SOLUTION OF SPARSE LINEAR LEAST SQUARES PROBLEMS USING GIVENS ROTATIONS*

by

Alan George[1]

and

Michael T. Heath[2]

Research Report CS-80-13

March 1980

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

[2] Mathematics and Statistics Research Department
Computer Sciences Division
Union Carbide Corporation, Nuclear Division
Oak Ridge, Tennessee, U.S.A. 37830

* Research sponsored by the Applied Mathematical Sciences Research Program, Office of Energy Research, U.S. Department of Energy under contract W-7405-eng-26 with the Union Carbide Corporation. Research of the first author also supported in part by Canadian Natural Science and Engineering Research Council grant A8111.

ABSTRACT

We describe a direct method for solving sparse linear least squares problems. The storage required for the method is no more than that needed for the conventional normal equations approach. However, the normal equations are not computed; orthogonal transformations are applied to the coefficient matrix, thus avoiding the potential numerical instability associated with computing the normal equations. Our approach allows full exploitation of sparsity, and permits the use of a fixed (static) data structure during the numerical computation. Finally, the method processes the coefficient matrix one row at a time, allowing for the convenient use of auxiliary storage and updating operations.

§1. Introduction

In this article we present a numerically stable method for solving the linear least squares problem

(1.1) $$\min_x \; ||Ax-b||_2 \;,$$

where A is m by n, $m \geq n$, and is assumed to have full column rank. Our method has been designed to deal effectively with (1.1) when m and n are large and A^TA is sparse.

The classical approach to solving this problem, and one which is still used in many contexts today, is via the system of normal equations

(1.2) $$Bx = \bar{b} \;,$$

where

(1.3) $$B = A^TA \quad \text{and} \quad \bar{b} = A^Tb \;.$$

The n by n symmetric positive definite matrix B is factored using Cholesky's method into R^TR, where R is upper triangular, and then x is computed by solving the two triangular systems $R^Ty = \bar{b}$ and $Rx = y$.

The following features make the normal equations approach attractive:

i. **The Cholesky algorithm does not require pivoting for stability** so that the ordering for B (i.e. column ordering of A) can be chosen based on sparsity considerations alone. Furthermore, there exists well developed software for exploiting sparsity in such linear systems. A good ordering can be determined in advance of any numerical computation, allowing use of a static data structure.

ii. The row ordering of A is irrelevant so that the rows of A can be processed sequentially from an auxiliary input file in arbitrary order. Thus only one row of A need be represented in fast storage at any given time.

iii. Explicit computation of the Cholesky factor R provides convenient access to the important statistical information contained in the unscaled covariance matrix $(A^TA)^{-1} = (R^TR)^{-1}$.

Unfortunately, the normal equations method also has several drawbacks:

i. Unless extended precision is employed, which would be costly both in space and time, there may be a serious loss of information in explicitly forming and processing A^TA and A^Tb.

ii. The condition number of B is the square of the condition number of A, so that the accuracy of the computed solution to the system (1.2) may be questionable, especially if A itself is already poorly conditioned.

iii. The sparsity of A does not necessarily imply that B will be comparably sparse. Indeed, if A has a full row then B is full.

Our aim is to provide a method which retains the advantages of the normal equations approach without having its disadvantages. Of course, this is not accomplished without some cost, and in this case the method we propose requires more arithmetic operations than the normal equations.

Perhaps the best known stable alternative to the normal equations is the orthogonal factorization approach (see, for example, [10]). An orthogonal matrix Q is computed which reduces A to upper trapezoidal form, so $||Ax-b||_2$ is transformed to

$$\left\| \begin{pmatrix} R \\ 0 \end{pmatrix} x - \begin{pmatrix} y \\ z \end{pmatrix} \right\|_2 \tag{1.4}$$

where

$$QA = \begin{pmatrix} R \\ 0 \end{pmatrix}, \quad Qb = \begin{pmatrix} y \\ z \end{pmatrix}, \tag{1.5}$$

and R is upper triangular. The application of Q does not change the two-norm, so the solution to (1.1) is obtained by solving the triangular system $Rx = y$. The matrix Q is usually obtained as a product of Householder or Givens transformations or by Gram-Schmidt orthogonalization.

There has been a reluctance to use this method for sparse problems, due in part to the generally accepted belief that orthogonal transformations cause an unacceptable amount of fill-in. For example, the application of Householder transformations or Gram-Schmidt orthogonalization can cause severe "intermediate" fill-in. Eventually this transient fill-in is itself reduced to zeros, but the phenomenon can cause minimum storage requirements to exceed greatly that which is ultimately required for R. In addition, the conventional use of Householder transformations or Gram-Schmidt orthogonalization requires access to all columns of the unreduced part of A during the computation.

The use of Givens rotations is much more attractive. The rows of A can be processed one by one, gradually creating R. Thus, no "intermediate swell" outside the working row need occur, and A can be accessed in the manner that it usually arises naturally or any other convenient or desirable order.

Although these advantages have been recognized, the use of Givens transformations for sparse linear least squares problems has not

gained wide acceptance. We conjecture that the main reason is that effective methods for permuting A so that R remains sparse have not been available. The method we propose in this paper provides a mechanism for solving this problem.

Several other methods for sparse least squares problems are surveyed in Björck [1], Duff and Reid [3], and Gill and Murray [8]. These include the elimination method of Peters and Wilkinson [16] and the augmented matrix method of Hachtel [11]. Although they can be quite effective in exploiting sparsity in many contexts, both of these methods require access to the whole matrix A at some stage of the computation. Moreover, both methods involve row and column pivoting for stability and therefore require dynamic data structures. Finally, neither method provides the Cholesky factor R explicitly and therefore they are not directly compatible with existing normal equations methodology.

§2. The Proposed Method

We begin our description by noting that the unique Cholesky factor of B in (1.3) and the R in (1.5) are the same, apart from possible sign differences in the rows. This is made apparent by the following.

$$A^TQ^TQA = (R^T\ 0)\begin{pmatrix} R \\ 0 \end{pmatrix} = R^TR \qquad \text{(from 1.5)}$$

$$= A^TA = B\ . \qquad \text{(from 1.3)} \tag{2.1}$$

Recall that R is independent of the ordering of the rows of A. This is obvious from (2.1) since Q could be simply a permutation matrix.

With these observations, we present the basic steps of our method, upon which we then elaborate.

1. Determine the structure (not the numerical values) of $B = A^TA$.
2. Apply an ordering algorithm to B, yielding a permutation matrix P such that $\tilde{B} = P^TBP$ has a sparse Cholesky factor $\tilde{R}$. Note that $\tilde{B} = P^TA^TAP = \tilde{A}^T\tilde{A}$.
3. Apply a "symbolic factorization" algorithm to $\tilde{B}$, generating a row-oriented data structure for $\tilde{R}$ [6].
4. Compute $\tilde{R}$ by processing the rows of $\tilde{A}$ one by one, using Givens rotations. Apply the same transformations to b. Thus, we have

$$Q\tilde{A} = \begin{pmatrix} \tilde{R} \\ 0 \end{pmatrix} \text{ and } Qb = \begin{pmatrix} y \\ z \end{pmatrix}.$$

5. Solve $\tilde{R}x = y$.

In Steps 1 through 3 we are exploiting well developed techniques for solving sparse positive definite systems. In Step 1, we determine the structure of A^TA, which of course need not involve any floating point operations and hence no rounding errors. We then apply an ordering

algorithm to the output of Step 1, providing a symmetric reordering of A^TA, which corresponds to a column permutation of A. We then apply an algorithm to $\tilde{B}$ which determines the structure of its Cholesky factor $\tilde{R}$, and sets up a data structure in which to store it.

These first three steps correspond exactly to what would be done in solving (1.2) using current sparse matrix techniques. The next step would then be to place the numerical values of $\tilde{B}$ in the data structure for $\tilde{R}$ and compute $\tilde{R}$ in place using Cholesky's method. However, we wish to avoid the explicit computation of $\tilde{B}$, so our method diverges from the normal equations approach at this point. Steps 1-3 will have provided us with a good column ordering for A, so that $\tilde{R}$ will be sparse, along with an efficient data structure which exploits that sparsity. Our Step 4 involves the application of Givens rotations to the rows of $\tilde{A} = AP$, one at a time, gradually building up $\tilde{R}$. These rotations are applied simultaneously to b, and when all rows of $\tilde{A}$ have been processed ("rotated into $\tilde{R}$"), Step 5 is executed to obtain x.

The advantages of our method are as follows:

1. Unlike the normal equations approach, we avoid the potential instability due to the explicit computation of B.
2. Storage requirements are essentially the same as that for the normal equations approach. Any sparsity exploitation available to the normal equations approach may also be used with our approach.
3. The use of row operations facilitates updating operations and the use of auxiliary storage.

4. The numerical computation is performed using a fixed (static) data structure; dynamic storage allocation to accommodate fill-in (transient or otherwise) is not necessary. The importance of this property of our method is difficult to overemphasize. It allows the use of a very efficient data structure for $\tilde{R}$ and efficient numerical computation.
5. Since the order in which the rows of A are processed in step 4 does not affect numerical stability or the ultimate sparsity of $\tilde{R}$, this order can be exploited, if desired, to reduce the total arithmetic operation count for the algorithm. Such strategies will be discussed in Section 4.

As was remarked earlier B, and hence R, may be quite full if A has one or more relatively full rows. In order to avoid this severe fill-in, such rows may be skipped in the initial processing and then be incorporated into the solution later by means of the updating procedure outlined in Section 5.

Our exploitation of sparsity in $\tilde{R}$ and its analysis in Section 3 is in a sense pessimistic, because the structure of $\tilde{R}$ obtained from step 3 of our algorithm results from a simulation of Cholesky's method assuming no cancellation occurs. Thus, $\tilde{R}^T + \tilde{R}$ is assumed to be at least as full as $\tilde{B}$. When $\tilde{R}$ is computed this way, such an assumption is entirely sensible since it is difficult to anticipate when such cancellation would occur. However, when $\tilde{R}$ is computed using Givens rotations, some such cancellation may be automatically identified, because the numbers which cancel during the application of the Cholesky algorithm to $\tilde{B}$ are never computed when using the orthogonal reduction method. See the example provided by Bjorck [1, p.181].

§3. Implementation Details

Steps 1-3 of our method involve standard sparse matrix techniques for solving sparse positive definite systems of equations, about which there is abundant literature and high quality software [5,17]. We have borrowed the majority of our implementation of Steps 1, 2, 3 and 5 from the software package SPARSPAK; a description of the algorithms and data structures used in the package can be found in [5]. Thus, our main purpose here is to describe and justify the implementation of Step 4 of our method.

For completeness, we include a brief description of the data structure chosen for $\tilde{R}$, an example of which is given in Figure 3.1. The storage scheme has an array RNZ which contains the nonzero off-diagonal entries of $\tilde{R}$, row by row, along with a pointer array XRNZ of length n which indicates where the elements of each row begin in RNZ. The diagonal elements of $\tilde{R}$ are stored in a separate array DIAG.

The column subscripts of the elements of $\tilde{R}$ in RNZ are stored in the array NZSUB, and the beginning position in NZSUB where the subscripts of row i of $\tilde{R}$ reside is given by XNZSUB(i). Note that NZSUB is <u>not</u> in general as long as RNZ; it has been "telescoped" in those cases where the leading column subscripts of a row form a final subsequence of those of a previous row. This clever storage scheme is due to Sherman [17].

$$\tilde{B} = \begin{bmatrix} b_{11} & b_{12} & & b_{14} & & & \\ & b_{22} & & & & & \\ & & b_{33} & & b_{35} & b_{36} & \\ & & & b_{44} & b_{45} & & \\ & \text{symmetric} & & & b_{55} & & b_{57} \\ & & & & & b_{66} & b_{67} \\ & & & & & & b_{77} \end{bmatrix} \qquad \tilde{R} = \begin{bmatrix} r_{11} & r_{12} & & r_{14} & & & \\ & r_{22} & & r_{24} & & & \\ & & r_{33} & & r_{35} & r_{36} & \\ & & & r_{44} & r_{45} & & \\ & 0 & & & r_{55} & r_{56} & r_{57} \\ & & & & & r_{66} & r_{67} \\ & & & & & & r_{77} \end{bmatrix}$$

A 7 by 7 matrix $\tilde{B}$ and its factor $\tilde{R}$.

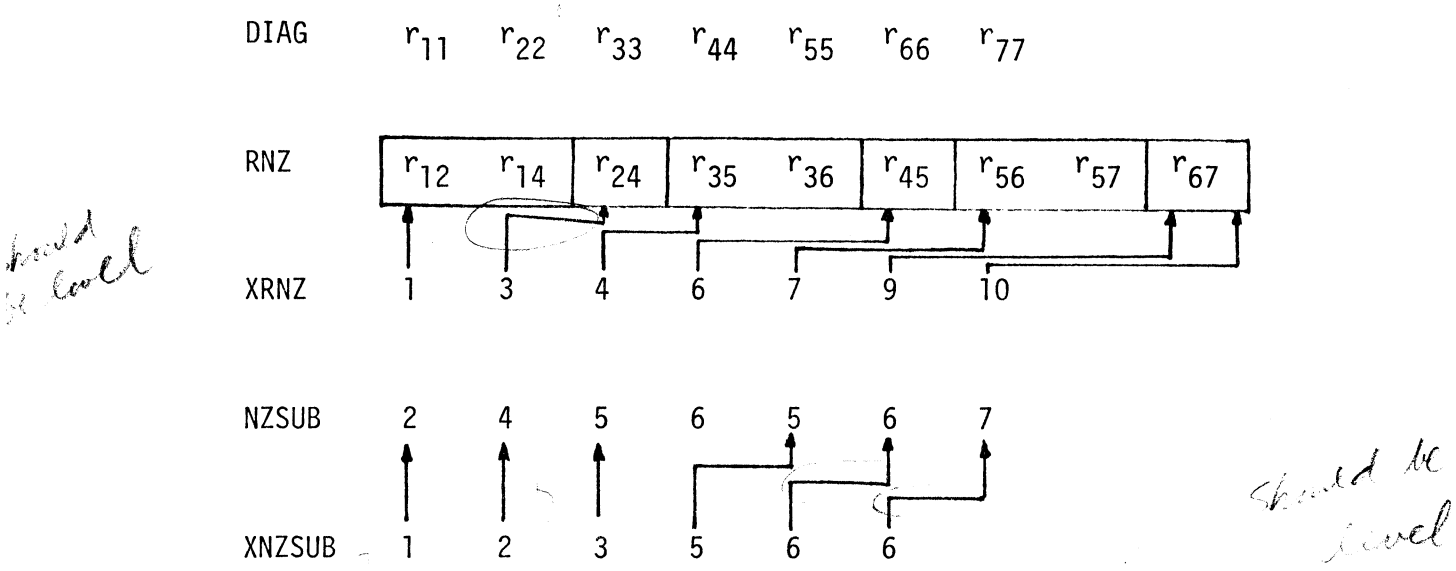

Figure 3.1 An example of the storage scheme used for $\tilde{R}$.

We assume that the rows of A and the corresponding elements of b are available on an external file, and can be read one at a time. The device need only provide serial access, and the order in which the rows of A appear on the file is immaterial. The row by row processing of $\tilde{A}$ and b to create $\tilde{R}$ is depicted in Figure 3.2.

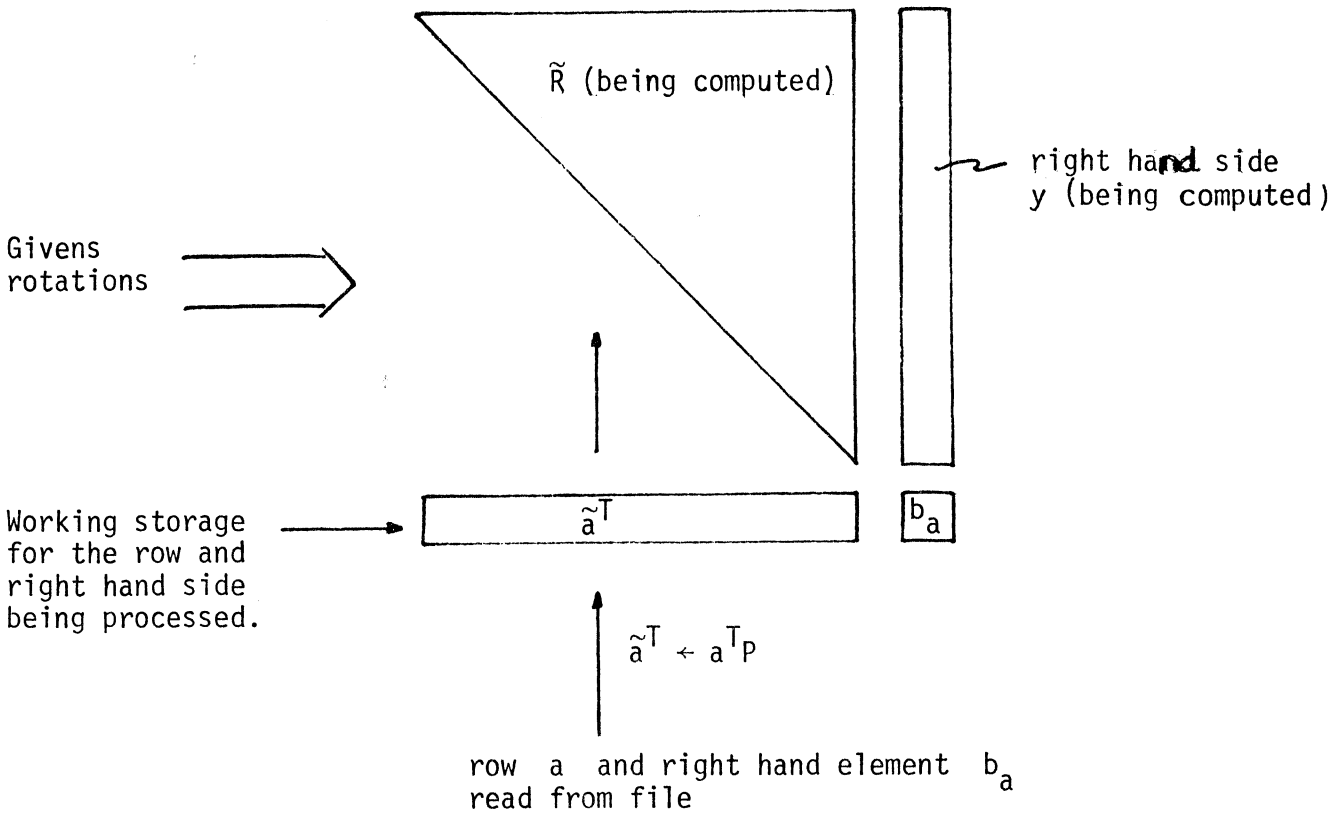

Figure 3.2 Diagram depicting the computational flow of the reduction of $[AP\vdots b]$ to $[\tilde{R}\vdots y]$.

The following is intended to "justify" the scheme just outlined. Specifically, we want to verify that the data structure for $\tilde{R}$ will indeed always have space to accommodate the fill-in created by the rotations. We begin with two observations, followed by a Lemma due to Parter [15].

Remark 1. Suppose a rotation is applied to the sparse vector pair (x,y), yielding $(\tilde{x},\tilde{y})$. Then in general (unless the rotation is a multiple of 90°) $\tilde{x}_i \neq 0$ and $\tilde{y}_i \neq 0$ whenever $x_i \neq 0$ or $y_i \neq 0$ (or both) (See Gentleman [4] for a thorough discussion of Givens rotations.)

Remark 2. $\tilde{B}_{ij} \neq 0 \Leftrightarrow \exists k \ni \tilde{A}_{ki} \neq 0$ and $\tilde{A}_{kj} \neq 0$

Lemma 1. (Parter) $\tilde{R}_{ij} \neq 0 \Leftrightarrow \tilde{B}_{ij} \neq 0$ or

$$\exists k < \min\{i,j\} \ni \tilde{R}_{ki} \neq 0 \text{ and } \tilde{R}_{kj} \neq 0 .$$

Lemma 2. There is space in the data structure of $\tilde{R}$ for any incoming row $\tilde{a}^T$.

Proof: Let $\tilde{a}$ have nonzeros in positions $c_1,c_2,\ldots,c_p$. By Remark 2, $\tilde{B}_{c_1c_\ell} \neq 0$, $\ell = 1,2,\ldots,p$. By Lemma 1, $\tilde{R}_{c_1c_\ell} \neq 0$, $\ell \neq 1,2,\ldots,p$.

So there is space in row c_1 of the data structure for $\tilde{a}^T$.

□

Except near the beginning of the computation, row c_1 of the data structure for $\tilde{R}$ will already be occupied by numerical values, so that row c_1 of $\tilde{R}$ will be used to reduce $\tilde{a}_{c_1}$ to zero, using a rotation. This will cause fill in $\tilde{a}$, according to Remark 1, so that the transformed $\tilde{a}$, which we denote by $\hat{a}$, will in general be nonzero in all positions which are nonzero to the right of the diagonal in row c_1 of $\tilde{R}$. That is, for $\ell > c_1$,

$$\tilde{R}_{c_1,\ell} \neq 0 \Rightarrow \hat{a}_\ell \neq 0 \ .$$

Let $D = \{d_1,d_2,\ldots,d_q\}$ be the subscripts of the nonzeros in $\hat{a}$, with $c < d_1 < d_2 < \ldots < d_q$.

Lemma 3. There is space for $\hat{a}$ in row d_1 of the data structure for $\tilde{R}$.

Proof: We need to verify that $\tilde{R}_{d_1 d_\ell} \neq 0$ for $d_\ell \in D$. This follows immediately from Lemma 1 by setting $k = c_1$, $i = d_1$ and $j = d_\ell$, $\ell = 2,3,\ldots,q$.

□

Of course row d_1 of the data structure for $\tilde{R}$ may already have numerical values in it, so it would be used to reduce $\hat{a}_{d_1}$ to zero, yielding a transformed $\hat{\hat{a}}$, and so on. Repeated application of Lemma 3 shows that eventually an empty row of the data structure will be encountered, or all elements of the incoming row will be annihilated.

In our implementation we have found it convenient to handle the intermediate fill in the working row by using a working vector of length n. The working vector is initialized to zero, and the nonzeros of the incoming row are inserted into the appropriate locations. New nonzeros generated by successive rotations can then be inserted into the working vector very efficiently without the need for a dynamic data structure. A table of subscripts of nonzeros in the working vector is maintained in order to avoid scanning through the entire n-vector. Arithmetic operations with zero operands may also be avoided.

A basic assumption of our method is that R can be stored in core, but there is room for little more. In particular, the Givens rotations are not stored; they are simply discarded after use. However,

if we wish to solve additional problems having the same matrix A but different right hand sides b, then the new right hand sides must be transformed by the same sequence of Givens rotations as were used in reducing the matrix. In this case the rotations could be written out on an external file for later use. Auxiliary storage could be economized by using the technique of Stewart [18] in representing each rotation by a single floating-point number. An alternative possibility for handling multiple right hand sides would be to solve the system

$$R^T R x = A^T b$$

using the R already on hand, so that only the original matrix, which is already stored on an external file, is needed to transform subsequent right hand sides. This latter approach is intermediate in numerical stability between the normal equations and orthogonalization [3].

§4. Numerical Experiments

Although the method presented in this paper is suitable for most sparse linear least squares problems, we have had a particular class of problems in mind in designing and testing the algorithm. Our paradigm is the class of network or grid problems having a collection of nodes which are interconnected by observations (equations). Each node is connected to only a few other nodes, usually nearest neighbors, and there may be considerably more observations than nodes. Such problems arise, for example, in geodetic surveying and in finite element analysis, and may reach truly spectacular size [9].

We have used two network problems in our numerical testing. One is a real geodetic network having 892 observation equations and 261 unknowns, with about five nonzeros per row, supplied to us by the U.S. National Geodetic Survey. This problem is a tiny portion of the North American datum to be readjusted in 1983 [13]. Our second example is characteristic of problems arising in the natural factor formulation of the finite element method. There is a k by k square grid of nodes and each observation equation involves the four nodes corresponding to one of the $(k-1)^2$ smallest subsquares at the grid. With $k = 20$ and with each observation equation replicated four times using randomly chosen coefficients, a problem was generated having 1444 rows and 400 columns. For both of these problems we compare the new algorithm with the normal equations method.

A few more remarks are in order concerning the details of our computer implementation. Steps 1 through 3, which are the same for both the normal equations approach and the new algorithm, are carried out using existing modules from SPARSPAK. We have found it

most convenient to determine the structure of A^TA in Step 1 by simply making a preliminary pass through the problem data stored on an external file, noting the column subscripts but not the values of nonzero elements of A. It is possible that the structure of A^TA could be determined a priori in some contexts, thus not requiring that the data be read twice. The ordering used in Step 2 for both methods is the quotient minimum degree ordering option of SPARSPAK. Other orderings are available in SPARSPAK but we have not tried these for our test problems. In Step 4 we have used standard Givens rotations (four multiplications, two additions, one square root) rather than the modified or "fast" Givens rotations (two multiplications, two additions, no square root) [4]. The modified Givens rotations could certainly be used in this context, but we have chosen not to do so in order to simplify the coding and to obtain maximum accuracy and stability. Moreover, in practice the modified scheme yields a far smaller reduction in actual running time than the reduction in operation count would imply, particularly since rescaling is often necessary to ensure stability [12].

The order in which the rows of $\tilde{A}$ are processed in Step 4 is immaterial to the numerical stability of the algorithm and to the ultimate sparsity of $\tilde{R}$. Thus, while the natural order in which the rows are stored will be most convenient, any other row ordering is also permissible. We have found that this freedom can be exploited to limit the amount of intermediate fill in the working row and thereby reduce substantially the overall operation count for the orthogonal factorization phase. An extreme instance of this behavior is shown by the contrived example depicted in Figure 4.1. For the row order shown in Fig. 4.1(a) the cost of using Givens rotations to reduce A to upper triangular form is $O(n^2)$. For the row order in Fig. 4.1(b) the cost

is $O(mn^2)$. Finding an optimal row ordering with respect to operation count in the orthogonal reduction is an obvious topic for further research. A suboptimal heuristic we have employed in our numerical experiments is simply to sort the data file into increasing order with respect to the maximum column subscript (in the permuted column order of $\tilde{A}$). For both of our example problems this is the "good" row ordering reported in Table 4.1. For the geodetic network problem the "bad" row ordering is the natural order in which the physical problem was originally presented. For the square grid problem the "bad" ordering is the reverse of the "good" ordering. Both test problems were sufficiently well conditioned that the answers from all methods agreed to essentially full machine precision.

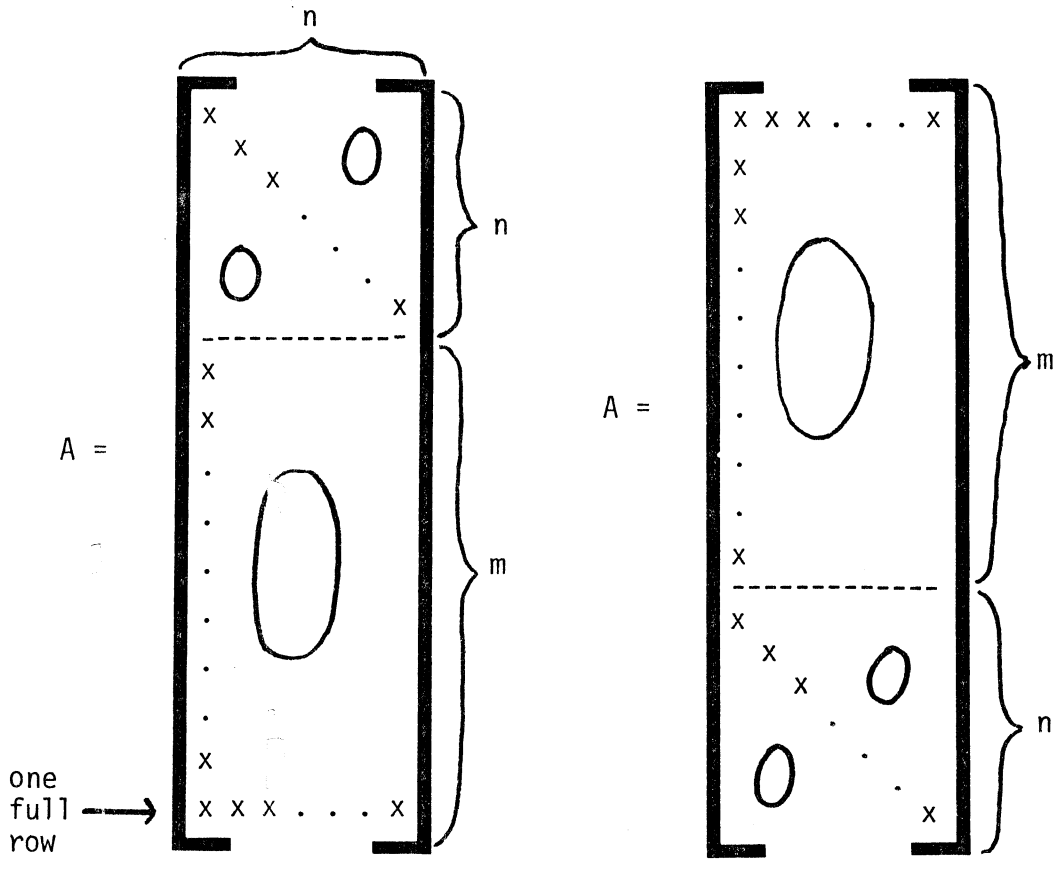

Figure 4.1

Table 4.1

	Geodetic problem	Grid problem
Number of rows	892	1444
Number of columns	261	400
Nonzeros in A	4342	5776
Nonzeros in A^TA	2785	1882
Nonzeros in $\tilde{R}$	5033	6229
Maximum storage used[1]	7531	9369
Time for ordering[2]	1.17	0.17
Normal equations		
Time for factor/solve	0.69	0.78
Operations for factor/solve[3]	83086	92539
Givens with "good" row order		
Time for factor/solve	1.90	2.62
Operations for factor/solve	1673586	2218705
Givens with "bad" row order		
Time for factor/solve	3.96	6.83
Operations for factor/solve	3133877	5760666

Notes:
1. Maximum storage used includes pointers and other overhead as well as space for nonzeros.
2. All times are in seconds for an IBM 3033 using Fortran double precision. Factor/solve times include reading in the data.
3. Operation count unit is a multiply-add pair.

§5. Addition of Rows

The processing of rows in the algorithm described in Section 2 is open ended in the sense that additional rows may be rotated into R at any time and the resulting new solution computed, provided the new rows fit into the existing data structure (i.e., the new row should add nothing to the structure of A^TA). If the new rows do not fit into the existing data structure, as will be the case, for example, with full rows which have been held out of initial processing in order to avoid catastrophic fill-in, then an updating procedure may be used to incorporate the effect of the new rows into the solution. Note that we do not update the factorization, but merely the solution. Such updating procedures are common in least squares applications [14]; we include details here for completeness.

We wish to solve the augmented problem

$$\min_x \left\| \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} - \begin{pmatrix} A_1 \\ A_2 \end{pmatrix} x \right\|_2 .$$

Let $r_1(x) = b_1 - A_1x$ and $r_2(x) = b_2 - A_2x$. Let y be the solution to the original problem

$$\min_x ||r_1(x)||_2$$

as determined by an algorithm such as that of Section 2 using the orthogonal factorization

$$A_1 = Q^T \begin{pmatrix} R \\ 0 \end{pmatrix} .$$

Letting x be the solution to the augmented problem, we wish to determine $z = x - y$. Now

$$r_1(x) = b_1 - A_1x = b_1 - A_1y - A_1z = r_1(y) - A_1z$$

and similarly $r_2(x) = r_2(y) - A_2z$. Since $A_1^T r_1(y) = A_1^T(b_1 - A_1y) = 0$, minimizing $||r_1(x)||_2^2$ is equivalent to minimizing $||A_1z||_2^2$.

Thus, the augmented problem can be recast as

$$\min_z \left\| \begin{matrix} A_1z \\ r_2(y)-A_2z \end{matrix} \right\|_2 = \min_z \left\| \begin{matrix} Q^TRz \\ r_2(y)-A_2R^{-1}Rz \end{matrix} \right\|_2 = \min_u \left\| \begin{matrix} u \\ r_2(y)-A_2R^{-1}u \end{matrix} \right\|_2 ,$$

with $u = Rz$. Now letting $v = r_2(y) - A_2R^{-1}u$, $d = r_2(y)$, $w = \binom{u}{v}$, and $C = [A_2R^{-1}\ I]$ we arrive at the problem

$$\min_w ||w||_2 \quad \text{subject to} \quad Cw = d$$

which is simply the problem of finding the minimum length solution to an underdetermined linear system. The row dimension of C is the same as that of A_2 so that the problem is presumably small and we can afford to ignore sparsity here. To solve this last problem we use another orthogonal factorization

$$C^T = U\begin{bmatrix} L^T \\ 0 \end{bmatrix} = [U_1\ U_2]\begin{bmatrix} L^T \\ 0 \end{bmatrix}$$

where U is orthogonal and L lower triangular, so that $Cx = d$ becomes

$$[L\ 0]\begin{bmatrix} U_1^T \\ U_2^T \end{bmatrix} x = d.$$

Using the change of variable $\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{bmatrix} U_1^T \\ U_2^T \end{bmatrix} x$, i.e. $x = U_1y_1 + U_2y_2$, the constraint equation becomes $[L\ 0]\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = d$.

We may now determine y_1 by solving $Ly_1 = d$, leaving y_2 free to be chosen so as to minimize $||x||_2$. The choice $y_2 = 0$ yields the desired result.

§6. Concluding Summary

We have developed a stable numerical method for solving sparse linear least squares problems using no more space than the traditional, but potentially unstable, normal equations method. As the figures in Table 4.1 show, our method requires more computer time than the normal equations, but the ratio is no worse than in dense problems of comparable m and n. Such a storage/time trade-off is quite acceptable, for example, on a small dedicated computer where space is at a premium but processing time is essentially free. Moreover, in fairness it should be noted that our implementation of the normal equations method is extremely efficient, using the most up-to-date sparse matrix technology.

Our method has several other interesting features. It does not need to see the whole matrix at any one time, and it can process pieces of the matrix in arbitrary order. This leads to the possibility of solving very large scale sparse problems by using auxiliary storage. Indeed, one of the most important uses we see for the algorithm is as the in-core module for an out-of-core sparse least squares solver [7].

Our method does require storage for A^TA during the ordering phase, and storage for R during the numerical computation. For sparse problems where m >> n, these quantities are usually much less than the space required for A.

In sparse least squares, as in many sparse matrix problems, there is a delicate and complicated interplay among considerations of sparsity, stability and efficiency. In the algorithm we have proposed these complex issues are isolated to a large extent. The use of orthogonal transformations rather than elimination assures stability **and allows pivoting, if any, to be done based on sparsity and efficiency**

alone. Similarly, the column and row orderings are based solely on sparsity and efficiency, respectively.

There are numerous areas for further research. We have tried only one column ordering scheme, based on the minimum degree algorithm. For other classes of problems, a different ordering strategy may be more appropriate. In particular, a column ordering which can be computed more quickly but yields a somewhat less sparse Cholesky factor might be preferable for some problems. The question of row ordering is not so well understodd and much research remains to be done.

References

[1] A. Björck, "Methods for sparse linear least squares problems", in Sparse Matrix Computations, ed. by J.R. Bunch and D.J. Rose, Academic Press, New York, 1976, pp. 177-199.

[2] Å. Björck and I.S. Duff, "A direct method for solving sparse linear least square problems", Linear Algebra and its Applications, (to appear in this issue).

[3] I.S. Duff and J.K. Reid, "A comparison of some methods for the solution of sparse overdetermined systems of linear equations", J. Inst. Maths. Applics. 17, (1976), pp. 267-280.

[4] W.M. Gentleman, "Least squares computations by Givens transformations without square roots", J. Inst. Math. Appl. 12 (1973), pp. 329-336.

[5] A. George and J.W.H. Liu, Computer Solution of Large Sparse Positive Definite Systems, to be published by Prentice Hall, Inc.

[6] A. George and J.W.H. Liu, "An optimal algorithm for symbolic factorization of symmetric matrices", SIAM J. on Comput., (to appear).

[7] A. George, M.T. Heath and R.J. Plemmons, "Solution of large scale sparse least squares problems using auxiliary storage", in preparation.

[8] P.E. Gill and W. Murray, "The orthogonal factorization of a large sparse matrix", in Sparse Matrix Computations, ed. by J.R. Bunch and D.J. Rose, Academic Press, New York, 1976, pp. 201-212.

[9] G.H. Golub and R.J. Plemmons, "Large scale geodetic least squares adjustment by dissection and orthogonal decomposition", Linear Algebra and its Applications, (to appear in this issue).

[10] G.H. Golub, "Numerical methods for solving linear least squares problems", Numer. Math. 7, (1965), pp. 206-216.

[11] G.D. Hachtel, "Extended application of the sparse tableau approach - finite elements and least squares", Technical Report, Elec. Sci, and Eng. Dept. and Computer Sci. Dept., UCLA, 1974.

[12] R.J. Hanson, "Is the fast Givens transformation really fast", ACM SIGNUM Newsletter, 8, No. 4, (1973), p.7.

[13] G.B. Kolata, "Geodesy: dealing with an enormous computer task", Science 200, (1978), pp. 421-422.

[14] C.L. Lawson and R.J. Hanson, Solving Least Squares Problems, Prentice-Hall, 1974.

[15] S.V. Parter, "The use of linear graphs in Gaussian elimination", SIAM Review, 3 (1961), pp. 364-369.

[16] G. Peters and J.H. Wilkinson, "The least squares problem and pseudo-inverses", Comput. J. 13 (1970), pp. 309-316.

[17] 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.

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