIX A

PLI SUBROUTINES

```
*************************************************************
VARIABLE DEFINITIONS NOT INCLUDED WITHIN PROGRAM

IA_TEMP: row indices for A_TEMP
LA_TEMP: indices for new columns in A_TEMP
EIA_TEMP: row indices for EA_TEMP
ELA_TEMP: indices for new columns in EA_TEMP
EIA:      row indices for EA
A_TEMP:   elements of â_j for nonbasic and inequality
          logical columns
BSTAT:    = '*' if incremental direction of change is
            infeasible
SSTAT:    = '*' if decremental direction of change is
            infeasible
TEMP_STAT: used for a 'G' constraint
TEMP_SHADOW:  used for a 'G' constraint
SWITCH:   row indices of 'G' constraints
TEMP: # of nonzeros in EA
EA_TEMP:  elements of â_j for equality logicals
EA:  right-hand-sides for equality logicals
EOBJ_INDEX: indicates where a new rhs starts in EA
ECOL_INDEX: column indices for EOBJ_INDEX
OBJ_INDEX: indicates where a new rhs starts in A
COL_INDEX: column indices for OBJ_INDEX
YFLAG: = false if new rhs is feasible for a new subproblem
DEG_INDEX: row indices of degenerate rows
IC_TEMP: column indices for C_TEMP
C_TEMP: reduced costs and y* values for inequality logicals
EIC_TEMP: column indices for EC_TEMP
EC_TEMP: y* values for equality logicals
IMAX: = false if i^th decremental price is missing
IMIN: = false if i^th incremental price is missing
IFLAG: = true if solving first subproblem
*************************************************************

OUTDATA: PROCEDURE;

dcl (i,ll,kk,j,k)              fixed bin,
    (value,pi,reduced_cost)    float bin(31);
value = -x(iobj);
allocate deg_index(nrow);
allocate ic_temp(ncol);
allocate c_temp(ncol);
allocate eic_temp(nrow);
allocate ec_temp(nrow);
```

```
allocate imax(nrow);
allocate imin(nrow);
c_temp(*) = 0;
ic_temp(*) = 0;
ec_temp(*) = 0;
eic_temp(*) = 0;
imax(*) = false;
imin(*) = false;
k = 0;  /* # of degenerate rows */
l = 0; /* # of elements in C_TEMP */
el = 0; /* # of elements in EC_TEMP */
do j = (nrow + 1) to ncol;
   value = 0;
   reduced_cost = 0;
   if ifbasic(j) \= 0 then do;
      value = x(ifbasic(j));
/* retrieval step (i) */
      if abs(value) < tolcost then do;
         k = k + 1;
         deg_index(k) = ifbasic(j);
         num_degen_rows = num_degen_rows + 1;
      end;
   end;
   else do;
      ll = la(j);
      kk = la(j + 1) - 1;
      do i = ll to kk;
         reduced_cost = reduced_cost + y(ia(i)) * a(i);
      end;
/* retrieval step (ii) */
      l = l + 1;
      c_temp(l) = reduced_cost;
      ic_temp(l) = j;
   end;
end;
do i = 1 to nrow;
   value = 0;
   if ifbasic(i) \= 0 then do;
      value = x(ifbasic(i));
      if abs(value) > tolcost then do;
         imax(i) = true;
         imin(i) = true;
      end;
      else do;
/* retrieval step (i) */
         imin(i) = true;
         k = k + 1;
         deg_index(k) = ifbasic(i);
         num_degen_rows = num_degen_rows + 1;
      end;
   end;
```

```
   if rowtype(i) \= 'g' then
      pi = y(i);
      else
         pi = -y(i);
   if abs(pi) < tolcost then
      imin(i) = true;
   if abs(value) < tolcost then do;
      if rowtype(i) = 'e' then do;
/* retrieval step (ii) */
         el = el + 1;
         ec_temp(el) = pi;
         eic_temp(el) = i;
      end;
      else do;
/* retrieval step (ii) */
         l = l + 1;
         c_temp(l) = abs(pi);
         ic_temp(l) = i;
      end;
   end;
end;
if num_degen_rows > 0 then
   call SUB_PROB;

END OUTDATA;

SUB_PROB: PROCEDURE;
dcl (mm,cc,k,i,j,ic,ll,kk,orig_nrow,n,nn)  fixed bin,
    (ia_temp(*),la_temp(*))          fixed bin controlled,
    (eia_temp(*),ela_temp(*),eia(*)) fixed bin controlled,
    a_temp(*)                         float bin controlled,
    (bstat(nrow),sstat(nrow),temp_stat) char(1),
    (temp_shadow,cobj)                   float bin(31),
    (switch(nrow - 1),temp)             fixed bin,
    (ea_temp(*),ea(*))           float bin controlled,
    (eobj_index(*),ecol_index(*)) fixed bin controlled,
    (obj_index(*),col_index(*))   fixed bin controlled,
    (yflag,pflag,nflag)           bit(1);

bstat(*) = ' ';
sstat(*) = ' ';
orig_nrow = nrow;   /* # of constraints in (P) */
nn = 0;             /* # of 'G' constraints in (P) */
/* used for reversing prices for 'G' constraints */
do i = 2 to orig_nrow;
   if rowtype(i) = 'g' then do;
      nn = nn + 1;
      switch(nn) = i;
   end;
end;
free b;  /* right-hand-side */
```

```
free y;  /* array used for FTRAN */
allocate a_temp(num_degen_rows);
allocate ia_temp(num_degen_rows * ncol);
allocate la_temp(ncol + 1);
allocate ea_temp(num_degen_rows * ncol);
allocate eia_temp(num_degen_rows * ncol);
allocate ela_temp(ncol + 1);
allocate incremental(orig_nrow);
allocate decremental(orig_nrow);
incremental(*) = 0;   decremental(*) = 0;
la_temp(*) = 0;  la_temp(1) = 1;
a_temp(*) = 0;   ia_temp(*) = 0;
cc = 0; /* # of elements in A_TEMP */
mm = 1; /* # of elements in LA_TEMP */
allocate y(nrow);  y(*) = 0;
/* retrieval step (iii) */
do i = 1 to 1;
   pflag = false;  nflag = false;
   call UNPACK(ic_temp(i);  call FTRAN(1);
   mm = mm + 1;     flag = false;
   do j = 1 to num_degen_rows;
      if abs(y(deg_index(j))) > tolcost |
         (num_degen_rows = j & flag = false) then do;
         flag = true;  cc = cc + 1;
         a_temp(cc) = y(deg_index(j));
         if a_temp(cc) > tolcost then pflag = true;
                                 else nflag = true;
         ia_temp(cc) = j + 1; la_temp(mm) = cc + 1;
      end;
   end;
   if abs(a_temp(cc) < tolcost & ic_temp(i) <= orig_nrow
      then do;
      incremental(ic_temp(i)) = c_temp(i);
      decremental(ic_temp(i)) = c_temp(i);
      imin(ic_temp(i)) = true;  imax(ic_temp(i)) = true;
   end;
   if pflag = true & nflag = false then do;
      incremental(ic_temp(i)) = c_temp(i);
      imin(ic_temp(i)) = true;
   end;
      else if pflag = false & nflag = true then do;
         decremental(ic_temp(i)) = c_temp(i);
         imax(ic_temp(i) = true;
      end;
end;
nnonz = cc + 1;
/* retrieval step (iv) */
if el > 0 then do;
cc = 0; /* # of elements in EA_TEMP */
   mm = 1; /* # of elements in ELA_TEMP */
   y(*) = 0;
```

```
   do i = 1 to el;
      pflag = false;  nflag = false;
      call UNPACK(eic_temp(i));  call FTRAN(1);
      mm = mm + 1;     flag = false;
      do j = 1 to num_degen_rows;
         if abs(y(deg_index(j))) > tolcost |
            (num_degn_rows = j & flag = false) then do;
            flag = true;  cc = cc + 1;
            ea_temp(cc) = y(deg_index(j));
            if ea_temp(cc) > tolcost then pflag = true;
                                     else nflag = true;
            eia_temp(cc) = j + 1; ela_temp(mm) = cc + 1;
         end;
      end;
      if abs(ea_temp(cc) < tolcost then do;
         incremental(eic_temp(i)) = ec_temp(i);
         decremental(eic_temp(i)) = ec_temp(i);
         imin(eic_temp(i)) = true; imax(eic_temp(i)) =true;
      end;
     if pflag = true & nflag = false then do;
        incremental(ic_temp(i)) = c_temp(i);
        imin(eic_temp(i)) = true;
     end;
     else if pflag = false & nflag = true then do;
        decremental(ic_temp(i)) = c_temp(i);
        imax(eic_temp(i)) = true;
     end;
   end;
end;
temp = cc + el;
if temp > nnonz then
   nnonz = temp;
allocate obj_index(nrow);
allocate col_index(nrow - 1);
allocate eobj_index(nrow);
allocate ecol_index(nrow - 1);
nrow = num_degen_rows + 1;  ncol = 1 + nrow;
ecol_index(*) = 0;
free a; free ia;  free la;
allocate a(nnonz); allocate ia(nnonz);
allocate ea(nnonz); allocate eia(nnonz);
col_index(*) = 0; obj_index(*) = 0;
/* retrieval step (v) */
allocate la;   la(nrow + 1) = 1;
k = 0;
ic = 0; /* # of elements in COL_INDEX */
do i = 1 to l;
   if ic_temp(i) <= orig_nrow then   ic = ic + 1;
   if abs(c_temp(i)) <= tolcost then do;
      la(nrow + i + 1) = la(nrow + i);
      if ic_temp(i) <= orig_nrow then do;
```

```
            obj_index(ic) = k + 1;
            col_index(ic) = ic_temp(i);
         end;
      end;
      else do;
         k = k + 1;
         if ic_temp(i) <= orig_nrow then do;
            obj_index(ic) = k;
            col_index(ic) = ic_temp(i);
         end;
         a(k) = c_temp(i);
         ia(k) = 1;
         la(nrow + i + 1) = la(nrow + i) + 1;
      end;
      ll = la_temp(i); /* first nonzero in column */
      kk = la_temp(i + 1) - 1; /* last nonzero in column */
      do n = ll to kk;
         a(k + 1) = a_temp(n);
         ia(k + 1) = ia_temp(n);
         la(nrow + i + 1) = la(nrow + i + 1) - 1;
         k = k + 1;
      end;
   end;
   obj_index(ic + 1) = k + 1;
   /* retrieval step (vi) */
   k = 0;
   ic = 0;
   ela_temp(1) = 1;
   if el > 0 then
      do i = 1 to el;
         ic = ic + 1;
         if abs(ec_temp(i)) < tolcost then do;
            eobj_index(ic) = k + 1;
            ecol_index(ic) = eic_temp(i);
         end;
         else do;
            k = k + 1;
            eobj_index(ic) = k;
            ecol_index(ic) = eic_temp(i);
            ea(k) = ec_temp(i);
            eia(k) = 1;
         end;
         ll = ela_temp(i);
         kk = ela_temp(i + 1) - 1;
         do n = ll to kk;
            ea(k + 1) = ea_temp(n);
            eia(k + 1) = eia_temp(n);
            k = k + 1;
         end;
      end;
   eobj_index(ic + 1) = k + 1;
   free c_temp;
```

```
free ic_temp;
free a_temp;
free ia_temp;
free la_temp;
rowtype(*) = 'g';
rowtype(1) = 'n';
direct = 'min';
allocate b(nrow)
do i = 1 to (orig_nrow - 1) while (col_index(i) \= 0);
   if imax(col_index(i)) = false then do;
      b(*) = 0;
      y(*) = 0;
      kk = obj_index(i);
      ll = obj_index(i + 1) - 1;
/* computation step I    */
      if iflag = true then do;
         call first_pass(kk,ll,col_index(i),a,ia,1);
         iflag = false;
      end;
      else do;
/* computation step II.1b */
         do y = kk to ll;
            y(ia(j)) = -a(j);
         end;
/* computation step II.2 */
         cobj = y(1);
         y(1) = 0;
/* computation step II.3 */
         yflag = false;
         call FTRAN(1);
/* computation step II.4 */
         do j = 2 to nrow;
            if y(j) < -tolcost then
               yflag = true;
         end;
         if yflag = false then
            decremental(col_index(i)) = -(y(iobj) + cobj);
         else do;
/* computation step II.5 */
            x(*) = y(*);
            call PRIMAL;
/* computation step II.6 */
            decremental(col_index(i)) = -(x(iobj) + cobj);
         end;
      end;
         if finstat = 'infeasible' then
            sstat(col_index(i)) = '*';
   end;
end;
do i = 1 to (orig_nrow - 1) while (ecol_index(i) \= 0);
   if imax(ecol_index(i)) = false then do;
      b(*) = 0;
```

```
      y(*) = 0;
      kk = obj_index(i);
      ll = obj_index(i + 1) - 1;
/* computation step I  */
      if iflag = true then do;
         call first_pass(kk,ll,col_index(i),ea,eia,1);
         iflag = false;
      end;
      else do;
/* computation step II.1a */
         do y = kk to ll;
            y(eia(j)) = -ea(j);
         end;
/* computation step II.2 */
         cobj = y(1);
         y(1) = 0;
/* computation step II.3 */
         yflag = false;
         call FTRAN(1);
/* computation step II.4 */
         do j = 2 to nrow;
            if y(j) < -tolcost then
               yflag = true;
         end;
         if yflag = false then
            decremental(ecol_index(i)) = -(y(iobj) + cobj);
         else do;
/* computation step II.5 */
            x(*) = y(*);
            call PRIMAL;
/* computation step II.6 */
            decremental(ecol_index(i)) = -(x(iobj) + cobj);
         end;
      end;
         if finstat = 'infeasible' then
            sstat(ecol_index(i)) = '*';
   end;
end;

do i = 1 to (orig_nrow - 1) while (col_index(i) \= 0);
   if imin(col_index(i)) = false then do;
      b(*) = 0;
      y(*) = 0;
      kk = obj_index(i);
      ll = obj_index(i + 1) - 1;
/* computation step I  */
      if iflag = true then do;
         call first_pass(kk,ll,col_index(i),a,ia,2);
         iflag = false;
      end;
      else do;
```

```
/* computation step II.1b */
         do y = kk to ll;
            y(ia(j)) = a(j);
         end;
/* computation step II.2 */
         cobj = y(1);
         y(1) = 0;
/* computation step II.3 */
         yflag = false;
         call FTRAN(1);
/* computation step II.4 */
         do j = 2 to nrow;
            if y(j) < -tolcost then
               yflag = true;
         end;
         if yflag = false then
            incremental(col_index(i)) = y(iobj) + cobj;
         else do;
/* computation step II.5 */
            x(*) = y(*);
            call PRIMAL;
/* computation step II.6 */
            incremental(col_index(i)) = x(iobj) + cobj;
         end;
      end;
         if finstat = 'infeasible' then
            bstat(col_index(i)) = '*';
   end;
end;
do i = 1 to (orig_nrow - 1) while (ecol_index(i) \= 0);
   if imin(ecol_index(i)) = false then do;
      b(*) = 0;
      y(*) = 0;
      kk = obj_index(i);
      ll = obj_index(i + 1) - 1;
/* computation step I  */
      if iflag = true then do;
         call first_pass(kk,ll,col_index(i),ea,eia,2);
         iflag = false;
      end;
      else do;
/* computation step II.1a */
         do y = kk to ll;
            y(eia(j)) = ea(j);
         end;
/* computation step II.2 */
         cobj = y(1);
         y(1) = 0;
/* computation step II.3 */
         yflag = false;
         call FTRAN(1);
```

```
/* computation step II.4 */
         do j = 2 to nrow;
            if y(j) < -tolcost then
               yflag = true;
         end;
         if yflag = false then
            incremental(ecol_index(i)) = y(iobj) + cobj;
         else do;
/* computation step II.5 */
            x(*) = y(*);
            call PRIMAL;
/* computation step II.6 */
            incremental(ecol_index(i)) = x(iobj) + cobj;
         end;
      end;
         if finstat = 'infeasible' then
            bstat(ecol_index(i)) = '*';
   end;
end;
do i = 1 to nn;
   temp_shadow = -decremental(switch(i));
   temp_stat = sstat(switch(i));
   decremental(switch(i)) = -incremental(switch(i));
   sstat(switch(i)) = bstat(switch(i));
   incremental(switch(i)) = temp_shadow;
   bstat(switch(i)) = temp_stat;
end;

END:SUB_POB;

FIRST_PASS: PROCEDURE(kk,ll,i,element,index,mode);
dcl (i,mode,ll,kk,j,index(*)   fixed bin,
    element(*)                 float bin;
/* computation step I.1a or I.1b */
do j = kk to ll;
   b(index(j)) = -element(j);
end;
if mode = 2 then
   b(*) = -b(*);
/* computation step I.2 */
cobj = b(1);
b(1) = 0;
/* computation step I.3 */
call INITIAL_BASIS;
/* computation step I.4 */
call PRIMAL;
/* computation step I.5 */
if mode = 1 then
   decremental(i) = -(x(iobj) + cobj);
   else incremental(i) = x(iobj) + cobj;

END FIRST_PASS;
```

APPENDIX B

SAMPLE OUTPUT FROM OTHER EXAMPLES

```
     MAX 3A + 2B -  C  + 4D
     ST   A +  B +  C  +  D  = 2.0
         2A -  B           <= 0.4
               B + 2C +  3D >= 2.366667
                   C +   D <= 0.5
          A,B,C,D          >= 0
```

VARIABLE	VALUE	REDUCED COST
A	6.333330000E-01	0.000000000E+00
B	8.666670000E-01	0.000000000E+00
C	0.000000000E+00	6.000000000E+00
D	5.000000000E-01	0.000000000E+00

ROW	LOGICAL	DUAL VALUE
1	-5.633333000E+00	-1.000000000E+00
2	0.000000000E+00	3.000000000E+00
3	9.999999995E-07	0.000000000E+00
4	0.000000000E+00	-1.000000000E+00
5	0.000000000E+00	4.000000000E+00

SHADOW PRICES:

ROW	INCREMENTAL	DECREMENTAL
2	2.333333333E+00	3.000000000E+00
3	0.000000000E+00	3.333333333E-01
4	-1.000000000E+00	0.000000000E+00
5	1.666666667E+00	4.000000000E+00

```
MAX 3A + 2B -  C  + 4D
ST   A +  B +  C  +  D  = 1.0
    2A -  B            <= 0.4
          B + 2C +  3D >= 2.0
              C +   D <= 0.5
     A,B,C,D           >= 0
```

VARIABLE	VALUE	REDUCED COST
A	0.000000000E+00	0.000000000E+00
B	5.000000000E-01	0.000000000E+00
C	0.000000000E+00	6.000000000E+00
D	5.000000000E-01	0.000000000E+00

ROW	LOGICAL	DUAL VALUE
1	-3.000000000E+00	-1.000000000E+00
2	0.000000000E+00	3.000000000E+00
3	9.000000000E-01	0.000000000E+00
4	0.000000000E+00	-1.000000000E+00
5	0.000000000E+00	4.000000000E+00

SHADOW PRICES:

ROW	INCREMENTAL		DECREMENTAL	
2	3.000000000E+00		3.000000000E+00	*
3	0.000000000E+00		0.000000000E+00	
4	-1.000000000E+00	*	-1.000000000E+00	
5	4.000000000E+00		4.000000000E+00	*

* direction of change is infeasible

NOTE: On LINDO, the range analysis for this problem had all r_i's = 0.

```
MIN 74A + 40B + 50C + 10D
ST   3A +   B +  2C        >=  8.0
     2A +  2B        +   D >= 11.0
     4A       +  3C        >= 10.66667
     2A +  3B +  2C +   D >= 11.0
     A,B,C,D             >= 0
```

VARIABLE	VALUE	REDUCED COST
A	2.666667500E+00	0.000000000E+00
B	0.000000000E+00	1.525000000E+01
C	0.000000000E+00	0.000000000E+00
D	5.666665000E+00	0.000000000E+00

ROW	LOGICAL	DUAL VALUE
1	-2.540000450E+02	1.000000000E+00
2	2.500000000E-06	0.000000000E+00
3	0.000000000E+00	-5.250000000E+00
4	0.000000000E+00	-1.350000000E+01
5	0.000000000E+00	-4.750000000E+00

SHADOW PRICES:

ROW	INCREMENTAL	DECREMENTAL
2	-1.800000000E+01	0.000000000E+00
3	-1.000000000E+01	-3.555555556E+00
4	-1.350000000E+01	0.000000000E+00
5	-6.444444444E+00	0.000000000E+00

```
MIN 10X1 + 16X2 + 32X3 + 14X4 + 22X5 +
    40X6 + 22X7 + 24X8 + 34X9

     ST   X1 + X2 + X3 >= 1
          X4 + X5 + X6 >= 1
          X7 + X8 + X9 >= 1
          X1 + X4 + X7 >= 1
          X2 + X5 + X8 >= 1
          X3 + X6 + X9 >= 1
          X1,X2,X3,X4,X5,X6 >= 0
```

VARIABLE	VALUE	REDUCED COST
X1	0.000000000E+00	0.000000000E+00
X2	1.000000000E+00	0.000000000E+00
X3	0.000000000E+00	6.000000000E+00
X4	1.000000000E+00	0.000000000E+00
X5	0.000000000E+00	2.000000000E+00
X6	0.000000000E+00	1.000000000E+01
X7	0.000000000E+00	4.000000000E+00
X8	0.000000000E+00	0.000000000E+00
X9	1.000000000E+00	0.000000000E+00

ROW	LOGICAL	DUAL VALUE
1	-6.400000000E+01	1.000000000E+00
2	0.000000000E+00	-1.000000000E+01
3	0.000000000E+00	-1.400000000E+01
4	0.000000000E+00	-1.800000000E+01
5	0.000000000E+00	0.000000000E+00
6	0.000000000E+00	-6.000000000E+00
7	0.000000000E+00	-1.600000000E+01

SHADOW PRICES:

ROW	INCREMENTAL	DECREMENTAL
2	-1.000000000E+01	0.000000000E+00
3	-1.400000000E+01	-4.000000000E+00
4	-1.800000000E+01	-2.000000000E+00
5	-1.000000000E+01	0.000000000E+00
6	-1.600000000E+01	-6.000000000E+00
7	-3.200000000E+01	-1.600000000E+01

VITA

Darren A. Smith was born in Springfield, Virginia on November 7, 1962. He attended Northern Virginia Community College for one year before transferring to the University of Tennessee, Knoxville. In March 1985 he received a Bachelor of Science degree in Accounting and was the top graduate in the Business school that quarter.

The author is an associate member of ORSA/TIMS and a CPA. He currently works for Federal Express in Memphis, TN.