# A Data Structure for Sparse $QR$ and $LU$ Factorizations [†]

*Alan George* [††]
*Joseph Liu* [†††]
*Esmond Ng* [††]

CS-85-16
June 1985

† Research supported in part by the Canadian Natural Sciences and Engineering Research Council under grants A8111 and A5509, by the Applied Mathematical Sciences Research Program, Office of Energy Research, U.S. Department of Energy under contract DE-AC05-84OR21400 with Martin Marietta Energy Systems Inc., and by the U.S. Air Force Office of Scientific Research under contract AFOSR-ISSA-84-00056.

†† Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1.

††† Department of Computer Science, York University, Downsview, Ontario, Canada M3J 1P3.

# A Data Structure for Sparse $QR$ and $LU$ Factorizations[†]

*Alan George*

Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada N2L 3G1

*Joseph Liu*

Department of Computer Science
York University
Downsview, Ontario, Canada M3J 1P3

*Esmond Ng*

Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada N2L 3G1

*ABSTRACT*

For a general $m$ by $n$ sparse matrix $A$, a new scheme is proposed for the structural representation of the factors of its sparse orthogonal decomposition by Householder transformations. The storage scheme is row-oriented and is based on the structure of the upper triangular factor obtained in the decomposition. The storage of the orthogonal matrix factor is particularly efficient in that the overhead required is only $m+n$ items, independent of the actual number of nonzeros in the factor. The same scheme is applicable to sparse orthogonal factorization by Givens rotations, and also to the recent implementation of sparse Gaussian elimination with partial pivoting developed by George and Ng [9]. Experimental results are provided to compare the sparse Gaussian elimination using the new storage scheme with that proposed in [9].

---

† Research supported in part by the Canadian Natural Sciences and Engineering Research Council under grants A8111 and A5509, by the Applied Mathematical Sciences Research Program, Office of Energy Research, U.S. Department of Energy under contract DE-AC05-84OR21400 with Martin Marietta Energy Systems Inc., and by the U.S. Air Force Office of Scientific Research under contract AFOSR-ISSA-84-00056.

# Table of Contents

## 1. Introduction

Consider the orthogonal decomposition of an $m$ by $n$ matrix $A$ by Householder transformations, with $m \geq n$:

$$A = Q \begin{pmatrix} R \\ O \end{pmatrix} ,$$

where $Q$ is $m$ by $m$ orthogonal and $R$ is $n$ by $n$ upper triangular. It is well known that the orthogonal matrix $Q$ can be represented in a factored form as a sequence of vectors which will be referred to as Householder vectors. When $A$ is dense, the vectors are simply stored in the lower trapezoidal part of the storage array. See, for example, LINPACK [2]. In this paper, we propose a new structural representation for this sequence of vectors which is appropriate when $A$ is sparse.

The new data structure is row-oriented, and the row structures are shown to be characterized implicitly by the structure of the triangular factor $R$. This same data structure is equally applicable to sparse Gaussian elimination with partial pivoting. Recently, George and Ng have devised efficient static storage schemes for sparse Gaussian elimination with partial pivoting [11,9] and sparse orthogonal decomposition using Householder transformations [10]. The scheme proposed in this paper provides an alternate storage method for this static representation.

One advantage of this new data structure is its reduction in overhead indexing storage. More importantly, since the storage method is row-oriented, it is much more appropriate for the partial pivoting scheme which involves row interchanges and row modifications. This leads to a more efficient implementation of sparse Gaussian elimination with partial pivoting.

The original motivation for this paper was to design an efficient data structure for sparse Gaussian elimination. However, we find that the structure is more easily understood and better characterized in the context of sparse orthogonal decomposition. Therefore, our presentation here covers sparse orthogonal decomposition before sparse Gaussian elimination with partial pivoting.

An outline of this paper is as follows. In Section 2, we provide a brief overview of sparse Householder transformations, and give a characterization of a structural representation of sparse Householder vectors. We use this characterization in Section 3 to set up a row-oriented data structure. This structure is appropriate for sparse orthogonal decomposition using Householder reflections or Givens rotations, when the orthogonal transformations are to be stored. In Section 4, we briefly review the static scheme of George and Ng [9] for sparse Gaussian elimination with partial pivoting. The row storage scheme described for orthogonal decomposition is also applicable in the static structural representation of the triangular factors obtained in Gaussian elimination with partial pivoting. In Section 5, we provide experimental results for an implementation of sparse Gaussian elimination with partial pivoting using this new

data structure. Its performance is compared with the original scheme developed by George and Ng [9], and with the Harwell MA28 package [3] for solving sparse nonsymmetric systems. Some concluding remarks are contained in Section 6.

## 2. A Structural Representation of Sparse Householder Vectors

### 2.1. Background Material

We provide a brief overview of results used in this paper, and also define the necessary terminology. We shall assume that the given $m$ by $n$ ($m \geq n$) sparse matrix $A$ has full column rank. Hence $A^TA$ is symmetric and positive definite.

Consider the orthogonal decomposition of $A$ into $Q\begin{pmatrix} R \\ O \end{pmatrix}$, where $Q$ is orthogonal and $R$ is upper triangular. It is well known that $R$ is mathematically the same as the Cholesky factor of $A^TA$ (apart from possible sign differences in some rows). However, it is also well known that the *structure* of the Cholesky factor obtained by a *symbolic* factorization of the structure of $A^TA$ may overestimate that of $R$. In [1], Coleman et. al. have shown that if the matrix $A$ has the *strong Hall property*, then the symbolic Cholesky factorization of $A^TA$ will correctly provide the structure of $R$. For the purpose of this paper, we shall assume that the matrix $A$ has this property so that we can refer to the structure of the Cholesky factor of $A^TA$ and the structure of the factor matrix $R$ interchangeably. This assumption is reasonable since if the matrix $A$ does not have this property, it can be permuted to block upper triangular form where each diagonal block submatrix has the strong Hall property.

Consider the structure of the factor matrix $R$. For each row $i \leq n$, define $\gamma[i]$ by

$$\gamma[i] \;=\; \min \{\, j > i \;|\; r_{ij} \neq 0 \,\} \quad ;$$

that is, $\gamma[i]$ is the column subscript of the first off-diagonal nonzero in row $i$ of $R$. If row $i$ does not have any off-diagonal nonzero, we set $\gamma[i]=i$. (Hence $\gamma[n]=n$.) We shall be using the quantities $\gamma[1]$, $\gamma[2]$, $\cdots$, $\gamma[n]$ to characterize a structural representation of sparse Householder transformations.

### 2.2. Structural Characterization

The use of Householder transformations in the orthogonal decomposition of a matrix is well known. We review it briefly as follows to establish the necessary notation. Consider the $m$ by $n$ matrix

$$A \;=\; \begin{pmatrix} d & v^T \\ u & E \end{pmatrix} \quad ,$$

where $u$ and $v$ are $(m-1)$- and $(n-1)$-vectors respectively. Let $\sigma$ be the 2-norm of the first column of $A$:

$$\sigma = (d^2 + u^T u)^{1/2} \quad .$$

Assume that $\sigma$ is nonzero. Then, a Householder reflection can be constructed to annihilate the vector $u$ in the matrix $A$. Let

$$\beta = 1 + \frac{d}{\sigma_d} \quad ,$$

$$w = \frac{u}{\sigma_d} \quad ,$$

and

$$\sigma_d = sgn(d) \times \sigma \quad ,$$

where $sgn(d)$ is a function whose value is $+1$ if $d$ is non-negative, and $-1$ otherwise. The Householder transformation is then given by

$$P = I - \frac{1}{\beta} \begin{pmatrix} \beta \\ w \end{pmatrix} \begin{pmatrix} \beta \\ w \end{pmatrix}^T \quad .$$

The vector $\begin{pmatrix} \beta \\ w \end{pmatrix}$ is referred to as the *Householder vector.*

It is easy to see that after the transformation the matrix becomes

$$PA = \begin{pmatrix} -\sigma_d & v^T - y^T \\ 0 & E - w y^T/\beta \end{pmatrix} \quad ,$$

where $y^T = \beta v^T + w^T E$; that is, $y^T$ is a linear combination of $v^T$ and the rows from $E$. The vector $y$ plays an important role in terms of structural modifications in the vector $v^T$ and the submatrix $E$.

Thus, the matrix $A$ can be reduced to upper triangular form by a sequence of Householder transformations

$$P_1 , P_2 , \cdots , P_n$$

defined by the corresponding Householder vectors

$$\begin{pmatrix} \beta_1 \\ w_1 \end{pmatrix}, \begin{pmatrix} \beta_2 \\ w_2 \end{pmatrix}, \cdots , \begin{pmatrix} \beta_n \\ w_n \end{pmatrix} \quad ,$$

where $\beta_i$ is a scalar, and $w_i$ is a vector of size $m-i$.

In this paper, we are interested in the structure of this set of Householder vectors when the given matrix $A$ is sparse. For this purpose, we define the following $m$ by $n$ *Householder matrix* $H$ to be

$$H = \begin{pmatrix} \beta_1 & 0 & 0 & & 0 \\ & \beta_2 & 0 & & 0 \\ w_1 & & \beta_3 & & 0 \\ & w_2 & & . \; . & . \\ & & w_3 & . \; . & \beta_n \\ & & & . \; . & w_n \end{pmatrix} \quad .$$

In what follows, we shall relate the structure of the matrix $H$ to that of the upper triangular factor matrix $R$. We shall also assume that the diagonal elements of $A$ are nonzero. This assumption is reasonable since the rows of a sparse matrix with full rank can be reordered so that the diagonal elements of the permuted matrix are nonzero. See [5] for details. When the diagonal elements of a matrix are nonzero, the matrix is said to have a *zero-free diagonal.*

Let us re-examine the effect of applying the Householder transformation $P$ to $A$ in order to annihilate the nonzero entries below the diagonal in the first column. From the transformation process, we make the following observations, where we assume throughout that exact cancellations do not occur. Recall that $\gamma[i]$ denotes the column subscript of the first off-diagonal nonzero in row $i$ of $R$.

**Observation 1:** The first row of the factor matrix $R$ is given by

$$R_{1*} = (-\sigma_d \;\; , \;\; v^T - y^T) \quad .$$

**Observation 2:** The *structure* of $R_{1*}$ is the *union* of the row structures of $A_{i*}$ where $a_{i1} \neq 0$, assuming $A$ has a zero-free diagonal.

**Observation 3:** $r_{1,\gamma[1]} \neq 0$, and $r_{1,j} = 0$ for $1 < j < \gamma[1]$.

**Observation 4:** If $a_{i1} \neq 0$, then

$$(PA)_{i,\gamma[1]} \neq 0$$

and

$$(PA)_{i,j} = 0 \;\; , \qquad \text{for } 1 < j < \gamma[1] \quad .$$

In fact, the structure of $(PA)_{i*}$ is the same as that of $R_{1*}$. Furthermore, the diagonal of $PA$ remains zero-free.

Observation 4 implies that if $a_{i1} \neq 0$, then after the Householder transformation is applied to $A$ the next nonzero entry in row $i$ of $PA$ appears in column $\gamma[1]$. Since the algorithm is to be applied recursively to the submatrix $E - wy^T/\beta$, observation 4 can be used repeatedly and the next theorem then follows.

**Theorem 2.1:** If $f_i$ is the column subscript of the first nonzero in row $A_{i*}$, then the locations of the nonzeros in row $H_{i*}$ are given by

$$f_i \ , \gamma[f_i] \ , \gamma[\gamma[f_i]] \ , \ \cdots \ , t \quad ,$$

where $t$ is defined as follows. If $i \leq n$, then $t=i$; otherwise, $t$ is some column subscript with $\gamma[t]=t$. ■

## 3. A Data Structure for Sparse Orthogonal Decomposition

### 3.1. Graph-Theoretic Interpretation

In this section, we provide a graph-theoretic interpretation of the row structure of the Householder matrix $H$ defined in the previous section. It is based on the notion of an elimination tree for the structure of the Cholesky factor of $A^TA$, and is closely related to the row storage scheme for sparse Cholesky factors developed by Liu [13].

Define an *elimination tree* for the structure of the Cholesky factor $R$ as follows. The tree has $n$ nodes, labelled from 1 to $n$. For each $i$, if $\gamma[i]>i$, then node $\gamma[i]$ is the *parent* of node $i$ in the elimination tree. We assume that the matrix $A^TA$ is irreducible, so that $n$ is the only node with $\gamma[n]=n$ and it is the root of the tree. (If $A^TA$ is reducible, then the elimination tree defined above is actually a forest which consists of several trees.)

Theorem 2.1 says that the structure of each row of the Householder matrix $H$ corresponds to a *chain* in the elimination tree of $R$. Since the structure of this tree is already given implicitly in the row structure of $R$ (by the $\gamma$'s), it is sufficient to know for each row of $H$ where the associated chain starts in the tree (that is, the $f$'s). This can be viewed as a special case of row structures in [13], where row structures are generally subtrees of the elimination tree. As we shall see in the next section, it is useful to know the location of a given node in the elimination tree relative to the root. Hence we define the *level number* $l_i$ of node $i$ in the elimination tree to be the number of nodes on the chain joining the root and node $i$. The graph-theoretic interpretation given in this section and the notion of level numbers will be useful in setting up a mechanism for directly accessing the nonzeros in a row of $H$.

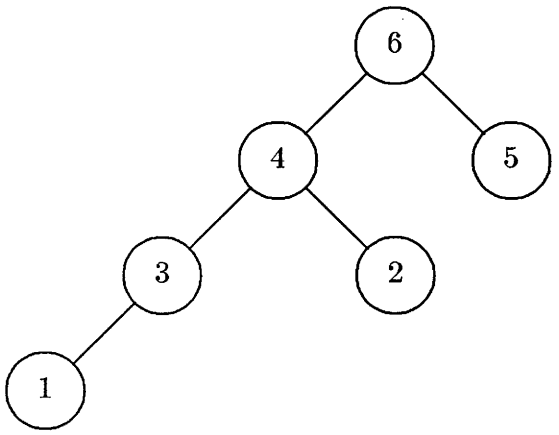
We conclude this section by providing an example to illustrate the notion of elimination trees. Consider the following 8 by 6 matrix

$$A = \begin{pmatrix} x & & & & & x \\ & x & & x & & \\ x & & x & & & \\ & x & & x & & x \\ & & & & x & x \\ x & & & & & x \\ & & x & x & & \\ & x & & x & & \end{pmatrix} .$$

It is easy to verify that the structures of its Householder matrix $H$ and its triangular factor $R$ are given by

$$H = \begin{pmatrix} x & & & & & \\ & x & & & & \\ x & & x & & & \\ & x & & x & & \\ & & & & x & \\ x & & f & f & & x \\ & & x & x & & f \\ & x & & x & & f \end{pmatrix} \qquad \begin{pmatrix} R \\ O \end{pmatrix} = \begin{pmatrix} x & & f & & & x \\ & x & & x & & f \\ & & x & f & & f \\ & & & x & & x \\ & & & & x & x \\ & & & & & x \\ & & & & & \\ & & & & & \\ & & & & & \end{pmatrix} .$$

Here "f" denotes fill-in. The elimination tree associated with $R$ is given below.




As an example, the column subscripts of the nonzeros in row 6 of $H$ are 1, 3, 4, 6. Indeed, (1,3,4,6) is a chain in the elimination tree.

### 3.2. The Data Structure

An efficient way to store the sparse triangular factor matrix $R$ is as follows. The diagonal elements of $R$ are stored in an array *DIAG*. The off-diagonal nonzeros in the matrix $R$ are stored by rows in a main storage vector *RNZ* and an accompanying index vector *XRNZ* is used to point to the beginning of each row in the storage array. The row structure of $R$ is represented using the *compressed subscript* structure which is described below. The compressed subscripts are stored in an integer vector *NZSUB*, and an accompanying index vector *XNZSUB* is used to point to the beginning of the subscript sequence for each row of $R$. This scheme is due to Sherman [14]. Readers are referred to [7] for details. It should be pointed out that the values of $\gamma[i]$ can be retrieved quite readily from the compressed subscript structure. There are efficient algorithms for generating the vector *XRNZ* and the compressed subscript structure. For example, the symbolic factorization algorithm described in [9], which applies directly to $A$, can be simplified for this purpose. Alternatively, symbolic factorization algorithms for sparse symmetric positive definite matrices, such as that described in Chapter 5 of [7], can be applied to $A^TA$. However, experience has shown that the latter approach is usually more time consuming and requires more storage.

The result in Theorem 2.1 and our discussions in Section 3.1 suggest a simple data structure to store the Householder vectors used in triangularizing the given sparse matrix $A$. The diagonal elements of the Householder matrix $H$ are stored in a one-dimensional array *HDIAG*. The off-diagonal nonzeros in the Householder matrix $H$ can be stored by rows in a one-dimensional array *HNZ*. We shall assume that the nonzeros in *HNZ* are stored in ascending order of their column subscripts. This turns out to be easy to arrange.

We need only two additional overhead integer vectors *HFIRST* and *LEVEL*, where *HFIRST*$[i]$ keeps the column subscript of the first nonzero in row $i$ of the matrix $H$ (or equivalently, in row $i$ of the original matrix $A$) and *LEVEL*$[i]$ is the level number $l_i$ of node $i$ in the elimination tree of the triangular factor $R$. Thus, the amount of overhead storage for the Householder matrix is $m+n$ integer locations.

As an example, we have provided in Figure 3.1 the storage representation of the Householder matrix $H$ and the upper triangular factor $R$ given in Section 3.1. The values of $\gamma[*]$ are also given for clarity; however, it should be noted that they are not explicitly stored in the overall storage scheme since they can be obtained directly from the row structure of the triangular factor $R$.

For example, to retrieve the structure of row 6 of $H$, we note that *HFIRST*$[6]=1$ so that the nonzero subscripts are given by

$$HFIRST[6]=1\ ,\ \gamma[1]=3\ ,\ \gamma[3]=4\ ,\ \gamma[4]=6\quad .$$

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $\gamma$ | 3 | 4 | 4 | 6 | 6 | 6 | | | | | |
| *LEVEL* | 4 | 3 | 3 | 2 | 2 | 1 | | | | | |
| *HFIRST* | 1 | 2 | 1 | 2 | 5 | 1 | 3 | 2 | | | |
| *HDIAG* | $h_{11}$ | $h_{22}$ | $h_{33}$ | $h_{44}$ | $h_{55}$ | $h_{66}$ | | | | | |
| *HNZ* | $h_{31}$ | $h_{42}$ | $h_{61}$ | $h_{63}$ | $h_{64}$ | $h_{73}$ | $h_{74}$ | $h_{76}$ | $h_{82}$ | $h_{84}$ | $h_{86}$ |
| *DIAG* | $r_{11}$ | $r_{22}$ | $r_{33}$ | $r_{44}$ | $r_{55}$ | $r_{66}$ | | | | | |
| *RNZ* | $r_{13}$ | $r_{16}$ | $r_{24}$ | $r_{26}$ | $r_{34}$ | $r_{36}$ | $r_{46}$ | $r_{56}$ | | | |
| *XRNZ* | 1 | 3 | 5 | 7 | 8 | 9 | 9 | | | | |
| *NZSUB* | 3 | 6 | 4 | 6 | | | | | | | |
| *XNZSUB* | 1 | 3 | 3 | 4 | 4 | 4 | 5 | | | | |

**Figure 3.1:** Storage representation of the Householder matrix $H$ and the triangular factor $R$.

Sometimes it is necessary to access an individual nonzero in the Householder matrix $H$. An example is when we have to put the nonzeros of the original matrix $A$ into the appropriate locations in the storage array *HNZ* or *RNZ*. If the nonzero is in the upper triangular portion of $A$, we can use the information in the compressed subscript structure, together with the array *XRNZ*, to put the nonzero in the appropriate location in the storage array *RNZ*. However, if the nonzero is in the lower trapezoidal portion of $A$, it is slightly more complicated to find the correct location in the storage array *HNZ* where the nonzero should be, since we do not have the column subscripts explicitly. Suppose for the moment that we have an index array *XHNZ* where *XHNZ*[$i$] points to the beginning of row $i$ of $H$ in the storage array *HNZ*. Assume again that $A^TA$ is irreducible. Thus, the elimination tree for the structure of $R$ has exactly one tree and $n$ is the only root of the tree. Now using the observation that the locations of the nonzeros in row $i$ of $H$ form a chain in the elimination tree (with starting node $f_i$ and terminating node $\min\{i,n\}$), the location for nonzero $a_{ij}$ $(i>j)$ must be *HNZ*[$p$], where

$$p = XHNZ[i] + (LEVEL[f_i] - LEVEL[j]) \quad .$$

Here $LEVEL[f_i]-LEVEL[j]$ is the distance between node $j$ and node $f_i$ (the first nonzero in row $i$) in the elimination tree. Note that we do not really have to include $XHNZ$ in the storage scheme since it can be generated very easily from the information in $HFIRST$ and $LEVEL$, as the following discussion shows. Let $XHNZ[1]=1$. Using Theorem 2.1 and the discussion in Section 3.1 again, it is easy to see that

$$XHNZ[i+1] = \begin{cases} XHNZ[i] + (LEVEL[f_i] - LEVEL[i]) & , \text{ for } 1 \leq i \leq n \quad , \\ XHNZ[i] + (LEVEL[f_i] - LEVEL[n]+1) & , \text{ otherwise} \qquad . \end{cases}$$

When $A^TA$ is reducible, the definition of $XHNZ[i+1]$ will be slightly different for $i=n+1, \cdots, m$.

With the use of this row-oriented storage representation for the Householder matrix $H$, the sequence of Householder transformations can be stored in factored form. To use it effectively, we need to adapt the computation to this data structure. There is a version of the Householder transformation described in [8] which consists of only row operations. That computational scheme is appropriate for the data structure proposed in this paper.

Orthogonal factorization can be used to solve overdetermined systems of linear equations in the least squares sense

$$A\,x \approx b \quad .$$

By storing the sequence of Householder transformations, least squares problems with the same coefficient matrix $A$ but different right hand side vectors $b$ can be solved effectively.

It is important to point out that the data structure described here is also appropriate for computing orthogonal decompositions using Givens rotations. If each plane rotation is represented by a single real number as proposed by Stewart [15], George and Ng [9] have shown that there is always enough room in the main storage vector $HNZ$ to store all the rotations required to reduce $A$ to $R$. In this case, the storage scheme may overestimate the number of rotations actually required.

## 4. A Data Structure for Sparse Gaussian Elimination with Partial Pivoting

We now turn to consider the problem of computing a triangular decomposition of an $n$ by $n$, sparse, nonsymmetric and nonsingular matrix $A$ using Gaussian elimination with partial pivoting. Such a decomposition is useful in many applications, the most important being the solution of linear systems

$$A\,x \;=\; b \quad .$$

Following [9], we denote the triangular decomposition by

$$A \;=\; P_1 M_1 P_2 M_2 \cdots P_{n-1} M_{n-1} U \quad ,$$

where $P_k$ is a permutation matrix that performs the row interchange at step $k$ of the partial pivoting algorithm, $M_k$ is an elementary lower triangular matrix with the $k$-th column containing the multipliers used at step $k$, and $U$ is the final upper triangular matrix.

In [11], George and Ng have provided an implementation of sparse Gaussian elimination with partial pivoting. Their scheme determines from the structure of the Cholesky factor of $A^TA$ a data structure that will accommodate the nonzeros in the factors $M_k$ and $U$ for *all* possible partial pivoting sequences. Such a data structure is said to be *static* since storage allocation and reorganization are not needed during the numerical computation. Needless to say, such a static data structure necessarily overestimates the actual structures of $M_k$ and $U$. However, it has the distinct advantages that the data structure is independent of the numerical values of $A$ (and of the partial pivoting sequence), and overhead in performing storage allocation and reorganization is eliminated during the numerical computation phase. Surprisingly, even with the overestimation of nonzeros in the actual factors $M_k$ and $U$, the approach is demonstrated to be effective and compares quite favorably with existing implementations of sparse Gaussian elimination with some form of pivoting for certain classes of problems. In [9], George and Ng further improve the static scheme by providing a new way to predict the structures of $M_k$. This results in a much tighter data structure for $M_k$ (and occasionally for $U$ as well). This makes the approach which uses static storage schemes even more attractive.

In what follows, we briefly review the new scheme proposed by George and Ng, and show that the structural modifications one has to apply to $A$ are the same as those in computing the orthogonal decomposition of $A$ using Householder transformations. Hence the data structure we have described in Section 3.2 is equally applicable to sparse Gaussian elimination with partial pivoting. We shall assume that the matrix $A$ is irreducible and has a zero-free diagonal.

As in Section 2.2, we partition $A$ into

$$A = \begin{pmatrix} d & v^T \\ u & E \end{pmatrix} ,$$

where $u$ and $v$ are $(n-1)$-vectors. In [9], George and Ng observe that, regardless of the actual row interchange at step 1, the final structure of the pivot row (or row 1 of $U$) must be contained in the structure of the following row vector

$$(d \ , \ \bar{v}^T) = (d \ , \ v^T+u^TE) \quad .$$

Thus, instead of considering $A$ at step 1, they propose to consider applying one step of Gaussian elimination *without* row interchange to the structure of the following modified matrix

$$\begin{aligned} \overline{A} &= \begin{pmatrix} d & \bar{v}^T \\ u & E \end{pmatrix} \\ &= \begin{pmatrix} 1 & 0 \\ u/d & I \end{pmatrix} \begin{pmatrix} d & \bar{v}^T \\ 0 & E-u\bar{v}^T/d \end{pmatrix} = \overline{M}_1\overline{A}_1 \quad . \end{aligned}$$

It is clear that the structure of $\overline{M}_1$ must contain that of $M_1$, and the structure of the matrix $(E-u\bar{v}^T/d)$ should also contain the structure of the matrix that would result from modifying $E$ in Gaussian elimination with partial pivoting, *irrespective of the actual row interchange that occurs.* One of the attractive features of this approach is that the structure of the matrix $(E-u\bar{v}^T/d)$ can be computed readily from the structure of $A$ without knowing the actual pivoting sequence.

If the same idea is applied recursively to the structure of $(E-u\bar{v}^T/d)$, we would obtain a lower triangular matrix, say $\overline{L}$, and an upper triangular matrix, say $\overline{U}$, such that the structure of $U$ is contained in that of $\overline{U}$, and the structure of column $k$ of $M_k$ is contained in that of column $k$ of $\overline{L}$, regardless of the partial pivoting sequence. An efficient algorithm has been given in [9] for determining the structures of $\overline{L}$ and $\overline{U}$. Now from the structures of $\overline{L}$ and $\overline{U}$, a static storage scheme that exploits the sparsity of $\overline{L}$ and $\overline{U}$ can then be set up for computing a triangular decomposition of $A$ using partial pivoting. In [9], the structure of $\overline{L}$ (and hence the structures of $M_k$) is represented by columns using a compressed subscript structure.

It is not hard to see that the structure of $\begin{pmatrix} 1 \\ u/d \end{pmatrix}$ is identical to that of $\begin{pmatrix} \beta \\ w \end{pmatrix}$, the Householder vector at step 1 in Section 2.2, and the structure of $(E-u\bar{v}^T/d)$ (or $\overline{A}_1$) is also identical to that of $(E-wy^T/\beta)$ (or $PA$) in Section 2.2. That is, structurally, $\overline{L}$ and $\overline{U}$ are identical respectively to the Householder matrix $H$ and the upper triangular factor $R$ described in Section 2.2. (Of course, here we assume both $H$ and $R$ are $n$ by $n$.) Thus, it should be clear that the row-oriented storage scheme described in Section 3.2 for $H$ is also applicable for $\overline{L}$, and no compressed subscript structure is required. Experimental results in [9] also provide the number of compressed subscripts for $\overline{L}$.

This will represent the saving in terms of overhead storage if the factor $\bar{L}$ is represented in the same way as the Householder matrix $H$. Typically, it ranges from 1/3 to 1/2 of the subscript storage for $\bar{U}$.

The actual saving in subscript storage for $\bar{L}$ is quite modest, in relative terms. A more important advantage of this storage scheme is that it can be adapted well to the numerical factorization phase with partial pivoting. Since partial pivoting involves row interchanges, the storage scheme described in this paper, which is row-oriented, allows a simple implementation. Furthermore, after the selection of the pivot row, the submatrix update operation

$$E - uv^T/d$$

can be organized as a sequence of row operations, which will be appropriate for this storage scheme. The next section will provide experimental results on the savings in factorization time.

## 5. Implementation of Sparse Gaussian Elimination with Partial Pivoting and Numerical Experiments

### 5.1. Implementation

In this section, we describe in detail an implementation of sparse Gaussian elimination with partial pivoting using the row-oriented storage scheme proposed in this paper. Some numerical experiments and comparisons with the implementation of George and Ng [9] and MA28 [3] will be presented in the next section.

As we have mentioned in the previous section, the data structure required for $\bar{L}$ and $\bar{U}$ is identical to that for the Householder matrix $H$ and the upper triangular factor $R$, and it has been described in detail in Section 3.2. The only difference is that, in the case of Gaussian elimination, we do not need the array *HDIAG* since all the diagonal elements of $M_k$ are equal to 1. This data structure can be set up efficiently using a simplified version of the symbolic factorization algorithm described in [9].

Now let us consider the implementation of the numerical factorization in detail. In order to implement partial pivoting, we have to be able to search for the pivot and, after finding the appropriate pivot, to perform the row interchange and row modifications efficiently. Note that all the *candidate* pivots are below the diagonal and in the same column. That is, they are in $\bar{L}$. However, since the proposed storage scheme for $\bar{L}$ is row-oriented, these candidate pivots are sprinkled in the data structure for $\bar{L}$. This raises two problems. First, how do we identify the *candidate* pivot rows at a given step, and second, how do we locate the candidate pivot in a candidate pivot row in the data structure?

Without loss of generality, let us consider the first step of Gaussian elimination with partial pivoting. First, for $1 \leq k \leq n$, define $Z_k$ as follows.

$$Z_k = \{ i \mid f_i = k \} \quad .$$

(Recall that $f_i$ is the column subscript of the first nonzero in row $i$ of $A$.) That is, $Z_k$ is the set of (indices of) rows such that their first nonzeros are all in column $k$. Clearly, at the first step of Gaussian elimination, the candidate pivot rows are given by the rows in $Z_1$. As George and Ng have shown in [9] and as stated in Theorem 2.1, after step 1, the next step in which these rows (except row 1) will again be used as candidate pivot rows has to be step $\gamma[1]$. Thus, all we need to do is to "merge" $Z_1$ and $Z_{\gamma[1]}$ after step 1; that is,

$$Z_{\gamma[1]} \leftarrow Z_{\gamma[1]} \bigcup Z_1 - \{1\} \quad .$$

This is done at each step. That is, at the end of step $k$, we replace $Z_{\gamma[k]}$ by $Z_{\gamma[k]} \bigcup Z_k - \{k\}$. Thus, $Z_k$ will contain the correct indices of candidate pivot rows at the beginning of step $k$. These sets $Z_1, Z_2, \cdots, Z_n$ can be maintained efficiently using linked lists. In our implementation, we have two integer vectors *LNKBEG* and *ROWLNK* for this purpose. At any time, the elements in $Z_k$ are given by

$$LNKBEG[k] ,$$

$$ROWLNK[LNKBEG[k]] ,$$

$$ROWLNK[ROWLNK[LNKBEG[k]]] ,$$

$$\cdots \quad .$$

If $p$ is the last row in the set, then $ROWLNK[p]=0$. The linked list of candidate pivot rows allows the search for the pivot row to be carried out extremely efficiently. The merging of two sets can also be performed easily. All we have to do is to modify the values of $ROWLNK[p]$ and $LNKBEG[\gamma[k]]$. Note that at the beginning of the elimination process, *LNKBEG* and *ROWLNK* can be generated very easily using the information in the *HFIRST* vector.

It is also easy to locate the candidate pivot in a candidate pivot row in the row-oriented data structure, as the following discussion shows. As in Section 3.2, we shall assume that the nonzeros in a row of $\bar{L}$ are stored in the storage array in ascending order of their column subscripts. Now before the elimination begins, we generate a vector *NXTNZ* of integers, where $NXTNZ[k]$ points to the beginning of row $k$ of $\bar{L}$. The initial content of *NXTNZ* is identical to that of the vector *XHNZ* described in Section 3.2 and can be constructed from *HFIRST* and *LEVEL*. Thus, the candidate pivots at step 1, for example, are found in $HNZ[NXTNZ[i]]$, where $i \in Z_1$. After the first step of elimination, we simply increment $NXTNZ[i]$ by 1, for $i \in Z_1$, to indicate where the nonzero in row $i$ to be used *next* time is. (Alternatively, one can use the

index vector *XHNZ* and the integer vectors *HFIRST* and *LEVEL* to *compute* the location of a certain nonzero. However, experimental results have shown that this approach is much more time consuming than the one described above.)

Finally, it is important to point out that it is possible to exploit completely the sparsity of the pivot row when we perform row modifications. More specifically, if we are at step $k$ of the elimination process and if row $i$ is a row to be modified, then we only have to modify column $j$ of row $i$, where $\bar{u}_{kj}$ is *actually* nonzero. This can be achieved by maintaining a list of locations of the actual nonzeros in row $k$ of $\bar{U}$. In our implementation, we use an integer vector *OFFSET* for this purpose. (Its name comes from the fact that we keep the location of each real nonzero in row $k$ of $\bar{U}$ relative to the first predicted nonzero in that row.) As we shall see in Section 5.2, there is indeed a significant reduction in factorization times in the new implementation where all the zeros in the pivot rows can be exploited.

The discussion above shows that we need 6 overhead integer vectors (*HFIRST*, *LEVEL*, *LNKBEG*, *ROWLNK*, *NXTNZ*, and *OFFSET*) in our implementation using the new row-oriented data structure. In an implementation of the approach due to George and Ng [9], these 6 integer vectors are not required, but we need two floating-point temporary arrays for accumulating intermediate results during the numerical factorization. Furthermore, we also need the index vector into the nonzeros of $\bar{L}$, and the space for the compressed subscript structure for $\bar{L}$ (which includes the space for the compressed subscripts and its index vector).

Let $\phi$ denote the number of compressed subscripts for $\bar{L}$. If we assume that one integer location and one floating-point location have the same number of bits, then the number of locations required by the overhead integer vectors in our implementation that are not in [9] is $6n$. Similarly, the number of locations required by the overhead integer and floating-point vectors in [9] that are not in our implementation is $\phi+4n+2$. Hence the "saving" in storage in our implementation using the row-oriented data structure is

$$\phi + 2 - 2n \quad .$$

Thus, there is a real saving if $\phi$ (the number of compressed subscripts for $\bar{L}$) exceeds $2n$.

The situation is quite different if a floating-point location and an integer location do not have the same number of bits, and if the machine (such as some of the IBM machines) allows the use of "short" and "long" integers. Suppose a floating-point location can hold $\xi_S$ short integers or $\xi_L$ long integers. Typically, $\xi_S=2$ and $\xi_L=1$ if single precision arithmetic is used. When double precision arithmetic has to be used, $\xi_S=4$ and $\xi_L=2$. If $n$ is not too big (that is, if $n$ can be represented as a short integer), then the vectors *HFIRST*, *LEVEL*, *LNKBEG*, *ROWLNK* and *OFFSET* can all be short integer arrays, but *NXTNZ* has to be a long integer array since the number of

predicted nonzeros in $\bar{L}$ may be so large that the pointers cannot be represented as short integers. By the same token, subscripts can be represented by short integers, but index vectors are usually long integer arrays. Hence, the overhead space required in our proposed scheme is

$$5\times\left\lceil\frac{n}{\xi_S}\right\rceil+\left\lceil\frac{n}{\xi_L}\right\rceil \quad ,$$

while the overhead space required in [9] is

$$2\times n+\left\lceil\frac{\phi}{\xi_S}\right\rceil+2\times\left\lceil\frac{(n+1)}{\xi_L}\right\rceil \quad .$$

If single precision arithmetic is used, and if $\xi_S=2$ and $\xi_L=1$, then our scheme always requires approximately $(n+\phi)/2$ fewer floating-point locations than the scheme in [9]. On the other hand, suppose we use double precision arithmetic, and $\xi_S=4$ and $\xi_L=2$. Then the saving in our new proposed scheme will be approximately $n+\phi/4$ floating-point locations.

Backward substitution can be implemented very efficiently since the upper triangular matrix $\bar{U}$ is represented by rows. However, implementation of forward substitution is a bit awkward since we want to access of nonzeros of $\bar{L}$ by columns, but they are stored by rows. However, it turns out that we can again use the vectors *LNKBEG*, *ROWLNK* and *NXTNZ* to facilitate accessing the nonzeros, although the time required to perform forward solution is still larger than those required when the nonzeros are stored by columns.

### 5.2. Numerical experiments

We have implemented sparse Gaussian elimination with partial pivoting using the row-oriented storage scheme proposed in this paper for solving sparse nonsymmetric linear systems

$$A\,x \;=\; b \quad .$$

The objective of this section is to present some numerical experiments to report on its performance, and to compare it with two other approaches for solving the same class of problems. The two other approaches are an implementation of the scheme proposed by George and Ng [9] and the Harwell package MA28 [3]. In the following discussion, the Harwell package will simply be referred to as MA28. The implementation of [9] and the implementation described in Section 5.1 will respectively be referred to as OLD-DS and NEW-DS. Here "DS" stands for "data structure". Following [9], in the implementation of OLD-DS and NEW-DS, we have preprocessed each coefficient matrix $A$ twice. First, we find a row permutation $Q_r$ using an algorithm from [5] so that the reordered matrix $Q_rA$ has a zero-free diagonal. Second, we find a minimum

degree ordering $P_c$ for $A^TA$ and then apply the ordering symmetrically to the columns and rows of $Q_rA$. Experience shows that this tends to minimize the number of nonzeros in $\bar{L}$ and $\bar{U}$ for the matrix $P_cQ_rAP_c^T$. The minimum degree algorithm used in our experiments is due to Liu [12]. Since dense rows of $A$ tend to destroy the sparsity of $A^TA$, and hence the quality of the minimum degree ordering, any rows with more than 100 nonzeros were withheld from $A$ when the minimum degree ordering was determined. The reader is referred to [9] for more details. Thus, we are effectively solving the following permuted system of linear equations

$$\left(P_cQ_rAP_c^T\right)\left(P_cx\right) = \left(P_cQ_rb\right) \quad .$$

It should be noted that the times required to find $Q_r$ and $P_c$, which are usually small compared with those for the numerical factorization, have not been reported here.

All experiments were carried out on a DEC VAX 11/780 computer running UNIX. The programs were written in ANSI standard FORTRAN using single precision floating-point arithmetic, and were compiled using the Berkeley FORTRAN compiler (f77). Throughout this section, storage requirements are expressed in terms of the number of floating-point locations required and execution times are in seconds. (One floating-point location and one integer location have the same number of bits.)

There were twenty-nine test problems used in our experiments. These problems arise from typical scientific and engineering applications, and were kindly provided by Iain Duff, Roger Grimes, John Lewis and Bill Poole [6]. Problems 1-14 were chemical engineering problems, problems 15-20 were supplied by Francois Cachard from Grenoble, and problems 21-29 were mathematical programming problems and survey problems. Problem 29 has the same structure as problem 28, and its numerical values were generated using a uniform random number generator. The characteristics of the test problems are summarized in Table 5.1. The right hand side vector $b$ for each linear system was chosen so that the solution vector consisted of all ones.

In Tables 5.2 and 5.3, we have provided the results of the experiments for OLD-DS and NEW-DS. The corresponding numerical results for MA28 are given in Table 5.4. Note that MA28 uses the technique of *threshold* pivoting which, depending on a so-called *threshold parameter* between 0 and 1, allows the multipliers to be larger than 1 in magnitude [3]. Loosely speaking, when the threshold parameter is small, MA28 tends to choose the pivots to preserve sparsity, and when the threshold parameter is large, MA28 will tend to choose pivots to enhance numerical stability. Thus, if the threshold parameter is equal to 1, the package effectively performs partial pivoting. In our experiments, we have tried using 0.1 and 1.0 as the threshold parameters. In the tables, "error" is the error in the computed solution measured using the $l_\infty$-norm. For MA28, execution time includes the time to factor the coefficient matrix and the time to solve a linear system using the triangular factors.

Based on the results of the numerical experiments in Tables 5.2-5.4, we make the following observations and remarks.

(1) Symbolic factorization in NEW-DS is more efficient than that in OLD-DS, both in storage requirements and execution times. This is expected since the compressed subscript structure for the lower triangular matrix $\bar{L}$ is not required in NEW-DS, and hence need not be generated. The symbolic factorization algorithm for NEW-DS is simpler than that for OLD-DS.

(2) For the numerical computation phase, both NEW-DS and OLD-DS require about the same amount of space. Although a compressed subscript structure is not required for the lower triangular matrix $\bar{L}$ in NEW-DS, extra integer arrays are needed for efficient implementation of numerical factorization (and numerical triangular solution). As a result, the overall storage requirement in NEW-DS may turn out to be larger than that in OLD-DS. This is found to be the case in some instances. In Section 5.1, we have shown that NEW-DS will require less space for numerical factorization than OLD-DS if the number of compressed subscripts for $\bar{L}$ in OLD-DS exceeds $2n$. As we can see from Tables 5.2 and 5.3, NEW-DS indeed requires less space than OLD-DS when the number of compressed subscripts for $\bar{L}$ is large.

(3) The numerical experiments confirm that NEW-DS is more efficient than OLD-DS in terms of factorization times in almost all cases. This is partly due to the fact that the data structure proposed in this paper is well suited for the row-oriented operations. The reduction in execution times is also due to the ability to exploit completely the sparsity of the pivot row, as explained in Section 5.1.

(4) The time required to compute the solution using the triangular factors is in general larger in NEW-DS than in OLD-DS. This is because $\bar{L}$ is now represented by rows in NEW-DS, but the nonzeros have to be accessed by columns in the forward substitution (since we only store the $k$-th column of $M_k$). In OLD-DS, $\bar{L}$ is represented by columns. Thus it is more complicated to access the nonzeros in the lower triangular portion in NEW-DS than in OLD-DS. However, from the numerical results, we should note that the reduction in factorization time is in general much larger than the increase in solution time so that the total time required for the numerical computation phase remains smaller in NEW-DS than in OLD-DS.

Although the increase in solution time in NEW-DS is small compared to the reduction in factorization time, this increase will become significant if we have to solve several linear systems that have the same coefficient matrix $A$ but different right hand side vectors $b$. In this case, it may be worthwhile to postprocess the nonzeros in $\bar{L}$ in order to speed up forward substitutions. Recall that Gaussian elimination of $A$ with partial pivoting may be stated as follows

$$PA = LU \quad ,$$

where

$$P = P_{n-1}P_{n-2} \cdots P_2P_1 \quad ,$$

and

$$L = P(P_1M_1P_2M_2 \cdots P_{n-1}M_{n-1}) \quad .$$

Thus, after the triangular factors $M_1$, $M_2$, $\cdots$, $M_{n-1}$ and $U$ have been computed, we may use the pivoting information ($P_1$, $P_2$, $\cdots$, $P_{n-1}$) to reconfigure the structure of $\bar{L}$ so that we obtain the off-diagonal nonzeros of $L$ stored by rows rather than those of $(M_1+M_2+\cdots+M_{n-1}-(n-1)I)$. Then both forward and backward solutions can be carried out very efficiently. Alternatively, we can generate the compressed subscript structure for the columns of $\bar{L}$ after $M_1$, $M_2$, $\cdots$, $M_{n-1}$ have been computed. Then the off-diagonal nonzeros of $\bar{L}$ can be accessed efficiently by columns in the forward solution. We are currently investigating how these two approaches can be implemented efficiently and how they compare with each other in terms of execution times.

(5) NEW-DS (and OLD-DS occasionally) competes well with MA28 both in terms of storage requirements and execution times. The maximum amount of space required by NEW-DS (which usually occurs in numerical factorization and solution) is, in most cases, about the same as that required by MA28, but the total time required for the symbolic factorization and numerical computation by NEW-DS (excluding the times for finding $Q_r$ and $P_c$) is usually smaller than that required by MA28.

(6) It should be pointed out that the storage requirement reported in Table 5.4 is the *minimal* amount of space required by MA28 to solve a problem. It is well known that if only the minimal amount of space is used, the execution time will be large due to the necessity to reorganize the data structure during the numerical computation. See [4,11] for more on this point. Hence extra storage is usually allocated in an attempt to prevent storage reorganization from occurring too frequently and hence to reduce the execution time required. In Table 5.4, the execution times reported were the times required when extra storage had been provided so that storage reorganization did not occur.

(7) Our experiments show that threshold pivoting with small threshold parameter does not necessarily produce a solution that is as accurate as that obtained with partial pivoting. The results also show that increasing the threshold parameter usually increases both the storage requirement and execution time. Similar observations have been made elsewhere, such as in [11].

(8) For problem 28, it is interesting to note that MA28 performed much better when the threshold parameter was 1.0 than when the parameter was 0.1. This is also the only problem in our test set for which MA28 with partial pivoting (that is, the threshold parameter was 1.0) performed significantly better than both OLD-DS and NEW-DS in terms of execution time. The following discussion provides an explanation. The coefficient matrix $A$ in problem 28 is a banded matrix with bandwidth close to 200. All off-diagonal nonzeros have value 1.0. If we denote the number of off-diagonal nonzeros in row $k$ by $\theta_k$, then the value of $a_{kk}$ is given by

$$a_{kk} = -\max\{\theta_k, 1\} \quad .$$

When we apply Gaussian elimination with partial pivoting to the original coefficient matrix, only three row interchanges occur. This implies that the $k$-th diagonal element at step $k$ usually has the largest magnitude. Now in MA28, if the threshold parameter is set to 1.0, the package will choose pivot elements to enhance numerical stability. That is, in most cases, the diagonal elements are the only suitable pivots. As a result the banded structure of the original matrix $A$ will likely be preserved. However, when the threshold parameter is 0.1, there may be several nonzeros that could be chosen as pivots. Now MA28 chooses the one to preserve sparsity. This may destroy the original banded structure of $A$ and increase the bandwidth. Thus the cost of factoring $A$ will subsequently be increased.

The explanation above is further supported by the results for problem 29. (Recall that problems 28 and 29 have the same structure but different numerical values. The numerical values for problem 29 were random numbers between 0 and 1.) In problem 29, since the numerical values are random numbers, it is unlikely that the $k$-th pivot (in partial pivoting) will lie on the diagonal. Hence, more row interchanges are likely to occur in this case when partial pivoting is employed. The banded structure will therefore be destroyed and more time will probably be needed to search for the appropriate pivots. This is confirmed by the numerical results. In fact, substantially more time was needed to solve the problem when the threshold parameter was 1.0 than when it was 0.1.

Note that, for problem 29, the factorization time is larger in NEW-DS than in OLD-DS. In fact, problem 29, which is a problem with random numerical values, is the only problem in our test set where NEW-DS behaves anomalously in terms of factorization time. This is apparently related to the number of row interchanges performed. Other experiments using matrices which differ from that in problem 29 only in (random) numerical values show that NEW-DS can be faster than OLD-DS in the numerical factorization phase. Also, it should be pointed out that random test problems are usually not capable of revealing the behavior of algorithms on practical problems.

(9) The results for problems 28 and 29 also illustrate the fact that data structures for OLD-DS and NEW-DS are insensitive to numerical values and pivoting sequences.

Recall that the structure of $\bar{L}$ and $\bar{U}$ depends on the structure of $A$ (and $P_c$). Since the structure of $A$ may be quite different from that of $A^T$, it may sometimes be more beneficial to work with $A^T$ than $A$. More specifically, one may alternatively use Gaussian elimination with partial pivoting to compute a triangular factorization of $\hat{P}_c\hat{Q}_r A^T\hat{P}_c^T$, where $\hat{Q}_r$ is chosen so that $\hat{Q}_r A^T$ has a zero-free diagonal, and $\hat{P}_c$ is a minimum degree ordering for $AA^T$. Then instead of solving the linear system $Ax=b$, one may solve the following system

$$\left(\hat{P}_c A\hat{Q}_r^T\hat{P}_c^T\right)\left(\hat{P}_c\hat{Q}_r x\right) = \left(\hat{P}_c b\right) \quad .$$

The important observation is that one can decide whether to use $A$ or $A^T$ by first applying a symbolic factorization to both $A$ and $A^T$ to determine how much space is needed for the numerical computation in each case. This is quite reasonable since the symbolic factorization algorithm for either OLD-DS or NEW-DS is very efficient. A similar suggestion has been made in [9].

## 6. Conclusion

In this paper, we have proposed a data structure for sparse orthogonal decomposition using Householder transformations and sparse Gaussian elimination with partial pivoting. The new data structure is row-oriented and is suitable for the two problems since the numerical computations can be described as row operations. Also, as in [9], the data structure is static since storage allocation and reorganization are not necessary during the numerical computation. This allows the numerical computations to be implemented efficiently. Another advantage is that the data structure is independent of the numerical values of $A$ (and the pivoting sequence in the case of sparse Gaussian elimination with partial pivoting), hence the data structure can be used to solve problems in which the structure of $A$ is fixed, but the numerical values vary. Numerical experiments are presented to show that an implementation of sparse Gaussian elimination with partial pivoting using the new row-oriented data structure indeed competes well with a static storage scheme proposed by George and Ng [9] in terms of storage requirements, and our new scheme is superior in terms of factorization times. We have also demonstrated that the implementation using the new static data structure is comparable to one in which storage allocation and reorganization are performed during the numerical computation, so that all sparsity is explicitly exploited. In some cases, our static implementation is more efficient than the dynamic implementation.

| problem | order | number of nonzeros | number of nonzeros per row | | number of nonzeros per column | | comment |
|---|---|---|---|---|---|---|---|
| | | | minimum | maximum | minimum | maximum | |
| 1 | 167 | 507 | 1 | 9 | 1 | 19 | rigorous model of a chemical stage |
| 2 | 381 | 2157 | 1 | 25 | 1 | 50 | multiply fed column, 24 components |
| 3 | 67 | 294 | 1 | 6 | 2 | 10 | cavett problem with 5 components |
| 4 | 655 | 2854 | 1 | 12 | 1 | 35 | 16 stage column section, some stages simplified |
| 5 | 479 | 1910 | 1 | 12 | 1 | 35 | 8 stage column section, all sections rigorous |
| 6 | 497 | 1727 | 1 | 28 | 1 | 55 | rigorous flash unit with recycling |
| 7 | 1505 | 5445 | 1 | 12 | 1 | 27 | 11 stage column section, all sections rigorous |
| 8 | 2021 | 7353 | 1 | 12 | 1 | 26 | 15 stage column sections, all sections rigorous |
| 9 | 989 | 3537 | 1 | 12 | 1 | 26 | 7 stage column section, all sections rigorous |
| 10 | 207 | 572 | 1 | 8 | 1 | 5 | heat exchange network |
| 11 | 59 | 312 | 2 | 7 | 1 | 12 | cavett's process |
| 12 | 137 | 411 | 1 | 8 | 1 | 8 | ethylene plant model |
| 13 | 425 | 1339 | 1 | 10 | 1 | 10 | nitric acid plant model |
| 14 | 225 | 1308 | 1 | 12 | 1 | 23 | hydrocarbon separation problem |
| 15 | 115 | 421 | 2 | 7 | 2 | 7 | unsymmetric matrix from Grenoble |
| 16 | 185 | 1005 | 3 | 7 | 3 | 7 | unsymmetric matrix from Grenoble |
| 17 | 216 | 876 | 2 | 5 | 2 | 5 | unsymmetric matrix from Grenoble |
| 18 | 343 | 1435 | 2 | 5 | 2 | 5 | unsymmetric matrix from Grenoble |
| 19 | 512 | 2192 | 2 | 5 | 2 | 5 | unsymmetric matrix from Grenoble |
| 20 | 1107 | 5664 | 2 | 7 | 2 | 6 | unsymmetric matrix from Grenoble |
| 21 | 113 | 655 | 1 | 20 | 1 | 27 | unsymmetric pattern supplied by Morven Gentleman |
| 22 | 199 | 701 | 1 | 6 | 2 | 9 | unsymmetric pattern of order 199 given by Willoughby |
| 23 | 130 | 1282 | 1 | 124 | 1 | 124 | unsymmetric matrix from laser problem given by A.R. Curtis |
| 24 | 663 | 1712 | 1 | 426 | 1 | 4 | unsymmetric basis from LP problem (Shell) |
| 25 | 363 | 3279 | 1 | 33 | 1 | 34 | unsymmetric basis from LP problem (Stair) |
| 26 | 822 | 4841 | 1 | 304 | 1 | 21 | unsymmetric basis from LP problem (BP) |
| 27 | 541 | 4285 | 1 | 11 | 5 | 541 | unsymmetric facsimile convergence matrix |
| 28 | 991 | 6027 | 1 | 16 | 1 | 16 | unsymmetric matrix from Philips Ltd., supplied by J.P. Whelan |
| 29 | 991 | 6027 | 1 | 16 | 1 | 16 | same as problem 28, but numerical values are random numbers |

**Table 5.1:** Characteristics of test matrices.

| problem | symbolic factorization | | | | | | numerical solution | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | storage | time | number of subscripts | | number of nonzeros | | storage | fact time | solve time | error |
| | | | lower | upper | lower | upper | | | | |
| 1 | 3720 | 0.200 | 173 | 524 | 349 | 830 | 3717 | 0.217 | 0.033 | 2.200(−1) |
| 2 | 17897 | 1.033 | 2462 | 6544 | 8780 | 17539 | 39520 | 12.800 | 0.567 | 1.766(−3) |
| 3 | 1953 | 0.100 | 171 | 385 | 426 | 843 | 2566 | 0.250 | 0.033 | 2.742(−6) |
| 4 | 20140 | 1.100 | 1579 | 4988 | 5358 | 11794 | 30928 | 6.483 | 0.417 | 4.292(−2) |
| 5 | 13771 | 0.800 | 1158 | 3040 | 3698 | 7203 | 20372 | 4.050 | 0.283 | 1.176(−1) |
| 6 | 12082 | 0.750 | 558 | 2101 | 1447 | 5748 | 15325 | 1.183 | 0.217 | 1.860(−2) |
| 7 | 38542 | 2.100 | 2464 | 7123 | 4688 | 12221 | 43055 | 3.633 | 0.550 | 2.629(−1) |
| 8 | 52017 | 2.800 | 3328 | 9726 | 6059 | 16347 | 57695 | 4.700 | 0.733 | 1.429(−1) |
| 9 | 25026 | 1.317 | 1523 | 4556 | 2771 | 7565 | 27298 | 2.000 | 0.350 | 6.625(−2) |
| 10 | 4404 | 0.217 | 180 | 591 | 321 | 845 | 4218 | 0.200 | 0.033 | 5.735(−4) |
| 11 | 1570 | 0.083 | 81 | 152 | 190 | 458 | 1534 | 0.100 | 0.017 | 2.157(−3) |
| 12 | 2810 | 0.133 | 48 | 291 | 61 | 309 | 2220 | 0.067 | 0.017 | 4.768(−7) |
| 13 | 10640 | 0.517 | 664 | 2193 | 1531 | 3561 | 12628 | 0.633 | 0.183 | 7.749(−6) |
| 14 | 6645 | 0.367 | 347 | 977 | 739 | 1964 | 6506 | 0.417 | 0.083 | 6.065(−4) |
| 15 | 3393 | 0.183 | 372 | 794 | 904 | 1697 | 5036 | 0.783 | 0.050 | 4.530(−6) |
| 16 | 7106 | 0.367 | 925 | 1946 | 2711 | 5083 | 12704 | 2.900 | 0.183 | 3.132(−3) |
| 17 | 7084 | 0.350 | 880 | 1855 | 3184 | 5828 | 14127 | 3.867 | 0.217 | 2.265(−6) |
| 18 | 12105 | 0.633 | 1784 | 3330 | 7573 | 13401 | 29865 | 11.367 | 0.483 | 7.987(−6) |
| 19 | 18911 | 1.167 | 2890 | 5488 | 15735 | 27923 | 57672 | 34.283 | 0.983 | 7.749(−6) |
| 20 | 49798 | 3.200 | 9794 | 15387 | 78081 | 137358 | 252801 | 269.400 | 4.600 | 9.675(−2) |
| 21 | 3351 | 0.183 | 111 | 569 | 313 | 1322 | 3562 | 0.433 | 0.050 | 1.629(−4) |
| 22 | 5393 | 0.267 | 440 | 1158 | 960 | 2194 | 6945 | 0.917 | 0.083 | 1.070(−3) |
| 23 | 4661 | 0.567 | 272 | 260 | 2720 | 7763 | 12449 | 2.517 | 0.217 | 3.906(−2) |
| 24 | 13168 | 1.017 | 429 | 1354 | 617 | 2216 | 11913 | 0.350 | 0.150 | 0.000(+0) |
| 25 | 15337 | 0.783 | 800 | 3618 | 2562 | 6972 | 17949 | 2.767 | 0.233 | 2.431(−3) |
| 26 | 27729 | 1.717 | 1875 | 6303 | 6889 | 17995 | 42108 | 8.183 | 0.633 | 4.913(−3) |
| 27 | 23868 | 1.250 | 2167 | 6634 | 7610 | 16600 | 38966 | 10.950 | 0.567 | 5.875(−2) |
| 28 | 46542 | 2.700 | 7681 | 14910 | 70772 | 119646 | 223914 | 334.400 | 4.233 | 1.383(−5) |
| 29 | 46542 | 2.667 | 7681 | 14910 | 70772 | 119646 | 223914 | 373.483 | 3.900 | 2.006(−4) |

**Table 5.2:** OLD-DS using $A$.

| problem | symbolic factorization | | | | | numerical solution | | | |
|---|---|---|---|---|---|---|---|---|---|
| | storage | time | number of subscripts | number of nonzeros | | storage | fact time | solve time | error |
| | | | upper | lower | upper | | | | |
| 1 | 2704 | 0.133 | 524 | 349 | 830 | 3876 | 0.100 | 0.067 | 1.603(−1) |
| 2 | 12514 | 0.717 | 6544 | 8780 | 17539 | 37818 | 8.167 | 0.683 | 8.606(−4) |
| 3 | 1352 | 0.083 | 385 | 426 | 843 | 2527 | 0.150 | 0.050 | 2.742(−6) |
| 4 | 14395 | 0.783 | 4988 | 5358 | 11794 | 30657 | 3.233 | 0.517 | 4.348(−2) |
| 5 | 9743 | 0.533 | 3040 | 3698 | 7203 | 20170 | 2.150 | 0.367 | 6.593(−2) |
| 6 | 8801 | 0.500 | 2101 | 1447 | 5748 | 15759 | 0.533 | 0.250 | 1.860(−2) |
| 7 | 27621 | 1.467 | 7123 | 4688 | 12221 | 43599 | 1.717 | 0.733 | 2.655(−1) |
| 8 | 37292 | 2.067 | 9726 | 6059 | 16347 | 58407 | 2.233 | 0.917 | 1.336(−1) |
| 9 | 17986 | 0.967 | 4556 | 2771 | 7565 | 27751 | 0.983 | 0.433 | 6.250(−2) |
| 10 | 3236 | 0.133 | 591 | 321 | 845 | 4450 | 0.100 | 0.067 | 2.558(−3) |
| 11 | 1057 | 0.067 | 152 | 190 | 458 | 1569 | 0.067 | 0.033 | 2.157(−3) |
| 12 | 2075 | 0.083 | 291 | 61 | 309 | 2444 | 0.033 | 0.033 | 4.768(−7) |
| 13 | 7785 | 0.367 | 2193 | 1531 | 3561 | 12812 | 0.267 | 0.183 | 7.987(−6) |
| 14 | 4538 | 0.250 | 977 | 739 | 1964 | 6607 | 0.267 | 0.117 | 6.065(−4) |
| 15 | 2368 | 0.133 | 794 | 904 | 1697 | 4892 | 0.583 | 0.100 | 4.530(−6) |
| 16 | 4804 | 0.267 | 1946 | 2711 | 5083 | 12147 | 2.100 | 0.233 | 2.789(−3) |
| 17 | 4894 | 0.283 | 1855 | 3184 | 5828 | 13677 | 2.600 | 0.250 | 4.530(−6) |
| 18 | 8198 | 0.483 | 3330 | 7573 | 13401 | 28765 | 8.933 | 0.583 | 5.245(−6) |
| 19 | 12803 | 0.750 | 5488 | 15735 | 27923 | 55804 | 25.850 | 1.183 | 8.464(−6) |
| 20 | 32124 | 2.400 | 15387 | 78081 | 137358 | 245219 | 213.983 | 5.817 | 9.942(−1) |
| 21 | 2357 | 0.150 | 569 | 313 | 1322 | 3675 | 0.233 | 0.050 | 1.629(−4) |
| 22 | 3852 | 0.183 | 1158 | 960 | 2194 | 6901 | 0.550 | 0.117 | 1.070(−3) |
| 23 | 2845 | 0.583 | 260 | 2720 | 7763 | 12435 | 1.967 | 0.283 | 3.906(−2) |
| 24 | 9699 | 0.783 | 1354 | 617 | 2216 | 12808 | 0.183 | 0.200 | 0.000(+0) |
| 25 | 10530 | 0.633 | 3618 | 2562 | 6972 | 17873 | 1.683 | 0.317 | 1.824(−3) |
| 26 | 19367 | 1.400 | 6303 | 6889 | 17995 | 41875 | 3.567 | 0.717 | 4.997(−3) |
| 27 | 16332 | 0.917 | 6634 | 7610 | 16600 | 37879 | 7.650 | 0.750 | 5.875(−2) |
| 28 | 30850 | 2.100 | 14910 | 70772 | 119646 | 218213 | 292.367 | 5.183 | 1.383(−5) |
| 29 | 30850 | 2.033 | 14910 | 70772 | 119646 | 218213 | 381.683 | 5.150 | 2.006(−4) |

**Table 5.3:** NEW-DS using $A$.

| problem | threshold parameter = 0.1 | | | threshold parameter = 1.0 | | |
|---|---|---|---|---|---|---|
| | storage | time | error | storage | time | error |
| 1 | 4087 | 0.517 | 8.572(−3) | 4119 | 0.667 | 8.572(−3) |
| 2 | 16353 | 6.833 | 7.697(−4) | 16459 | 7.933 | 8.597(−4) |
| 3 | 2412 | 0.500 | 6.080(−6) | 2541 | 0.550 | 3.278(−6) |
| 4 | 23526 | 10.933 | 3.324(−2) | 25866 | 25.417 | 7.245(−3) |
| 5 | 15058 | 4.433 | 2.642(−2) | 16182 | 8.500 | 1.372(−2) |
| 6 | 12795 | 1.683 | 4.547(−3) | 12907 | 2.017 | 4.917(−3) |
| 7 | 42263 | 26.517 | 1.551(−1) | 43947 | 46.633 | 1.378(−1) |
| 8 | 56335 | 46.167 | 7.695(−2) | 58745 | 87.600 | 1.697(−1) |
| 9 | 27001 | 12.367 | 1.253(−1) | 27913 | 20.933 | 1.814(−1) |
| 10 | 4706 | 0.533 | 1.422(−3) | 4724 | 0.533 | 1.377(−3) |
| 11 | 2056 | 0.350 | 1.009(−4) | 2056 | 0.333 | 4.926(−4) |
| 12 | 3199 | 0.267 | 0.000(+0) | 3199 | 0.283 | 4.768(−7) |
| 13 | 10597 | 1.750 | 5.960(−7) | 10657 | 1.783 | 2.384(−7) |
| 14 | 7384 | 1.133 | 7.439(−5) | 7384 | 1.133 | 2.861(−5) |
| 15 | 4281 | 1.050 | 8.357(−5) | 4437 | 1.250 | 1.192(−6) |
| 16 | 12995 | 6.717 | 4.076(−1) | 12947 | 5.833 | 2.420(−3) |
| 17 | 12145 | 3.800 | 3.731(−5) | 12727 | 5.983 | 3.695(−6) |
| 18 | 23471 | 9.900 | 1.270(−4) | 26449 | 25.017 | 8.225(−6) |
| 19 | 42515 | 49.117 | 1.376(−2) | 55184 | 67.350 | 6.676(−6) |
| 20 | 128074 | 223.917 | 3.141(+2) | 125742 | 222.517 | 6.891(−1) |
| 21 | 3905 | 0.817 | 8.571(−5) | 4243 | 1.317 | 5.198(−5) |
| 22 | 6525 | 1.550 | 1.486(−3) | 7689 | 3.850 | 9.044(−4) |
| 23 | 5934 | 1.517 | 9.766(−2) | 5934 | 1.517 | 9.766(−2) |
| 24 | 14418 | 1.200 | 0.000(+0) | 14418 | 1.200 | 0.000(+0) |
| 25 | 17297 | 5.417 | 1.636(−4) | 17829 | 6.083 | 1.359(−5) |
| 26 | 27817 | 5.083 | 1.361(−3) | 28351 | 5.783 | 1.225(−3) |
| 27 | 45191 | 36.317 | 7.265(−3) | 42824 | 41.517 | 7.095(−3) |
| 28 | 176755 | 409.733 | 6.399(−2) | 145345 | 149.817 | 4.649(−6) |
| 29 | 168444 | 445.950 | 1.078(−2) | 278690 | 2492.400 | 3.242(−4) |

**Table 5.4:** MA28 using $A$.

## 7. References

[1] T.F. COLEMAN, A. EDENBRANDT, AND J.R. GILBERT, "Predicting fill for sparse orthogonal factorization", Technical Report 83-578, Dept. of Computer Science, Cornell University (1983).

[2] J.J. DONGARRA, C.B. MOLER, J.R. BUNCH, AND G.W. STEWART, *LINPACK users' guide,* SIAM, Philadelphia (1980).

[3] I.S. DUFF, "MA28 - A set of FORTRAN subroutines for sparse unsymmetric linear equations", Tech. Report AERE R-8730, Harwell (1977).

[4] I.S. DUFF, "Practical comparisons of codes for the solution of sparse linear systems", in *Sparse Matrix Proceedings 1978,* ed. I.S. Duff and G.W. Stewart, SIAM Press, pp. 107-134 (1979).

[5] I.S. DUFF, "Algorithm 575. Permutations for a zero-free diagonal", *ACM Trans. on Math. Software* **7**, pp. 387-390 (1981).

[6] I.S. DUFF, R.G. GRIMES, J.G. LEWIS, AND W.G. POOLE, JR., "Sparse matrix test problems", *ACM SIGNUM Newsletter* **17**(2), p. 22 (1982).

[7] J.A. GEORGE AND J. W-H. LIU, *Computer Solution of Large Sparse Positive Definite Systems,* Prentice-Hall, Inc., Englewood Cliffs, New Jersey (1981).

[8] J.A. GEORGE AND J.W-H. LIU, "Householder reflections versus Givens rotations in sparse orthogonal decomposition", *Linear Algebra and its Appl.*, (1985). (To appear.)

[9] J.A. GEORGE AND E.G-Y. NG, "Symbolic factorization for sparse Gaussian elimination with partial pivoting", Research Report CS-84-43, Department of Computer Science, University of Waterloo (1984).

[10] J.A. GEORGE AND E.G-Y. NG, "Orthogonal reduction of sparse matrices to upper triangular form using Householder transformations", Research Report CS-84-05, Department of Computer Science, University of Waterloo (1984). (To appear in SIAM J. Sci. Stat. Comput.)

[11] J.A. GEORGE AND E.G-Y. NG, "An implementation of Gaussian elimination with partial pivoting for sparse systems", *SIAM J. Sci. Stat. Comput.* **6**, pp. 390-409 (1985).

[12] J.W-H. LIU, "Modification of the minimum degree algorithm by multiple elimination", *ACM Trans. on Math. Software* **11**(1985). (To appear.)

[13] J.W-H. LIU, "A compact row storage scheme for sparse Cholesky factors using elimination trees", *ACM Trans. on Math. Software*, (1985). (To appear.)

[14] A.H. SHERMAN, "On the efficient solution of sparse systems of linear and nonlinear equations.", Research Report #46, Dept. of Computer Science, Yale University (1975).

[15] G.W. STEWART, "The economical storage of plane rotations", *Numer. Math.* **25**, pp. 137-138 (1976).