ORNL/CSD-114

UCC-ND

NUCLEAR DIVISION

UNION CARBIDE

Numerical Methods for Large Sparse Linear Least Squares Problems

M. T. Heath

OPERATED BY
UNION CARBIDE CORPORATION
FOR THE UNITED STATES
DEPARTMENT OF ENERGY

Printed in the United States of America. Available from
National Technical Information Service
U.S. Department of Commerce
5285 Port Royal Road, Springfield, Virginia 22161
NTIS price codes—Printed Copy: A03; Microfiche A01

ORNL/CSD-114
Distribution Category UC-32

NUMERICAL METHODS FOR LARGE SPARSE LINEAR LEAST SQUARES PROBLEMS

Michael T. Heath

Date Published - April 1983

COMPUTER SCIENCES
at
Oak Ridge National Laboratory
Post Office Box Y
Oak Ridge, Tennessee 37830

Research sponsored by the
Applied Mathematical Sciences Research Program
Office of Energy Research

Union Carbide Corporation - Nuclear Division
operating the
Oak Ridge Gaseous Diffusion Plant Oak Ridge National Laboratory
Oak Ridge Y-12 Plant Paducah Gaseous Diffusion Plant
under contract No. W-7405-eng-26
for the
Department of Energy

CONTENTS

NUMERICAL METHODS FOR LARGE SPARSE LINEAR LEAST SQUARES PROBLEMS*

MICHAEL T. HEATH†

Abstract. Large sparse least squares problems arise in many applications, including geodetic network adjustments and finite element structural analysis. Although geodesists and engineers have been solving such problems for years, it is only relatively recently that numerical analysts have turned attention to them. In this paper we present a survey of numerical methods for large sparse linear least squares problems, focusing mainly on developments since the last comprehensive surveys of the subject published in 1976. We consider direct methods based on elimination and on orthogonalization, as well as various iterative methods. The ramifications of rank deficiency, constraints, and updating are also discussed.

Key words. sparse least squares, normal equations, elimination, orthogonalization, Givens rotations, iterative methods

1. Introduction. The method of least squares often means different things to different people. For statisticians, least squares is a method for estimating unknown parameters. For engineers and experimental scientists, it is a method for curve fitting or data smoothing. For numerical analysts, least squares has come to mean solving nonsquare systems of equations, that is, systems whose equations and unknowns differ in number. Of course, such a system does not necessarily have a solution, and so instead we settle for minimizing some norm of the residual. Although other norms are sometimes used, the (possibly weighted) Euclidean norm is by far the most common, hence the name "least squares." If such a minimum residual solution is not unique, then an additional requirement is imposed, such as that the norm of the solution itself also be minimal.

From both practical and computational points of view, linear problems are an extremely important subclass. This is true not only because many systems are inherently linear, but also because many algorithms for solving nonlinear problems require the solution of a succession of linear subproblems. For this reason, a great deal of effort over an extended period of time has gone into the development of a number of effective algorithms for solving linear least squares problems. In many cases the seeds of these algorithms are contained in computing practices which predate electronic computers and were not necessarily originally intended for least squares problems. These techniques were later sharpened and refined, particularly with regard to their numerical properties, for use on digital computers before finally receiving a systematic modern implementation for solving least squares problems. The present survey is concerned with a further phase of development, the extension of these algorithms to handle large sparse least squares problems.

To fix notation, the problem we wish to consider is

$$\min_x \ \|b - Ax\|_2, \tag{1.1}$$

* Research sponsored jointly by the National Geodetic Survey of the National Ocean Survey, NOAA, U. S. Department of Commerce, under Interagency Agreement No. 40-1108-80, and 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.

† Mathematics and Statistics Research Department, Computer Sciences Division, Oak Ridge National Laboratory, Oak Ridge, Tennessee 37830.

where A is an $m \times n$ matrix and b and x are vectors of dimension m and n, respectively. Standard assumptions are that $m \geq n$ and $rank(A)=n$, in which case the solution to (1.1) is unique. Problems in which $m<n$ or $rank(A)<\min\{m,n\}$ do occur, however, in many important contexts. For such underdetermined or rank deficient problems, (1.1) does not have a unique solution, so that the minimum norm solution, or some other particular solution, is usually what is required. Other important generalizations of (1.1) include the imposition of constraints and updating the solution when new data are added. For a comprehensive discussion of algorithms for least squares problems see [44]. For statistical interpretation see [58].

This survey is primarily concerned with methods which are effective for solving (1.1) when the matrix A is large and sparse, which means that A contains relatively few nonzero elements. Areas in which large sparse least squares and related problems arise include geodesy, photogrammetry, image enhancement, structural analysis, spline data smoothing, and mathematical programming.

In order to make the solution of very large sparse least squares problems computationally feasible with respect to execution time and storage requirements, algorithms for such problems should store and operate on only the nonzero entries of the matrix and should try to minimize the creation of new nonzeros as computations proceed. Another important consideration is the manner in which the data are accessed, since auxiliary storage is often required for large problems. Conventionally, the rows of the matrix A correspond to observations (successive measurements or replications), whereas the columns of A correspond to the unknown variables or parameters of the problem. Thus it is more convenient to generate, store, and access the data matrix by rows rather than by columns.

In addition to sparsity preservation and data management, questions of numerical stability must also be taken into account when evaluating algorithms for sparse least squares problems. Indeed, the main purpose of much of the research on least squares algorithms in the past twenty years has been to improve on the numerical shortcomings of more traditional methods. As in other areas of sparse matrix computations, here, too, the tradeoffs between sparsity and stability are of vital importance.

2. Normal Equations. If $rank(A)=n$, then the solution to (1.1) is given by the solution to the system of normal equations

$$A^TAx=A^Tb. \tag{2.1}$$

If $rank(A)=m$, the minimum norm solution to (1.1) is given by $x=A^Ty$, where y is the solution to the linear system

$$AA^Ty=b. \tag{2.2}$$

In either case, the matrix of the linear system is symmetric and positive definite so that the solution can be obtained by Cholesky factorization. For system (2.1) the latter factorization takes the form $A^TA=R^TR$, with R upper triangular, so that (2.1) can be solved by forward and backward substitution. Most of the following discussion will be given in terms of system (2.1) for the overdetermined case; an analogous discussion is applicable to system (2.2) for the underdetermined case. (See [13] for a survey of basic methods for underdetermined problems.)

For dense problems, algorithms based on this approach are quite efficient, requiring only about half as many arithmetic operations as competing methods when $m \gg n$,

although this advantage diminishes for more nearly square problems. The normal equations method is also very attractive for sparse problems, since excellent software packages are available for solving large sparse symmetric positive definite linear systems (eg., YSMP [20], SPARSPAK [27] and MA27 [19]). These software packages symmetrically reorder the equations and unknowns so that the Cholesky factor suffers relatively little fill (creation of new nonzeros). A number of reordering heuristics are known which can dramatically reduce the fill resulting from factorization. These include the minimum degree algorithm, various dissection schemes, and various bandwidth or profile minimization schemes.

Let P and Q be permutation matrices of order n and m, respectively. Then

$$(QAP)^TQAP = P^TA^TQ^TQAP = P^TA^TAP.$$

We conclude that the row ordering of A has no bearing on the normal equations, but reordering the columns of A corresponds to a symmetric row and column permutation of A^TA. In particular, the rows of A can be processed sequentially from an auxiliary file in arbitrary order. In the software packages mentioned in the previous paragraph, the symmetric reordering is determined by analyzing the structure of the graph corresponding to the symmetric matrix. For the symmetric matrix of the system of normal equations this graph is easily determined: the nonzeros of each row of A generate a clique in the graph of A^TA. (See [27] for relevant graph-theoretic concepts and terminology.) It is important to emphasize that, since pivoting is not required for numerical stability in the Cholesky algorithm, the reordering phase is entirely symbolic and takes place before any floating point computation. Furthermore, by anticipating all fill in advance, dynamic storage allocation is unnecessary, so that an efficient static data structure can be used.

Thus, an algorithm implementing (2.1) for sparse least squares problems goes like this (an analogous algorithm implements (2.2) for the underdetermined case):

ALGORITHM 1. Normal Equations.

1. Determine the structure (not the numerical values) of A^TA.
2. Find a permutation matrix P such that P^TA^TAP has a sparse upper triangular Cholesky factor R.
3. Factor P^TA^TAP symbolically, generating a row-oriented data structure for R.
4. Compute A^TA and A^Tb numerically, processing the rows of A one by one from an external file.
5. Factor $P^TA^TAP = R^TR$ numerically.
6. Solve $R^Tz = P^TA^Tb$.
7. Solve $Ry = z$.
8. $x = Py$.

The use of the normal equations is as old as the method of least squares itself. The normal equations are easy to derive and understand, and, as we have seen, they adapt nicely to deal with sparse problems. Indeed, for many people "normal equations" and "least squares" are virtually synonymous, and algorithms based on the normal equations are still by far the most commonly used for solving least squares problems (see, e.g., [36]). The only trouble in this apparent paradise is that use of the normal equations may lead to numerical difficulties which are relatively harmless in some cases, but can be disastrous in others.

The potential numerical problems associated with the normal equations spring from two sources. One is the potential loss of information in explicitly computing the

cross-product matrix A^TA and vector A^Tb. The other source of difficulty is the fact that the condition number of the matrix A^TA is the square of that of A, so that an accurate solution of (2.1) may be difficult or impossible to compute if A itself is already poorly conditioned (see, e.g., [60], [48], [32]). If A is not of full column rank, then A^TA is singular and the Cholesky factorization algorithm breaks down. Near rank degeneracy causes similar numerical problems in finite precision arithmetic. The upshot of all this is that forming and solving the normal equations requires relatively high working precision in order to guarantee suitable accuracy, which would often entail an unacceptable increase in storage requirements for very large problems. The principal motivation for the other methods we will discuss is to alleviate these numerical difficulties.

Another problem which the reader may have noticed is that the sparsity of A does not guarantee that A^TA is comparably sparse. Indeed, if an otherwise sparse A has even one dense row, then, barring numerical cancellation, A^TA is completely full. Clearly, such a situation is disastrous for the normal equations algorithm. The day can be saved, however, by initially omitting any rows of A which would cause excessive fill in A^TA (assuming full rank is still maintained) and later updating the solution to incorporate the effect of the omitted rows. Such updating procedures are common in least squares applications when new observations are added to a previously solved problem. The possibility of excessive fill in forming A^TA is sometimes given as justification for some alternative method. Barring accidental cancellation, however, all direct methods suffer the same or a similar fate when confronted with a matrix A having one or more dense rows. On the other hand, the updating procedures necessary to cope with such situations can be more stably implemented using other techniques, such as orthogonal transformations.

3. Elimination Methods. For simplicity of presentation, in this section we assume $rank(A)=n$. Carrying out Gaussian elimination with row and column interchanges on the $m \times n$ matrix A leads to a factorization of the form

$$P_1AP_2=LU, \tag{3.1}$$

where P_1 and P_2 are permutation matrices of order m and n, respectively, L is an $m \times n$ unit lower trapezoidal matrix, and U is an $n \times n$ upper triangular matrix. Utilizing this factorization, together with the orthogonal invariance of the Euclidean norm and the change of variable

$$UP_2^Tx=y, \tag{3.2}$$

problem (1.1) becomes

$$\min_y \|P_1b-Ly\|_2. \tag{3.3}$$

One way to solve problem (3.3) is by means of the system of normal equations

$$L^TLy=L^TP_1b. \tag{3.4}$$

The solution x to (1.1) is then recovered by solving the triangular system (3.2). It appears that little has been gained by this approach compared to solving system (2.1) directly. As observed by Peters and Wilkinson [52], however, by using an appropriate pivoting strategy in the elimination (i.e., choice of P_1 and P_2), it is possible to control the conditioning of L, isolating any ill conditioning of A in U. Thus, it should be safe numerically to form and solve system (3.4). Other authors have suggested the use of

orthogonalization techniques to solve (3.3), but there is no real advantage over applying such methods directly to (1.1) unless the problem is only slightly overdetermined (see [12] and [53]).

This elimination scheme of Peters and Wilkinson has been extended and adapted for sparse problems by Björck and Duff [8]. When A is sparse, the permutations P_1 and P_2 must be chosen to preserve sparsity in L and U as well as to enhance the conditioning of L and the numerical stability of the factorization. A threshold pivoting strategy is used by Björck and Duff as a compromise between considerations of sparsity and stability. Note that here, too, one or more dense rows in A could cause unacceptable fill in L^TL, and so an updating scheme should be used to account for such rows. Usually the nonzero pattern of L is very much like that of A, and so solving system (3.4) requires about the same work and storage as system (2.1). Thus, the improved numerical behavior of the Peters-Wilkinson scheme is bought at the possibly high price of computing the factorization (3.1). One saving grace is that since U is not involved in solving (3.3), U may be written out on an external file and then recalled for the back substitution (3.2), thereby conserving main storage.

Björck and Duff introduce a modification of the Peters-Wilkinson scheme in which the solution x is split into two parts, one of which does not involve the normal equations (3.4). Using the partitioning

$$L=\begin{bmatrix}L_1\\L_2\end{bmatrix}, \qquad P_1b=\begin{bmatrix}b_1\\b_2\end{bmatrix}, \tag{3.5}$$

where L_1 is a matrix of order n and b_1 is a vector of dimension n, let the n-vector c be defined by

$$L_1c=b_1.$$

Note that this is equivalent to taking c to be the first n components of the transformed right hand side vector, if the latter is processed simultaneously with A during the elimination. Let the $(m-n)-$vector$d$ be defined by

$$d=b_2-L_2c.$$

We now observe that

$$P_1(b-Ax)=\begin{bmatrix}o\\d\end{bmatrix}-Lz,$$

where

$$z=UP_2^Tx-c.$$

Thus, if z is the solution of the problem

$$\min_z \left\|\begin{bmatrix}o\\d\end{bmatrix}-Lz\right\|_2,$$

and x_1 and x_2 are given by the triangular systems

$$UP_2^Tx_1=c, \qquad UP_2^Tx_2=z,$$

then $x=x_1+x_2$ is the solution of (1.1). There are two principal advantages to splitting the solution into two parts in this manner. First, since $\|b-Ax_1\|_2=\|d\|_2$, if $\|d\|_2$ is sufficiently small (i.e., the original problem (1.1) is nearly consistent), then x_1 is already an adequate solution, so that x_2, and hence z, need not be computed at all.

Second, any ill conditioning in L affects only the computation of the "correction" term x_2.

The steps of the Peters-Wilkinson method as modified by Björck and Duff are summarized in the following algorithm.

ALGORITHM 2. Elimination.

1. Compute the factorization (3.1), choosing P_1 and P_2 to produce sparse U and L and well conditioned L.
2. Solve $L_1 c = b_1$, with L_1 and b_1 as in (3.5).
3. Solve $UP_2^T x_1 = c$.
4. $d = b_2 - L_2 c$.
5. If $\|d\|_2 < \epsilon$, set $x = x_1$ and stop.
6. Solve $L^T L z = L^T \begin{bmatrix} o \\ d \end{bmatrix}$ using Algorithm 1.
7. Solve $UP_2^T x_2 = z$.
8. $x = x_1 + x_2$.

In their paper [8], Duff and Björck show how to extend this algorithm to handle rank deficient problems, weighted problems, constraints, and updating. Delves and Barrodale [15] have published a related elimination algorithm in which first a square subsystem of (1.1) is solved by the usual square LU factorization, then the remaining rows of A are treated as updates. Such an approach is advantageous for problems which are only slightly overdetermined. Their algorithm does not appear to have been implemented as yet specifically for sparse problems.

Another family of elimination methods is based on the fact that the solution x to (1.1) and the residual $r = b - Ax$ must satisfy the augmented $(m+n) \times (m+n)$ linear system

$$\begin{bmatrix} I & A \\ A^T & O \end{bmatrix} \begin{bmatrix} r \\ x \end{bmatrix} = \begin{bmatrix} b \\ o \end{bmatrix}. \tag{3.6}$$

This system has been used by Björck [2] in studying iterative refinement for least squares solutions, and its use for sparse problems has been advocated by Hachtel, who calls this the "sparse tableau" approach [37]. The idea here is simply to use a standard sparse solver for square linear systems to solve (3.6). Note that block elimination applied to (3.6) without pivoting yields the usual normal equations for x. The hope is that the sparse square system solver will use the additional freedom in choosing pivots to find a considerably more sparse factorization. Although system (3.6) is symmetric, it is indefinite, so that the pivoting must take account of numerical stability as well as sparsity. While a sparse symmetric indefinite factorization is certainly possible [19], there are actually certain advantages to ignoring the symmetry of (3.6) and using a nonsymmetric system solver [18]. On the other hand, ignoring symmetry incurs the heavy penalty of having to store two copies of A.

4. Orthogonalization Methods.

4.1. Basic Methods. We have seen that the chief difficulty with the normal equations is numerical: the information loss and ill conditioning associated with the explicit formation of the cross-product matrix. The elimination method of Peters and Wilkinson tries to lessen these numerical effects. Another alternative is to avoid explicit formation of the cross-product matrix altogether by computing its triangular Cholesky

factor R directly from A, and this can be done by means of orthogonal factorization. An orthogonal matrix Q of order m is computed which reduces $[A, b]$ to the form

$$QA=\begin{bmatrix}R\\O\end{bmatrix}, \qquad Qb=\begin{bmatrix}c\\d\end{bmatrix}, \tag{4.1}$$

where R is an upper triangular matrix of order n and c is a vector of dimension n. (We consider first the $m \geq n$ case; the underdetermined case will be taken up later in this section.) To see that R is indeed the Cholesky factor of the cross-product matrix (assuming Q is chosen so that R has positive diagonal entries), we need merely note that

$$A^TA=A^TQ^TQA=[R^T \ O]\begin{bmatrix}R\\O\end{bmatrix}=R^TR.$$

Since the Euclidean norm is invariant under orthogonal transformation, the solution to (1.1) may be obtained by solving the triangular system $Rx=c$.

There are three principal methods for computing the factorization (4.1): Gram-Schmidt orthogonalization ([55], [1]), Householder reflections ([41], [32], [9]), and Givens rotations ([31], [21-23], [38]). Both Gram-Schmidt and Householder reduce A to triangular form by annihilating all the subdiagonal elements in an entire column at each step. Though effective for dense problems, this column-oriented, "sledge hammer" approach has serious drawbacks for large sparse problems. The trouble is that at each step each column in the remaining unreduced portion of the matrix which has a nonzero inner product with the column being reduced takes on the sparsity pattern of their union. Although this newly created fill scattered throughout the unreduced matrix will eventually be annihilated by the orthogonalization process, in the mean time it must be stored, greatly increasing storage requirements beyond that required for R (see Fig. 1). Givens rotations are a much more appropriate tool in this context because of their ability to introduce zeros more selectively and in a more flexible order [21]. In particular, R can be built up gradually as the rows of A are processed one by one in their natural order (or any other desired order), intermediate fill is confined to R and the working row, and the unreduced rows of A can remain untouched on external storage until their turn for actual reduction (see Fig. 2).

A problem which plagues all orthogonalization methods is that even if A and R are sparse, it is unlikely that the orthogonal matrix Q will be particularly sparse. There has been some study of maintaining sparsity in Q ([61],[11],[16]), but the outlook in general is quite unpromising, especially since sparsity must be maintained simultaneously in R. Instead, most practical procedures simply discard the orthogonal transformations which make up Q as they are used in processing the matrix and right hand side vector. For a simple problem with a single right hand side no real harm is done, since Q is not actually needed to compute the least squares solution once R and c have been obtained. But for more complicated problems, such as those having multiple right hand sides which are not known in advance, or certain applications which require explicit computation of an orthogonal basis, getting along without Q can be a headache. One alternative is to write the orthogonal transformations on auxiliary storage as they are generated. Each Givens rotation can be represented by a single floating point number [59], thereby economizing storage. In other cases the need for Q can be circumvented, although not always with equivalent numerical stability. For example, one way to handle multiple right hand sides is to solve the system

FIG. 1. *Reducing a sparse matrix by Householder reflections.*

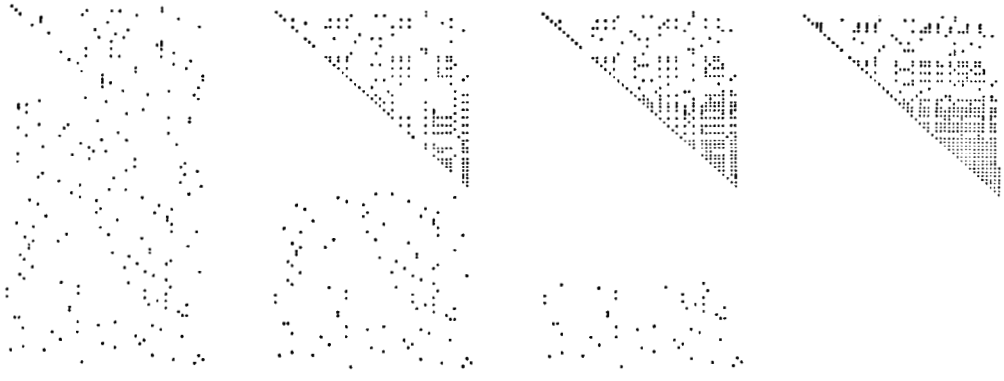

FIG. 2. *Reducing a sparse matrix by Givens rotations.*

$$A = \begin{pmatrix} x & & & & & \\ & x & & & & \\ & & x & & & \\ & & & x & & \\ & & & & x & \\ x & & & & & \\ x & & & & & \\ x & & & & & \\ x & x & x & x & x & \end{pmatrix} \begin{matrix} \left.\vphantom{\begin{matrix}a\\a\\a\\a\\a\\a\\a\\a\\a\end{matrix}}\right\} n \\ \left.\vphantom{\begin{matrix}a\\a\\a\\a\end{matrix}}\right\} k \end{matrix}, \qquad PA = \begin{pmatrix} x & x & x & x & x \\ x & & & & \\ x & & & & \\ x & & & & \\ x & & & & \\ & x & & & \\ & & x & & \\ & & & x & \\ & & & & x \end{pmatrix}$$

FIG. 3. *Cost for reducing A: $O(n^2)$, PA: $O(kn^2)$.*

$$R^TRx = A^Tb \tag{4.2}$$

using the R already computed, so that only the original matrix A, which is available on an external file, is needed to transform subsequent right hand sides. Although system (4.2) shares some of the numerical shortcomings of the normal equations method, at least R is accurately computed, and the accuracy of the solution can be improved by a few iterations of iterative refinement [4].

Having decided to use Givens rotations, there remains the problem of choosing a good row and column ordering for A. There has been a good deal of work ([16],[23],[62]) on choosing these orderings dynamically: the ordering is determined according to some local minimization-of-fill criterion as numerical computations proceed, and storage is inserted into the data structure as needed to accommodate any fill generated. Such algorithms are very similar in spirit to square, nonsymmetric linear system solvers in that access to the whole unreduced matrix is required at each step for possible pivot selection. An important difference, however, is that the stability of orthogonal transformations allows the selection to be based solely on sparsity considerations (assuming the problem is not too disparately weighted: see further comments on row ordering below).

A different approach is taken in [24], which is patterned more after symmetric positive definite linear system solvers. Recall our earlier remark in discussing the normal equations that the row ordering for A has no bearing on the structure of A^TA, but that the column ordering for A determines the structure of A^TA and hence that of R, its Cholesky factor. Thus the structure of A^TA can be analyzed symbolically in advance of any numerical computation in order to find a good column order for A which will limit fill in R, and also to set up a data structure which will accommodate any fill in R which does occur. Such a static data structure can be very efficient, requiring none of the garbage collection and other overhead associated with dynamic data structures.

An algorithm based on these considerations is as follows:

ALGORITHM 3. Triangularization by Givens Rotations.

1. Determine the structure (not the numerical values) of A^TA.
2. Find a permutation matrix P such that P^TA^TAP has a sparse upper triangular Cholesky factor R.
3. Factor P^TA^TAP symbolically, generating a row-oriented data structure for R.
4. Compute R and c numerically, processing the rows of $[AP, b]$ one by one using Givens rotations.
5. Solve $Ry = c$.
6. $x = Py$.

Steps 1 through 3 of Algorithm 3 are the same as those of Algorithm 1 and can be carried out very efficiently using standard sparse matrix software designed for symmetric positive definite linear systems. In particular, assuming the Givens rotations are either discarded or written on secondary storage, Algorithm 3 exploits sparsity to the same degree and requires the same amount of primary storage as Algorithm 1, but is more stable numerically.

The details of step 4 of Algorithm 3 are of some interest. Let a^T be a given row of AP to be processed next. Let j be the subscript of the first nonzero component of a^T. It is shown in [24] that there is space in row j of the data structure of R to

accommodate a^T. If row j of the data structure is still vacant, as will likely be the case early in the row by row processing, then a^T may simply be placed into row j of the data structure. If, on the other hand, row j of the data structure is already occupied by previously stored numerical values, then row j may be used to annihilate the first nonzero of a^T with a Givens rotation. It is further shown in [24] that the resulting "shorter" row can also be accommodated in the data structure, even though some fill may have occurred as a result of the transformation. Thus, the process may be repeated until either an unoccupied row is found in which to place the working row or all its nonzeros have been annihilated.

Although the order in which the rows of A are processed in step 4 does not affect the structure of R, it does affect the amount of fill created in the working row and hence the total cost of computing R. Fig. 3 gives an extreme example of the difference row ordering can make with respect to numerical factorization cost. Heuristic row ordering rules have been suggested which can substantially reduce computational costs for certain classes of problems, but the general relationship between row and column orderings is not yet well understood (see [24] and [28]). Another factor which may dictate a particular row ordering is that any heavily weighted rows should be processed first for optimum numerical stability (see [44], pp. 103-106).

Specialized algorithms which adapt orthogonalization techniques to problems having banded or similar structure are given in [14], [44], and [54].

4.2. Extensions and Generalizations. Algorithm 3 lends itself to a number of useful extensions and generalizations in order to handle more difficult or complicated problems. For example, the problem may be so large that R will not fit in main memory, and hence auxiliary storage must be used. Indeed, for extremely large problems such as the geodetic readjustment of the North American Datum [42], storage requirements may even exceed the virtual address space of the largest computers, so that auxiliary space cannot be managed implicitly by a paging algorithm. In [26] an algorithm is given in which such large problems are partitioned by incomplete nested dissection into a sequence of smaller subproblems, each of which is processed by the basic Givens algorithm, eventually producing the solution to the original problem.

It is clear that Algorithm 3 suffers the same catastrophic fill as the other methods we have discussed when confronted with a matrix A having one or more dense rows. It is also sometimes desired that some of the equations in a linear system be satisfied exactly while the remaining equations are satisfied only in the least squares sense. For example, it may be required that the sum of all the variables be equal to 1 or some other prescribed constant. Extensions to Algorithm 3 which enable it to incorporate such constraints and/or updating are derived in [39]. Further generalization to allow arbitrary rank along with constraints and updating is given in [7].

Another implicit assumption in Algorithm 3 is that $rank(A)=n$. The usual generalization of the orthogonal factorization (4.1) for dense rank deficient problems is to use column interchanges during the orthogonalization process to obtain a factorization of the form

$$QAP=\begin{bmatrix} R & S \\ O & T \end{bmatrix}, \tag{4.3}$$

where R is an upper triangular matrix of order $k=rank(A)$, P is a permutation matrix which performs the column interchanges, and the elements of T are negligible

in magnitude. In Algorithm 3, however, the column ordering for A represented by P is fixed in advance to preserve sparsity, and cannot be altered during the numerical phase. It is shown in [39] that a factorization of the form (4.3) can nevertheless be obtained without explicit column pivoting, leading in turn to a basic or minimum-norm solution for underdetermined and/or rank deficient sparse linear least squares problems.

An alternate approach for underdetermined problems is to apply Algorithm 3 to A^T rather than A, yielding an orthogonal factorization of the form

$$A=[R^T \ O]Q. \tag{4.4}$$

This enables us to replace system (2.2) by the system

$$R^TRy=b, \tag{4.5}$$

which can be solved by forward and back substitution. Such an approach fits in well with the philosophy of discarding Q, but it would appear to have the same condition squaring effect as that suffered by (2.2) and (4.2). In practice, however, the use of (4.5) usually yields results which are comparable to the theoretically more accurate algorithm which involves Q explicitly [56]. This surprising behavior has been explained by Paige [49], who shows that as long as A is not too ill conditioned, the error resulting from (4.5) depends essentially on the condition number of A rather than the condition number squared. In using (4.4) and (4.5) for the sparse case via Algorithm 3, we must now avoid dense columns rather than dense rows. Appropriate updating procedures are given in [29].

5. Iterative Methods. For some large sparse least squares problems iterative methods are a useful alternative to the direct methods we have discussed thus far. Unlike direct methods, which compute an approximation to the exact solution in a finite number of steps, iterative methods successively improve an initial approximate solution until the approximation is acceptably close to the exact solution. One advantage of this approach is that one need not spend time computing an unnecessarily accurate solution if the data do not warrant it. Direct methods, on the other hand, generally have fixed accuracy and do not produce meaningful intermediate results. Iterative methods are also especially appropriate for problems in which the entries of the matrix are easily generated on demand. In such cases the matrix need not be stored at all, but instead can be defined by its action on vectors.

In principle any iterative method for symmetric positive definite (or semidefinite) linear systems can simply be applied to the system of normal equations (2.1). Explicit formation of the cross-product matrix A^TA can be avoided by keeping the normal equations in factored form

$$A^T(b-Ax)=0. \tag{5.1}$$

In this way the matrix A is used only to compute matrix-vector products of the form Ax and A^Ty. Although this formulation avoids some of the numerical difficulties and fill which can result from explicitly forming A^TA, the convergence rate of iterative methods based on (5.1) still depends on the spectrum of A^TA, and can therefore be very slow. Because of this the main emphasis in research on iterative methods for least squares problems has been to accelerate convergence by means of various splittings and preconditioners.

A splitting takes the form $A=M-N$, leading to a sequence of least squares problems of the form

$$Mx_{k+1} \cong Nx_k + b.$$

Preconditioning is a change of variable $z=Cx$, where C is a nonsingular matrix of order n, so that

$$b - Ax = b - AC^{-1}z.$$

In either case the strategy is to choose M or C so that M or AC^{-1} gives a more favorable spectrum than A, thereby speeding convergence. Since these two approaches are essentially equivalent [10], we will concentrate on preconditioning methods.

In implementing a preconditioner for least squares problems matrix-vector products of the form Ax and A^Ty become $AC^{-1}z$ and $C^{-T}A^Ty$, respectively. Of course the matrix product AC^{-1} is never explicitly computed, but instead is treated as the product of two successive operators. Thus each iteration will require solution of linear systems of the form

$$Cx = z \qquad \text{and} \qquad C^Tx = y. \tag{5.2}$$

For this reason C is usually chosen to be diagonal or triangular so that systems (5.2) can be solved easily. Several different preconditioners have been used effectively for least squares problems:

1. Diagonal scaling. $C=diag(d_i)$, where the d_i are norms of the columns of A.
2. SSOR preconditioning [5]. $C=I+\omega L^T$, where A has been scaled so that $A^TA=L+I+L^T$ with L strictly lower triangular, and ω is a scalar relaxation parameter.
3. Incomplete Cholesky factorization ([45],[46]). $C=R'$, where R' is an approximation to the Cholesky factor R but is more sparse. Note that the true Cholesky factor R would be the ideal choice for C since then AC^{-1} is orthogonal.
4. LU preconditioning [57]. $C=U$, where U is the upper triangular factor from a factorization of the form (3.1).
5. Gauss-Jordan preconditioning [43]. $C=A_1^{-1}$, where A_1 is a square, nonsingular submatrix of A of order n.

Notice that this last preconditioner gives an algorithm that is essentially the same as the direct method of Delves and Barrodale [15] mentioned in section 3, except that the updating is done iteratively rather than in closed form. This is just one example of the many ways in which preconditioning blurs the distinction between direct and iterative methods.

As a concrete example of an iterative method for least squares problems we now consider the method of conjugate gradients. There are many variants of conjugate gradients which are theoretically equivalent but differ in their numerical behavior. One of the most effective for reasonably well conditioned least squares problems is this early version due to Hestenes and Stiefel [40]:

ALGORITHM 4. Conjugate Gradients.

1. $x_0=0$
2. $r_0=b$
3. $s_0=A^Tr_0$
4. $\gamma_0=\|s_0\|_2^2$
5. $p_1=s_0$
6. For $k=1, 2, \ldots$ repeat the following:
   a. $q_k=Ap_k$
   b. $\alpha_k=\gamma_{k-1}/\|q_k\|_2^2$
   c. $x_k=x_{k-1}+\alpha_k p_k$
   d. $r_k=r_{k-1}-\alpha_k q_k$
   e. $s_k=A^Tr_k$
   f. $\gamma_k=\|s_k\|_2^2$
   g. $\beta_k=\gamma_k/\gamma_{k-1}$
   h. $p_{k+1}=s_k+\beta_k p_k$

Although this algorithm theoretically produces the exact solution to (1.1) in at most n iterations, with the rounding errors of finite precision arithmetic it may require far more or far fewer than n iterations to yield a satisfactory solution. If a preconditioner is used to accelerate convergence, then the matrix-vector products Ap and A^Tr are replaced by $AC^{-1}p$ and $C^{-T}A^Tr$ implemented by solving systems of the form (5.2). In using a preconditioner there is a tradeoff between the reduction in the number of iterations required and the increase in work per iteration.

A variant of conjugate gradients which is effective for more ill conditioned least squares problems has been developed by Paige and Saunders in [50], which should be consulted for algorithmic details. Their algorithm is based on the bidiagonalization procedure of Golub and Kahan [33] in the same way that the conjugate gradient algorithm is related to Lanczos tridiagonalization. The bidiagonalization approach has also been adapted to obtain regularized solutions of ill-posed problems ([6],[47]).

These bidiagonalization algorithms can be thought of as a natural extension of the singular value decomposition method to solve sparse problems. The singular value decomposition, which has the form

$$A=U\Sigma V^T, \tag{5.3}$$

where U and V are orthogonal matrices of order m and n, respectively, and Σ is an $m \times n$ nonnegative diagonal matrix, is in many ways the most satisfactory numerical method for solving least squares problems ([33], [35], [44]). Unfortunately the full decomposition (5.3) is not computationally useful for large sparse problems because the orthogonal matrices U and V are generally too dense. Through Lanczos-style bidiagonalization, however, a few dominant triples (σ,u,v) of singular values and vectors can be generated even for very large matrices [34].

6. Concluding Remarks. In this paper we have surveyed the principal numerical methods available for solving large sparse linear least squares problems. We have concentrated on developments since the surveys [3], [18], and [30] published in 1976. Several of these methods have been compared in numerical experiments using a variety of test problems ([18],[25],[50],[57]). Although much more testing is needed on a broader spectrum of test examples, some preliminary conclusions are possible:

1. For well conditioned problems the traditional method of normal equations implemented using modern sparse matrix techniques is very effective and hard to beat, especially on problems for which $m \gg n$ and which are quite sparse.
2. For more difficult problems which require the greater numerical stability of orthogonal factorization, the method of Givens rotations can be implemented so as to use essentially the same storage as the normal equations method (see Algorithm 3).
3. For problems which are only slightly overdetermined, or are overdetermined but consistent, a method based on elimination is likely to be best.
4. Several effective techniques are available for problems which are well suited to iterative solution. If conditioning is a problem, the method of Paige and Saunders [50] is especially to be recommended. A preconditioner can help speed convergence but must be chosen carefully.

Most of the methods we have discussed have been implemented in computer software which is publicly available, or soon will be:

1. Several symmetric positive definite linear system solvers (e.g., SPARSPAK [27], YSMP[20], MA27 [19]) are available which could form the basis of an efficient implementation of the normal equations method. Since it handles indefinite problems as well, MA27 could also be used to solve the augmented system (3.6).
2. The implementation of the Peters-Wilkinson elimination method by Björck and Duff [8] is to be included in the Harwell Subroutine Library. Their code uses a modified version of the Harwell subroutine MA28 [17] to compute the LU factorization (3.1).
3. Software implementing the algorithms of George and Heath [24], [39] is to be included in a new, expanded version of SPARSPAK, giving it the capability of solving nonsymmetric and nonsquare problems. The new modules rely heavily on the existing SPARSPAK for the symbolic parts of the computation.
4. The work of Zlatev [62] is implemented in the software package LLSS01 [63]. To conserve storage, this code allows for an incomplete triangular factorization (determined by a drop tolerance), followed by iterative refinement.
5. The iterative algorithm of Paige and Saunders has been implemented as subroutine LSQR and is available from ACM TOMS [51]. This code solves damped least squares problems as well as ordinary least squares and nonsymmetric equations.

There is considerable room for further research on algorithms for sparse least squares problems. Better row and column orderings are needed, as well as a better understanding of the relationship between them. The tradeoffs between static and dynamic data structures need further study. In elimination methods, the structural relationships between A and L and between U and R need to be better understood. Also worthy of exploration are row elimination schemes using stabilized elementary eliminators, as opposed to the use of a general threshold (partial or complete) pivoting approach. More sophisticated criteria are needed for withholding rows which lead to excessive fill in the Cholesky factor. The numerical behavior of updating schemes needs to be further scrutinized and improved. Existing algorithms should be extended

to allow more generality with regard to constraints, updating and rank. An important problem for iterative methods is automating the choice of an effective preconditioner to speed convergence. With all of these methods relatively little attention has been given to handling problems which are too large to fit in main storage or to the use of advanced computer architectures such as pipelined, array, or parallel processors.

The answers to these and many other outstanding questions will undoubtedly lead to more effective and efficient methods for solving large sparse least squares problems. In addition, advances in sparse least squares computations will have an effect on other areas of sparse matrix computations, such as the application of sparse orthogonal factorizations to problems in optimization, control, and eigenvalue and singular value computations.

REFERENCES

[1] Å. BJÖRCK, *Solving linear least squares problems by Gram-Schmidt orthogonalization*, BIT, 7 (1967), pp. 1-21.

[2] Å. BJÖRCK, *Iterative refinement of linear least squares solutions*, BIT, 7 (1967), pp. 257-278 and 8 (1968), pp. 8-30.

[3] Å. BJÖRCK, *Methods for sparse least squares problems*, in Sparse Matrix Computations, J. R. Bunch and D. J. Rose, eds., Academic Press, New York, 1976, pp. 177-199.

[4] Å. BJÖRCK, *Comment on the iterative refinement of least squares solutions*, J. Amer. Stat. Assoc., 73 (1978), pp. 161-166.

[5] Å. BJÖRCK, *SSOR preconditioning methods for sparse least squares problems*, Proc. Comput. Sci. Stat. Symposium on the Interface, 12 (1979), pp. 21-25.

[6] Å. BJÖRCK, *A bidiagonalization algorithm for solving ill-posed systems of linear equations*, Report LiTH-MAT-R-80-33, Dept. of Mathematics, Linkoping University, Linkoping, Sweden, October 1980.

[7] Å. BJÖRCK, *A general updating algorithm for constrained linear least squares problems*, SIAM J. Sci. Stat. Comput., to appear.

[8] Å. BJÖRCK AND I. S. DUFF, *A direct method for the solution of sparse linear least squares problems*, Linear Algebra Appl., 34 (1980), pp. 43-67.

[9] P. BUSINGER AND G. H. GOLUB, *Linear least squares solutions by Householder transformations*, Numer. Math., 7 (1965), pp. 269-276.

[10] Y. T. CHEN, *Iterative methods for linear least squares problems*, Research Report CS-75-04, Dept. of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, February 1975.

[11] Y. T. CHEN AND R. P. TEWARSON, *On the fill-in when sparse vectors are orthonormalized*, Computing, 9 (1972), pp. 53-56.

[12] A. K. CLINE, *An elimination method for the solution of linear least squares problems*, SIAM J. Numer. Anal., 10 (1973), pp. 283-289.

[13] R. E. CLINE AND R. J. PLEMMONS, *ℓ_2-solutions to underdetermined linear systems*, SIAM Review, 18 (1976), pp. 92-106.

[14] M. G. COX, *The least squares solution of overdetermined linear equations having band or augmented band structure*, IMA J. Numer. Anal., 1 (1981), pp. 3-22.

[15] L. M. DELVES AND I. BARRODALE, *A fast direct method for the least squares solution of slightly overdetermined sets of linear equations*, J. Inst. Math. Appl., 24 (1979), pp. 149-156.

[16] I. S. DUFF, *Pivot selection and row ordering in Givens reduction on sparse matrices*, Computing, 13 (1974), pp. 239-248.

[17] I. S. DUFF, *MA28 - A set of Fortran subroutines for sparse unsymmetric linear equations*, Report R.8730, AERE, Harwell, England, July 1977.

[18] I. S. DUFF AND J. K. REID, *A comparison of some methods for the solution of sparse overdetermined systems of linear equations*, J. Inst. Math. Appl., 17 (1976), pp. 267-280.

[19] I. S. DUFF AND J. K. REID, *MA27 - A set of Fortran subroutines for solving sparse symmetric sets of linear equations*, Report R.10533, AERE, Harwell, England, 1982.

[20] S. C. EISENSTAT, M. H. SCHULTZ AND A. H. SHERMAN, *Algorithms and data structures for sparse symmetric Gaussian elimination*, SIAM J. Sci. Stat. Comput., 2 (1981), pp. 225-237.

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

[22] W. M. GENTLEMAN, *Error analysis of QR decompositions by Givens transformations*, Linear Algebra Appl., 10 (1975), pp. 189-197.
[23] W. M. GENTLEMAN, *Row elimination for solving sparse linear systems and least squares problems*, Proc. 6th Dundee Conf. Numer. Anal., Springer-Verlag, 1976, pp. 122-133.
[24] A. GEORGE AND M. T. HEATH, *Solution of sparse linear least squares problems using Givens rotations*, Linear Algebra Appl., 34 (1980), pp. 69-83.
[25] A. GEORGE, M. T. HEATH AND E. NG, *A comparison of some methods for solving sparse linear least squares problems*, SIAM J. Sci. Stat. Comput., to appear.
[26] A. GEORGE, M. T. HEATH AND R. J. PLEMMONS, *Solution of large-scale sparse least squares problems using auxiliary storage*, SIAM J. Sci. Stat. Comput., 2 (1981), pp. 416-429.
[27] A. GEORGE AND J. W.-H. LIU, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice-Hall, Englewood Cliffs, NJ, 1981.
[28] A. GEORGE AND E. NG, *On row and column orderings for sparse least squares problems*, SIAM J. Numer. Anal., to appear.
[29] A. GEORGE AND E. NG, *Solution of sparse underdetermined systems of linear equations*, Research Report CS-82-39, Dept. of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, September 1982.
[30] P. E. GILL AND W. MURRAY, *The orthogonal factorization of a large sparse matrix*, in Sparse Matrix Computations, J. R. Bunch and D. J. Rose, eds., Academic Press, New York, 1976, pp. 201-212.
[31] W. GIVENS, *Computation of plane unitary rotations transforming a general matrix to triangular form*, J. SIAM, 6 (1958), pp. 26-50.
[32] G. H. GOLUB, *Numerical methods for solving linear least squares problems*, Numer. Math., 7 (1965), pp. 206-216.
[33] G. H. GOLUB AND W. KAHAN, *Calculating the singular values and pseudo-inverse of a matrix*, SIAM J. Numer. Anal., 2 (1965), pp. 205-224.
[34] G. H. GOLUB, F. T. LUK AND M. L. OVERTON, *A block Lanczos method for computing the singular values and corresponding singular vectors of a matrix*, ACM Trans. Math. Software, 7 (1981), pp. 149-169.
[35] G. H. GOLUB AND C. REINSCH, *Singular value decomposition and least squares solutions*, Numer. Math., 14 (1970), pp. 403-420.
[36] J. H. GOODNIGHT, *A tutorial on the SWEEP operator*, Amer. Statistician, 33 (1979), pp. 149-158.
[37] G. D. HACHTEL, *Extended application of the sparse tableau approach - finite elements and least squares*, Technical Report, Computer Science Dept., UCLA, 1974.
[38] S. HAMMARLING, *A note on modifications to the Givens plane rotation*, J. Inst. Math. Appl., 13 (1974), 215-218.
[39] M. T. HEATH, *Some extensions of an algorithm for sparse linear least squares problems*, SIAM J. Sci. Stat. Comput., 3 (1982), pp. 223-237.
[40] M. R. HESTENES AND E. STIEFEL, *Methods of conjugate gradients for solving linear systems*, J. Res. Nat. Bur. Stds., B49 (1952), pp. 409-436.
[41] A. S. HOUSEHOLDER, *Unitary triangularization of a nonsymmetric matrix*, J. ACM, 5 (1958), pp. 339-342.
[42] G. B. KOLATA, *Geodesy: dealing with an enormous computer task*, Science, 200 (1978), pp. 421-422.
[43] P. LÄUCHLI, *Jordan-Elimination und Ausgleichung nach kleinsten Quadraten*, Numer. Math., 3 (1961), pp. 226-240.
[44] C. L. LAWSON AND R. J. HANSON, *Solving Least Squares Problems*, Prentice-Hall, Englewood Cliffs, NJ, 1974.
[45] T. A. MANTEUFFEL, *An incomplete factorization technique for positive definite linear systems*, Math. Comp., 34 (1980), pp. 473-497.
[46] N. MUNKSGAARD, *Solving sparse symmetric sets of linear equations by preconditioned conjugate gradients*, ACM Trans. Math. Software, 6 (1980), pp. 206-219.
[47] D. P. O'LEARY AND J. A. SIMMONS, *A bidiagonalization-regularization procedure for large scale discretizations of ill-posed problems*, SIAM J. Sci. Stat. Comput., 2 (1981), pp. 474-489.
[48] E. E. OSBORNE, *On least squares solutions of linear equations*, J. ACM, 8 (1961), pp. 628-636.
[49] C. C. PAIGE, *An error analysis of a method for solving matrix equations*, Math. Comp., 27 (1973), pp. 355-359.
[50] C. C. PAIGE AND M. A. SAUNDERS, *LSQR: An algorithm for sparse linear equations and sparse least squares*, ACM Trans. Math. Software, 8 (1982), pp. 43-71.

[51] C. C. PAIGE AND M. A. SAUNDERS, *Algorithm 583. LSQR: Sparse linear equations and least squares*, ACM Trans. Math. Software, 8 (1982), pp. 195-209.
[52] G. PETERS AND J. H. WILKINSON, *The least squares problem and pseudo-inverses*, Comput. J., 13 (1970), pp. 309-316.
[53] R. J. PLEMMONS, *Linear least squares by elimination and MGS*, J. ACM, 21 (1974), 581-585.
[54] J. K. REID, *A note on the least squares solution of a band system of linear equations by Householder reductions*, Comput. J., 10 (1967), pp. 188-189.
[55] J. R. RICE, *Experiments on Gram-Schmidt orthogonalization*, Math. Comp., 20 (1966), pp. 325-328.
[56] M. A. SAUNDERS, *Large-scale linear programming using the Cholesky factorization*, Report No. CS 252, Computer Science Dept., Stanford University, Stanford, CA, January 1972.
[57] M. A. SAUNDERS, *Sparse least squares by conjugate gradients: a comparison of preconditioning methods*, Proc. Comput. Sci. Stat. Symposium on the Interface, 12 (1979), pp. 15-20.
[58] G. A. F. SEBER, *Linear Regression Analysis*, John Wiley and Sons, New York, 1977.
[59] G. W. STEWART, *The economical storage of plane rotations*, Numer. Math., 25 (1976), pp. 137-138.
[60] O. TAUSSKY, *Note on the condition of matrices*, Math. Comp., 4 (1950), pp. 111-112.
[61] R. P. TEWARSON, *On the orthonormalization of sparse vectors*, Computing, 3 (1968), pp. 268-279.
[62] Z. ZLATEV, *Comparison of two pivotal strategies in sparse plane rotations*, Comp. Math. Appl., 8 (1982), pp. 119-135.
[63] Z. ZLATEV AND H. B. NIELSEN, *LLSS01 - A Fortran subroutine for solving least squares problems (user's guide)*, Report No. 79-07, Inst. Numer. Anal., Tech. Univ. Denmark, Lyngby, Denmark, 1979.

ORNL/CSD-114
Distribution Category UC-32

INTERNAL DISTRIBUTION

1. Central Research Library
2. K-25 Plant Library
3. ORNL Patent Office
4. Y-12 Technical Library, Document Reference Section
5. Laboratory Records - RC
6-7. Laboratory Records Department
8. K. O. Bowman
9. J. A. Carpenter
10. H. P. Carter/CSD Library
11. J. B. Drake
12. R. E. Funderlic
13. P. W. Gaffney
14. D. A. Gardiner
15. L. J. Gray
16-20. M. T. Heath
21. L. P. Lewis
22. T. J. Mitchell
23. M. D. Morris
24. N. J. Price
25. R. J. Renka
26. C. A. Serbin
27. K. E. Shultz/ Biometrics Library
28. V. R. R. Uppuluri
29. R. C. Ward
30. D. G. Wilson

EXTERNAL DISTRIBUTION

31. Prof. V. Ashkenazi, Department of Civil Engineering, The University of Nottingham, Nottingham, England NG7 2RD
32. Dr. Donald M. Austin, Division of Engineering, Mathematical, & Geosciences, Office of Basic Energy Sciences, Germantown Building, Department of Energy, Washington, DC 20545
33. Prof. Åke Björck, Department of Mathematics, Linköping University, Linköping 58183, Sweden
34. Dr. Paul Boggs, Scientific Computing Division, National Bureau of Standards, Boulder, CO 80303
35. Dr. James Bunch, UCSD, C-012, APM Buildng, La Jolla, CA 92093
36. Dr. T. D. Butler, T-3, Hydrodynamics, Los Alamos National Laboratory, P.O. Box 1663, Los Alamos, NM 87545
37. Dr. Bill L. Buzbee, C-3, Applications Support & Research, Los Alamos National Laboratory, P.O. Box 1663, Los Alamos, NM 87545
38. Dr. Tony Chan, Department of Computer Science, Yale University, P.O. Box 2158 Yale Station, New Haven, CT 06520
39. Dr. L. Lynn Cleland, Engineering Research Division, Lawrence Livermore National Laboratory, P.O. Box 808, Livermore, CA 94550
40. Dr. James S. Coleman, Division of Engineering, Mathematical and Geosciences, Office of Basic Energy Sciences, Department of Energy, ER-17, MC G-256, Germantown, Washington, DC 20545
41. Dr. Thomas F. Coleman, Computer Science Department, Cornell University, Ithaca, NY 14853

42. Dr. James Corones, Ames Laboratory, Iowa State University, Ames, IA 50011

43. Dr. Jane K. Cullum, IBM T. J. Watson Research Center, P.O. Box 218, Yorktown Heights, NY 10598

44. Dr. Jack J. Dongarra, Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60439

45. Dr. Iain S. Duff, CSS Division, Building 8.9, AERE Harwell, Didcot, Oxon, England

46. Dr. George J. Davis, Department of Mathematics, Georgia State University, Atlanta, GA 30303

47. Dr. Stanley Eisenstat, Department of Computer Science, Yale University, P.O. Box 2158 Yale Station, New Haven, CT 06520

48. Dr. Marvin D. Erickson, Computer Technology, Systems Department, Pacific Northwest Laboratory, P.O. Box 999, Richland, WA 99352

49. Dr. Albert M. Erisman, Boeing Computer Services, 565 Andover Park West, Tukwila, WA 98188

50. Dr. Kirby W. Fong, L-560, Lawrence Livermore National Laboratory, P.O. Box 5509, Livermore, CA 94550

51. Dr. Fred N. Fritsch, L-300, Mathematics and Statistics Division, Lawrence Livermore National Laboratory, P.O. Box 808, Livermore, CA 94550

52. Dr. David M. Gay, Bell Laboratories, 600 Mountain Avenue, Murray Hill, NJ 07974

53. Dr. J. Alan George, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1

54. Dr. John R. Gilbert, Computer Science Department, Cornell University, Ithaca, NY 14853

55. Prof. Max Goldstein, Courant Institute of Mathematical Sciences, New York University, 251 Mercer Street, New York, NY 10012

56. Prof. Gene H. Golub, Department of Computer Science, Stanford University, Stanford, CA 94305

57. Dr. Fred Gustavson, IBM T. J. Watson Research Center 33-205, P.O. Box 218, Yorktown Heights, NY 10598

58. Dr. Richard J. Hanson, Numerical Mathematics Division 5122, Sandia National Laboratories, Albuquerque, NM 87115

59. Dr. Robert E. Huddleston, Applied Mathematics Division 8332, Sandia National Laboratories, Livermore, CA 94550

60. Prof. Alan Jennings, Department of Civil Engineering, Queen's University of Belfast, David Keir Building, Stranmillis Road, Belfast BT7 1NN, Northern Ireland

61. Dr. Linda Kaufman, Bell Laboratories, 600 Mountain Avenue, Murray Hill, NJ 07974

62. Dr. Robert J. Kee, Applied Mathematics Division 8331, Sandia National Laboratories, Livermore, CA 94550

63. Dr. Aram K. Kevorkian, Technical Computing Department, General Atomic Company, P.O. Box 81608, San Diego, CA 92138

64. Prof. Peter D. Lax, Director, Courant Institute of Mathematical Sciences, New York University, 251 Mercer Street, New York, NY 10012

65. Dr. Charles Lawson, Jet Propulsion Laboratory, Pasadena, CA 91103

66. Dr. Joseph Liu, Department of Computer Science, York University, 4700 Keele Street, Downsview, Ontario, Canada M3J 1P3

67. Dr. Franklin Luk, Computer Science Department, Cornell University, Ithaca, NY 14853

68. Ms. Judith A. Mahaffey, Statistics, Systems Department, Pacific Northwest Laboratory, P.O. Box 999, Richland, WA 99352

69. Dr. Thomas A. Manteuffel, Numerical Mathematics Division 5642, Sandia National Laboratories, Albuquerque, NM 87185

70. Dr. Paul C. Messina, Applied Mathematics Division, Argonne National Laboratory, Argonne, IL 60439

71. Dr. George Michael, Computation Department, Lawrence Livermore National Laboratory, P.O. Box 808, Livermore, CA 94550

72. Prof. Cleve Moler, Department of Computer Science, University of New Mexico, Albuquerque, NM 87131

73. Dr. Jorge J. Moré, Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60439

74. Dr. Esmond Ng, Department of Computer Science, University of Waterloo, Waterloo, Ontario, Canada N2L 3G1

75. Dr. Basil Nichols, T-7, Mathematical Modeling and Analysis, Los Alamos National Laboratory, P.O. Box 1663, Los Alamos, NM 87545

76. Dr. Dianne P. O'Leary, Computer Science Department, University of Maryland, College Park, MD 20742

77. Prof. Chris Paige, Computer Science Department, McGill University, 805 Sherbrooke Street W., Montreal, Quebec, Canada H3A 2K6

78. Prof. Beresford N. Parlett, Department of Mathematics, University of California, Berkeley, CA 94720

79. Dr. Ronald Peierls, Applied Mathematics Department, Brookhaven National Laboratory, Upton, NY 11973

80. Dr. Robert J. Plemmons, Departments of Mathematics and Computer Science, North Carolina State University, Raleigh, NC 27650

81. Dr. William G. Poole, Boeing Computer Services, 565 Andover Park West, Tukwila, WA 98188

82. Mr. Allen Pope, NOAA/NOS/NGS, OA/C-121, 6001 Executive Boulevard, Rockville, MD 20852

83. Dr. Carl Quong, Computer Science and Applied Mathematics Department, Lawrence Berkeley National Laboratory, Berkeley, CA 94720

84. Dr. John K. Reid, CSS Division, Buildng 8.9, AERE Harwell, Didcot, Oxon, England

85. Dr. Donald J. Rose, Bell Laboratories, 600 Mountain Avenue, Murray Hill, NJ 07974

86. Dr. Milton E. Rose, Director, ICASE, M/S 132C, NASA Langley Research Center, Hampton, VA 28665

87. Dr. Bert W. Rust, Scientific Computing Division, Technology Building, National Bureau of Standards, Washington, DC 20234

88. Dr. Michael Saunders, Systems Optimization Laboratory, Operations Research Department, Stanford University, Stanford, CA 94305

89. Dr. David S. Scott, Computer Sciences Department, University of Texas, Austin, TX 78712

90. Dr. Lawrence F. Shampine, Numerical Mathematics Division 5642, Sandia National Laboratories, P.O. Box 5800, Albuquerque, NM 87115

91. Dr. Danny C. Sorensen, Mathematics and Computer Science Division, Argonne National Laboratory, 9700 South Cass Avenue, Argonne, IL 60439

92. Dr. Trond Steihaug, Department of Mathematical Sciences, Rice University, P.O. Box 1892, Houston, TX 77251

93. Prof. G. W. Stewart, Computer Science Department, University of Maryland, College Park, MD 20742

94. Prof. Charles Van Loan, Department of Computer Science, Cornell University, Ithaca, NY 14853

95. Dr. Ray A. Waller, S-1, Statistics, Los Alamos National Laboratory, P.O. Box 1663, Los Alamos, NM 87545

96. Dr. James H. Wilkinson, Division of Numerical Analysis and Computer Science, National Physical Laboratory, Teddington, Middlesex TW11 OLW, England

97. Dr. Ralph A. Willoughby, Mathematical Sciences Department, IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598

98. Dr. Margaret Wright, Systems Optimization Laboratory, Operations Research Department, Stanford University, Stanford, CA 94305

99. Mr. Robert Ziegler, Defense Mapping Agency, DMAHTC-GSGC, 6500 Brookes Lane, Washington, DC 20315

100. Office of Assistant Manager for Energy Research and Development, Department of Energy, Oak Ridge Operations Office, Oak Ridge, TN 37830

101-
282. Given Distribution as shown in TID-4500 under Mathematics and Computers Category