Dense Factors of Sparse Matrices [1]

Roger Fletcher

Department of Mathematics and Computer Science, University of Dundee, Dundee DD1 4HN, Scotland, UK.

Numerical Analysis Report NA/170, July 1996

Abstract

The method of Implicit LU factors for factorizing a nonsingular matrix A, and hence solving systems of equations, is described. It is shown how the factors are related to the regular LU factors computed by Gaussian Elimination. A backward error analysis is given and discussed. Implicit LU factors are shown to be advantageous when a copy of A is kept for other purposes, and particularly so when A is a sparse matrix stored in compact form. It is shown how the method can be implemented efficiently when there are many unit columns in A.

Techniques for updating implicit LU factors are described when one column in A is replaced by a new column, with applications to linear programming and related calculations. Ideas in the Fletcher-Matthews method for updating regular LU factors are adapted to update implicit factors, and the resulting method is seen to be more efficient.

Other possibilities for implicit factors are also described which use more storage but are potentially more stable. Some comparisons on a range of linear programming problems are presented and discussed.

1 Introduction

The research in this paper arises in the context of Simplex-like methods for Linear Programming (LP) and related calculations. Such methods involve a nonsingular $n \times n$ matrix A, and systems of linear equations involving both A and A^T are required to be solved. On each iteration of the Simplex method a column is removed from A and a new column is added (the Simplex update). The columns that arise in the LP problem, and hence in A, are usually sparse and are stored in compact format.

[1]This paper is dedicated to Mike Powell on the occasion of his 60th birthday, and was presented at the Conference on Numerical Mathematics, Cambridge, July 1996.

To solve the very largest LP problems, say 10^5 variables or more, then very specialised product form methods have been devised (see for example Suhl [21]). Development of codes for these methods is very intricate and it is common for some man years of effort to be required. Also there are some concerns about numerical stability.

For smaller LP problems, say 1000 variables or less, it becomes more attractive to compute dense factors, avoiding problems with fill-in and product forms. This gives a simple code which is readily understood and maintained, and hopefully permits better numerical stability. It is nonetheless desirable to take advantage of the fact that A is stored in compact form.

An obvious candidate would be to use regular LU or QR factors which are readily updated and have exhibited excellent numerical stability in practice. This papers argues that it can be much better to use so called Implicit LU factors with significantly less storage and operations counts, yet with good numerical stability.

Ideas of implicit factorizations go back many years and I am grateful to Michele Benzi of CERFACS for information on some of the early papers. Different types of implicit factors are possible, and there are relationships with iterative methods such as conjugate gradients and projection methods. More recently, those familiar with null-space methods will also see a connection. To give an idea of the frequency with which these ideas have been discovered and rediscovered, some the earlier papers are itemized in the following table

Fox, Huskey and Wilkinson	[9]	1948
Hestenes and Stiefel	[12]	1952
Purcell	[16]	1953
Householder	[13]	1955
Pietrzykowski	[15]	1960
Faddeev and Faddeeva	[7]	1963
Stewart	[19]	1973
Enderson and Wassyng	[6]	1978
Sloboda	[18]	1978
Wassyng	[23]	1982
Abaffy, Broyden and Spedicato	[1]	1984
Hegedus	[11]	1986

Most of the papers develop the method as an iterative method based on conjugacy and orthogonality properties, although Purcell does recognise the method as being a competitor of the Crout decomposition. The term Implicit LU factors is first used in the paper of Abaffy, Broyden and Spedicato and they give more references to previous related work. Abaffy, Broyden and Spedicato regard implicit LU factors as a special case of a so-called ABS method, and there are many more recent contributions by Spedicato and his co-workers. Other recent researchers to use the idea are Tuma [22], Benzi and Meyer [3] and Benzi, Meyer and Tuma [4].

Despite these many references, the ideas involved are not well known in the numerical analysis community, perhaps because they are not aired in standard texts, and perhaps because the diversity of previous papers has not established a common style for presentation of the algorithms. Also, algorithm (2.15) given below does not appear to be referenced elsewhere, so the potential of Implicit LU factors for facilitating solves with both A and A^T may not have been recognised.

This paper makes some other contributions which are thought to be new, including a backward error analysis (Section 3), application to block systems (Section 4), and the derivation of updating techniques (Section 5). A comparison of other types of implicit factors (which are special cases of ABS methods) is described and discussed in Section 7, and provides evidence that the efficiency of Implicit LU factors is obtained with only marginal loss of numerical stability. Hopefully this paper will also lead to an enhanced awareness of the existence of implicit factors, leading to their increased use in other situations.

2 Implicit LU factors

In this section the idea of Implicit LU factors of an $n \times n$ nonsingular matrix A is explained and its relationship to the regular LU factors that may be computed by Gaussian Elimination (GE) is described. It is also shown how Implicit LU factors may be used to solve well-determined systems of linear equations.

First the computation and use of regular LU factors by GE is reviewed, using a compact matrix notation. The details of the computation are well-known (e.g Wilkinson [24]) and can be expressed in the form

$$A^{(k+1)} = M^{(k)}A^{(k)} \qquad k = 1, 2, \ldots, n-1 \tag{2.1}$$

where $A^{(1)} = A$. $M^{(k)}$ differs from a unit matrix only in that $(M^{(k)})_{jk} = -m_{jk}$ for $j = k+1, \ldots, n$, where

$$m_{jk} = a_{jk}^{(k)}/a_{kk}^{(k)} \qquad j = k+1, \ldots, n \tag{2.2}$$

are the so-called *multipliers* on stage k. The operation in (2.1) eliminates the sub-diagonal elements in column k of $A^{(k)}$ so that the generic form of $M^{(k)}$ and $A^{(k)}$ is typified for $n = 6$ and $k = 3$ by

$$M^{(k)} = \begin{bmatrix} 1 & & & & & \\ & 1 & & & & \\ & & 1 & & & \\ & & -m_{43} & 1 & & \\ & & -m_{53} & & 1 & \\ & & -m_{63} & & & 1 \end{bmatrix} \quad \text{and} \quad A^{(k)} = \begin{bmatrix} a_{11}^{(k)} & a_{12}^{(k)} & a_{13}^{(k)} & a_{14}^{(k)} & a_{15}^{(k)} & a_{16}^{(k)} \\ & a_{22}^{(k)} & a_{23}^{(k)} & a_{24}^{(k)} & a_{25}^{(k)} & a_{26}^{(k)} \\ & & a_{33}^{(k)} & a_{34}^{(k)} & a_{35}^{(k)} & a_{36}^{(k)} \\ & & a_{43}^{(k)} & a_{44}^{(k)} & a_{45}^{(k)} & a_{46}^{(k)} \\ & & a_{53}^{(k)} & a_{54}^{(k)} & a_{55}^{(k)} & a_{56}^{(k)} \\ & & a_{63}^{(k)} & a_{64}^{(k)} & a_{65}^{(k)} & a_{66}^{(k)} \end{bmatrix}.$$

It is noted that the inverse of $M^{(k)}$ is obtained simply by changing the signs of the subdiagonal elements of $M^{(k)}$. Then it is readily verified that $A = LU$ where

$$L = M^{(1)^{-1}}M^{(2)^{-1}} \ldots M^{(n-1)^{-1}} \quad \text{and} \quad U = A^{(n)}. \tag{2.3}$$

Equivalently L is obtained simply by adding the multipliers m_{ij} into the corresponding sub-diagonal elements of a unit matrix, for all $i > j$. A row interchange may be carried out at the start of each stage (partial pivoting) to ensure that the pivot $a_{kk}^{(k)}$ is non-zero, and commonly the largest modulus pivot is chosen. This ensures that the elements of L are bounded in modulus by 1. The operations count for the computation of L and U is $\frac{1}{3}n^3 + O(n^2)$ multiplications, with a similar number of additions.

To solve a system $A\mathbf{x} = \mathbf{b}$, the systems

$$L\mathbf{y} = \mathbf{b} \qquad \text{and} \qquad U\mathbf{x} = \mathbf{y} \tag{2.4}$$

are solved in turn. These are triangular systems and may be solved by forward and backward substitution respectively, each substition requiring $\frac{1}{2}n^2 + O(n)$ multiplications. Likewise to solve $A^T\mathbf{x} = \mathbf{b}$, the systems

$$U^T\mathbf{y} = \mathbf{b} \qquad \text{and} \qquad L^T\mathbf{x} = \mathbf{y} \tag{2.5}$$

are solved in a similar way.

An algorithm to determine one particular type of Implicit LU factors, which we refer to as LIU factors, may now be described. It computes a lower triangular matrix $\mathbb{L}$ and a diagonal matrix $D = \operatorname{diag}(d_1, d_2, \ldots, d_n)$. It is convenient to introduce an intermediate lower triangular matrix $\mathbb{L}^{(k)}$, the sub-diagonal elements of which are zero in columns $k, \ldots, n-1$. For $n = 6$ and $k = 3$ this would be typified by

$$\mathbb{L}^{(k)} = \begin{bmatrix} 1 & & & & & \\ l_{21}^{(k)} & 1 & & & & \\ l_{31}^{(k)} & l_{32}^{(k)} & 1 & & & \\ l_{41}^{(k)} & l_{42}^{(k)} & & 1 & & \\ l_{51}^{(k)} & l_{52}^{(k)} & & & 1 & \\ l_{61}^{(k)} & l_{62}^{(k)} & & & & 1 \end{bmatrix}. \tag{2.6}$$

Initially $\mathbb{L}^{(1)} = I$ and the algorithm is

```
for $k = 1, 2, \ldots, n-1$
    $d_k = \mathbf{l}_k^{(k)T}\mathbf{a}_k$
    for $j = 1, \ldots, k$
        $\mathbf{l}_j^{(k+1)} = \mathbf{l}_j^{(k)}$
    end
    for $j = k+1, \ldots, n$                                  (2.7)
        $r_{jk} = \mathbf{l}_j^{(k)T}\mathbf{a}_k / d_k$
        $\mathbf{l}_j^{(k+1)} = \mathbf{l}_j^{(k)} - r_{jk}\mathbf{l}_k^{(k)}$
    end
end
```

where $\mathbf{l}_j^{(k)T}$ denotes row j of $\mathbb{L}^{(k)}$ and $\mathbf{a}_k$ denotes column k of A. Finally $\mathbb{L} = \mathbb{L}^{(n)}$ and $d_n = \mathbf{l}_n^{(n)T}\mathbf{a}_n$.

The relationship of this algorithm to GE is now considered. The proposition that

$$\mathbb{L}^{(k)}A = A^{(k)} \tag{2.8}$$

is established inductively, where $A^{(k)}$ is the matrix introduced in (2.1). Clearly (2.8) is true for $k = 1$. If (2.8) is true for some $k \geq 1$, it follows that

$$a_{jk}^{(k)} = \mathrm{l}_j^{(k)T}\mathbf{a}_k \qquad j = k, k+1, \ldots, n. \tag{2.9}$$

Hence the ratios

$$r_{jk} = \frac{\mathrm{l}_j^{(k)T}\mathbf{a}_k}{d_k} = \frac{a_{jk}^{(k)}}{a_{kk}^{(k)}} = m_{jk} \qquad j = k+1, \ldots, n \tag{2.10}$$

are just the multipliers in GE. Thus the LIU calculation in the inner loops of (2.7) can be expressed as

$$\mathbb{L}^{(k+1)} = M^{(k)}\mathbb{L}^{(k)}. \tag{2.11}$$

It follows that

$$A^{(k+1)} = M^{(k)}A^{(k)} = M^{(k)}\mathbb{L}^{(k)}A = \mathbb{L}^{(k+1)}A.$$

This is (2.8) with k replaced by $k+1$, which establishes (2.8) for all $k > 1$.

In the case $k = n$ it is deduced from (2.8) and (2.3) that

$$\mathbb{L}A = U \tag{2.12}$$

and it is seen that

- $\mathbb{L}$ for the LIU factors is the *inverse* of L for the regular LU factors
- D for the LIU factors is the *diagonal* of U for the regular LU factors

(since $U = A^{(n)}$ and $D = \mathrm{diag}(a_1^{(1)}, a_2^{(2)}, \ldots, a_n^{(n)})$). The rest of the matrix U is not explicitly available in the LIU approach but is given implicitly by the equations

$$u_{ij} = a_{ij}^{(n)} = \mathrm{l}_i^T\mathbf{a}_j \qquad i < j \tag{2.13}$$

where l_i^T denotes row i of $\mathbb{L}$. This explains the use of the term LIU (for L-Implicit-U) factors. It is also readily possible to calculate (Implicit-L)-U factors of A by carrying out *column* operations on A, eliminating elements above the diagonal. Indeed, this form seems to be the one that is used in all previous work. Of course this is equivalent to obtaining the LIU factors of A^T. However the form used in this paper has the advantage of being more closely related to the familiar form of GE in which L is unit triangular, and row operations and row pivoting are used. This discussion also brings out the lack of symmetry in LIU factors and it can be seen that the algorithm does not compete with

Choleski factors when A is symmetric and positive definite (except possibly for parallel computation, see Benzi and Meyer [3]).

For dense A, it is possible for $\mathbb{L}$ and D to overwrite the lower triangle of A as the calculation proceeds, in which case the LIU algorithm (2.7) requires $\frac{1}{3}n^3 + O(n^2)$ multiplications, as for GE. Also it will be seen below that only the strict upper triangle of A is required when solving systems with LIU factors. Likewise, only the strict upper triangle of A is required to recover U from (2.13). Thus the lower triangle of A can be replaced by $\mathbb{L}$, D and its strict upper triangle, without any essential loss of information (ignoring round-off). Of course for sparse A this property is unlikely to be advantageous.

Partial pivoting in the LIU algorithm is possible in a similar way to GE. To do this, the elements $a_{jk}^{(k)} = \mathbf{l}_j^{(k)T}\mathbf{a}_k$ are assembled at the start of stage k of (2.7), and a row interchange is made which brings the maximum modulus element onto the diagonal, to become d_k. This gives rise to factors

$$\mathbb{L}PA = U \tag{2.14}$$

and is exactly equivalent to partial pivoting in GE. However, in contrast to GE, complete pivoting is not practicable in the LIU algorithm because $A^{(k)}$ is not stored during the calculation of $\mathbb{L}$ and D, and computation of the relevant elements of $A^{(k)}$ would be too expensive. To simplify the subsequent presentation, it is assumed that the row permutations have been incorporated into A so that (2.14) can be written as (2.12).

The use of LIU factors in the solution of linear systems is now considered. Equations (2.4) and (2.5) cannot be used directly as U is not explicitly available. To solve a system $A\mathbf{x} = \mathbf{b}$, it is written as $\mathbb{L}A\mathbf{x} = \mathbb{L}\mathbf{b}$ and hence as $U\mathbf{x} = \mathbb{L}\mathbf{b}$ using (2.12). Thus, using (2.13),

$$\begin{aligned} x_i &= \left(\mathbf{l}_i^T\mathbf{b} - \sum_{j=i+1}^{n} u_{ij}x_j\right) \Big/ u_{ii} \\ &= \left(\mathbf{l}_i^T\mathbf{b} - \sum_{j=i+1}^{n} \mathbf{l}_i^T\mathbf{a}_j x_j\right) \Big/ d_i \\ &= \mathbf{l}_i^T\left(\mathbf{b} - \sum_{j=i+1}^{n} \mathbf{a}_j x_j\right) \Big/ d_i. \end{aligned}$$

The bracketed quantity is seen to be the partial sum in the calculation of the residual vector $\mathbf{b} - A\mathbf{x}$, giving rise to the algorithm

```
$\mathbf{b}^{(n)} = \mathbf{b}$
for $i = n, n-1, \dots, 1$
    $x_i = \mathbf{l}_i^T\mathbf{b}^{(i)}/d_i$                    (2.15)
    $\mathbf{b}^{(i-1)} = \mathbf{b}^{(i)} - \mathbf{a}_i x_i$
end.
```

This algorithm may be new and is certainly not well known. Because of the structure of $\mathbf{l}_i$, only a column of the strict upper triangle of A need be used in the update of $\mathbf{b}^{(i+1)}$. When A is a dense matrix, this enables the calculation to be completed in $n^2 + O(n)$ multiplications, the same as for (2.4).

To solve a system $A^T\mathbf{x} = \mathbf{b}$, the vector $\mathbf{y}$ defined by $\mathbf{x} = \mathbb{L}^T\mathbf{y}$ is introduced, as in (2.5). Then $A^T\mathbf{x} = \mathbf{b}$ can be written as $A^T\mathbb{L}^T\mathbf{y} = \mathbf{b}$ and hence as $U^T\mathbf{y} = \mathbf{b}$ using (2.12). Thus, using (2.13),

$$\begin{aligned} y_i &= \left(b_i - \sum_{j=1}^{i-1} u_{ji}y_j\right) \Big/ u_{ii} \\ &= \left(b_i - \sum_{j=1}^{i-1} \mathbf{a}_i^T\mathbf{l}_j y_j\right) \Big/ d_i \\ &= \left(b_i - \mathbf{a}_i^T \sum_{j=1}^{i-1} \mathbf{l}_j y_j\right) \Big/ d_i \end{aligned}$$

In this case, the summation is seen to involve the partial sum in the calculation of the solution vector $\mathbf{x} = \mathbb{L}^T\mathbf{y} = \sum_{i=1}^n \mathbf{l}_i y_i$, giving rise to the algorithm

```
$\mathbf{x}^{(0)} = \mathbf{0}$
for $i = 1, 2, \dots, n$
    $y_i = (b_i - \mathbf{a}_i^T\mathbf{x}^{(i-1)})/d_i$
    $\mathbf{x}^{(i)} = \mathbf{x}^{(i-1)} + \mathbf{l}_i y_i$                    (2.16)
end
$\mathbf{x} = \mathbf{x}^{(n)}$.
```

In this algorithm, $\mathbf{x}^{(i-1)}$ has the same structure as $\mathbf{l}_{i-1}$, so only column i of the strict upper triangle of A need be used in the calculation of y_i. As before, when A is a dense matrix, this enables the calculation to be completed in $n^2 + O(n)$ multiplications. In fact the algorithm may be initialized with $\mathbf{x}^{(0)}$ other than $\mathbf{0}$, and can be interpreted as an iterative projection method based on the orthogonality properties $\mathbf{l}_i^T\mathbf{a}_j = 0, \quad i > j$ that derive from (2.12). Many references have developed the LIU algorithm from this point of view.

3 An Error Analysis of LIU factors

The floating point error analysis of Gaussian Elimination and triangular substitution (Wilkinson [24]) has provided a thorough understanding of the numerical stability properties of these algorithms. The computed factors in GE are shown to be the exact factors $LU = A + E$ of a perturbed system, where E is referred to as the backward error. E can be bounded by an expression of the form $3\rho n\varepsilon$ in which ρ measures the growth in $A^{(k)}$ during the elimination and ε is the relative precision of the arithmetic. Thus the purpose of the pivoting process is seen to be one of inhibiting growth during the elimination. For a triangular system the computed solution is also the exact solution of a perturbed system in which the perturbation is bounded by a modest multiple of ε. These bounds may be combined to give a total backward error bound for solving $A\mathbf{x} = \mathbf{b}$ by GE of the form $\rho\phi(n)\varepsilon$ in which $\phi(n)$ is a modest quadratic in n (Stewart [20], Theorem 5.3). This bound usually overstates the dependence on n which is unlikely to be a dominant factor. If partial pivoting is used, exponential growth is possible, but in practice this is very rare. Ill-conditioned systems usually do not lead to growth in the backward error, and pose no particular difficulty for the algorithm.

It is with this background in mind that an error analysis of the LIU factorization and solution algorithms has been attempted. It is shown that a backward error expression for the factorization algorithm (2.7) may be obtained. Bounds on the backward error, akin to those for GE, are derived. The main feature is seen to be the need to avoid growth in the matrix $\mathbb{L}$. It is argued that, with partial pivoting, growth is no more likely with LIU factors than with LU factors. Hence it is concluded that there is little to choose between the two algorithms on the grounds of numerical stability. It has not proved possible to obtain a satisfactory backward error bound for the solution algorithms (2.15) and (2.16). The reasons for this are explained. It is argued that this is unlikely to cause significant deterioration in accuracy unless the magnitude of the solution $\mathbf{x}$ is large.

The main result for algorithm (2.7) in floating point arithmetic of relative precision ε is

Theorem 1. *There exists a lower triangular matrix E such that the computed LIU factors $\mathbb{L}$ and D are the exact factors of a matrix $A + E$, that is*

$$\mathbb{L}(A + E) = U, \tag{3.1}$$

where U is upper triangular, $D = \mathrm{diag}(u_{11}, u_{22}, \dots, u_{nn})$ and u_{jk} is given implicitly by $\mathbf{l}_j^T\mathbf{a}_k$ for all $j < k$. E may be bounded by

$$|e_{jk}| \le (\tfrac{3}{2}n^2 + O(n))\varepsilon\gamma_{jk} + O(\varepsilon^2) \tag{3.2}$$

where

$$\gamma_{jk} = \max_{p \ge k}\{\max_i |l_{ji}^{(p)} a_{ik}|\}. \tag{3.3}$$

Proof The errors that arise in processing a column $\mathbf{a}_k$ of A are considered. This column first arises on stage k of (2.7), and is used to calculate

$$a_{jk}^{(k)} = \mathrm{fl}(\mathbf{l}_j^{(k)T}\mathbf{a}_k)) \qquad j = k, \dots, n$$

($\mathrm{fl}(\cdot)$ indicates the outcome of a calculation in floating point arithmetic), followed by

$$m_{jk} = \mathrm{fl}(a_{jk}^{(k)}/a_{kk}^{(k)}) \qquad j = k+1, \dots, n.$$

A vector $\boldsymbol{\delta}^{(k)}$ to account for these errors may be defined by

$$\delta_j^{(k)} = 0 \quad j < k, \qquad \delta_k^{(k)} = a_{kk}^{(k)} - \mathbf{l}_k^{(k)T}\mathbf{a}_k, \qquad \delta_j^{(k)} = m_{jk}a_{kk}^{(k)} - \mathbf{l}_j^{(k)T}\mathbf{a}_k \quad j > k. \tag{3.4}$$

It then follows by virtue of the structure of $I\!\!L^{(k)}$ (see (2.6)) that

$$\mathbf{l}_k^{(k)T}(\mathbf{a}_k + \boldsymbol{\delta}^{(k)}) = a_{kk}^{(k)} \qquad \text{and} \qquad \mathbf{l}_j^{(k)T}(\mathbf{a}_k + \boldsymbol{\delta}^{(k)}) = m_{jk}a_{kk}^{(k)} \quad j > k,$$

so that if the calculation of $a_{jk}^{(k)}$ and m_{jk} is carried out on $\mathbf{a}_k + \boldsymbol{\delta}^{(k)}$ in exact arithmetic, then $a_{jk}^{(k+1)} = 0$ is obtained. Thus in exact arithmetic we can write

$$M^{(k)}I\!\!L^{(k)}(\mathbf{a}_k + \boldsymbol{\delta}^{(k)}) = \mathbf{u}_k \tag{3.5}$$

where $u_{kk} = a_{kk}^{(k)} = d_k$, $u_{jk} = 0$ for $j > k$, and u_{jk} is implicitly defined by $\mathbf{l}_j^{(k)T}\mathbf{a}_k$ for $j < k$.

To obtain bounds on $\boldsymbol{\delta}^{(k)}$, standard error analysis techniques are used. Thus we write $m_{jk} = (1 + f_{jk})a_{jk}^{(k)}/a_{kk}^{(k)}$ where $|f_{jk}| \le \varepsilon$ so that

$$|m_{jk}a_{kk}^{(k)} - a_{jk}^{(k)}| \le \varepsilon|a_{jk}^{(k)}| \qquad j > k.$$

Similarly we may express

$$\begin{aligned} a_{jk}^{(k)} = \mathrm{fl}(\mathbf{l}_j^{(k)T}\mathbf{a}_k)) = \; & (\ldots((a_{jk} + l_{j1}a_{1k}(1 + e_1^{jk}))(1 + f_1^{jk}) + l_{j2}a_{2k}(1 + e_2^{jk}))(1 + f_2^{jk}) + \ldots \\ & + l_{j,k-1}a_{k-1,k}(1 + e_{k-1}^{jk}))(1 + f_{k-1}^{jk}) \end{aligned}$$

where e_i^{jk} and f_i^{jk} are relative errors bounded by in modulus by ε. Collecting the errors on each term, and using $a_{jk} = l_{jj}^{(k)}a_{jk}$ and (3.4), gives

$$\begin{aligned} |\delta_k^{(k)}| &\le k^2\varepsilon \max_i |l_{ki}^{(k)}a_{ik}| + O(\varepsilon^2) \\ |\delta_j^{(k)}| &\le k^2\varepsilon \max_i |l_{ji}^{(k)}a_{ik}| + \varepsilon|a_{jk}^{(k)}| + O(\varepsilon^2) \\ &\le (k^2 + k)\varepsilon \max_i |l_{ji}^{(k)}a_{ik}| + O(\varepsilon^2) \qquad j > k. \end{aligned} \tag{3.6}$$

Further errors are incurred when $I\!\!L^{(p+1)} = \mathrm{fl}(M^{(p)}I\!\!L^{(p)})$ is calculated, both for $p = k$ and $p > k$, which can be accounted for by making a further perturbation of $\mathbf{a}_k$. The detailed effect of errors in calculating $I\!\!L^{(p+1)}$ is now considered. Since $l_{ji}^{(p+1)} = l_{ji}^{(p)}$ for $j \le p$ and all i, there is no error in this case. Likewise for $j > p$ and $i > p$. Moreover $l_{jp}^{(p+1)} = -m_{jp}$ for $j > p$ so there is also no error in this case. Rounding errors only arise for $j > p$ and $i < p$, defined by

$$\begin{aligned} l_{ji}^{(p+1)} &= (l_{ji}^{(p)} - m_{jp}l_{pi}^{(p)}(1 + g_{ji}^{(p)}))(1 + h_{ji}^{(p)}) \\ &= l_{ji}^{(p)} - m_{jp}l_{pi}^{(p)} + h_{ji}^{(p)}l_{ji}^{(p+1)} + g_{ji}^{(p)}(l_{ji}^{(p+1)} - l_{ji}^{(p)}) + O(\varepsilon^2) \end{aligned}$$

where $g_{ji}^{(p)}$ and $h_{ji}^{(p)}$ are relative errors bounded in modulus by ε. This can be expressed in the form

$$\mathbb{L}^{(p+1)} = M^{(p)}\mathbb{L}^{(p)} + E^{(p)} \tag{3.7}$$

where

$$e_{ji}^{(p)} = h_{ji}^{(p)} l_{ji}^{(p+1)} + g_{ji}^{(p)}(l_{ji}^{(p+1)} - l_{ji}^{(p)}) + O(\varepsilon^2). \tag{3.8}$$

Note that $E^{(p)}$ has zeros throughout rows $1, \ldots, p$ and columns $p, \ldots, n$. It also follows that

$$\left(E^{(p)}\mathbf{a}_k\right)_j = \sum_{i=1}^{p-1}(h_{ji}^{(p)} l_{ji}^{(p+1)} + g_{ji}^{(p)}(l_{ji}^{(p+1)} - l_{ji}^{(p)}))a_{ik} + O(\varepsilon^2)$$

and we can deduce the bound

$$\left|\left(E^{(p)}\mathbf{a}_k\right)_j\right| \le 3(p-1)\varepsilon \max_i\{\max(|l_{ji}^{(p+1)}a_{ik}|, \, |l_{ji}^{(p)}a_{ik}|)\} + O(\varepsilon^2) \tag{3.9}$$

for $j > p$ and $p \ge k$.

Equation (3.7) with $p = k$ can be substituted into (3.5) to give

$$(\mathbb{L}^{(k+1)} - E^{(k)})(\mathbf{a}_k + \boldsymbol{\delta}^{(k)}) = \mathbf{u}_k$$

and hence

$$\mathbb{L}^{(k+1)}(\mathbf{a}_k - E^{(k)}\mathbf{a}_k + \boldsymbol{\delta}^{(k)}) = \mathbf{u}_k$$

since $E^{(k)}\boldsymbol{\delta}^{(k)} = \mathbf{0}$. Subsequent operations on $\mathbb{L}^{(p)}$ for $p > k$ can be accommodated in a similar way (note that $M^{(p)}\mathbf{u}_k = \mathbf{0}$) to give

$$\mathbb{L}\left(\mathbf{a}_k + \boldsymbol{\delta}^{(k)} - \sum_{p=k}^{n-1} E^{(p)}\mathbf{a}_k\right) = \mathbf{u}_k. \tag{3.10}$$

The error terms in (3.10) can all be bounded using the growth factor (3.3) and the bounds in (3.6) and (3.9). The leading coefficient for the total error E_{jk} is

$$k^2 + k + 3\sum_{p=k}^{n} p$$

which reduces to $\frac{3}{2}n^2 - \frac{1}{2}k^2$ and hence yields the result in (3.1).

End of proof

Some discussion on the bounds given by Theorem 1 is appropriate. The quantities $l_{ji}^{(p)}a_{ik}, \;\; i = 1, 2, ..., n$ in (3.3) are the individual terms in the scalar product $\mathbf{l}_j^{(p)}\mathbf{a}_k$ that defines $a_{jk}^{(p)}$. It is the factor $\max_p |a_{jk}^{(p)}|$ that features in the error bound for the computation of LU factors by GE. Thus these error bounds are closely related. The factor $\frac{3}{2}n^2$ in place of $3n$ arises in (3.3), essentially because the individual terms in the scalar product have been used. However the total bounds for solving $A\mathbf{x} = \mathbf{b}$ by GE (Stewart, [20]) also contain an n^2 factor. These factors of n^2 or n are likely to overestimate the actual dependence of the error on n, and it is unlikely that the occurrence of an n^2 factor indicates a significantly worse dependence on n. However the calculation of any term $l_{ji}^{(p)}a_{ik}$ does create an error $\sim \varepsilon l_{ji}^{(p)}a_{ik}$ and so the dependence of the error bound on the growth factor γ_{jk} defined in (3.3) is likely to be realistic in practice. Note also that, as for the GE analysis, Theorem 1 does not make an assumption that partial pivoting has been used.

The message conveyed by (3.1) is that growth in the backward error is correlated with growth in any of the matrices $\mathbb{L}^{(p)}$ for $p = 1, \ldots, n$. If partial pivoting is used, then $|m_{jk}| \leq 1$ and the growth can be bounded by 2^{n-2}. This is attained for the matrix

$$A = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & 1 \\ -1 & 1 & 0 & \cdots & 0 & 1 \\ -1 & -1 & 1 & \cdots & 0 & 1 \\ & & & \ddots & & \\ -1 & -1 & -1 & \cdots & 1 & 1 \\ -1 & -1 & -1 & \cdots & -1 & 1 \end{bmatrix}.$$

This is the same matrix for which the growth bound on U for GE is obtained. In practice serious growth is very rare, even if the bound on the multipliers is relaxed, as in some sparse matrix methods. These considerations, and the similarity of the backward error bounds, suggest that the numerical stability of the calculation of both LIU factors and LU factors is very satisfactory, and there is little to choose between the algorithms on this account.

Another consequence of (3.1) is that

$$\mathbb{L}A = U + \mathbb{L}^{-1}E \tag{3.11}$$

where $\mathbb{L}^{-1}E$ is also lower triangular. Now $\mathbb{L}^{-1}$ is the matrix L in the LU factors of A. If partial pivoting is used then the elements of L are in $[-1, 1]$ and hence $\mathbb{L}^{-1}E$ has a similar sort of bound to that for E.

Turning to the error analysis of the solution algorithms (2.15) and (2.16) that use LIU factors, most of the errors that arise can be thrown onto the data, as in Wilkinson's [24] analysis of triangular substitution. However the errors arising from the partial sums $\mathbf{b}^{(i)}$ (for (2.15)) and $\mathbf{x}^{(i)}$ (for (2.16)) are not obviously disposed of in this way. This may indicate a potential source of instability if significant growth in the partial sums occurs. For GE there is a good backward error bound for the entire process of solving $A\mathbf{x} = \mathbf{b}$ or $A^T\mathbf{x} = \mathbf{b}$. An important consequence is that if $\mathbf{x} \sim O(1)$ then the solution process gives an accurate residual error *that is unaffected by any ill-conditioning in* A. However in this case then the partial sums arising from (2.15) and (2.16) are unlikely to grow, in which case accurate residuals are also likely to be obtained. This outcome has been verified by some Matlab experiments on increasingly ill-conditioned matrices.

4 LIU factors for Sparse Matrices

LIU factors have some advantages when A is a sparse matrix that is stored in compact form. Assuming that there is a low proportion of non-zero elements in A, the operations involving A in (2.7), (2.15) and (2.16) have negligible cost. However there is a disadvantage that $\mathbb{L}$ is not usually very sparse (if A is block irreducible, $\mathbb{L}$ is generally fully dense), although Benzi and Meyer [3] have had some success with the use of drop tolerances in conjunction with Implicit LU factors. However, for very large sparse matrices, sparse LU factors are likely to provide the method of choice since both L and U often retain a significant amount of the sparsity in A. Nevertheless coding a sparse LU solver is a complex task because of the need to account for fill-in during the factorization. Updating such representations in algorithms such as the Simplex method for Linear Programming is also complex and there are some concerns in regard to numerical stability. If A is sparse but not too large (n is less than 1000, say) then it becomes attractive to consider the use of dense LIU factors. The storage required is $\frac{1}{2}n^2 + O(n)$ locations and the major cost is that of sdot or saxpy operations with rows of $\mathbb{L}$, which can be coded very efficiently. Moreover LIU factors are readily and stably updated when simplex column updates are made to A, as is demonstrated in Section 5.

In fact the situation can be even more favourable when A has many unit columns, a situation that often arises in Linear Programming applications. In this case we may express

$$A = \begin{bmatrix} A_1 & 0 \\ A_2 & I \end{bmatrix}$$

without loss of generality, where A_1 is $m \times m$ nonsingular, and I has dimension $n - m$. However more is gained if we permute this matrix so that A_2 is in the upper triangle, and so is handled implicitly. Thus we consider the matrix

$$A = \begin{bmatrix} I & A_2 \\ 0 & A_1 \end{bmatrix} \tag{4.1}$$

for which the subdiagonal elements of $\mathbb{L}$ are zero in columns $1, \ldots, n-m$. A particularly nice feature that does not previously seem to have been observed, is that the solution algorithms do not need to partition A into A_1 and A_2, but can work with columns of A itself. It is readily verified that algorithm (2.15) for solving $A\mathbf{x} = \mathbf{b}$ simplifies to become

```
$\mathbf{b}^{(n)} = \mathbf{b}$
for $i = n, n-1, \ldots, n-m+1$
    $x_i = \mathbf{l}_i^T \mathbf{b}^{(i)} / d_i$
    $\mathbf{b}^{(i-1)} = \mathbf{b}^{(i)} - \mathbf{a}_i x_i$
end                                                    (4.2)
for $i = 1, \ldots, n-m$
    $x_i = b_i^{(n-m)}$
end.
```

Also the algorithm for solving $A^T\mathbf{x} = \mathbf{b}$ becomes

```
for $i = 1, \dots, n-m$
    $\mathbf{x}_i^{(n-m)} = b_i$
end
for $i = n-m+1, \dots, n$
    $\mathbf{x}_i^{(n-m)} = 0$
end                                                        (4.3)
for $i = n-m+1, \dots, n$
    $y_i = (b_i - \mathbf{a}_i^T\mathbf{x}^{(i-1)})/d_i$
    $\mathbf{x}^{(i)} = \mathbf{x}^{(i-1)} + \mathbf{l}_i y_i$
end
$\mathbf{x} = \mathbf{x}^{(n)}$.
```

The dominant cost for each is $\sim \frac{1}{2}m^2$ multiplications, assuming that the operations with $\mathbf{a}_i$ have negligible cost.

5 Simplex updates of LIU factors

In the Simplex method for Linear Programming and related calculations, an important step is to update factors of A efficiently when one column of A is replaced by a new column. The ordering of the columns in A is immaterial. An efficient and reasonably stable algorithm for updating $PA = LU$ factors is that of Fletcher and Matthews [8], and this section shows how the same ideas can be adapted for updating LIU factors. These ideas are then extended to allow matrices of the form (4.1) to be updated, whilst retaining the block structure.

It is convenient to regard the Simplex update as taking place in two stages. First a column is removed from A and columns to the right are moved one place to the left. The factors of the resulting matrix are restored to triangular form. Then the matrix is extended by adding the new column as column n. We now review how the FM algorithm updates LU factors under these transformations.

When a column is removed from A the resulting matrix has factors in which the corresponding column has been removed from U. For example if column 2 is removed from a 5×5 matrix A, the resulting matrix has factors

$$\begin{bmatrix} a_{11} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{43} & a_{44} & a_{45} \\ a_{51} & a_{53} & a_{54} & a_{55} \end{bmatrix} = \begin{bmatrix} 1 & & & & \\ l_{21} & 1 & & & \\ l_{31} & l_{32} & 1 & & \\ l_{41} & l_{42} & l_{43} & 1 & \\ l_{51} & l_{52} & l_{53} & l_{54} & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{13} & u_{14} & u_{15} \\ & u_{23} & u_{24} & u_{25} \\ & u_{33} & u_{34} & u_{35} \\ & & u_{44} & u_{45} \\ & & & u_{55} \end{bmatrix}.$$

The aim of the FM method is to eliminate the resulting subdiagonal elements of U in a stable way. Without loss of generality we assume that column 1 of A has been removed, and A and U are renumbered to reflect this change. The first stage of the method determines an elementary matrix B which eliminates u_{21} in the equation

$$\begin{bmatrix} a_{11} \\ a_{21} \end{bmatrix} = \begin{bmatrix} 1 & \\ l_{21} & 1 \end{bmatrix} B^{-1} B \begin{bmatrix} u_{11} \\ u_{21} \end{bmatrix}$$

whilst retaining the unit triangular structure of L. If u_{11} is not zero, the matrix

$$B = \begin{bmatrix} 1 & \\ r & 1 \end{bmatrix} \tag{5.1}$$

can be used, where $r = -u_{21}/u_{11}$. The is referred to as the "no-perm" operation. Alternatively, an interchange may first be made to rows 1 and 2 of A, which also interchanges rows 1 and 2 of L. Then B may be made up of the product of two elementary matrices, as defined by the equation

$$\begin{bmatrix} a_{21} \\ a_{11} \end{bmatrix} = \begin{bmatrix} l_{21} & 1 \\ 1 & \end{bmatrix} \begin{bmatrix} & 1 \\ 1 & -l_{21} \end{bmatrix} \begin{bmatrix} 1 & \\ -s & 1 \end{bmatrix} \begin{bmatrix} 1 & \\ s & 1 \end{bmatrix} \begin{bmatrix} l_{21} & 1 \\ 1 & \end{bmatrix} \begin{bmatrix} u_{11} \\ u_{21} \end{bmatrix} \tag{5.2}$$

where $s = -u_{11}/\Delta$ and $\Delta = l_{21}u_{11} + u_{21}$. The outer pair of matrices in $B^{-1}B$ induce a unit submatrix in L, and the inner pair eliminate the resulting subdiagonal element of U. This is referred to as the "perm" operation. One of these possibilities is always valid. To minimize a bound on growth in the factors, Fletcher and Matthews essentially use the no-perm operation when the test

$$|\Delta| \le |u_{11}| \max(1, |l_{21}|) \tag{5.3}$$

is satisfied.

Once B is determined, it is applied from the left to rows 1 and 2 of U. Also B^{-1} is applied from the right to columns 1 and 2 of L. This requires two saxpy-like operations in the no-perm case and four in the perm case, and is the major cost of the stage. The same process is repeated in subsequent stages to eliminate the remaining subdiagonal elements of U. The total cost of the elimination is $O(n^2)$, with the constant depending on which column is removed and the extent to which perm operations are used. Finally a new column, $\mathbf{a}_n$ say, is added to A and the column $\mathbf{u}_n$ to be added to U is determined by solving the triangular system

$$L\mathbf{u}_n = \mathbf{a}_n, \tag{5.4}$$

where L is the lower triangular factor resulting from the elimination stages. Solving this equation is a significant proportion of the cost of the whole update.

The means by which the FM algorithm may be adapted to update LIU factors is now considered. The same stages are performed and the same matrices B are determined at each stage. The same test (5.3) is used to decide on a perm or no-perm operation. The differences are itemized in the following list

- B operations are applied from the left to $\mathbb{L}$, rather than B^{-1} being applied to the right of L.

- The current matrix U as the elimination proceeds is always available implicitly from the equation $\mathbb{L}A = U$, using the current copy of $\mathbb{L}$. Thus, although the u_{21} elements are directly available from D, the u_{11} elements must be generated by a scalar product between the relevant row of $\mathbb{L}$ and column of A.

- No operations on U need be performed. The new diagonal elements of D are just the pivot elements (either u_{11} or Δ) on the elimination stages.

- The element l_{21} of L is the negative of the corresponding element of $\mathbb{L}$.

- For the perm operation, the row permutation in L corresponds to a column permutation in $\mathbb{L}$.

- The final triangular substitution (5.4) to determine $\mathbf{u}_n$ is not required, only $d_n = \mathbf{l}_n^T\mathbf{a}_n$ is needed.

Thus a no-perm operation simply applies the matrix B in (5.1) from the left to rows 1 and 2 of the equation $\mathbb{L}A = U$. In the perm operation, columns 1 and 2 of $\mathbb{L}$ are first interchanged. Then an elementary operation is applied from the left, which induces a unit submatrix in $\mathbb{L}$. Finally another elementary operation is applied from the left to eliminate the resulting subdiagonal of U.

The costs of the algorithm in the case of LIU factors is $O(n^2)$, as for LU factors, but there are some differences. There is a considerable saving in not having to update U in the LIU case, but this is partially offset by the need to calculate u_{11} at each stage by a scalar product. Also more arithmetic operations are required to update $\mathbb{L}$ than L, except when it is column 1 that is removed from A. However not having to solve (5.4) is a considerable saving for the LIU algorithm, and overall I would expect the LIU algorithm to be noticeably more efficient in practice.

Updating strategies can also be determined when A is kept in the block form (4.1). If a non-unit vector is added or removed then the procedure described above for LIU factors also applies. Therfore only the cases in which a unit vector is added or removed need to be considered.

When a unit vector, $\mathbf{e}_{n-m}$ say, is removed from A in (4.1), it may be the case that the vector $\mathbf{s}$ defined by solving

$$A^T\mathbf{s} = \mathbf{e}_{n-m} \tag{5.5}$$

is available. This is often the case in LP calculations. One way to extend $\mathbb{L}$ is to move the unit element in row $n-m$ of $\mathbf{s}$ to row n, and shift up the intermediate elements. The same permutation is made to rows of $\mathbb{L}$ and columns of A. Then, using the new values, column n of A is deleted and row n of $\mathbb{L}$ is replaced by $\mathbf{s}$. It follows from the orthogonality conditions implied by (5.5) that $\mathbb{L}A$ is still upper triangular. Unfortunately it is possible that growth in the elements of $\mathbf{s}$ and hence $\mathbb{L}$ can occur with this method. A more stable alternative is simply to extend A_1 by moving the partition lines in (4.1) one column to the left and one row upwards. Then the procedure described earlier in this section may be used to update the LIU factors of A_1 when its first column is deleted. However, if $\mathbf{s}$ is available, it is worth using the more simple alternative if no significant growth is observed.

The main differences in procedure occur when a unit vector, $\mathbf{e}_p$ say, is added to A. To introduce this unit vector as the last column of A would not conform to the structure defined in (4.1) in which the unit columns are to the left of the non-unit columns. However the correct structure can be maintained by moving row p of A into the A_2 partition, and including a unit column in A. Hence what is required is a procedure for updating the LIU factors of a matrix A when a *row* is removed from A. Now it is possible (Matthews [14]) to adapt the Fletcher-Matthews method so as to update LU factors when a row of A is removed. What is described here is a modified form of this procedure which allows the LIU factors to be updated. It is assumed that a column has previously been deleted from A, leaving A and U as $n\times(n-1)$ matrices, and $\mathbb{L}$ as an $n\times n$ matrix, with $\mathbb{L}A=U$. The row to be deleted from A is denoted by p. The procedure takes place in a number of stages, indexed by p, $p+1,\ldots,n-1$. It is convenient to regard each stage as starting off by interchanging rows p and $p+1$ of A and columns p and $p+1$ of $\mathbb{L}$. The net effect over all stages is to cyclically permute row p of A to row n, when it can then be deleted.

The generic procedure within a stage is now described, in the case that $p=1$. After the initial interchange, the leading submatrix satisfies the equation

$$\begin{bmatrix} & l_{11} \\ 1 & l_{21} \end{bmatrix}\begin{bmatrix} a_{21} & a_{22} \\ a_{11} & a_{12} \end{bmatrix}=\begin{bmatrix} u_{11} & u_{12} \\ & u_{22} \end{bmatrix}. \tag{5.6}$$

In fact $l_{11}=1$ holds on the first stage, but this is not necessarily the case on subsequent stages, although $l_{11}\neq 0$ since $\mathbb{L}$ is always nonsingular. Having $l_{11}\neq 1$ enables an additional scaling operation to be avoided, The elements u_{11} and u_{22} in (5.6) are available as d_1 and d_2 respectively, but u_{12} must be evaluated at the start of the stage by a scalar product between the relevant row of $\mathbb{L}$ and column of A. It is required to multiply equation (5.6) on the left by a matrix B in order to restore a lower triangle in $\mathbb{L}$ and retain the upper triangle in U. A straightforward possibility is to choose B as the elementary matrix

$$B=\begin{bmatrix} r & 1 \\ & 1 \end{bmatrix} \tag{5.7}$$

where $r=-l_{21}/l_{11}$, and this is referred to as the "no-perm" operation. This fails when $l_{21}=0$ as the resulting lower triangular matrix has zero on the diagonal. The alternative choice defines B as the product of two elementary matrices

$$B = \begin{bmatrix} 1 & \\ s & 1 \end{bmatrix} \begin{bmatrix} r & 1 \\ 1 & \end{bmatrix} \tag{5.8}$$

where $s = -u_{12}/\Delta$ and $\Delta = u_{22} - l_{21}u_{12}/l_{11}$, and this is referred to as the "perm" operation. In this case a subsequent column interchange in A and U is used to make the leading submatrix in U upper triangular. This operation fails if $\Delta = 0$. A test analogous to (5.3) is to choose the no-perm operation when

$$|\Delta| \leq \left|\frac{l_{21}}{l_{11}}\right| \max(|u_{11}|, |u_{12}|). \tag{5.9}$$

The same procedure is also applied on subsequent stages. On stages other than $k = p = 1$ the operation defined by B is also applied to the other elements in rows p and $p+1$ of $\mathbb{L}$. This is the dominant cost of the updating process, which therefore requires $O(n^2)$ arithmetic operations in all.

6 Other implicit factorizations

An interesting observation is that the algorithms (2.15) and (2.16) for respectively solving $A\mathbf{x} = \mathbf{b}$ and $A^T\mathbf{x} = \mathbf{b}$ do *not* require $\mathbb{L}$ to be triangular, only U. Hence another practicable implicit factorization would be to have factors

$$SA = U \tag{6.1}$$

where S is square. These are referred to as SIU factors. As before the diagonal elements of U are available as a matrix D, but the rest of U is implicit. An advantage of this format is that Simplex updates take a more simple form in which only one elementary operation per stage is required. This is described by reference to the case in which column 2 has been deleted from a 4×4 matrix A, leaving a matrix A'. The first stage of the updating process is to eliminate u_{33} from the SIU factors. If $|u_{23}| \geq |u_{33}|$ then the operation

$$\begin{bmatrix} 1 & & & \\ & 1 & 0 & \\ & r & 1 & \\ & & & 1 \end{bmatrix} \begin{bmatrix} s_{11} & s_{12} & s_{13} & s_{14} \\ s_{21} & s_{22} & s_{23} & s_{24} \\ s_{31} & s_{32} & s_{33} & s_{34} \\ s_{41} & s_{42} & s_{43} & s_{44} \end{bmatrix} A' = \begin{bmatrix} 1 & & & \\ & 1 & 0 & \\ & r & 1 & \\ & & & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{13} & u_{14} \\ & u_{23} & u_{24} \\ & u_{33} & u_{34} \\ & & u_{44} \end{bmatrix}.$$

can be used, where $r = -u_{33}/u_{23}$. Otherwise the operation

$$\begin{bmatrix} 1 & & & \\ & 0 & 1 & \\ & 1 & s & \\ & & & 1 \end{bmatrix} \begin{bmatrix} s_{11} & s_{12} & s_{13} & s_{14} \\ s_{21} & s_{22} & s_{23} & s_{24} \\ s_{31} & s_{32} & s_{33} & s_{34} \\ s_{41} & s_{42} & s_{43} & s_{44} \end{bmatrix} A' = \begin{bmatrix} 1 & & & \\ & 0 & 1 & \\ & 1 & s & \\ & & & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{13} & u_{14} \\ & u_{23} & u_{24} \\ & u_{33} & u_{34} \\ & & u_{44} \end{bmatrix}.$$

is used, where $s = -u_{23}/u_{33}$, and the situation is somewhat reminiscent of the Bartels-Golub method [2]. Subsequent stages are used to eliminate all the subdiagonal elements of U. As in the previous section, the elements u_{kk} are available as the elements d_k, but $u_{k-1,k}$ must be evaluated when needed by a scalar product between row $k-1$ of the current S matrix and column k of A. Once U has been returned to triangular form, a new column $\mathbf{a}_n$ can be added to A. As U is implicit, all that needs to be done is to calculate $d_n = \mathbf{s}_n^T \mathbf{a}_n$, where $\mathbf{s}_n^T$ denotes row n of S.

A related possibility is to update implicit factors

$$QA = U \tag{6.2}$$

(QIU factors) in which Q is an orthogonal matrix. In this case the elimination of a subdiagonal element of U can be done using Givens' rotations. This is illustrated by

$$\begin{bmatrix} 1 & & & \\ & c & s & \\ & s & -c & \\ & & & 1 \end{bmatrix} \begin{bmatrix} q_{11} & q_{12} & q_{13} & q_{14} \\ q_{21} & q_{22} & q_{23} & q_{24} \\ q_{31} & q_{32} & q_{33} & q_{34} \\ q_{41} & q_{42} & q_{43} & q_{44} \end{bmatrix} A' = \begin{bmatrix} 1 & & & \\ & c & s & \\ & s & -c & \\ & & & 1 \end{bmatrix} \begin{bmatrix} u_{11} & u_{13} & u_{14} \\ & u_{23} & u_{24} \\ & u_{33} & u_{34} \\ & & u_{44} \end{bmatrix},$$

where $c = u_{23}/\sqrt{(u_{23}^2 + u_{33}^2)}$ and $s = u_{33}/\sqrt{(u_{23}^2 + u_{33}^2)}$.

Both these methods require $n^2 + O(n)$ storage, so both this and the operations count are greater than for LIU factors. However the numerical stability of these methods is likely to be superior. It can be assumed that numerical stability is correlated with any growth that occurs in the matrices $\mathbb{L}$, S or Q. Of course the elements of Q are bounded in modulus by 1, so there is no growth and this can be expected to give the most stable method. The growth in S is limited by a factor of 2 per stage, which is a better bound than can be obtained for the Fletcher-Matthews method, and hence presumably for the methods derived in Section 5. Thus it would be expected that the SIU method is more stable than the LIU method. In practice it is likely that the actual growth is much less than is predicted by the bound. It is also observed that the determinants of $\mathbb{L}$, S and Q are all ± 1. Some experiments which attempt to quantify the cost and stability of these methods are described in the next section.

Another implicit factorization that is quite different from the methods discussed in this paper is to have QR factors

$$A = QR \tag{6.3}$$

in which A and R are stored, and Q is defined implicitly by $Q = AR^{-1}$. This only requires $\sim \frac{1}{2}n^2$ storage so is comparable with LIU factors in this respect. It is also possible to update the factors using Givens' rotations when columns are added to or removed from A (Gill and Murray [10], Siegel [17]). However a disadvantage is that two solves with R are needed to solve $A\mathbf{x} = \mathbf{b}$. Thus the forward error in $\mathbf{x}$ behaves like $\kappa(A)^2\varepsilon$ rather than $\kappa(A)\varepsilon$ for LIU factors, so the method is more seriously affected by ill-conditioning. For this reason this method is little used, although there is some evidence (Björck [5]) that iterative refinement can improve matters.

7 Numerical results and Conclusions

To quantify the relative costs and numerical stability of the different implicit factorizations, a range of calculations on eleven LP problems from the SOL test set with n up to 688 has been carried out. These calculations involve a mixture of Simplex updates to A and solves with A and A^T. These calculations dominate the cost of solving the LP problems and so provide an overall measure of the efficiency of the different implicit factorization schemes. The calculations have been repeated with different types of prescaling strategies and the use or non-use of steepest edge pricing. The coefficient matrices in the LP problems, and hence the columns of A, are sparse and are stored in compact form.

Not much is gained by reporting the detailed outcome of all these tests (tables are available from me on request) but the following statistics provide an overview.

Computer Time

$$\frac{\text{LIU method}}{\text{SIU method}} \qquad \text{typical range} \qquad 0.7 \sim 0.8$$

$$\frac{\text{QIU method}}{\text{SIU method}} \qquad \text{typical range} \qquad 1.5 \sim 2.0$$

The first of these comparisons reflects the improvement in efficiency obtained by working with a triangular matrix $\mathbb{L}$ as against a square matrix S, moderated by the extra operations required by a perm elimination and the added complexity of manipulating triangular matrices. The overall effect is about what might have been predicted. The second comparison indicates the additional cost of using Givens' rotations rather than elementary elimination operations, and the ratio of cost is somewhat larger than I would have expected.

Two different indicators of numerical stability are given. The first is to monitor the maximum growth in $\mathbb{L}$ and S over *all* LP iterations.

Maximum growth

LIU method	typical range	$100 \sim 2000$
SIU method	typical range	$5 \sim 50$

Of course no growth in Q is obtained. It should be stressed that these figures represent the worst that happens during the LP calculation and that very large growth only arises on the occasional iteration, so that the average growth is much less. Scaling and the use of steepest edge pricing also tend to reduce the maximum growth that occurs. Nonetheless one would expect that when large growth does occur, then a corresponding growth in backward error would be obtained which would persist through subsequent updates.

Another indication of numerical stability is to calculate the accumulated error, by evaluating the residual of the equation $\mathbb{L}A = U$ (or $SA = U$ or $QA = U$) at the end of each LP calculation. The outcome is very uneven but an overall impression has been obtained by averaging (in log space) the maximum residual for each LP problem. Scaling columns of A to have unit length has a significant effect, so results for unscaled problems are given separately. The results are obtained on a SUN ELC in double precision, for which the machine precision is $\varepsilon \simeq 1.1_{10}-16$.

Accumulated error

	LIU	SIU	QIU
Scaled problems	$1.8_{10}-14$	$1.5_{10}-14$	$4.3_{10}-15$
Unscaled problems	$2.4_{10}-12$	$1.6_{10}-12$	$2.5_{10}-13$

In fact the SIU method did somewhat better than these results indicate, as the averages include 3 very poor results (out of 44) for the scaled problems. If these results are removed from the comparison then the average improves to $5.2_{10}-15$ which is almost as good as the QIU method. Also, if a bad result for the SIU method is deleted from the 22 runs on scaled problems, the average improves to $8.4_{10}-13$. Given that the variance of the results is large, the most that can be concluded is that the LIU method is about a factor of 10 worse on average than the QIU method, with the SIU method somewhere in the middle.

Given the much higher maximum growth for the LIU method, it is somewhat surprising that a more substantial discrepancy in the accumulated error is not obtained. The reason may be that when large growth occurs, the associated large errors are confined to the column that has been introduced. When this column is subsequently removed, it may be that the associated large error is also removed. Similar conclusions were reached in [7] in regard to updates of LU factors.

Overall the LIU method requires only half the storage of the SIU and QIU methods, and it also performs best in regard to computing time. On average it is about a factor of 10 worse in regard to accuracy than the QIU method. Since the QIU method obtains accuracy close to the machine precision, this seems to be a small price to pay for the better performance. Similar conclusions could be expected if the method were compared with regular (dense) LU or QR factors.

8 References

[1] Abaffy J., Broyden C. and Spedicato E. (1984), A class of direct methods for linear systems, *Numer. Math.*, **45**, 361-376.

[2] Bartels R.H. (1971), A stabilization of the simplex method, *Numer. Math.*, **16**, 414-434.

[3] Benzi M. and Meyer C.D. (1995), A direct projection method for sparse linear systems, *SIAM J. Sci. Comput.*, **16**, 1159-1176.

[4] Benzi M., Meyer C. D. and Tuma M. (1996), A sparse approximate inverse preconditioner for the conjugate gradient method, *SIAM J. Sci. Comput.*, **17**, 1135-1149.

[5] Björck Å. (1987), Stability analysis of the method of semi-normal equations for least squares problems, *Linear Algebra Applns.*, **88/89**, 31-48.

[6] Enderson D. and Wassyng A. (1978), A new method for the solution of $Ax = b$, *Numer. Math.*, **29**, 287-289.

[7] Faddeev D.K. and Faddeeva V.N. (1963), *Computational Methods of Linear Algebra*, W.H. Freeman, San Francisco.

[8] Fletcher R. and Matthews S.P.J. (1984), Stable modification of explicit LU factors for simplex updates, *Math. Progr.*, **30**, 367-284.

[9] Fox L., Huskey H.D. and Wilkinson J.H. (1948), Notes on the solution of algebraic linear simultaneous equations, *Quart. J. Mech. Appl. Math.*, **1**, 149-173.

[10] Gill P.E. and Murray W. (1973), A numerically stable form of the simplex algorithm, *Linear Algebra Applns.*, **7**, 99-138.

[11] Hegedüs C.J. (1986), Newton's recursive interpolation in $\mathbb{R}^n$, *Colloquia Math. Soc. János Bolyai*, **50**, 605-623.

[12] Hestenes M.R. and Stiefel E. (1952), Methods of conjugate gradients for solving linear systems, *J. Res. NBS*, **49**, 409-436.

[13] Householder A.S. (1955), Terminating and non-terminating iterations for solving linear systems, *J. SIAM*, **3**, 67-72.

[14] Matthews S.P.J. (1984), Matrix algebra and other aspects of linear and quadratic programming, Ph.D. Thesis, Dept. of Math. Sci., Univ. of Dundee, Scotland.

[15] Pietrzykowski T. (1960), Projection method, *Zak. Apar. Mat. Polskiej Acad. Nauk*, Praca A8.

[16] Purcell E.W. (1953), The vector method of solving simultaneous linear equations, *J. Math. Phys.*, **32**, 180-183.

[17] Siegel D. (1992) Implementing and modifying Broyden class updates for large scale optimization, Report DAMTP 1992/NA12, Univ. of Cambridge.

[18] Sloboda F. (1978), A parallel projection method for linear algebraic systems, *Apl. Mat. Ceskosl. Akad. Ved.*, **23**, 185-198.

[19] Stewart G.W. (1973), Conjugate direction methods for solving linear systems, *Numer. Math.*, **21**, 283-297.

[20] Stewart G.W. (1973), *Introduction to Matrix Computations*, Academic Press, New York.

[21] Suhl U.H. (1994), MOPS – Mathematical OPtimization System, *European J. Operational Research*, **72**, 312-322.

[22] Tuma M. (1993), Solving sparse unsymmetric sets of linear equations based on implicit Gauss projection, Tech report 555, Inst. of Comp. Sci., Czech Acad. Sci., Prague.

[23] Wassyng A. (1982) Solving $Ax = b$: a method with reduced storage requirements, *SIAM J. Num. Anal.*, **19**, 197-204.

[24] Wilkinson J.H. (1965), *The Algebraic Eigenvalue Problem*, Oxford University Press, Oxford.