Mathematical Programming 13 (1977) 241–254.
North-Holland Publishing Company

MATRIX AUGMENTATION AND PARTITIONING IN THE UPDATING OF THE BASIS INVERSE

Johannes BISSCHOP and Alexander MEERAUS

Development Research Center, World Bank, Washington, DC, U.S.A.

Received 2 August 1976
Revised manuscript received 3 May 1977

A compact and flexible updating procedure using matrix augmentation is developed. It is shown that the representation of the updated inverse does not grow monotonically in size, and may actually decrease during certain simplex iterations. Angular structures, such as GUB, are handled naturally within the partitioning framework, and require no modifications of the simplex method.

Key words: Large-scale programming, Basis updating, Matrix modifications, Partitioning methods.

1. Introduction

There are basically two methods which are being used for the inversion and updating of the basis matrix in linear programming. Until recently the product form of the inverse (PFI) was used predominantly because of its simplicity to implement. Since then the usage of triangular (LU) decomposition has become widely recognized because of the superiority of this form of the inverse over the original product form in terms of speed, compactness, and accuracy (see [1, 15]). Research in this area is still extensive, and several sparsity-exploiting variants of the basic LU decomposition are under study (see [6, 11, 13]).

In this paper we propose yet another method for updating the inverse basis using augmentation and partitioning. Our main interest lies in solving large sparse systems that are kept entirely in core. We start out showing how general modifications of a reinverted basis B can be handled using augmentation and partitioning. Primal simplex iterations are then viewed as a special case, and studied in detail. It is shown that for problems with an angular structure (such as GUB), a reduced working basis can be maintained without requiring a specialization of the simplex method. Finally, the methodology is analyzed and compared to existing procedures.

2. Matrix modifications

The study of matrix modifications or matrix tearing dates back to the works of Kron [9] and Sherman and Morrison [14]. The updating of a basis B in linear

programming is an example of a matrix modification where several columns previously not in B are interchanged for columns in B. We will consider various modifications of the matrix B. The basic ideas in the following theorem stem from the work of Kron [9], and can also be found in Rose and Bunch [12].

Theorem 1. *The matrix B is $m \times m$, U is $m \times p$, V is $p \times m$, and S is $p \times p$. Assume that S^{-1} exists. Let $\hat{B} = B + USV$ be a modification of the matrix B. Then*

(i) *x is a solution of $\hat{B}x = b$ if and only if x is part of the solution of the augmented system*

$$\begin{bmatrix} B & U \\ V & -S^{-1} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} b \\ 0 \end{bmatrix};$$

(ii) *x is a solution of $x\hat{B} = c$ if and only if x is part of the solution of the augmented system*

$$[x \; y] \begin{bmatrix} B & U \\ V & -S^{-1} \end{bmatrix} = [c \;\; 0].$$

Proof. By construction, the vector x solves $\hat{B}x = b$ if and only if it solves the augmented system

$$\begin{bmatrix} \hat{B} & 0 \\ V & -S^{-1} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} b \\ 0 \end{bmatrix}. \tag{2.1}$$

To solve (2.1) is equivalent to solving

$$\begin{bmatrix} B & U \\ V & -S^{-1} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} b \\ 0 \end{bmatrix}. \tag{2.2}$$

One verifies this by applying the non-singular transformation

$$\begin{bmatrix} I_m & -US \\ 0 & I_p \end{bmatrix} \tag{2.3}$$

to both sides of (2.1) to obtain the system (2.2). Similarly, the vector x solves $x\hat{B} = c$ if and only if it solves the augmented system

$$[x \;\; y] \begin{bmatrix} \hat{B} & U \\ 0 & -S^{-1} \end{bmatrix} = [c \;\; 0]. \tag{2.4}$$

To solve (2.4) is equivalent to solving

$$[x \;\; y] \begin{bmatrix} B & U \\ V & -S^{-1} \end{bmatrix} = [c \;\; 0]. \tag{2.5}$$

One verifies this by applying the non-singular transformation

$$\begin{bmatrix} I_m & 0 \\ -SV & I_p \end{bmatrix} \tag{2.6}$$

to both sides of (2.4) to obtain the system (2.5). This concludes the proof of the theorem.

Corollary. *If* $\hat{B} = B + USV$, *and* S^{-1} *is not readily available, one may use the above theorem for* $\hat{B} = B + (US)I_pV = B + UI_p(SV)$ *and consider either of the matrices*

$$\begin{bmatrix} B & US \\ V & -I_p \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} B & U \\ SV & -I_p \end{bmatrix}.$$

In actual applications the Corollary to Theorem 1 is of interest as the inversion of S can be avoided when inverting the augmented matrix. If one adds the assumption that B^{-1} exists, Theorem 1 provides us with a representation of $\hat{B}^{-1}$. Without loss of generality, assume that $\hat{B} = B + UV$. Then using Kron's partitioning scheme, $x = \hat{B}^{-1}b$ can be computed via

(i) $y = -Q^{-1}VB^{-1}b$,

(ii) $x = B^{-1}(b - Uy)$;

and $x = c\hat{B}^{-1}$ can be computed via

(i) $y = -cB^{-1}UQ^{-1}$,

(ii) $x = (c - yV)B^{-1}$

where

$$Q = (-I_p - VB^{-1}U).$$

The following theorem concerning the expansion and contraction of matrices will be used in the sequel.

Theorem 2. *Consider the partitioned matrices*

$$A = \begin{bmatrix} A_1 & A_2 \\ A_3 & A_4 \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} B_1 & B_2 \\ B_3 & B_4 \end{bmatrix},$$

where $B = A^{-1}$, *and* A_1^{-1} *and* B_4^{-1} *are assumed to exist. Then knowledge of* A_1^{-1}, A_2, A_3, *and* $Q^{-1} = (A_4 - A_3A_1^{-1}A_2)^{-1}$ *is sufficient to determine the inverse of the larger matrix* A, *and knowledge of* B_4^{-1}, B_1, B_2, *and* B_3 *is sufficient to determine the inverse of the smaller matrix* A_1.

Proof. It is straightforward to verify that $AB = I$ whenever

$$B_1 = A_1^{-1} + A_1^{-1}A_2Q^{-1}A_3A_1^{-1},$$
$$B_2 = -A_1^{-1}A_2Q^{-1},$$
$$B_3 = -Q^{-1}A_3A_1^{-1},$$
$$B_4 = Q^{-1},$$

and

$$Q = (A_4 - A_3A_1^{-1}A_2).$$

Then, by substitution, one can verify that $A_1^{-1} = B_1 - B_2B_4^{-1}B_3$. This concludes the proof of the theorem.

Limiting our study of modification to linear programming, there are four basic modifications of a basis matrix B that result in an augmented system.

(a) *Column replacement.* Let the column vector a_i of the $m \times m$ matrix B be interchanged for the column vector a_j not yet in B. Then $\hat{B} = B + UV$ where the $1 \times m$ vector V is a unit vector with the one in the ith position, and the $m \times 1$ vector U contains the vector difference $a_j - a_i$.

(b) *Row replacement.* Let the row vector r_i of B be interchanged for the row vector r_j not yet in B. Then $\hat{B} = B + UV$ where the $1 \times m$ vector V contains the vector difference $r_j - r_i$, and the $m \times 1$ vector U is a unit vector with the one in the i-th position.

(c) *Row and column addition.* A modification adding a row and a column to the previous basis results immediately in an augmented system.

(d) *Row and column deletion.* After a row and column have been added during one stage of the solution procedure, a subsequent deletion will result in a reduced system of the same form.

It is of interest to point out that the above modifications can be performed in any order. This flexibility associated with augmentation and partitioning is not present when modifications are executed by either product form or LU decomposition techniques.

3. Primal simplex iterations after reinversion

It is assumed that the sparse matrix A representing the LP problem is stored in core, together with a compact representation of the basis inverse B^{-1}. By Theorem 1, finding an inverse representation for the modified matrix $\hat{B} = B + UV$ is equivalent to finding an inverse representation for the augmented system

$$\begin{bmatrix} B & U \\ V & -I \end{bmatrix}. \tag{3.1}$$

This involves the matrices B^{-1}, U, V, and Q^{-1}, where $Q = (-I - VB^{-1}U)$. The size of Q is bounded from above by the number of simplex iterations following the latest reinversion of the basis matrix. As we will see, the dimensions of Q may in some simplex iterations be unaltered, or even diminished. It is important to note that only Q^{-1} needs to be stored explicitly, and that both U and V are stored with pointers. Recall that U contains vector differences involving columns of A, and that V contains unit vectors. It is effectively the size of the matrix Q that determines the build-up of additional non-zero elements in the simplex method. It will be necessary to distinguish between four situations that may arise during simplex iterations following the latest reinversion of a basis B.

Case 1. A non-B vector replaces a B vector. The dimensions of the Q matrix associated with the augmented system (3.1) will increase by one.

Case 2. A non-B vector replaces another non-B vector (which had entered the basis in a previous iteration). The dimensions of Q will be unaltered during such interchange of vectors not belonging to B.

Case 3. A B vector (which had left the basis in a previous iteration) replaces another B vector. As in Case 2, the dimensions of Q will be unaltered.

Case 4. A B vector replaces a non-B vector. The dimensions of the Q matrix will actually decrease by one.

Each of the above cases will be considered separately when discussing the modification of the Q matrix in a simplex iteration.

Assume that k iterations have passed since the latest reinversion, and that a total of $l \leq k$ non-B column vectors have been interchanged for B column vectors. The resulting augmented system is

$$\begin{bmatrix} B & U_l \\ V_l & I_l \end{bmatrix}. \tag{3.2}$$

The $m \times l$ matrix $U_l = U_l^1 - U_l^2$, where U_l^1 contains all columns that have entered into the basis, and U_l^2 contains all columns that have left the basis. As part of an inductive argument (the case $l = 1$ is straightforward) we assume that

$$V_l B^{-1} U_l^2 = I_l \tag{3.3}$$

after k iterations. It will be shown for each of the above cases that this assumption will also be satisfied after the $k+1^{\text{st}}$ iteration. Note that

$$\begin{aligned} Q_l &= (-I_l - V_l B^{-1} U_l), \\ &= (-I_l - V_l B^{-1} U_l^1 + V_l B^{-1} U_l^2), \\ &= (-V_l B^{-1} U_l^1). \end{aligned}$$

At the end of the kth iteration after a reinversion during the revised simplex method we have the matrices/vectors B^{-1}, Q_l^{-1}, X_0^k (the solution vector), a^k (the entering vector), and $\lambda^k = c_B^k B^{-1}$. Here λ^k is a partially updated pricing vector, and c_B^k denotes the vector of objective function coefficients corresponding to the basic variables of the kth iteration. Next we will discuss the computations that are needed for a primal iteration.

Step 1. *Transform the Entering Vector.* Compute

$$\begin{aligned} w &= Q_l^{-1}(-V_l B^{-1} a^k), && \text{(save } B^{-1}a^k\text{)}, \\ y^k &= B^{-1}(a^k - U_l w) \\ &= B^{-1}a^k - B^{-1}U_l^1 w + B^{-1}U_l^2 w. \end{aligned}$$

In this step one needs to access the matrix B^{-1} twice when computing $B^{-1}a^k$ and $B^{-1}U_l^1 w$. In the event that a^k belongs to B, the matrix B^{-1} needs to be accessed only once. The matrix $B^{-1}U_l^2$ is a set of l unit vectors where the location of the ones is known from the set of pointers determining U_l^2.

Step 2. *Determine the Leaving Vector.* Use x_0^k to determine the vector a_i, the ith column of B, that will leave the basis. Establish which of the four cases discussed previously has occurred. Update x_0^k to become x_0^{k+1} using the vector y^k.

Step 3. *Update the Inverse Representation.* Modify Q_l^{-1} into $\hat{Q}^{-1} = (-\hat{V}B^{-1}\hat{U}^1)^{-1}$. This will be done separately for each case following Step 4.

Step 4. *Compute the Pricing Vector.* Let c_B^{k+1} denote the vector of cost coefficients associated with the basic variables of iteration $k+1$. Then compute

$$w = (-c_B^{k+1}B^{-1}\hat{U})\hat{Q}^{-1},$$
$$\pi^{k+1} = (c_B^{k+1} - w\hat{V})B^{-1},$$

where π^{k+1} is the current pricing vector. Here

$$c_B^{k+1}B^{-1}\hat{U} = c_B^{k+1}B^{-1}\hat{U}^1 - c_B^{k+1}B^{-1}\hat{U}^2.$$

Note that the matrix $B^{-1}\hat{U}^2$ is a set of unit vectors where the location of the ones is known from the set of pointers determining $\hat{U}^2$. Also note that

$$\begin{aligned} c_B^{k+1}B^{-1} &= c_B^kB^{-1} + d^{k+1}e_iB^{-1}, \\ &= \lambda^k + d^{k+1}e_iB^{-1}, \\ &= \lambda^{k+1}, \end{aligned}$$

where d^{k+1} denotes the difference between the cost coefficients associated with a^k and a_i respectively, and e_i denotes a unit vector with the one in the ith position, where i is the number of the position of the leaving vector a_i in B. In this step one needs to access the matrix B^{-1} when computing $(c_B^{k+1} - w\hat{V})B^{-1}$, and e_iB^{-1} (a partial access). Some of the information used in Steps 1 and 4 is also needed when Q_l^{-1} is modified into $\hat{Q}^{-1}$. Each of the four cases discussed previously will be considered next.

Case 1. The non-B vector a_j replaces the B vector a_i. The matrix $\hat{U}$ is equal to the matrix U_l plus an added column vector containing the difference $a_j - a_i$. The matrix $\hat{V}$ is equal to the matrix V_l plus an added row vector containing an m-dimensional unit vector e_i with a one in the ith position, where i is the number of the position of the leaving vector a_i in B. The induction argument of (3.3) is completed for this case by observing that $\hat{V}B^{-1}\hat{U}^2 = \hat{I} = I_{l+1}$. Here

$$\hat{Q} = \begin{bmatrix} Q_l & q^C \\ q^R & q^I \end{bmatrix},$$

where $q^C = V_lB^{-1}a_j$ and $[q^R \quad q^I] = e_iB^{-1}\hat{U}^1$. Both $B^{-1}a_j$ and e_iB^{-1} are needed in each simplex iteration so that no extra sweep through B^{-1} is required for the updating of $\hat{Q}^{-1}$. Let

$$\hat{Q}^{-1} = \begin{bmatrix} \hat{Q}_1 & \hat{Q}_2 \\ \hat{Q}_3 & \hat{Q}_4 \end{bmatrix}.$$

Then, using Theorem 2,

$$\begin{aligned} \hat{Q}_1 &= Q_l^{-1} + \rho Q_l^{-1}q^Cq^RQ_l^{-1}, \\ \hat{Q}_2 &= -\rho Q_l^{-1}q^C, \\ \hat{Q}_3 &= -\rho q^RQ_l^{-1}, \\ \hat{Q}_4 &= \rho, \\ \rho &= 1/(q^I - q^RQ_l^{-1}q^C). \end{aligned}$$

Case 2. The non-B vector a_j replaces the non-B vector a_i which occupies the pth column of U_l^1. The matrix $\hat{U}^1$ differs from the matrix U_l^1 in that its pth

column contains the vector a_j. The matrices $\hat{U}^2$ and $\hat{V}$ are exactly equal to U_l^2 and V_l respectively. The induction argument of (3.3) is completed for this case by observing that $\hat{V}B^{-1}\hat{U}^2 = V_lB^{-1}U_l^2 = \hat{I} = I_l$. Here

$$\hat{Q} = Q_l + sr,$$

where

$$s = q - q_p \quad (l \times 1),$$
$$r = e_p \quad (1 \times l),$$

with

$$q = -\hat{V}B^{-1}a_j = -V_lB^{-1}a_j,$$

q_p, the pth column of Q_l,

e_p, the $(1 \times l)$ unit vector with a one in the pth position.

Note that q is available from Step 1 in the simplex method. Using Kron's tearing formula [9], one can compute

$$\hat{Q}^{-1} = Q_l^{-1} + Q_l^{-1}s(-1 - rQ_l^{-1}s)^{-1}rQ_l^{-1}.$$

Note that

$$\begin{aligned}(-1 - rQ_l^{-1}s) &= (-1 - rQ_l^{-1}(q - q_p)) \\ &= (-1 - rQ_l^{-1}q + 1) \\ &= (-rQ_l^{-1}q)\end{aligned}$$

and that

$$\hat{Q}^{-1} = Q_l^{-1} + (-rQ_l^{-1}q)^{-1}(Q_l^{-1}qrQ_l^{-1} - Q_l^{-1}q_prQ_l^{-1}),$$

where $Q_l^{-1}q_prQ_l^{-1}$ is the zero matrix with the exception of its pth row which is equal to the pth row of Q_l^{-1}.

Case 3. The B vector a_j (corresponding to the jth column of B) replaces the B vector a_i (corresponding to the ith column of B). Assume that a_j occupies the pth column of U_l^2. The matrix $\hat{U}^2$ differs from U_l^2 in that its pth column contains the vector a_i. The matrix $\hat{V}$ differs from the matrix V_l in that its pth row contains a unit vector with the one in the ith position. The matrix $\hat{U}^1$ is exactly equal to the matrix U_l^1. The induction argument of (3.3) is completed for this case by observing that $\hat{V}B^{-1}\hat{U}^2 = \hat{I} = I_l$ by construction. Here

$$\hat{Q} = Q_l + sr,$$

where

$$s = e_p \quad (l \times 1),$$
$$r = q - q_p \quad (1 \times l),$$

with

$$q = -e_iB^{-1}\hat{U}^1,$$

q_p, the pth row of Q_l,

e_p, e_i, unit vectors of length l and m with a one in the pth and ith position respectively.

Note that $e_I B^{-1}$ is used in Step 4 in the simplex method. Using Kron's tearing formula [9], one can compute

$$\hat{Q}^{-1} = Q_l^{-1} + Q_l^{-1}s(-1 - rQ_l^{-1}s)^{-1}rQ_l^{-1}.$$

Note that

$$\begin{aligned}(-1 - rQ_l^{-1}s) &= (-1 - (q - q_p)Q_l^{-1}s),\\ &= (-1 - qQ_l^{-1}s + 1),\\ &= (-qQ_l^{-1}s),\end{aligned}$$

and that

$$\hat{Q}^{-1} = Q_l^{-1} + (-qQ_l^{-1}s)^{-1}(Q_l^{-1}sqQ_l^{-1} - Q_l^{-1}sq_pQ_l^{-1})$$

where $Q_l^{-1}sq_pQ_l^{-1}$ is the zero matrix with the exception of its pth column, which is equal to the pth column of Q_l^{-1}.

Case 4. The B vector a_j (corresponding to the jth column of B) replaces the non-B vector a_i which occupies the pth column of U_l^1. Assume that a_j occupies the qth column of U_l^2. The matrix $\hat{U}$ differs from the matrix U_l in that its pth column will be empty, while the qth column of $\hat{U}^2$ will contain the pth column of U_l^2. The matrix $\hat{V}$ differs from the matrix V_l in that its pth row will be empty, while its qth row will contain the pth row of V_l. Let P be a $l \times l$ permutation matrix which equals the identity matrix with columns p and q interchanged. Note that P is symmetric, and therefore $P^{-1} = P^{\tau} = P$. Let R be an $l \times l$ reduction matrix which equals the identity matrix minus the one in the pth column. Let $\hat{U} = \hat{U}^1 - \hat{U}^2$, then

$$\begin{aligned}\hat{U}^1 &= U_l^1 R,\\ \hat{U}^2 &= U_l^2 PR,\\ \hat{V} &= RPV_l,\\ \hat{I} &= RI_lR.\end{aligned}$$

The resulting matrix $\hat{Q} = (-\hat{I} - \hat{V}B^{-1}\hat{U})$ has its pth column and pth row equal to the zero vector, and therefore contains a smaller matrix with dimensions $(l-1)\times(l-1)$. One may verify this as follows:

$$\begin{aligned}\hat{Q} &= -RI_lR - RPV_lB^{-1}U_l^1R + RPV_lB^{-1}U_l^2PR,\\ &= R[-I_l - PV_lB^{-1}U_l^1 + PV_lB^{-1}U_l^2P]R,\\ &= R[-I_l - PV_lB^{-1}U_l^1 + PI_lP]R,\\ &= R[-PV_lB^{-1}U_l^1]R,\\ &= RPQ_lR.\end{aligned}$$

This shows that no new information is needed to update the basis inverse in this case. For the sake of convenience, redefine $\hat{Q}$ to be the $(l-1)\times(l-1)$ matrix contained in RPQ_lR. To determine $\hat{Q}^{-1}$ we use the larger inverse $(PQ_l)^{-1}$, and rely upon Theorem 2. Let γ be the intersection of the pth row and pth column of $(PQ_l)^{-1}$ which is equivalent to the intersection of the pth row and qth

column of Q_l^{-1}. Let the $1\times(l-1)$ vector q^R be equal to the pth row of Q_l^{-1} without the element γ. Similarly, let the $(l-1)\times 1$ vector q^C be equal to the qth column of Q_l^{-1} without the element γ. Also, let Q_1 be the matrix Q_l^{-1} without the pth row and qth column. Then, by Theorem 2,

$$\hat{Q}^{-1} = Q_1 - (1/\gamma)q^C q^R.$$

This completes the discussion for each of the four cases that can occur while updating the inverse basis representation in the revised simplex method.

4. Angular structures

Special attention has been given to angular structures in linear programming in an effort to reduce the size of the working basis (see [3, 8]). Chapter 6 in [10] gives an excellent discussion of Generalized Upper Bounding, and its generalization to block-triangular form. Such special linear programs with $(m+p)$ rows are solved with a working basis that has size m, where m is usually much smaller than p. The result is a substantial saving in storage. The methods used, however, are a specialization of the simplex method, and require special implementation because of the increased complexity in updating the reduced working basis. Since the growth of the inverse representation of this paper is a function of the number of iterations following reinversion and *not* the size of the original tableau, a specialization of the simplex method for problems with an angular structure is not required. Only a partitioning of the original basis is needed to obtain the reduced working basis developed in [10]. Assume that for $i = 0, 1, 2, \ldots, p$, A_i is $m_0 \times n_i$, x_i is $n_i \times 1$, D_i is $m_i \times n_i$, and $(x_0)_1$ is the first component of x_0. Then consider the following class of linear programming problems. Minimize $(x_0)_1$ subject to

$$\begin{aligned}
D_1 x_1 &= b_1,\\
D_2 x_2 &= b_2,\\
&\ddots \quad \vdots\\
D_p x_p &= b_p,\\
A_0 x_0 + A_1 x_1 + A_2 x_2 + \cdots + A_p x_p &= b,\\
x_i \geq 0 \quad &\text{for all } i.
\end{aligned}$$

Assume that the row rank of the matrices D_i is m_i. Then any basic feasible solution must include at least m_i components of the vector x_i. The initial basis B can therefore always be partitioned into the form

$$\begin{bmatrix} D & U \\ V & \hat{B} \end{bmatrix} = B.$$

Let $\hat{p} = \sum_{i=1}^{p} m_i$. Then

(i) The ith block of columns in B, $i = 1, 2, \ldots, p$ corresponds to m_i components of the vector x_i.

(ii) The $\hat{p} \times \hat{p}$ matrix D is a block diagonal matrix consisting of p nonsingular blocks with dimensions $(m_i \times m_i)$.

(iii) The $\hat{p} \times m_0$ matrix U is a set of vectors which are either zero vectors, or vectors containing m_i nonzero elements.

(iv) The $m_0 \times \hat{p}$ matrix V and $m_0 \times m_0$ matrix $\hat{B}$ consist of selected columns from the last m_0 rows of the original tableau.

The inverse representation of B requires only knowledge of D^{-1}, U, V and Q^{-1}, where $Q = (\hat{B} - VD^{-1}U)$ and D^{-1} consists of p inverted blocks with dimensions $(m_i \times m_i)$. The $(m_0 \times m_0)$ matrix Q is exactly the reduced working basis developed in [8].

Let the $(\hat{p} + m_0)$-dimensional vectors z, d, and c be partitioned into (z_1, z_2), (d_1, d_2) and (c_1, c_2) respectively. Then $Bz = d$ is computed via

(i) $z_2 = Q^{-1}(d_2 - VD^{-1}d_1)$,

(ii) $z_1 = D^{-1}(d_1 - Uz_2)$

and $zB = c$ is computed via

(i) $z_2 = (c_2 - c_1D^{-1}U)Q^{-1}$,

(ii) $z_1 = (c_1 - z_2V)D^{-1}$.

5. Empirical evidence and computational aspects

We studied in detail the pivoting sequence for an agricultural production scheduling problem with 1124 rows, 2982 columns and 30,700 non-zero elements

Table 1
A pivoting sequence of 9500 iterations

Iter	Case 1	Case 2	Case 3	Case 4	Q-size
10	9.29	0.60	0.05	0.06	9.23
	(0.82)	(0.75)	(0.22)	(0.26)	(0.94)
20	17.49	1.94	0.21	0.36	17.13
	(1.48)	(1.34)	(0.45)	(0.66)	(1.84)
30	24.95	3.66	0.43	0.96	23.99
	(2.18)	(1.91)	(0.69)	(1.10)	(2.83)
40	32.01	5.81	0.66	1.52	30.49
	(2.90)	(2.51)	(0.89)	(1.44)	(3.82)
50	38.69	7.85	1.01	2.45	36.24
	(3.35)	(2.93)	(1.09)	(1.90)	(4.64)
60	44.92	10.18	1.47	3.43	41.49
	(3.65)	(3.36)	(1.36)	(2.35)	(5.26)
70	51.33	12.98	1.74	3.95	47.38
	(4.45)	(4.11)	(1.65)	(2.65)	(6.25)
80	57.30	15.60	2.20	4.90	52.40
	(4.87)	(4.66)	(1.82)	(2.83)	(6.72)
90	62.94	17.82	2.96	6.28	56.66
	(5.42)	(5.01)	(1.96)	(3.24)	(7.64)
100	68.55	20.57	3.45	7.43	61.12
	(5.74)	(5.62)	(2.25)	(3.81)	(8.47)

The table entries are averages (and standard deviations) observed for each of the cases discussed in Section 3 for a fixed number of iterations between reinversions.

using a commercial LP code (APEX-III, developed by Control Data Corporation). The average occurrences of cases 1 to 4 between reinversion after a fixed number of iterations are shown in Table 1. The resulting matrix Q is surprisingly small.

There are a number of characteristics that are important in algorithms for large scale systems. Some of the most important aspects to consider are: (a) storage requirements, (b) numerical stability, (c) speed, and (d) flexibility. In the absence of actual implementation, the following computational analysis will be somewhat superficial.

(a) *Storage Requirements.* Let A be $M \times N$, and let K denote the maximum size that Q_l^{-1} is allowed to assume (usually $K \leq 50$). Then, using previously developed notation, the basic core requirements involve storage for B^{-1}, A (both in super-sparse form), and N-dimensional array for c (usually packed), K^2 storage locations for Q_l^{-1}, one K-dimensional working array and three M-dimensional working arrays to perform steps 1 through 4 of the simplex method. Total core requirements beyond that needed for B^{-1}, A and c are therefore roughly $K^2 + K + 3M$ between every reinversion. The quantity K^2 compares favorable with the buildup of new nonzero elements in either the PFI or LU decomposition. The following table is extracted from the paper of Tomlin [15] in which he compares the two methods.

Table 2
Growth of nonzero elements

Iterations after latest reinversion	Problem A (822 rows)		Problem B (2978 rows)		Problem C (3496 rows)		Partitioned form of
	PFI	LU	PFI	LU	PFI	LU	Updated inverse*
10	4340	450	10807	972	3692	409	100
20	8572	868	21453	1792	22193	1975	400
30	12965	1118	33400	2971	40677	4357	900
40	17232	1874	45792	4443	59297	6113	1600
50	21582	2691	57903	5662	77935	8240	2500

* It should be noted that core allocation is fixed in advance, so that every number in this column is actually the constant 2500.

In this table the partitioned form of the updated inverse is superior to either the PFI or LU decomposition from a storage viewpoint. A similar pattern is shown in Table 3, which is a typical sequence after reinversion of the agricultural model used previously.

(b) *Numerical Stability.* As our representation of the updated inverse assumes a fixed partitioning involving B^{-1} and Q^{-1}, there are situations in which this factorization is unstable. For instance, if a series of basis matrices progres-

Table 3
Typical non-zero build-up[a]

Iterations after last reinversion	Nonzero elements of inverse using LU[b]	Size of Q-matrix	Nonzero elements of inverse using partitioning
0	9129	0	9129
10	15642	10	9229
20	22807	18	9453
30	29388	24	9705
40	36178	31	10090
50	44072	36	10335

[a] Agricultural production scheduling problem, see Table 1.
[b] Commercial LP code (APEX III) employing LU decomposition.

ses from an ill-conditioned to a well-conditioned state, the final updated factorization will not be well-conditioned. This is mainly because the elements of Q are determined by B^{-1}. It is, therefore, important that the representation of B^{-1} is stable. In that case the stability associated with the overall representation of the updated basis inverse can be monitored essentially via the condition number of Q. The size of the condition number of Q may be partially controlled via the rejection of unacceptable pivots produced by incoming vectors.

As we have manipulated the small but explicit matrix Q^{-1}, we have the option to improve it numerically via the iteration

$$Q_{i+1}^{-1} = Q_i^{-1}(2I - QQ_i^{-1}), \quad i = 0, 1, 2, \ldots,$$

where Q_0^{-1} is the latest version of Q^{-1}. This process converges rapidly whenever $\|I - QQ_0^{-1}\| \leq \epsilon < 1$, in which case $\|I - QQ_i^{-1}\| \leq \epsilon^{2^i}$. If $\epsilon \geq 1$, convergence is not guaranteed and this may be taken as a signal for refactorizing the current basis [4]. This iterative improvement of Q^{-1} removes the potentially negative effect of a fixed pivoting sequence induced by the order in which additions to the matrix Q were made.

The above scheme requires only additional storage for Q if the elements of Q_i^{-1} are immediately replaced by the newly calculated elements of Q_{i+1}^{-1}. The matrix Q, however, does not have to be kept throughout all iterations between reinversions. If the final size of Q^{-1} is fixed at K, thus reserving K^2 storage locations, we can pack Q and Q^{-1} into the same space until they reach the size $K/\sqrt{2}$.

As both the monitoring of the condition of Q and the iterative improvement of Q^{-1} could be considered expensive if done at every iteration, one may employ stable methods for updating a factorization of Q rather than using the methods for manipulating Q^{-1}. One such method uses the orthogonal factorization $\bar{Q}Q = R$, where R is upper triangular. In this case, monitoring the condition of Q is equivalent to monitoring the condition of R [5, 16].

(c) *Speed.* Comparing the LU decomposition and the partitioned form of the

updated inverse from a computational viewpoint is not a straightforward matter. At the time of this writing, no actual comparisons on real-world problems using efficient implementations of the two methods have been made. In the LU decomposition one forward sweep, one backward sweep, and one partial backward sweep through the updated inverse is required (see [5]). The computations involved are comparable to steps 1 and 4 as described previously where U_l, V_l, and Q_l^{-1} are accessed twice, and B^{-1} three times fully and one time partially. In the event that the incoming vector is a B vector, this reduces to two complete and one partial sweep through B^{-1}. Considering that the time spent in forward and backward transformations is proportional to the number of nonzero elements of the inverse, it looks like the partitioned form will take twice as long for FTRAN and BTRAN operations. This is true for the first iterations following reinversion. The PF or LU inverse grows virtually always faster than the Q-matrix and the breakeven point with the partitioned form is reached in some cases very soon. Table 3 shows that a single iteration of the partitioned form is already faster after approximately 14 iterations. Additional time could be saved by storing $B^{-1}U$ (available from step 1) during the first few iterations, thereby eliminating two accesses of B^{-1} per iteration.

(d) *Flexibility.* The partitioned form involves a mixture of sparse and full matrix technology. The method is flexible in that one is not locked into just column operations. It allows dynamic changes in computational strategies. Special structures (partitioning of the A matrix) are handled quite naturally and are straightforward to implement.

6. Summary and conclusions

In this paper we address the general problem of matrix modifications, and the specific problem of updating a basis inverse in linear programming. Our interest is in large sparse systems to be solved entirely in core. We develop a partitioned form of the updated inverse which we found to be more compact than either the PFI or the LU decomposition. This partitioned form involves an interesting mixture of sparse and full matrix technology. The method is flexible in that any of the four modifications discussed in Section 2 can be made in subsequent iterations. Relative to LU decomposition the method is straightforward to implement, and no excessive work needs to be done in the actual updating of the inverse. A thorough comparison on real-world problems using well implemented codes is needed to make any definite conclusions regarding computational efficiency. A superficial analysis has shown that the partitioned form of the updated inverse has a clear disadvantage regarding the number of computations required in backward and forward transformations. On the other hand, the procedure has the distinct advantage that the growth of additional nonzero elements is not related to the size of the problem, but only to the number of iterations following reinversion. It is shown that the representation of the updated inverse does not grow monotonically in size, and that it may actually contract during certain simplex iterations. This is determined by the pivoting

sequence as it occurs during the solution process. It is shown that special structures such as GUB or other angular forms may take advantage of this partitioned form of the updated inverse.

Realizing that the method has clear advantages over other methods in terms of compactness and implementation, we feel that the partitioned form of the updated inverse deserves definite attention when developing linear programming codes for large sparse problems.

References

[1] R.H. Bartels and G.H. Golub, "The simplex method of linear programming using LU decomposition", *Communications of the Association for Computing Machinery* 12 (1969) 266–268.

[2] J. Bisschop and A. Meeraus, "A recursive form of the inverse of general sparse matrices", DRC technical note #1, Development Research Center, World Bank (Washington, DC, 1976).

[3] G.B. Dantzig and R.M. Van Slyke, "Generalized upper bounding techniques for linear programming", *Journal of Computer and Systems Sciences* 1 (1967) 213–226.

[4] G. Forsythe and C. Moler, *Computer solution of linear algebraic systems* (Prentice-Hall, Englewood Cliffs, NJ, 1967).

[5] P.E. Gill, G.H. Golub, W. Murray and M.A. Saunders, "Methods for modifying matrix factorizations", *Mathematics of Computation* 28 (1974) 505–535.

[6] D. Goldfarb, "On the Bartels–Golub decomposition for linear programming bases", Report CSS, Atomic Energy Research Establishment (Harwell, Didcot, Oxfordshire, 1975).

[7] E. Hellerman and D. Rarick, "Reinversion with the preassigned pivot procedure", *Mathematical Programming* 2 (1971) 195–216.

[8] R.N. Kaul, "An extension of generalized upper bounding techniques for linear programming", ORC 65-27, Operations Research Center, University of California at Berkeley (1965).

[9] G. Kron, *Diakoptics* (MacDonald, London, 1956).

[10] L.S. Lasdon, *Optimization theory for large systems* (MacMillan Company, New York, 1970).

[11] J.K. Reid, "A sparsity-exploiting variant of the Bartels–Golub decomposition for linear programming bases", Report CSS, Atomic Energy Research Establishment (Harwell, Didcot, Oxfordshire, 1975).

[12] D.J. Rose and J.R. Bunch, "The role of partitioning in the numerical solution of sparse systems", in: D.J. Rose and R.A. Willoughby, eds., *Sparse matrices and their applications* (Plenum Press, New York, 1972).

[13] M.A. Saunders, "The complexity of LU updating in the simplex method" in: R.S. Anderssen and R.P. Brent, eds., *The complexity of computational problem solving* (University Press, Queensland, 1972).

[14] J. Sherman and W.J. Morrison, "Adjustment of an inverse matrix corresponding to changes in the elements of a given column or a given row of the original matrix", *Annals of Mathematical Statistics* 20 (1949) 621.

[15] J.A. Tomlin, "Modifying triangular factors of the basis in the simplex method", in: D.J. Rose and R.A. Willoughby, eds., *Sparse matrices and their applications* (Plenum Press, New York, 1972).

[16] J.A. Tomlin, "An accuracy test for updating triangular factors", *Mathematical Programming Study* 4 (1975) 142–145.