UPDATING LU FACTORS OF LP SIMPLEX BASES

G. SANDE *

Abstract. Methods for updating the LU factors of simplex basis matrices are reviewed. An alternative derivation of the Fletcher and Matthews method is given. This leads to generalizations of their method which avoids problems with both the Bartels and Golub method and the Fletcher and Matthews method. The improvements are to both numerical stability and data access locality. The resulting updating algorithm is preferred to the Reid variant of the Bartels and Golub method for both numerical stability and cost of execution.

Key words. LP basis updating, LU factor updating, Bartels and Golub method, Fletcher and Matthews method

AMS subject classifications. 65F05, 65F50, 90C05

Version timestamp: 10:25 – Friday 9th October, 2015

1. Introduction. Methods for updating the LU factors of LP simplex bases have been in successful use since the introduction of LU factoring for LP bases by Bartels and Golub [1] in the 1960s. Experience with the method shows that the factors have both growth in the number of nonzero elements in their sparse representations and growth in the size of the elements of the factors. The number of nonzero elements growth is much like that of the earlier method of product form inverses in use with LP bases so was to be expected. The growth in size of the elements is unwelcome by those who value numerical stability.

In the 1960s computers were few in number, usually shared and operated with budgets. The processors were slower than the memories. Execution time for even moderate sized problems could be of concern. Memory was a limiting resource, for other than small problems, which had to be carefully managed. The most important memory distinction was whether it was internal or external. Analysis of algorithms was a matter of careful counting of arithmetic operations. Benchmarking was an empirical confirmation of the analysis results and was not seriously influenced by programming details.

Current computers are common with only very large computers shared. Processors have become much faster and memories are both faster and larger. Processors are typically faster than memories. Execution time can become a concern for larger problems. Memory is rarely a limiting resource, except for large problems, although it is no longer homogeneous with caches often used to improve performance. Data movement can easily be more time consuming than the operations to be performed on the data. Locality of data access has become very important with some data having redundant storage, sometimes combined with redundant computation, to decrease access times. Benchmarking has become subject to many details so that a single analysis is rarely definitive.

In the intervening time the field of analysis of algorithms has developed with the notion of amortized costing where several types of operations alternate. Here we have the operations of calculating the factors of the basis, updating the factors of the basis and solving equations with the resulting factors of the basis. The question is whether some extra effort in one of the operations might lower the overall computational cost

*Sande and Associates, 10 Regency Park Drive #604, Halifax, Nova Scotia B3S 1P2, email:Gordon.Sande@EastLink.ca

of the combined usage. The Fletcher and Matthews [3] method is such a change where more work in the factor updating method is balanced by less growth in the size of the factors. Their method has its own operational problems but does suggest generalizations which avoid those problems.

2. Review. The Revised Simplex method for Linear Programming partitions the coefficient matrix into basic and nonbasic portions, $A = [\ B\ N\ ]$. The operations performed with the basis are solving the systems $B\ x = r$ and $B^T\ x = r$ as well as updating the basis by replacing a column by a new column. The basis matrix

$$B = [\ b_1\ b_2\ \ldots\ b_j\ \ldots\ b_n\ ]$$

is updated by replacing a column to become

$$\tilde{B} = [\ b_1\ b_2\ \ldots\ \tilde{b_j}\ \ldots\ b_n\ ].$$

The simplex basis update maintains a nonsingular basis although the basis may be as ill conditioned as the linear programming problem.

The basis update may be expressed algebraically as

$$\tilde{B} = B + (\ \tilde{b_j} - b_j\ )\ \cdot\ e_j^t$$

where e_j is the j^{th} unit basis vector. This is an application for the Sherman-Morrison formulae for updating a matrix inverse with a rank one modification. After a suitable number of updates the inverse would be calculated directly to limit the accumulation of numerical errors. This method was used in the early applications of linear programming.

Alternately we could write

$$\tilde{B} = B\ (\ I + B^{-1}\ (\ \tilde{b_j} - b_j\ )\ \cdot\ e_j^t\ )$$

and notice that a matrix of the form $I + x \cdot y^t$ has an inverse of the same form. Multiplying by such an update factor is less costly than updating the entire inverse matrix. Successive updates would lead to more factors of this form, which is called the product form of the inverse. After a suitable number of updates the inverse would be calculated directly either to limit the accumulation of numerical errors or to lower the cost of applying the several update factors.

Elble and Sahinidis [2] provide a recent review of LU updates for the simplex algorithm.

3. Bartels and Golub Method. Bartels and Golub (B&G) suggested that the basis matrix should be factored rather than inverted. The storage requirements are reduced and the operation count is lower as a sparse B leads to sparse factors but typically the inverse will not be sparse.

In practice we will have $B = PLUQ^{-1}$ where L is a unit lower triangular matrix, U is an upper triangular matrix and both P and Q are permutation matrices. With these factors it is easy to solve $B\ x = r$ and $B^T\ x = r$. The first step of the updating is to calculate $\tilde{u_k} = L^{-1}\ P^{-1}\ \tilde{b_j}$, a column of $\tilde{U}$ where $\tilde{B} = PL\tilde{U}Q^{-1}$, which converts the problem to one in which

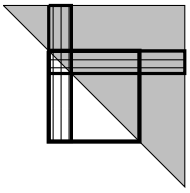
$$U = [\ u_1\ u_2\ \ldots\ u_k\ \ldots\ u_n\ ]$$

is updated by replacing a column to become

$$\tilde{U} = [\ u_1\ u_2\ \ldots\ \tilde{u_k}\ \ldots\ u_n\ ].$$

LP computations typically provide this computation so the first step is completed at no extra cost.

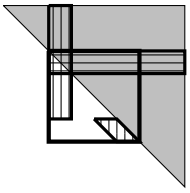
3.1. Usual variants. B&G provided a method for bringing $\tilde{U}$ to a form which enables easy solution of the required equations. They factor $\tilde{U}$ to obtain products of P and L terms with the Q terms multiplied together. Four variants of their method have been developed. The variants are examples of zero chasing using more and more elaborate permutations to avoid arithmetic operations.



Active block with column spike.

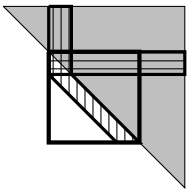
The new column of $\tilde{U}$ is called a column spike as it may have elements that are below the diagonal. If there are no elements below the diagonal then $\tilde{U}$ will be upper triangular and no further processing is required. The smallest diagonal block which includes the column spike would be called the active block. We have a column spike inserted into an upper triangular matrix. The active block is centred on the diagonal and encloses the column spike. The first row of the active block is indicated so it can be followed.

3.1.1. Stange et al. [7] variant. This variant is a direct application of simple Gaussian elimination with attention to reducing fillin of the active block. The combination of rows which produces the least fillin would be the last two rows which have few nonzero elements in the active block. By construction the column spike will be nonzero for the last row of the active block. The row above it can be used to eliminate this element. A row exchange may be required to avoid a zero pivot element or to enhance numerical stability.



Column spike partially eliminated.

A nonzero will be introduced in the subdiagonal of the last row and the column spike will now be nonzero in the second last row. This process can be repeated to reduce the column spike to a single element in the subdiagonal. All of the subdiagonal elements of the active block will be nonzero in an upper Hessenberg form.



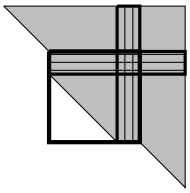
Column spike completely eliminated.

A sequence of eliminations, starting from the top row of the active block, will remove the subdiagonal of the upper Hessenberg form and provide more permutation and elimination matrix terms. Each step will generate both a permutation matrix, which might be an identity if there was no row interchange, and an elimination matrix, which might be an identity if there had been a row interchange required for a zero pivot.

These matrices will be part of a product form of the L factor, where L will connote left rather than lower and will be an operator rather than a matrix. If this operator were to be multiplied out it would not be a lower triangle matrix due to the presence of the permutation matrices and could not be readily back solved.

Although this form appears to be a simple direct application of zero chasing with elimination matrices it was reported after the other variants. The method shown here is implicit in the solution of a more general related problem that uses the absence of a column permutation.

3.1.2. Bartels and Golub variant. This original variant produces a nonzero subdiagonal of the Hessenberg form by permuting the columns rather than by use of repeated eliminations of the column spike of the previous variant.

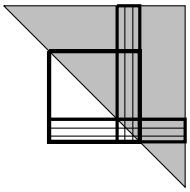


Column spike permuted.

The first column of the active block is moved to the last column and the remaining columns are moved one column closer to the beginning of the active block. This is a cyclic permutation applied to the column indices of the active block. The column permutation matrix can be combined with the existing column permutation matrix Q. The subdiagonal of the upper Hessenberg form can be eliminated in the same fashion as with the previous variant.

3.1.3. Forrest and Tomlin [4] variant. This variant uses both column and row permutations to produce a row spike version of the active block. In the B&G variant the column spike is permuted to be in the upper triangular portion of the active block in exchange for a nonzero subdiagonal.

A cyclic permutation of row indices moves the first row of the active block to the last row. The other rows are moved up one closer to the beginning of the active block. The subdiagonal will return to the diagonal except for the last row. This is a cyclic permutation of subscripts of the diagonal elements with each diagonal element carrying along its row and column.

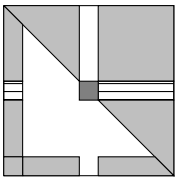


Row spike permuted.

The column permutation can be combined with the existing column permutation. The row permutation can be combined with the row permutation that may be required to eliminate the row spike. The row spike can be eliminated by combining with the successive rows of the active block using the diagonal element as the pivot in a standard Gaussian elimination. The diagonal will be nonzero so row exchanges will only be required for numerical stability. A common optimization of this method is to omit any row exchanges needed for numerical stability.

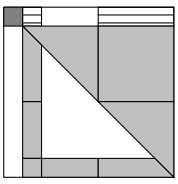
3.1.4. Reid [6] variant. This variant uses repeated diagonal element permutations to reduce the size of the active block.

The basic observation of this variant is that when the row spike form is constructed the row may not require the full active block. If there are leading zeroes in the row spike the active block can be made smaller by removing leading columns until the row spike has a leading nonzero.



Singleton column.

Once this has been done the active block can be returned to a column spike form to check if the new column spike requires the full active block. If not, the active block can have some rows removed until the bottom element of the column spike is nonzero. This can be repeated until both the column and row spike require the same sized active block.



Singleton column permuted.

As well, if any column of the active block has only one nonzero element, which would be on the diagonal, that element can be moved to the first row and column of the active block using a cyclic permutation of a portion of the diagonal elements. The first column, and row, can then be removed from the active block.

If any row of the active block has a single nonzero element, which would be on the diagonal, that element can be moved to the last row and column of the active

block using a cyclic permutation of a portion of the diagonal elements. The last row, and column, can then be removed from the active block.

The four types of operations can be repeated until the column and row spike forms of the active block both require the same sized active block and all rows and columns in the active block have two or more nonzero entries. The result may be that no eliminations are required as the active block has been effectively removed. If eliminations are required a row spike form would be convenient.

3.2. Other variants. We can use other methods to obtain two additional variants. These methods do not follow the zero chasing row oriented elimination paradigm of the usual B&G Methods. These variants are like the B&G Method but would not be considered a usual B&G variant. Zero chasing provides low computational cost and little fillin as each step causes only minor change in the sparsity structure. Row oriented methods are commonly used in numerical linear algebra algorithms. These two related variants were implemented as part of this study but have had only limited experimental use.

3.2.1. Irreducible Blocks variant. Without additional structure desirable initial row and column permutations would be ones that yield the irreducible blocks of the active block. The irreducible blocks found would not be in any special form. Finding the irreducible blocks is the first step in many factoring procedures. Factoring the irreducible blocks would yield further eliminations and permutations. The experimental code developed to use this strategy was found to be less costly than the Reid variant of B&G Methods. The explanation of this unexpected result is that the irreducible blocks of the active block are typically very small even when the active block becomes fairly large with a sparse column spike. The row and column permutations for the Irreducible Blocks variant are not the same as they would be for methods, like the Reid variant, that are limited to permuting the elements of the diagonal.

3.2.2. Column Pivoting variant. Later in this note we will observe that the active block can also be factored using column pivoting with no need for any row permutations. Column pivoting would be expected both to require more computation and to result in more fillin than the zero chasing row eliminations and related permutations used by the usual B&G variants. Although column pivoting systematically introduces zeros it does not preserve the known zeros of the column spike form. The absence of a row permutation means that the L terms can be combined so the form of the factoring is preserved. Zero chasing row eliminations and related permutations lead to both P and L terms with their fillin and eventual need for refactoring. The usual B&G Methods seek to lower the cost of the update without considering any effect on the other operations. Such a cost analysis will favour zero chasing row eliminations and related permutations. Analyzing the cost of column elimination is a more difficult discounted analysis of a method which would have other differing properties. The experimental code developed to use this strategy was found to be more costly than the Reid variant of B&G Methods.

3.3. Multiple permutations. If we follow the usual B&G Methods we would arrive at a form like

$$B = P_1 L_1 P_2 L_2 \dots P_M L_M U_M Q_M^{-1}$$

which would require representing many lower triangular matrices L and permutation matrices P. With an algebraic rearrangement the need for storing intermediate

permutations can be eliminated. Starting with

$$B = P_1 L_1 U_1 Q_1^{-1}$$

we form

$$B = (P_1 L_1 P_1^{-1}) P_1 U_1 Q_1^{-1}.$$

If U_1 is replaced by

$$\tilde{U_1} = P_2 L_2 U_2 Q_2^{-1}$$

we would have

$$B = (P_1 L_1 P_1^{-1}) P_1 P_2 L_2 U_2 Q_2^{-1} Q_1^{-1}$$

which is

$$B = (P_1 L_1 P_1^{-1})(P_1 P_2 L_2 P_2^{-1} P_1^{-1}) P_1 P_2 U_2 Q_2^{-1} Q_1^{-1}$$

or

$$B = (P_1 L_1 P_1^{-1})(\tilde{P_2} L_2 \tilde{P_2^{-1}}) \tilde{P_2} U_2 \tilde{Q_2^{-1}}$$

if products of permutations are combined. The general formulae is both obvious and long. A form of PLP^{-1} is a matrix in the original indices although not lower triangular. When $L = I + y \cdot x^t$, which includes the elimination matrices constructed above, we have $PLP^{-1} = I + (Py) \cdot (Px)^t$ where the permutation can be combined with the vector indices. All of the products of intermediate permutation matrices can be combined with vectors and need not be stored. Any initial L matrix can be represented as a product matrices of the form $I + y \cdot x^t$ as can be demonstrated by applying Gaussian elimination to factor L. The composite permutation matrices P and Q are only needed for U as all of the L terms are represented in the original subscripts.

3.4. Balancing factoring, updating and solving. A practical problem is how many updates should be applied before new factors are calculated. Each update causes the backsolving to take slightly more time as there are more terms in L and U may have some degree of fillin. If only a few updates are applied before new factors are calculated the factoring will be the more time consuming part of the computation. If many updates are applied before new factors are calculated the backsolving will be the more time consuming part of the computation. A practical policy is to calculate new factors when the accumulated fillin becomes excessive. A doubling of size was found to provide a good compromise for examples of interest. This usually took about 30 updates with some examples of doubling in size in under 10 updates and other examples taking as many as 100 updates.

As part of the study of timings it was observed that not all backsolve right-hand-sides took the same time. Some right-hand-sides were very sparse, as would be true to find a column of the inverse of the basis B^{-1}, while others were merely sparse, as might be true to find a column of the simplex tableau $B^{-1}N$, and others were dense, as might be $B^{-1}(b - Nx_N)$. The very sparse backsolve problems could be done with priority queue based methods which have too much overhead to be used effectively with the sparse or dense problems. The sparse problems could be done

with the common gather-scatter methods for sparse matrices which have too much overhead to be used effectively with either the very sparse or dense problems. The dense methods have too much overhead to be used effectively with either the very sparse or sparse problems. Many sparse matrix packages provide sparse and dense solvers without additional methods for very sparse problems.

For typical LPs, the sparsity of B can be attributed to two sources. When suitably permuted, B is highly structured as a block matrix of zero blocks and nonzero blocks that form a block triangular matrix. The nonzero blocks are themselves sparse. Sparse matrix storage methods do not require explicit representation of the block structure as it is used implicitly. Many factoring procedures will determine the irreducible blocks to make explicit use of the block structure to help in lowering the fillin of the factors. The inverse of B will be highly structured as a block triangular matrix but will not be sparse as the nonzero blocks will not generally be sparse. When solving equations the structure of the inverse will cause some right-hand-sides to have more fillin than others.

To deal with both B and B^T it is convenient to have both row and column oriented representations. The extra effort required to maintain a redundant representation is more than repaid by the savings obtained by the use of the suitable representation for each task. Redundant representations cost less than the scanning of a single representation that would be needed without the redundant representations. Work vectors can be represented in both gathered and scattered forms, with the requirement for scanning reduced by the use of the phase markers method, for overall savings in effort. Having a column representation of the coefficient matrix A of the LP problem is convenient for use with the LU decomposition of the basis while a row representation is more convenient for the various update operations required in the simplex method. A sparse row representation of A will make implicit use of the block structure of A. The LP updates corresponding to $B^{-1}N$ will typically be structured from the structures of A and the inverse of B. The redundant representation allows the sparse matrix techniques to avoid computations with and of many zeroes. Sparse methods can even be used to track the infeasible components of the solution, which become a smaller fraction of the solution as optimality is approached. Much of the early sparse techniques literature was done when memory was a limiting resource so the advantages of redundant representations, which typically double the storage, were not explored.

3.5. Discussion of B&G Method. It was also observed that some examples had substantial growth in the size of elements of U. When this became excessive new factors would be calculated even before indicated to lower fillin. The individual L factors had no growth in the size of their elements as would be expected from a partial pivoting policy. It was assumed that if L were to be multiplied out that it would show corresponding growth in the size of its elements. When memory sizes became larger it was possible to accumulate the product the L factors, and their inverses, and it was then observed that some examples had growth in the size of the elements of the product even with no growth in the size of elements of U. Whenever refactoring happened the size of the elements in both L and U returned to their nominal values.

The experience with the B&G Methods is that they have been effective for many years. The multiple terms of the L factors is part of the design and much like the product form of the inverse algorithms which had been in use before B&G. The growth of the size of the elements of U is an undesirable aspect of these methods that is usually

treated as one part of dealing with numerical accuracy issues in simplex algorithms. The growth of the size of elements in the expanded L is also undesirable even though not directly observed except in the occasions when L is explicitly expanded.

The growth of the size of the elements of L has a simple heuristic plausibility explanation. If we multiply two partial pivoting elimination matrices the product may no longer be a partial pivoting elimination matrix. For

$$\begin{bmatrix} 1 & \\ x & 1 \end{bmatrix} \begin{bmatrix} 1 & \\ y & 1 \end{bmatrix} = \begin{bmatrix} 1 & \\ x+y & 1 \end{bmatrix}$$

the absolute values of x and y are bounded by one for partial pivoting but the absolute value of x+y is not bounded by one and may have more growth than expected under partial pivoting. Larger growth would require repeated reinforcement which would only happen in special circumstances. We would not expect much repetition in sparse matrices. Even with repetition we would expect both reinforcement and cancellation in the accumulated size. The presence of any permutations P will both loose the lower triangular nature of L and make any self correcting capability of partial pivoting less effective.

The numerical problems are most evident when the basis is refactored. What had appeared to be a feasible solution before the refactoring may become an infeasible solution afterward. Recovering from a loss of feasibility can be dealt with by a temporary return to phase one of the usual two phase simplex algorithm. An optimal solution may no longer be optimal after refactoring and require either feasibility recovery or more iterations. In extreme cases the refactoring may report a singular basis which can be addressed by returning to previous bases to find an earlier nonsingular basis. Truncating small values to aid sparsity makes the numerical problems more severe.

4. Fletcher and Matthews Method. Fletcher and Matthews (F&M) sought a form preserving method for the basis update. This would avoid the many L factors with the need for periodic refactoring. Their method starts after U has been brought to upper Hessenberg form by a column permutation. The first step is the elimination of the first subdiagonal in the upper Hessenberg form which leaves us with a new problem of the same form but with one less subdiagonal element. The process is repeated until there are no subdiagonal elements.

4.1. Separate cases. They provide a method which has two distinct cases depending upon whether a row interchange is needed. The following development is equivalent to that of F&M but with differing motivation and notation.

4.1.1. No row interchange case. The objective of the numerical processing to produce $L\tilde{U} = \bar{L}\bar{U}$ where $\bar{L}$ is unit lower triangular and $\bar{U}$ is upper triangular. We will abuse the notation by dropping the diacritical markings and using an order four matrix as illustrative of the general case.

The first stage would be to eliminate the first element of the subdiagonal in U using Gaussian elimination. We would form $LU = LM^{-1}MU$ where

$$M = \begin{bmatrix} 1 & & & \\ & 1 & 0 & \\ & -m & 1 & \\ & & & 1 \end{bmatrix} \text{ and } M^{-1} = \begin{bmatrix} 1 & & & \\ & 1 & 0 & \\ & m & 1 & \\ & & & 1 \end{bmatrix}$$

are the standard form and $m = u_{k+1,k}/u_{k,k}$. LM^{-1} will be modified in column k and MU will be modified in row $k+1$. m may not be defined or we may prefer to not use this elimination because of a loss of numerical stability. We then would require an alternate method to eliminate the first element of the subdiagonal of U.

4.1.2. Row interchange case. In the standard development of partial pivoting we would interchange rows to avoid lack of definition or loss of numerical stability. Here we interchange the two rows of U. The row interchange matrix will be

$$M_1 = \begin{bmatrix} 1 & & & \\ & 0 & 1 & \\ & 1 & 0 & \\ & & & 1 \end{bmatrix}$$

which is its own inverse. We would form LM_1^{-1} and M_1U. But LM_1^{-1} is not in a convenient form as

$$LM_1^{-1} = \begin{bmatrix} 1 & & & \\ \cdots & 0 & 1 & \\ \cdots & 1 & l_{k+1,k} & \\ \cdots & \cdots & \cdots & 1 \end{bmatrix}$$

with columns k and $k+1$ exchanged but we can restore the diagonal by forming $M_1LM_1^{-1}$ to have

$$M_1LM_1^{-1} = \begin{bmatrix} 1 & & & \\ \cdots & 1 & l_{k+1,k} & \\ \cdots & 0 & 1 & \\ \cdots & \cdots & \cdots & 1 \end{bmatrix}$$

which has both rows k and $k+1$ as well as columns k and $k+1$ exchanged. It is lower Hessenberg as well as having a zero in its subdiagonal. The row permutation can be used to update the permutation P. We have applied three permutations which interchange rows of U, columns of L and rows of L to have a product of a lower Hessenberg matrix with an upper Hessenberg matrix.

If the element that is permuted to be above the diagonal is zero then the lower Hessenberg form is lower triangular. The row interchange can be applied to L without complication other than restoring the diagonal and the elimination in U is like the no row interchange case. The formal development is the same although an implementation would benefit from this special case.

We may reduce the lower Hessenberg form to lower triangle with a transposed Gaussian elimination. We would form$M_1LM_1^{-1}M_2^{-1}$ and M_2M_1U where

$$M_2 = \begin{bmatrix} 1 & & & \\ & 1 & -m & \\ & & 1 & \\ & & & 1 \end{bmatrix} \text{ and } M_2^{-1} = \begin{bmatrix} 1 & & & \\ & 1 & m & \\ & & 1 & \\ & & & 1 \end{bmatrix}$$

for $m = l_{k+1,k}$ to eliminate the superdiagonal element of L. The result will be a modification of column $k+1$ of L and row k of U which is the pivot row after the row interchange.

We now have the product of a lower triangle matrix and an upper Hessenberg matrix which was the intended result of the row interchange in U. Gaussian elimination can now be applied. So we form $M_1 L M_1^{-1} M_2^{-1} M_3^{-1}$ and $M_3 M_2 M_1 U$ where

$$M_3 = \begin{bmatrix} 1 & & & \\ & 1 & 0 & \\ & -m & 1 & \\ & & & 1 \end{bmatrix} \text{ and } M_3^{-1} = \begin{bmatrix} 1 & & & \\ & 1 & 0 & \\ & m & 1 & \\ & & & 1 \end{bmatrix}$$

for $m = u_{k,k}/(u_{k+1,k} + l_{k+1,k} \cdot u_{k,k})$. This may be undefined or numerically unstable so that the elimination without the row interchange might be preferred. F&M suggest a modified test for which of the two forms of the elimination to use after examining several possibilities. It is the usual partial pivoting rule of using the larger of the two pivots, which has the complication that the alternate pivot needs some computation to be determined. They observe that it is not possible for both elimination forms to be undefined so that one of the two forms is always possible. This derivation shows that their suggested test is the usual test used for partial pivoting row interchanges.

If the alternate elimination form is declined we might interchange the now modified rows of U again. The subdiagonal zero in L means that L will remain lower triangular. The final elimination will fill in that zero and achieve the original elimination in two stages. This variant provides logical completeness and an analysis which may be of use elsewhere. There is an obvious optimization of combining the two eliminations which reinforces the utility of being able to judge which of the two cases should be used.

4.2. Alternative description. We can use this analysis to provide a different description of the F&M Method. The first step would be to introduce the subdiagonal zero in L using a column elimination, if it is not already zero, with its corresponding modification of the rows of U. The decision of whether a row interchange is appropriate can be made and the corresponding permutation applied. The form of L will not be changed whether that permutation is an identity, for no row interchange, or not, for a row interchange. The elimination of the subdiagonal of U can be completed with the corresponding modifications of the columns of L The possible requirement for a row interchange has become a question limited to the second step under this different description. A careful implementation would notice that the two eliminations under no row interchange can be combined so that the original form of the F&M Method would be recovered. This analysis shows that the F&M Method is just an alternate implementation of an elementary Gaussian elimination applied to adjacent rows. Stange et al. use the F&M Method with their version of their update as the only row permutation is interchange of adjacent rows.

4.3. Discussion of F&M Method. F&M provide a direct derivation of the matrix identified above as $M_3 M_2 M_1$ by starting with the assumption of a row interchange in U. The $M_3 M_2 M_1$ matrix is then recognized as being a product and two factors are exhibited. The permutations are not factored out to provide standard form elimination matrices. The present derivation starts with a row interchange used to facilitate Gaussian elimination and provides additional terms to preserve the structure of the matrices. The additional terms are constructed in much the same style as is used in the zero chasing algorithms of many numerical decomposition algorithms. One of the additional terms is the row interchange in L. In this derivation all of the operations are seen to be standard operations which are commonly used in the numerical manipulation of sparse matrices. The elimination matrices are in a form in which

trivial cases can be easily skipped. The numerical properties of the two derivations are identical as the same terms are calculated. F&M provides an extensive analysis of the numerical properties. Powell [5] has an additional discussion of the numerical properties. The data motions will be different for the two formulations. The explicit factoring of the permutations provides additional implementation flexibility.

Various discussions of the F&M Method suggest that it is an algorithm more suitable for dense problems. It is not clear if this was based on fillin experience or was avoiding the nonstandard manipulations apparently required of sparse matrices.

When the F&M Method was used the numerical properties were as well behaved as expected. Usually the size of the factors would either grow or shrink slightly for little change in size with no ongoing growth in size. Upon occasion an update would take considerable time and the sizes of the updated factors would have considerably increased. The very bad examples of fillin in some papers were not just possible constructs but were being observed. The increase in size was so large that a refactoring was indicated rather than another update. The practical solution to the long time and increased size was to notice when such fillin became apparent before the update was completed so that the update could be terminated before completion and the matrix factored instead. The new factors did not show the increase in size. This form of failure was attributed to the updating algorithm not having adequate flexibility in choice of possible permutations to avoid the fillin although the factoring algorithm did have adequate flexibility of choice. The final judgement was that the F&M Method was an improvement in numerical properties but was not adequate in avoiding excess computational time due to its poorer fillin behaviour.

5. Generalizations of F&M Method. A solution for the inadequacies of the F&M Method would be to seek a form preserving method like the F&M Method but one with more flexibility in its choice of permutations. An immediate possibility would be the F&M Method but with elimination applied to blocks larger than just two by two.

5.1. Block matrices. We use block matrices to provide a general development. We would identify active blocks on the diagonals of U and L which will be labelled U_{22} and L_{22}. The initial permutation to upper Hessenberg form is not required here. The active block U_{22} will not be upper triangular but the active block L_{22} will be lower triangular. We have

$$B = \begin{bmatrix} L_{11} & & \\ L_{21} & L_{22} & \\ L_{31} & L_{32} & L_{33} \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} & U_{13} \\ & U_{22} & U_{23} \\ & & U_{33} \end{bmatrix}$$

.

5.1.1. Remultiply and Factor variant. We remultiply U_{22} and L_{22} to form a block which would be full in general. In matrix terms we would form $B = LM^{-1}MU$ where

$$M = \begin{bmatrix} 1 & & \\ & L_{22} & \\ & & 1 \end{bmatrix}.$$

The result is

$$B = \begin{bmatrix} L_{11} & & \\ L_{21} & I & \\ L_{31} & L_{32}L_{22}^{-1} & L_{33} \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} & U_{13} \\ & L_{22}U_{22} & L_{22}U_{23} \\ & & U_{33} \end{bmatrix}.$$

The block $L_{22}U_{22}$ can be factored and the factors multiplied into the block matrices. If $L_{22}U_{22} = P\bar{L}\bar{U}Q^{-1}$, then

$$B = \begin{bmatrix} L_{11} & & \\ L_{21} & P\bar{L} & \\ L_{31} & L_{32}L_{22}^{-1}P\bar{L} & L_{33} \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} & U_{13} \\ & \bar{U}Q^{-1} & \bar{L}^{-1}P^{-1}L_{22}U_{23} \\ & & U_{33} \end{bmatrix}.$$

We can separate out the permutation factors to obtain

$$B = \mathcal{P} \begin{bmatrix} L_{11} & & \\ P^{-1}L_{21} & \bar{L} & \\ L_{31} & L_{32}L_{22}^{-1}P\bar{L} & L_{33} \end{bmatrix} \begin{bmatrix} U_{11} & U_{12}Q & U_{13} \\ & \bar{U} & \bar{L}^{-1}P^{-1}L_{22}U_{23} \\ & & U_{33} \end{bmatrix} \mathcal{Q}^{-1}$$

where

$$\mathcal{P} = \begin{bmatrix} I & & \\ & P & \\ & & I \end{bmatrix} \text{ and } \mathcal{Q}^{-1} = \begin{bmatrix} I & & \\ & Q^{-1} & \\ & & I \end{bmatrix}.$$

Some of the internal blocks will have their rows or columns permuted and the overall row and column permutations will be modified. The main computation will be remultiplying and then factoring $L_{22}U_{22}$, forming $L_{32}L_{22}^{-1}P\bar{L}$ and forming $\bar{L}^{-1}P^{-1}L_{22}U_{23}$. These seem unlikely terms until we recall that they are just intermediates that would be formed in the course of a Gaussian elimination with the newly formed permutations. The presence of the L_{22}^{-1} and $\bar{L}^{-1}$ terms are uncomfortable to those who try to avoid multiplying by matrix inverses but a close examination of a block Gaussian elimination shows that any element size growth attributable to L_{22}^{-1} or $\bar{L}^{-1}$ would also be present in the block Gaussian elimination.

5.1.2. Column Pivoting variant. We might factor U_{22} directly to try to find a simpler method. If $U_{22} = P\bar{L}\bar{U}Q^{-1}$, then

$$B = \begin{bmatrix} L_{11} & & \\ L_{21} & L_{22}P\bar{L} & \\ L_{31} & L_{32}P\bar{L} & L_{33} \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} & U_{13} \\ & \bar{U}Q^{-1} & \bar{L}^{-1}P^{-1}U_{23} \\ & & U_{33} \end{bmatrix}.$$

We can separate out the permutation factors to obtain

$$B = \mathcal{P} \begin{bmatrix} L_{11} & & \\ P^{-1}L_{21} & P^{-1}L_{22}P\bar{L} & \\ L_{31} & L_{32}P\bar{L} & L_{33} \end{bmatrix} \begin{bmatrix} U_{11} & U_{12}Q & U_{13} \\ & \bar{U} & \bar{L}^{-1}P^{-1}U_{23} \\ & & U_{33} \end{bmatrix} \mathcal{Q}^{-1}$$

where

$$\mathcal{P} = \begin{bmatrix} I & & \\ & P & \\ & & I \end{bmatrix} \text{ and } \mathcal{Q}^{-1} = \begin{bmatrix} I & & \\ & Q^{-1} & \\ & & I \end{bmatrix}$$

which has the problem that $P^{-1}L_{22}P$ may no longer be lower triangular. The previous development had preliminary processing that turned L_{22} into an identity matrix which allowed P to be arbitrary. Here the solution is to restrict P to be an identity permutation so L_{22} can be arbitrary with $U_{22} = \bar{L}\bar{U}Q$. This simpler and restricted

factoring is recognized as column pivoting. It has limited flexibility that might be exploited for some sparsity control. The simpler method becomes

$$B = \begin{bmatrix} L_{11} & & \\ L_{21} & L_{22}\bar{L} & \\ L_{31} & L_{32}\bar{L} & L_{33} \end{bmatrix} \begin{bmatrix} U_{11} & U_{12}Q & U_{13} \\ & \bar{U} & \bar{L}^{-1}U_{23} \\ & & U_{33} \end{bmatrix} \begin{bmatrix} I & & \\ & Q^{-1} & \\ & & I \end{bmatrix}.$$

The lower triangular factors multiply to provide an updated lower triangular factor. The upper triangular factor has a column permutation which is part of its factorization. This would have provided a form preserving variant on the B&G method but did not follow the zero chasing paradigm so does not seem to have been previously reported.

5.2. Discussion of the block matrix forms. Both block matrix generalizations of the F&M Method were implemented for experimental use and run with quite sparse examples. The unexpected result was that the Remultiply and Factor variant was less costly than the Column Pivoting variant. The Remultiply and Factor variant used a sparse matrix factorization with a first step of finding the irreducible blocks. The cost of finding the irreducible blocks more than saved its cost as it produced many blocks of size one and provided better sparsity control. This seems to occur in many problems but may not occur in all as it is undoubtedly problem dependent.

If the active block is a small portion of the full matrix its remultiplication and factoring will take less work than factoring the full matrix. When the active block is most of the full matrix the remultiplication and factoring will take more work than factoring the full matrix. An elementary analysis, confirmed by simple experiments, would suggest that direct factoring is less work when the active block has more than half of the elements of the matrix, or equivalently more than about 70 per cent of the columns.

These refactorings would happen occasionally and serve as a pragmatic solution to how often a refactoring should be forced to control possible accumulated rounding errors.

A very abstract and telegraphic description of the Remultiply and Factor variant can be given as starting from the update forming an improper block matrix factorization

$$B = \begin{bmatrix} L_{11} & & \\ L_{21} & L_{22} & \\ L_{31} & L_{32} & L_{33} \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} & U_{13} \\ & U & U_{23} \\ & & U_{33} \end{bmatrix}$$

in which U is no longer upper triangular. This is a factoring but it is improper as it is redundant and U is not in a form which permits ready backsolving. We try to make the problem with U more tractable by forming a new block matrix factorization

$$B = \begin{bmatrix} L_{11} & & \\ L_{21} & I & \\ L_{31} & \bar{L_{32}} & L_{33} \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} & U_{13} \\ & M & \bar{U_{23}} \\ & & U_{33} \end{bmatrix}.$$

This could have been formed by a block Gaussian elimination of a form that we are unlikely to see in practice as it has bypassed a diagonal block. This might be described as partially undoing the existing improper factorization structure so that it can be redone as a proper elimination. We now complete the reduction of the reconstructed M to triangular factors and update any blocks as required. This is useful in practice as M typically has only a few small irreducible blocks.

The numerical properties of the Remultiply and Factor variant matched those of the F&M Method as no spurious growth in size of the elements of L or U was observed in use. The size of the factors after updating was observed to both increase and decrease while remaining stable. The size of the factors after refactoring was always close to the size before refactoring with both increase and decrease being observed. The final judgement would be that the generalization was successful both in terms of numerical properties and fillin.

There is a simple heuristic plausibility explanation for the lack of growth in the size of elements of L. The newly factored active block will have no element size growth in $\bar{L}$ with partial pivoting in the factoring. The other block which is modified is post multiplied first by an inverse matrix and then by a permuted matrix that is related to the inverse matrix. This suggests that there will usually be no material change in size of its elements. The active block is only part of the L matrix but over successive updates all of the elements of L will be either unchanged elements of an earlier active block or will have been modified, perhaps several times, by the inverse matrix times related matrix combination. This follows from the observation that L could be produced produced by a Gaussian elimination although not with the usual computational formulae.

The basic processor costs of the Remultiply and Factor variant can be easily estimated. The factorings will have the same cost as for a B&G Method although there will be fewer factorings. The solving will be less costly than for a B&G Method as the LU factors are smaller since there is less fillin. The higher numerical accuracy has a tendency to require slightly fewer simplex iterations, or pivotal exchange steps. There will be the same number of updates as for a B&G Method, after correcting for the change in the number of factorings and simplex iterations. The updates will operate on smaller factors but with more elaborate processing.

Using the Reid B&G variant as a reference we find, as already noted, that the Irreducible Blocks variants is less costly and the Column Pivoting variant is more costly than the Reid B&G variant. The Remultiply and Factor variant was found to be less costly that the Reid B&G variant but more costly than the Irreducible Blocks variant. The costing order of these four variants is from the least costly being the Irreducible Blocks variant, then the Remultiply and Factor variant, then the Reid B&G variant with the most costly being the Column Pivoting variant. The better than expected performance of the Remultiply and Factor variant resulted in its being promoted from experimental to regular use. The major effect is the reduction in cost from finding the irreducible blocks with the increase in cost from the form preservation being the secondary effect. This is at most a rough approximation as there are many other differences between the variants. The improved data localization means that the memory system can operate more efficiently on systems with hierarchical memories.

The presence of numerical issues is very evident in linear programming test suites. Some test problems are noted as being ill conditioned. Others are noted as sensitive to tolerance parameters. Yet others have the reported optimum value updated multiple times. In solutions where one expects to see integer values the reported values are only nearly integer. Some formulation techniques are prone to numerical issues. Common questions from beginners are what value for M (the so called Big-M) will be effective both in achieving the intended result and in avoiding numerical problems as well as the whole issue of how to deal with tolerances. Although linear programming is stated as an optimization problem many applications are more interested in determining the solution vector than in determining the optimum value. Some applications are

concerned with the values of the smaller components of the solution vector although error estimates are commonly only given relative to the larger components.

The presence of numerical errors should be of no surprise as the observed growth in the size of the elements in the U factor of a B&G Method greatly exceeds what would be judged tolerable by the standards of numerical analysis. As noted above, the size of elements in the L factor of a B&G Method also grows but is rarely observed. Actual errors in solving depend on both the matrix factors and the right-hand-side. Summaries like the growth in the size of elements of the factors are part of worst case estimates over all possible right-hand-sides. The extensive use of updating formulae in linear programming codes is a source of numerical error that is made worse by numerical errors in determining the updates. The numerical errors in determining the updates are those arising from the numerical errors in solving which are made worse by the numerical problems in factoring the basis. Many LP codes pay considerable attention to guarding against numerical errors as if the problem was with the updating rather than in both the updating and the solving. For problems where time is not a limiting factor the increased numerical stability and ease of formulation may justify the increase in computing cost to lower overall user and programming costs.

5.3. Related Problems. There are related problems that can be solved with these techniques. Sometimes the problem is to update the basis factorization after replacement of a row in the basis. If all updates are row updates then the problem is just the transpose of the column update problem and requires only a minor change of notation. If there is a rare row update with mostly column updates then we might view the row update as several column updates where the several columns have only been changed in the row being updated. We can no longer assume that the result of an update will be nonsingular. The complications of allowing for updates that lower the rank followed by other updates that restore the rank can be analysed but it would be considerable work for a rare case that may trigger refactoring in any case. It is probably easier to treat the rare row update as another opportunity to refactor.

If there is a mix of row and column updates we would seek a more symmetric form of the method. Rather than a LU factoring we could use a LDU factoring with unit triangular factors and a diagonal matrix. For a column update the initial processing would use LD to produce a modified U much as has been done here. For a row update the initial processing would use DU to produce a modified L. The two update procedures would be conceptual transposes although the requirement that they work on the same data would make separate programs more obvious.

The replacement of a basis column, which yields the column spike, is a restricted form of rank one update. The methods that use zero chasing, the usual B&G methods or the F&M Method, rely on this restriction in their processing except the Stange et al. variant uses the absence of column permutations to permit general rank one updates. The methods that do not use zero chasing, the Irreducible Blocks, column pivoting or remultiply and factor methods, have no restriction on the rank of the update other than that implied by the size of the active block.

5.4. Further Work. The original suggestion was to develop a generalization of the F&M Method which worked on several rows rather than the two rows of the F&M Method. Following this suggestion we could use a column permutation to form a Hessenberg form as used in the F&M Method. Then we could apply the methods developed above to several rows.

We could start by developing alternate descriptions of the methods we already know. We might choose to operate only on the first two rows of the Hessenberg form

in the same way as the F&M Method. Notice that the two by two matrix we seek to factor could be of rank either one or two and that we should not use a column exchange. These properties are implicit in the description of the F&M Method but need to be explicitly observed in a matrix formulation. We might choose to operate on the entire active block using the block matrix generalizations developed above but with an extra column permutation. Notice that it has no effect other than to the form of some formulae as other terms compensate for the presence of the permutation.

We could then extend the development to operate only on an initial portion of the active block. In such a partial active block the sparsity pattern could result in the first row or the last column being all zero. Either of these would be a structural rank deficiency or there might be a numerical rank deficiency without the presence of a structural rank deficiency. The presence of the nonzero subdiagonal limits the rank deficiency to being either one or none. The rank deficiency is only in the upper triangular factor with the corresponding block of the lower triangular factor of full rank. We might choose to use the remultiply and factor method. When the partial active blocks are remultiplied the rank deficiency will remain. When we factor the remultiplied partial active block we need to restrict any column permutation to leave the last column in that position. Both of these problems were observed in the F&M Method when we used a matrix description. Rather than an irreducible block representation we would need something like a Dulmage-Mendelsohn decomposition to deal with both the possible low rank and the restricted column permutation. A succeeding partial active block will overlap the current partial active block so there will be serial dependence which prevents parallel operation. Or we might choose to use column pivoting as the remultiply and factor method seems to have fewer advantages for smaller partial active blocks. Column pivoting has the attractive feature that rows can be added at any time to form a sliding partial active block. A partial active block can be interpreted as a restriction on the choice of the pivot column. A rank deficiency may require a row permutation so column pivoting may not always be possible.

When we use a partial active block there will be several new questions. There is the question of what size of partial active block we should choose both for computational efficiency and adequate flexibility in choice of permutation to avoid the sparsity control problems of the F&M Method. Should we use only one of the two methods available or should we have some way to choose between them for each partial active block. The issues in the choice between remultiply and factor or column pivoting are more complex when we use partial active blocks. If we use multiple partial active blocks to avoid the need for refactoring for a large active block we will have the issue of how many updates can be applied before refactoring is suggested.

All of these methods use chained matrix products that may permit a more efficient order of forming the products. There are instances of the product of a matrix and the permuted inverse of a minor modification of the same matrix that may benefit from special methods. There is a need for both more and wider experience with these methods.

6. Conclusion. The original LP basis update methods for explicit inverses were based on the rank one modification of an inverse as given by the Sherman-Morrison formula. This became an update in the form of multiplicative factors. The B&G updates for the LU factored LP basis were a natural development of these methods. The irreducible blocks variant uses a stronger analysis than is possible within the constraints of zero chasing methods.

The F&M Method for LU updating has not been widely used. Experience shows that it may generate considerable fillin for some problems. The alternate derivation given here has been in terms of operations that are common for sparse matrices with the trivial cases readily recognized so they may be omitted. The derivation is in the style of zero chasing algorithms in common use. This revised derivation makes the pivoting test suggested by F&M appear to be the simple and natural test for pivoting. The revised derivation suggests generalizations which are better able to control fillin.

The difficulty encountered in trying to develop more direct update methods was that a mixed product of lower triangular and permutation matrices was not lower triangular. The methods developed here use one of two techniques to address this issue.

One method is that the modified upper triangular factor may be factored into a lower triangular and upper triangular factor with column pivoting which uses only column permutations. This permits the lower triangular factors to be multiplied without lose of their special structure. The switch from row, rook or complete pivoting to column pivoting would be of little significance in the dense case but would be expected to often have considerable impact on fillin in the sparse case. Column pivoting was not found to be cost competitive with methods that could use pivoting based on irreducible blocks.

Many iterative numerical methods are based on improving the current approximate solution. The factoring and then combining the new factors with the existing factors follows this style. The difficulty is that sometimes the new factors do not combine well with the existing factors as we have seen in the variants on the B&G Method. The result shown here is that there is an alternative that does combine cleanly although it requires further analysis to see if it also provides benefits in fillin and numerical stability. The result was that other methods are more cost effective.

The other method uses analysis developed here that shows that an identity block in the lower triangular factor which permits permutations of rows or columns without losing the lower triangular property is very useful. Constructing such an identity block requires modification of other blocks as a preliminary step before the main updating processing can be done. The updating can then use all the techniques for sparse matrix factoring rather than conforming to the requirements imposed by zero chasing. The general form is very flexible and can be applied to related problems. No spurious growth in either size of the factors or of the elements in the factors was observed.

This method has a style which is more like the backtracking typical of combinatorial optimization algorithms. Rather than improving the current solution the method seeks to improve an earlier solution but by using different improvements. Since this is not being done in an explicit backtracking environment the problem is how to look for and find the earlier solution. The answer is to find an identity matrix multiplying a full matrix as the earlier solution. Once the earlier solution has been found all of the usual sparse matrix tools can be used to full advantage.

There have been eight variants on updating LU factors described here. Five have been in the literature and three are new. The Irreducible Blocks variant is of interest as it follows the style of the B&G Method but with stronger methods. It could be viewed as what the Reid B&G variant would be with a stronger analysis of the active block. However it has the same type of problems. The Column Pivoting variant is of interest as it both follows the style of the B&G Method and is form preserving. It achieves both ends in a very simple fashion using well known methods. Like the

F&M Method it may be of more interest for dense applications. The Remultiply and Factor variant is of interest as it is form preserving, avoids the problems of the usual B&G variants as well as those of the F&M Method and is less costly than the Reid B&G variant. It is technically more elaborate than the other variants.

When we have several possible methods it is often convenient to classify them on both their design attributes and their operational properties. B&G has the attribute of zero chasing as their algorithm design technique. F&M has the attribute of form preserving to deal with the product form of the L factor. In examining these methods the attribute of the algorithm depending on only U or on both L and U is useful. Dependence on both L and U requires the access to L that comes from form preserving. The properties of interest are the ability to maintain sparsity and to maintain numerical stability as indicated by not having growth in the number of or the size of the elements of L or U. With three possible design attributes we would expect eight algorithm forms. In practice there are only five possible forms. Dependence on both L and U requires form preservation which rules out two combinations. Being dependent on only U would require row permutations to be zero chasing and require no row permutations to be form preserving which rules out the combination of having both attributes.

The simplest form would be no zero chasing and no form preservation thus dependent on only U by the use of dense or sparse matrix methods. An example of this form would be the irreducible blocks method although the distinction between zero chasing and identifying the irreducible blocks maybe be minor and arbitrary. Simpler common methods like row, rook or complete pivoting make less use of the structure and have not been used as they generate more fillin. Notice that row pivoting becomes the original B&G variant with the column permutation to Hessenberg form. The four usual B&G variants are zero chasing and not form preserving thus dependent on only U. The Column Pivoting variant is not zero chasing, is form preserving and is dependent on only U as it avoids row permutations. The F&M Method is zero chasing, form preserving and dependent of both L and U as its zero chasing combines row permutations with dependence on both L and U. The Remultiply and Factor variant is not zero chasing, is form preserving and depends on both L and U as the remultiply is not zero preserving although the factoring benefits from the identification of the irreducible blocks of the product. There are five observed combinations of attributes with one combination having four variants and four combinations having one variant each.

The zero chasing attribute is realized in two forms with a total of five variants which have been described in the literature. The three forms with a total of three variants which are not zero chasing are newly described in this note. The forms which are either form preserving or dependent on both L and U are both in the literature or new to this note.

The variants which are zero chasing, dependent on only U or both do not have the property of preserving sparsity well. The variants which depend on both L and U, which in practice requires it to be form preserving, have the property of numerical stability. The design technique of using the irreducible blocks is associated with lowered execution costs. We notice that the Remultiply and Factor variant achieves both sparsity preservation and numerical stability by being not zero chasing, form preserving and dependent on on both L and U which systematically differs from the design of the B&G Method.

Linear programming codes benefit from the improved numerical properties that

lower the need for feasibility recovery and allow for the use of tighter tolerances that reflect the problem being solved rather than the introduced error from factoring problems. This will be of particular benefit to those applications of linear programming that repeatedly solve variations on the same underlying problem. The ability to use tighter tolerances may also make use of linear programming simpler for those end users who are not experts in applying the methods and dealing with possible sources of error.

The form preservation both lowers the immediate size of the factors and allows the factors to be stored in more contiguous data structures. Current processors often have memory hierarchies that operate more efficiently with greater data locality. It is common for the data movements within the memory hierarchies to be more expensive than the arithmetic carried out on the data.

Summary of Algorithm Attributes and Properties

Design Attributes				Operational Properties			
Zero Chasing	Form Preserving	Dependency	Possible Combination	Low Fillin	Low Numerical Growth	Speed Ordering	Algorithm
Yes	No	U	Yes	No	No	3 (Reid)	Bartels & Golub (4 variants)
Yes	No	L & U	No				
Yes	Yes	U	No				
Yes	Yes	L & U	Yes	No	**Yes**		Fletcher & Matthews
No	No	U	Yes	No	No	1	Irreducible Blocks
No	No	L & U	No				
No	Yes	U	Yes	No	No		Column Pivoting
No	Yes	L & U	Yes	**Yes**	**Yes**	2	Remultiply & Factor

REFERENCES

[1] R. H. Bartels and G. H. Golub, *The Simplex Method of Linear Programming Using the LU Decomposition*, Communications of the ACM, 12 (1969), pp 266-268.

[2] J. M. Elble and N. V. Sahinidis, *A Review of the LU Update in the Simplex Algorithm*, Int. J. Mathematics in Operations Research, 4 (2012), pp 366-399.

[3] R. Fletcher and S. P. F. Matthews, *Stable Modification of Explicit LU Factors for Simplex Updates*, Mathematical Programming, 30 (1984), pp 267-284.

[4] J. J. H. Forrest and J. A. Tomlin, *Updating Triangular Factors of the Basis to Maintain Sparsity in the Product Form Simplex Method*, Mathematical Programming, 2 (1972) pp. 263-278.

[5] M. J. D. Powell, *On Error Growth in the Bartels-Golub and Fletcher-Matthews Algorithms for Updating Matrix Factorizations*, Linear Algebra and Its Applications, 88/89 (1987), pp 597-621.

[6] J. K. Reid, *A Sparsity-Exploiting Variant of the Bartels-Golub Decomposition for Linear Programming Bases*, Mathematical Programming, 24 (1982), pp 55-69.

[7] P. Stange, A. Griewank and M. Bollhoefer. *On The Efficient Update of Rectangular LU Factorizations Subject to Low Rank Modifications*, Electronic Transactions On Numerical-Analysis, 26 (2007), pp 161-177.