SOLUTION OF LARGE-SCALE SPARSE LEAST SQUARES PROBLEMS USING AUXILIARY STORAGE

J.A. George
M.T. Heath[1]
R.J. Plemmons[2]

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

Research Report CS-80-52

November 1980

1. Computer Sciences Division, Oak Ridge National Laboratory, Post Office Box Y, Oak Ridge, Tennessee 37830.

2. University of Tennessee, Knoxville, Tennessee.

Issued simultaneously as Union Carbide Corp. report ORNL/CSD-63.

CONTENTS

ILLUSTRATIONS

TABLES

SOLUTION OF LARGE-SCALE SPARSE LEAST SQUARES PROBLEMS USING AUXILIARY STORAGE

A. George
M. T. Heath
R. J. Plemmons

ABSTRACT

Very large sparse linear least squares problems arise in a variety of applications, such as geodetic network adjustments, photogrammetry, earthquake studies, and certain types of finite element analysis. Many of these problems are so large that it is impossible to solve them without using auxiliary storage devices. Some problems are so massive that the storage needed for their solution exceeds the virtual address space of the largest machines. In this paper we describe a method for solving such problems on a typical (large) computer and provide the results of some experiments illustrating the effectiveness of our approach. The method includes an automatic partitioning scheme which is essential to the efficient management of the data on auxiliary files.

1. Introduction and Overview

1.1 Introduction

In this paper a method is presented for solving the linear least squares problem

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

when the $m \times n$ matrix A is very large and sparse and has full column rank n. The problems we wish to solve are so large that the use of auxiliary storage is essential regardless of the numerical method employed. In some cases storage requirements may even exceed the virtual address space of the largest machines, and therefore auxiliary space cannot be managed implicitly by a paging algorithm. Our approach is to break the large problem up into smaller subproblems which are processed sequentially, eventually producing the solution to the original problem. Such an approach requires a method for partitioning the large problem, a computational module for processing the subproblems, and an algorithm for managing external files containing intermediate results. In the remainder of this section, after giving some specific examples of large-scale least squares problems, we survey possible numerical methods and then describe the particular technique we have chosen for the computational module. In Section 2, problem partitioning and data management are discussed. Section 3 presents numerical test results and observations.

1.2 Examples of Large-Scale Least Squares Problems

In recent years least squares problems of ever-increasing size have arisen with ever-increasing frequency. One reason for this is

that modern data acquisition technology allows the collection of massive amounts of data. Another factor is the tendency of scientists to formulate more and more complex and comprehensive models in order to obtain finer resolution and more realistic detail in describing physical systems. Particular areas in which such large-scale least squares problems occur include geodetic surveying (Avila and Tomlin [1979], Golub and Plemmons [1980]), photogrammetry (Golub, Luk and Pagano [1980]), earthquake studies (Vanicek, Elliott and Castile [1979]), and in the natural factor formulation of the finite element method (Argyris and Brönlund [1975], Argyris et al [1978]). An example of truly spectacular size is the least squares adjustment of coordinates (latitudes and longitudes) of stations comprising the North American Datum, to be completed in 1983 by the U.S. National Geodetic Survey (Kolata [1978]). This enormous task requires solving, perhaps several times, a least squares problem having six million equations in four hundred thousand unknowns.

1.3 Numerical Methods for Sparse Least Squares

Several methods have been proposed for solving sparse linear least squares problems (Duff and Reid [1976], Björck [1976], Gill and Murray [1976]). For our purposes the most important qualities in a numerical method will be storage requirements, numerical stability, and convenience in utilizing auxiliary storage.

The classical approach to solving the linear least squares problem is via the system of normal equations

$$A^TAx = A^Tb. \tag{1.2}$$

The n by n symmetric positive definite matrix $B = A^TA$ 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 $R^Ty = A^Tb$ and $Rx = y$. This algorithm has several attractive features for large sparse problems. The Cholesky factorization does not require pivoting for stability so that the ordering for B (i.e., column ordering for A) can be chosen based on sparsity considerations alone. Moreover, there exists well-developed software for determining a good ordering in advance of any numerical computation, thereby allowing use of a static data structure. Another advantage is that 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, and A need never be represented in fast storage in its entirety at any one time. Unfortunately the normal equations method may be numerically unstable. This is due to the potential loss of information in explicitly forming A^TA and A^Tb, and to the fact that the condition number of B is the square of that of A.

A well-known stable alternative to the normal equations is provided by orthogonal factorization (Golub [1965]). An orthogonal matrix Q is computed which reduces A to upper trapezoidal form

$$QA = \begin{bmatrix} R \\ 0 \end{bmatrix}, \qquad Qb = \begin{bmatrix} y \\ z \end{bmatrix}, \tag{1.3}$$

where R is n by n and upper triangular. Since Q does not change the two-norm, we have

$$||Ax - b||_2 = \left|\left|\begin{bmatrix} R \\ 0 \end{bmatrix} x - \begin{bmatrix} y \\ z \end{bmatrix}\right|\right|_2, \tag{1.4}$$

and therefore the solution to (1.1) is obtained by solving the triangular system $Rx = y$. The matrix Q usually results from Gram-Schmidt orthogonalization or from a sequence of Householder or Givens transformations. Both the Gram-Schmidt and Householder algorithms process the unreduced part of the matrix A by columns and can cause severe intermediate fill-in. The use of Givens rotations is much more attractive in that the matrix is processed by rows, gradually building up R, and intermediate fill-in is confined to the working row. This approach, implemented with a good column ordering and an efficient data structure, is the basis for the computational module described in section 1.4.

Other direct non-normal-equations methods for sparse least squares, including those of Peters and Wilkinson [1970] (as implemented by Björck and Duff [1980]) and Hachtel [1976], were considered, but these elimination methods perform row and column pivoting for stability and to preserve sparsity, necessitating access to the entire matrix.† This feature greatly inhibits the partitioning of large problems and the flexible use of auxiliary storage.

1.4 The Computational Module

The numerical method we use is developed in detail in George and Heath [1980]. Our motivation is to combine the flexibility, convenience and low storage requirements of the normal equations with the stability of orthogonal factorization. The basic steps of the algorithm are as follows:

Algorithm 1 Orthogonal Decomposition of A

1. Determine the structure (not the numerical values) of $B = A^TA$.

† For least squares problems where (A|b) has rows of about equal magnitude, elimination methods require only partial pivoting for stability. However, both row and column pivoting are needed to effectively limit fill-in.

2. Find an ordering for B (column ordering for A) which has a sparse Cholesky factor R.

3. Symbolically factorize the reordered B, generating a row-oriented data structure for R.

4. Compute R by processing the rows of A one by one using Givens rotations.

Steps 1 through 3 of Algorithm 1 are the same as would be used in a good implementation of the normal equations method. These steps may be carried out very efficiently using existing well-developed sparse matrix software (George and Liu [1979]). It is important to emphasize that the data structure for R is generated in advance of any numerical computation, and therefore dynamic storage allocation to accommodate fill-in during the numerical computation is unnecessary. The order in which the rows of A are processed in Step 4 does not affect the structure of R or the stability of computing it. Therefore the rows may be accessed from an external file one at a time in arbitrary order. A suboptimal row ordering to reduce the amount of computation associated with intermediate fill-in during the orthogonal decomposition phase was shown to be effective in George and Heath [1980] and is used here. Thus Algorithm 1 requires the same storage and exploits sparsity at least to the same degree as the normal equations, allows convenient use of auxiliary storage, and in addition is numerically stable.

Alternatively, a row ordering scheme can also be used to enhance accuracy in problems having widely varying row norms or weights (Gay [19]).

2. Decomposition of A Using Auxiliary Storage

2.1 Introduction

This section consists of two parts. The first describes an algorithm for finding a column ordering and partitioning of A which lends itself to the efficient use of auxiliary storage. The method uses slightly modified ideas and techniques which have already been described in detail by George and Liu [1978], so our presentation is brief and limited to showing our modifications and the relevance of the scheme to the least squares problem. It is important to note that in some contexts such partitionings/orderings arise as a natural by-product of the modelling procedure (finite element analysis) or data acquisition (geodesy). In these cases, this "preprocessing" algorithm would not be required.

The second part of this section deals with the utilization of the software described in Section 1.4 and the partitioning provided by Algorithm 2 of Section 2.2, along with files on auxiliary storage, to solve very large least squares problems.

2.2 Finding an Appropriate Ordering and Partitioning for the Columns of A

In the sequel it is convenient to work with the symmetric graph associated with the normal equations matrix $B = A^TA$. The graph $G = (X,E)$ associated with B has n nodes x_i, $i = 1, 2, \ldots, n$, which form X, and an edge set E consisting of unordered pairs of nodes with $\{x_i, x_j\} \in E$ if and only if $B_{ij} = B_{ji} \neq 0$, $i \neq j$. Thus the nodes x_i correspond to the variables of the least squares problem; i.e., to the columns of A. Implicit here is the assumption that the nodes of G have been labelled as the columns of A. Thus, a relabelling of the nodes of

G corresponds to a symmetric permutation of the matrix B, or equivalently, a column permutation of A. Given G <u>without</u> any labels, finding an appropriate permutation of the columns of A can be viewed as finding an appropriate labelling for G.

A graph $G' = (X',E')$ is a <u>subgraph</u> of $G = (X,E)$ if $X' \subset X$ and $E' \subset E$. For $Y \subset X$, $G(Y)$ refers to the subgraph $(Y,E(Y))$ of G, where $E(Y) = \{\{u,v\} \in E | u,v \in Y\}$.

Nodes x and y are said to be <u>adjacent</u> if $\{x,y\}$ is an edge in E. For $Y \subset X$, the <u>adjacent set</u> of Y is defined as

$$Adj(Y) = \{x \in X - Y | \{x,y\} \in E \text{ for some } y \in Y\}.$$

A <u>path</u> of length ℓ is a sequence of ℓ edges $\{x_0,x_1\}$, $\{x_1,x_2\}$, ... $\{x_{\ell-1},x_\ell\}$ where all the nodes are distinct except for possibly x_0 and x_ℓ. A graph G is <u>connected</u> if there is a path joining each pair of distinct nodes.

A partitioning P of G is an ordered collection of node sets

$$P = \{Y_1,Y_2, \ldots, Y_p\},$$

where $Y_i \cap Y_j = \emptyset$, $i \neq j$, and $\bigcup_{i=1}^{p} Y_i = X$.

Given a partitioning P of G, the only numberings (labellings) we will be concerned with in this paper will be <u>compatible</u> with P. That is, each Y_i is numbered consecutively, and nodes in Y_i are numbered before those in Y_{i+1}.

A subset $Y \subset X$ of the node set of G is a <u>separator</u> of G if $G(X-Y)$ consists of two or more connected components. A separator is minimal if no subset of it is a separator.

As we shall see below, a desirable column ordering and partitioning for A is provided by a nested dissection partitioning of the graph of B (George, Poole and Voigt [1978]). The algorithm we use here can be described as follows, where G = (X,E) is the graph of B and μ is a user-supplied parameter.

Algorithm 2 Incomplete Nested Dissection Partitioning

1. Set V = X, p = 0, and n = |X|.

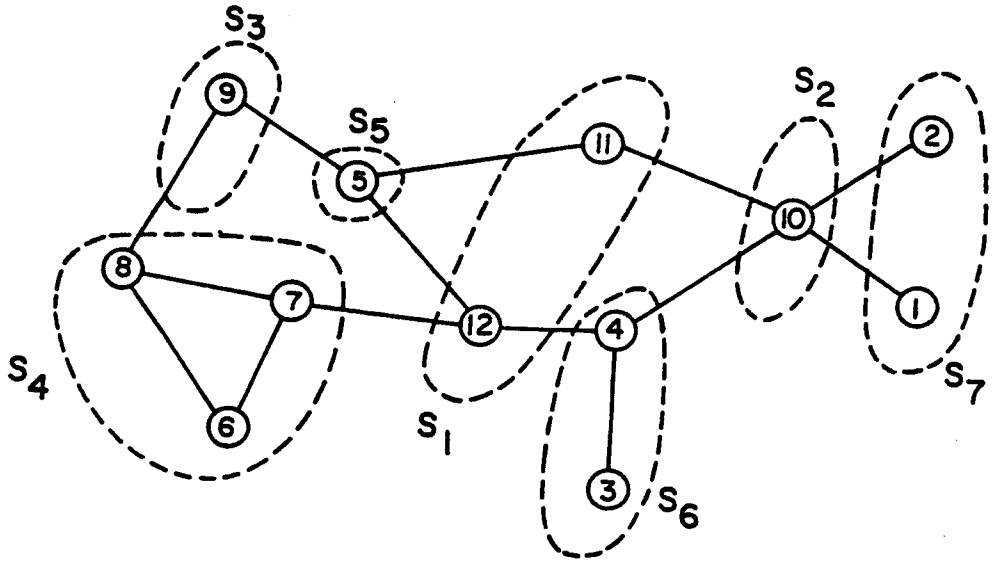
2. If V = $\emptyset$, go to step 4. Otherwise, let G(T) be a connected component of G(V) and set p = p + 1. If $|T| \leq \mu$, set $S_p = T$; otherwise, find S_p, a minimal separator of G(T) which disconnects it into two or more components of approximately equal size.

3. Set $V = V - S_p$, $n = n - |S_p|$, and go to step 2.

4. Set $P = \{Y_1, Y_2, \ldots, Y_p\}$, where $Y_i = S_{p+1-i}$, i = 1, 2, ..., p.

Apart from the inclusion of the threshold μ, and the omission of any specific labelling strategy for each S_p, our implementation corresponds exactly to that described by George and Liu [1978, pp. 1054-1060], so the reader is referred there for details. For our purposes, any ordering compatible with P is acceptable, and the choice of μ is discussed in Section 3. It should be obvious that μ governs the relative "completeness" of the dissection procedure. An example of a nested dissection partitioning along with a compatible ordering is given in Figure 2.1.

In order to avoid unnecessarily complicated notation in the following section, we assume from now on that A has been reordered by columns and that the Y_i simply consist of the appropriate consecutive subsequences of the first n integers. In other words, the nodes of Y_i have been replaced by their labels.



$\mu = 3$

$p = 7$

$P = \{Y_1, Y_2, \ldots, Y_7\}$ where $Y_i = S_{8-i}$.

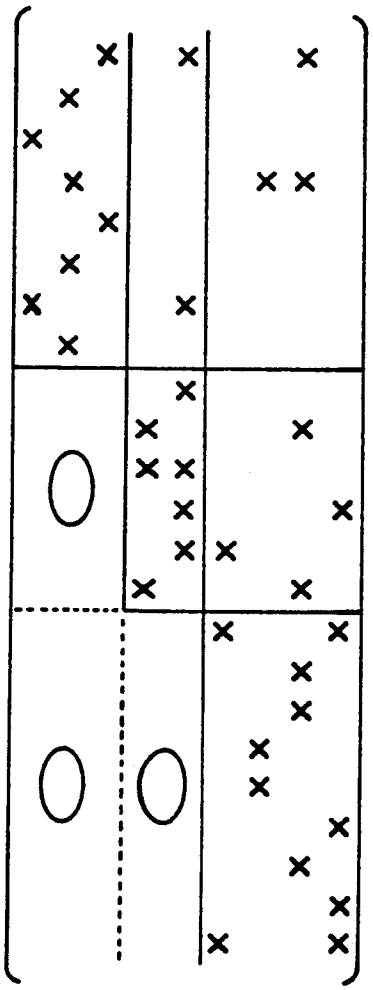
Figure 2.1 An example of a nested dissection partitioning of a graph and an induced numbering of its nodes.

We now define a partitioning $R = \{Z_1, Z_2, \ldots, Z_p\}$ of the row numbers of A, induced by the column partitioning P, as follows. Let

$$Z_i = \left\{ k \mid \exists j \in Y_i \ni a_{kj} \neq 0 \right\} - \left\{ \bigcup_{\ell=1}^{i-1} Z_\ell \right\}, \quad i = 1,2,\ldots, p,$$

with $Z_0 = \emptyset$. The manner in which the Z_i are defined above implies that in general, for a sparse matrix A, if the rows of A are permuted so that those specified by Z_i appear above those specified by Z_{i+1}, then A will have

block upper trapezoidal form, as depicted in Figure 2.2, for three diagonal blocks. Again, to keep the presentation simple, we assume that the rows of A have been relabelled so that the Z_i consist of consecutive integers, with those in Z_i preceding those in Z_{i+1}. We denote the submatrices of A by A_{ij}, $1 \leq i, j \leq p$, and our objective is to find a form for A such that $A_{ij} = 0$ for $i > j$.



A

$Y_1 = \{1,2,3\}$ $\quad Y_1' = \{5,7,8\}$

$Y_2 = \{4,5\}$ $\quad Y_2' = \{6,8,9\}$

$Y_3 = \{6,7,8,9\}$ $\quad Y_3' = \emptyset$

$Z_1 = \{1,2,\ldots,8\}$

$Z_2 = \{9,10,\ldots,14\}$

$Z_3 = \{15,16,\ldots,23\}$

$\mathcal{P} = \{Y_1, Y_2, Y_3\}$

$\mathcal{R} = \{Z_1, Z_2, Z_3\}$

Figure 2.2 A 23 by 9 sparse matrix A partitioned by columns according to $\mathcal{P}$ and the induced row partitioning $\mathcal{R}$.

As mentioned earlier, this process of dissection of the problem variables into independent subsets may arise more or less automatically or may be carried out by some means other than the one proposed above.

For example, in geodesy, observations between or among control points are collected and tabulated by geographic region, so the variables (coordinates) and observations are naturally arranged in a hierarchical structure. Moreover, automatic subdivision can be implemented on the basis of specifying latitude and longitude boundaries as separators for the coordinate sets. For a more detailed description of this process, called Helmert blocking by geodesists, see Golub and Plemmons [1980] and the references cited therein.

In the analysis of structures, the variables in the model are often subdivided according to subassemblies, with each component being processed by an individual design group. This may occur at several levels, leading to "multi-level substructuring." This partitioning procedure together with the natural factor formulation of the finite element method (Argyris et al [1978]) leads to a sparse least squares problem having block upper trapezoidal structure, as illustrated in Figure 2.2.

A special case of the block upper trapezoidal form is the so-called block angular form, where $A_{ij} = 0$ unless $j = i$ or $j = p$. This form arises naturally in many mathematical programming contexts, but more important from our point of view is that Weil and Kettler [1971] have provided a heuristic algorithm for permuting a general sparse matrix into block angular form. We have not used their algorithm, but in some contexts it might serve as an alternate "preprocessor" (partition generator) to the one proposed in this section. Golub and Plemmons [1980]

have exploited this block angular form structure in connection with computing orthogonal decompositions of problems arising in geodesy.

2.3 Reduction of A Using Auxiliary Storage

Before describing in detail how the partitioning P is used in exploiting auxiliary storage, we first review the basic computational procedure, without any reference to the actual implementation. For definiteness, we assume $|P| = 3$ and that A has the form shown in Figure 2.3. The computation consists of p major steps; the i-th step results in the generation of $|Y_i|$ rows of the upper triangular factor R of A. During the i-th step, the columns of Y_i and a subset of columns from $\bigcup_{j=i+1}^{p} Y_j$ are involved in the computation. We denote the column numbers of this subset by Y_i'.

The computation involved in the first step is depicted in Figure 2.3, and all the other major steps are similar, generating the sequence of successively smaller matrices A_1, A_2, A_3, ..., A_p. At the first step, which transforms $A = A_1$ to A_2, the matrix $[A_{11} \vdots \tilde{A}_1]$ is reduced to upper triangular form using Algorithm 1 described in section 1.4. The matrix $\tilde{A}_1$ consists of those columns of A_{12} and A_{13} which are non-null. (Thus, $\tilde{A}_1$ is simply a "compressed" version of A_{12} and A_{13}.) The first $|Y_1|$ columns of the resulting upper triangular matrix are the first $|Y_1|$ columns of R; the remaining columns are "compressed", bearing the same relationship to R as $\tilde{A}_1$ does to $[A_{12} \vdots A_{13}]$. The rows corresponding to these latter columns are "expanded" and put on the top of the second row-block of A, yielding A_2. The next step of the computation is depicted in Figure 2.4, and the final step has no special features.

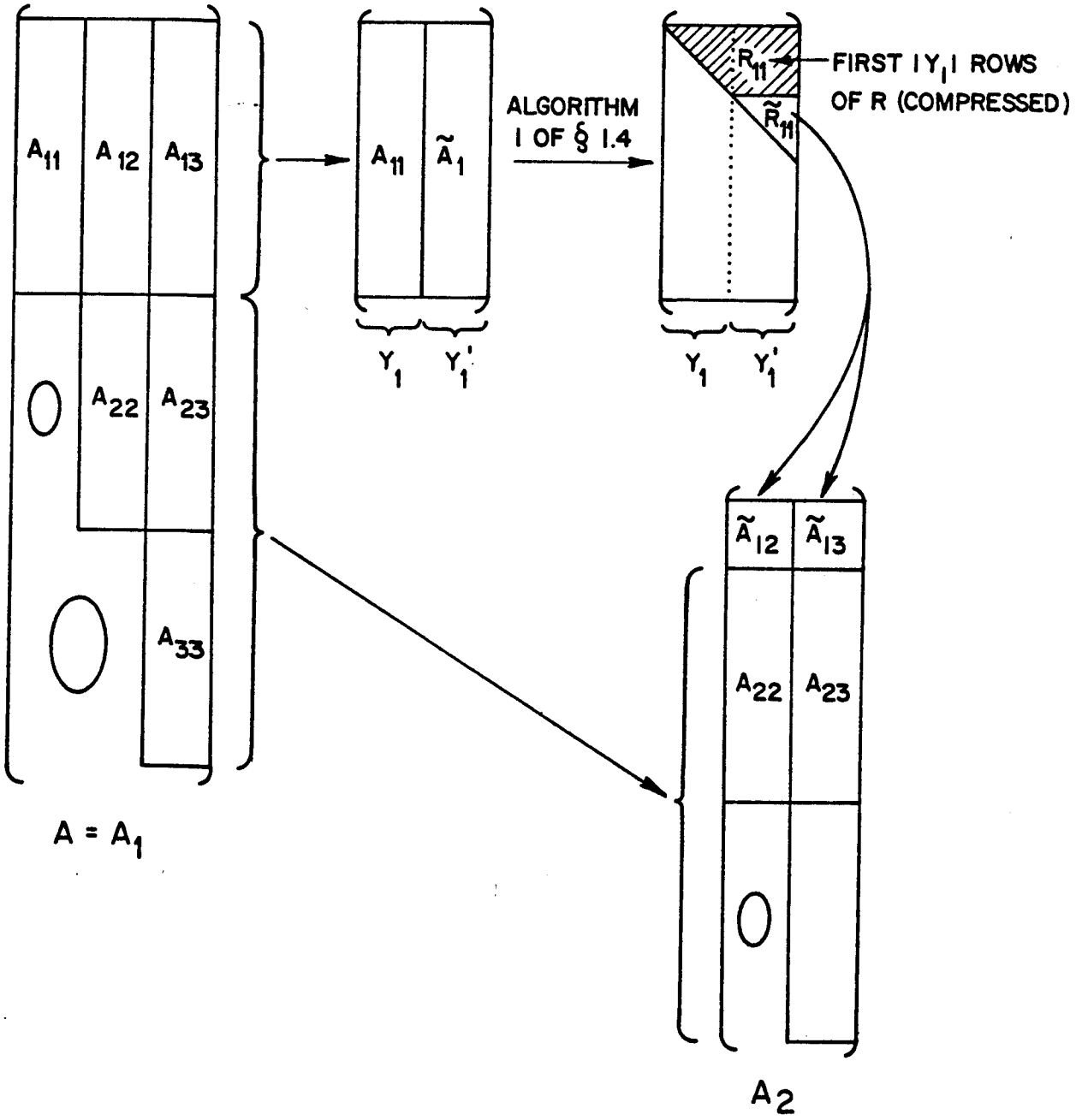


Figure 2.3 Pictorial description of the first step of the $|P|$ step reduction of A to upper trapezoidal form.

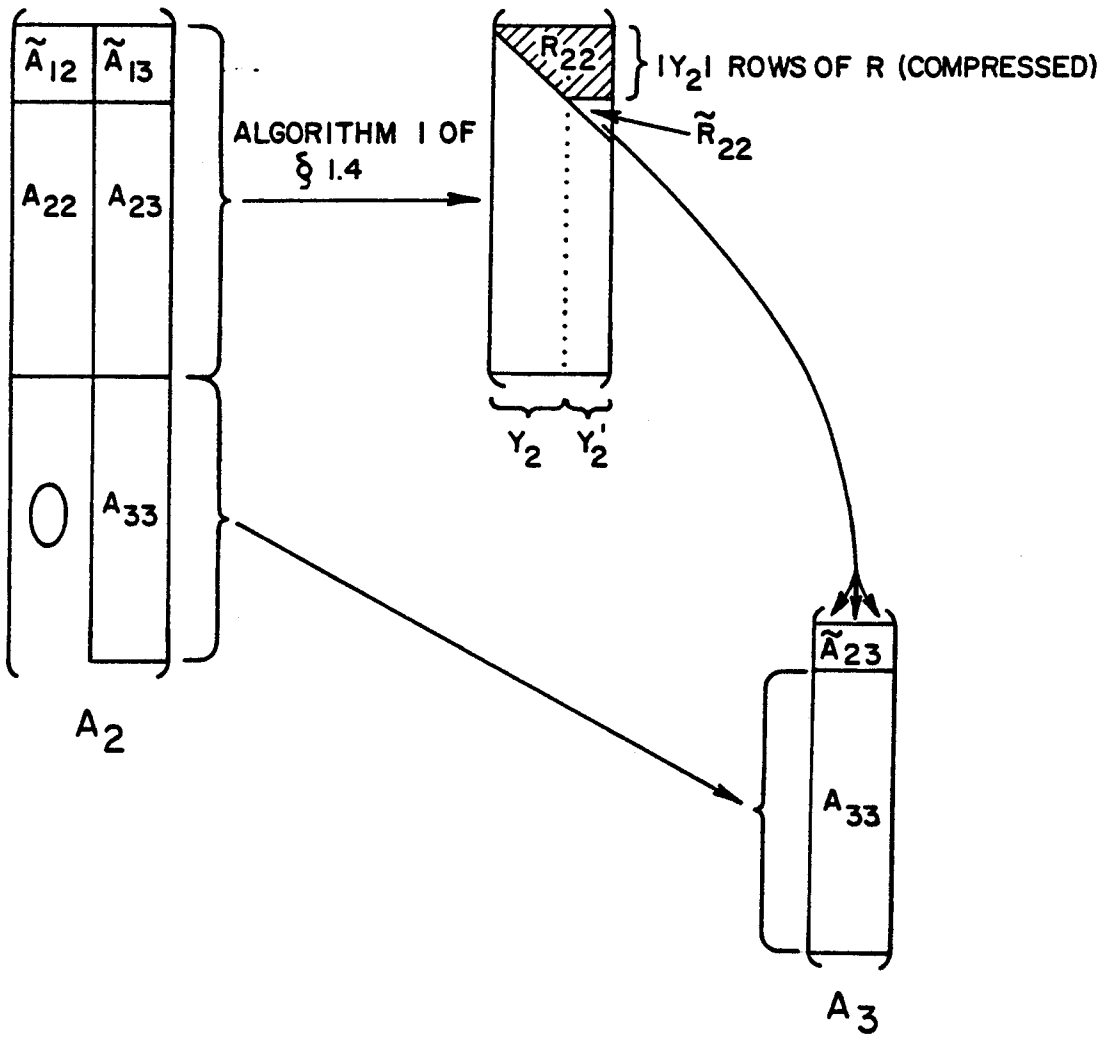


Figure 2.4 Depiction of the second step of the block-reduction of A to upper trapezoidal form.

Thus in the general case, after i-1 steps of the reduction process, the matrix A_i will have the form shown in Figure 2.5.

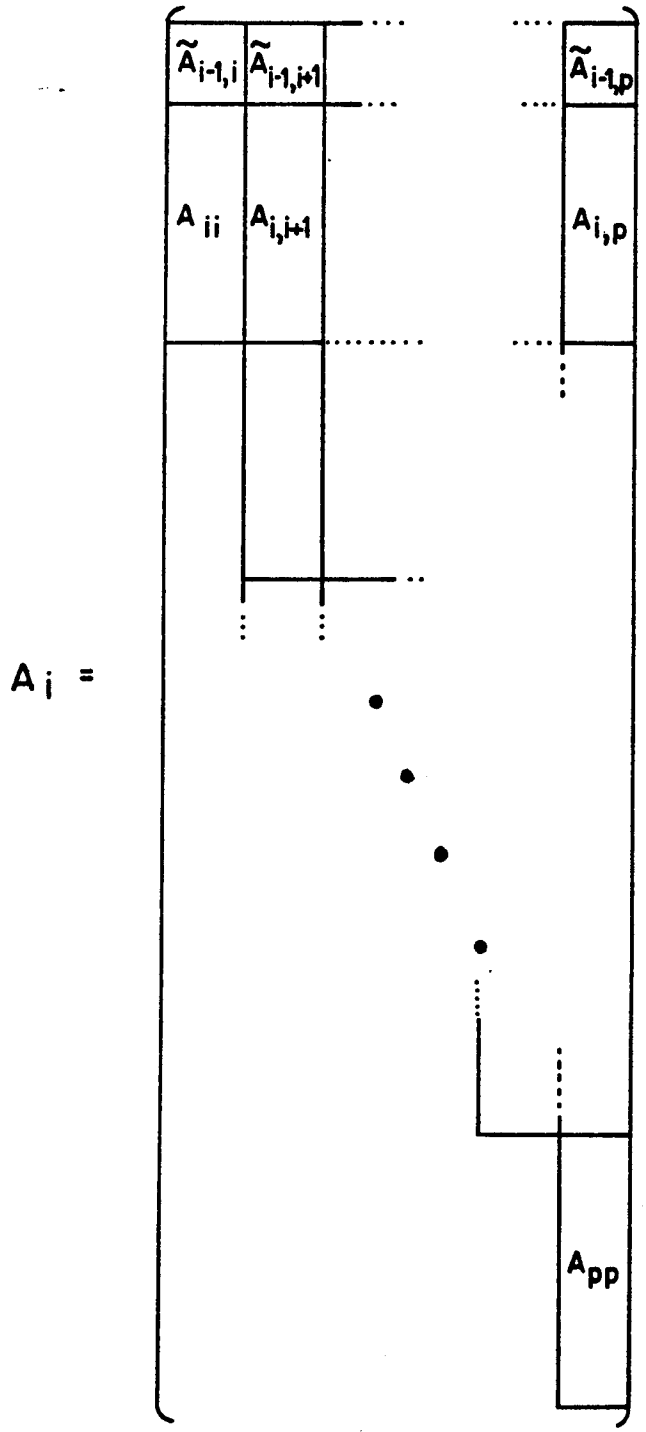
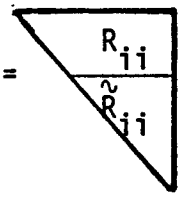


Figure 2.5 Block structure of A_i.

Note that since we assume that the rows of A_i are stored on auxiliary storage at all times, the only storage needed at the i-th stage of the computation is that required for the presumably sparse upper triangular matrix R_i, where

$R_i =$ 

Another important practical observation is that in Algorithm 1 of Section 1.4, and in the one that is described in the sequel, there is no restriction on the order that the rows of any of the A_i are stored so that any beneficial row ordering may be used.

In what follows it is helpful to have names for certain subsets of the rows of A_i. We define the set of rows of A_i having nonzeros in columns which intersect Y_i by $\overline{Z}_i$, and we use Z_i' to denote the remaining rows of A_i. Thus, it is precisely those rows in $\overline{Z}_i$ which are involved in the i-th major step of the computation.

For simplicity, we have not included the right hand side in our Figures 2.3-2.5, but we intend that it be processed simultaneously with the rows of A, and carried along in parallel. In particular, throughout this paper, when we refer to processing a "row of A" or an "equation," we implicitly include the corresponding element of b, the weight (if any), and so on. Our program allows for weights, and in order to handle them uniformly (i.e., to avoid distinguishing between virgin rows in A_i and those that have been transformed), a weight of 1 is associated with transformed equations.

We are now ready to describe the algorithm for reducing A, which involves the use of four files. These files may be stored on tapes, disks, drums; only serial access to the files is necessary. The files are:

1. R-file: accepts the rows of R as they are computed.
2. C-file: (current file) at the beginning of the i-th major step of the computation, contains the rows of A_i in arbitrary order.
3. $\overline{Z}$-file: during the i-th step of the computation, contains the rows in the set $\overline{Z}_i$; that is, those rows of A that are involved in the computation at the i-th major step.
4. Z'-file: at the beginning of the i-th step, this file is empty. The C-file is read, and split into the $\overline{Z}$-file and this file, which receives the rows of A_i in the set Z_i'. After the computation of R_i is performed, the rows of $\tilde{R}_{ii}$ are written on this file, and the Z'-file becomes the C-file for the next step of the computation. (It now contains the rows of A_{i+1}.)

Thus the $\overline{Z}$-file is a scratch file, and the roles of the C-file and Z'-file alternate at each succeeding major step of the computation.

Algorithm 3 Block Reduction of A to Upper Triangular Form

For i = 1, 2, ..., p do the following:

1. Rewind the C-file, $\overline{Z}$-file, and Z'-file. Read the equations from the C-file one by one, and for each do the following:

1.1. If the equation number is in $\bar{Z}_i$, write the equation on the $\bar{Z}$-file and record the structure it contributes to the normal equations matrix $H_i^T H_i$, corresponding to the matrix $H_i \equiv [A_{ii} \vdots \tilde{A}_i]$.

1.2 If the equation is in Z_i', write the equation on the Z' file.

2. Order the equations so that $H_i^T H_i$ suffers low fill-in, restricting the ordering so that the variables in Y_i' appear last.

3. Create the data structure for R_i.

4. Rewind the $\bar{Z}$-file. Read the equations from it and compute R_i, also applying the transformations to b.

5. Write the rows of R_{ii} on the R-file along with the transformed elements of b.

6. Write the rows of $\tilde{R}_{ii}$ on the Z'-file, along with the transformed elements of b.

7. Reverse the names of the C-file and Z'-files.

It should be clear that the combination of the structure recording part of Step 1.1 along with Steps 2, 3, and 4 is essentially Algorithm 1, described in Section 1.4. The only modification is the adjustment of the ordering provided by Algorithm 1 so that variables in Y_i' appear last; all others remain in their same relative position. The ordering of the variables of Y_i that is provided in Step 2 is recorded so that the rows of R written on the R file can be processed in the correct order in the back substitution. (Note that Algorithm 2 in Section 2.1 only provided the partitioning and did not provide an ordering for each Y_i.)

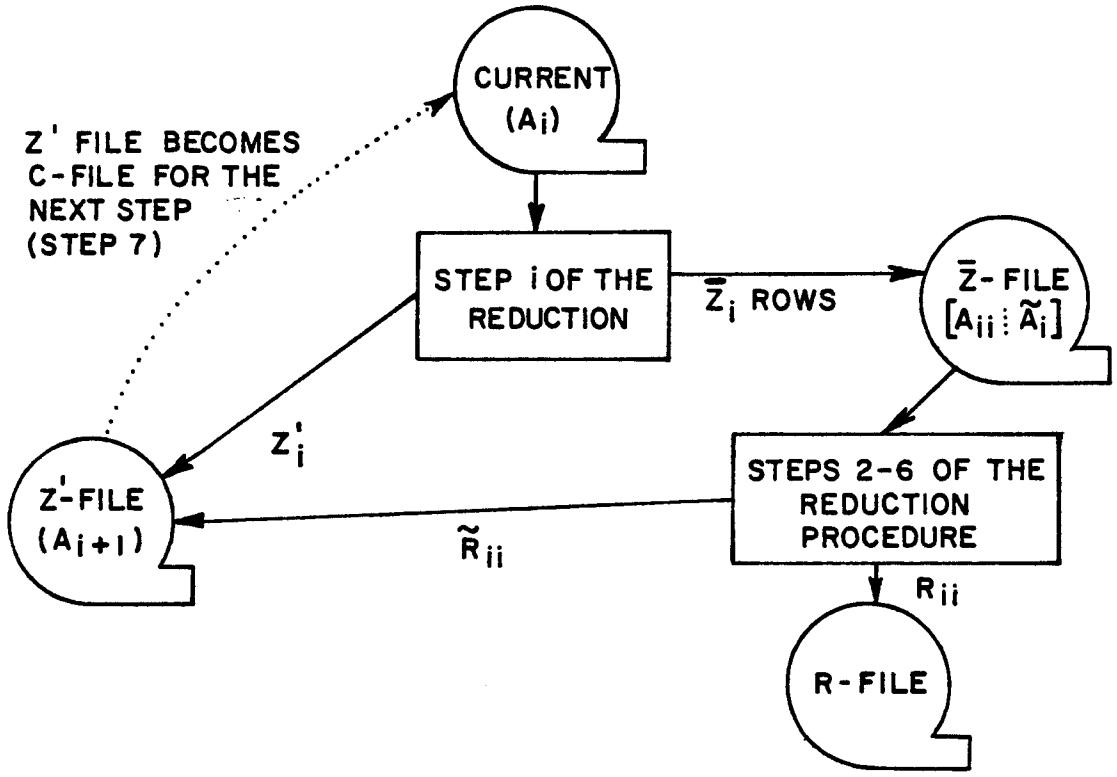


Figure 2.6 Diagram of the data flow of the algorithm for reducing A to upper trapezoidal form.

At the conclusion of the reduction of A to upper trapezoidal form, along with the simultaneous reduction of b, the rows of R and corresponding right-hand side elements (y in (1.3) of Section 1) will be on the R-file, with the "write head" positioned at the end of the last record. The following simple back substitution is then used to compute x.

For i = n, n - 1, ..., 1 do the following:

1. backspace the R-file;
2. read the i-th row of R and corresponding right-hand side element y_i and compute x_i;
3. backspace the R-file.

3. Numerical Experiments and Observations

3.1 Introduction

This section contains results of some experiments performed using an implementation of the algorithms described in Section 1.4 and Section 2. Our test problems are of various sizes (ranging from 1,444 equations and 400 unknowns up to 17,946 equations and 4,554 unknowns) and of two different types. One class of problems is typical of those that would arise in the natural factor formulation of the finite element method (Argyris and Brönlund [1975]), and the second class typifies those arising in geodetic adjustment problems (Golub and Plemmons [1980]). The descriptions of the problems we provide include only the details necessary to characterize their size and structure; for important information on the physical origins and their mathematical models, the reader is referred to the references.

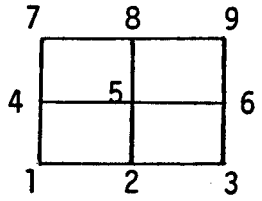
3.2 Test Problems

Our finite element test problems are associated with a q by q grid consisting of $(q-1)^2$ small squares, as shown in Figure 3.1 with $q = 3$. Associated with each of the $n = q^2$ grid points is a variable, and associated with each small square are four equations (observations) involving the four corner grid points (variables) of the square. Thus, the associated coefficient matrix is $n = q^2$ by $m = 4(q-1)^2$, as illustrated in Figure 3.1.

Our second set of test problems were also derived from a q by q mesh, but of a rather different type. The purpose here is to construct problems typical of those arising in geodetic adjustments. The mesh can be viewed as being composed of q^2 "junction boxes," connected to

	1	2	3	4	5	6	7	8	9
1	x	x		x	x				
2	x	x		x	x				
3	x	x		x	x				
4	x	x		x	x				
5		x	x		x	x			
6		x	x		x	x			
7		x	x		x	x			
8		x	x		x	x			
9				x	x		x	x	
10				x	x		x	x	
11				x	x		x	x	
12				x	x		x	x	
13					x	x		x	x
14					x	x		x	x
15					x	x		x	x
16					x	x		x	x

A



Finite element grid

Figure 3.1 A 3 by 3 finite element grid and its associated 16 by 9 least squares coefficient matrix arising in the natural factor formulation of the finite element method.

their neighbors by chains of length ℓ, as shown in Figure 3.2 where $q = 3$ and $\ell = 4$. There are two variables associated with each of the $5q^2 + 2q(q-1)(3(\ell-1)+1)$ vertices in the mesh, n observations associated with each pair of nodes joined by an edge (involving four variables),

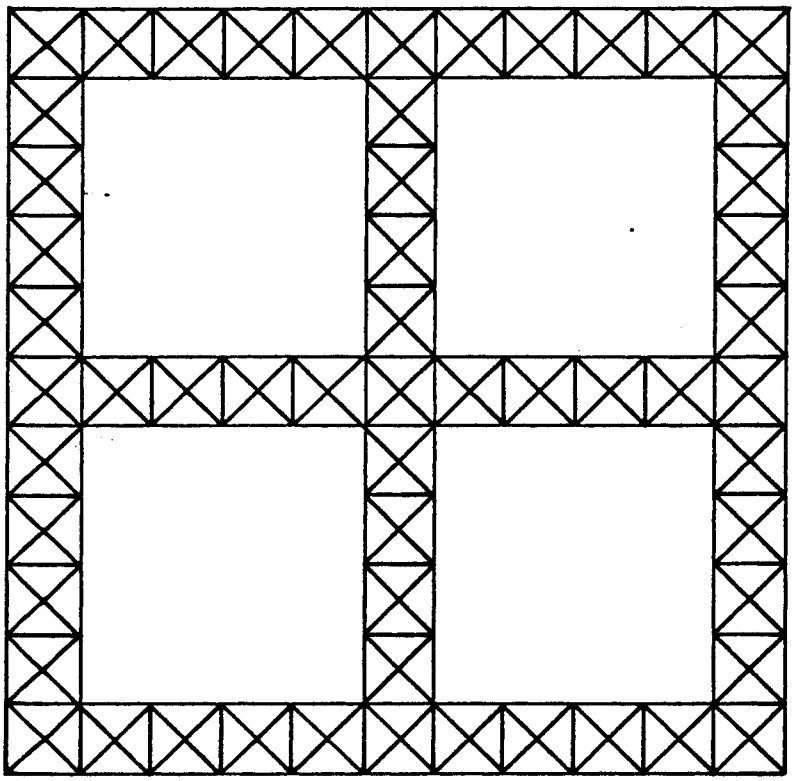


Figure 3.2 Idealization of a 3 by 3 geodetic network having connecting survey chains of length 4.

and η observations associated with each triangle in the mesh (involving six variables). Thus, the associated least squares problems have $n = 10q^2 + 4q(q-1)(3(\ell-1)+1)$ variables, and $m = (12q^2 + 2q(q-1)(11\ell-1))\eta$ observations. In typical real problems, ℓ is around 5 or 6, so we set $\ell = 5$ in our experiments, yielding $n = 62q^2 - 52q$ and $m = \eta(120q^2-108q)$. In our experiements we set $\eta = 2$, yielding $m/n \approx 4$ for large q.

3.3 Numerical Experiments

Since one of our main objectives is to provide a means of solving very large problems on computers having limited main storage, our first experiments involve solving a sequence of problems of increasing size, using a fixed amount of array storage. Of course, as the problems increase in size, the dissection process which provides the partitioning (Algorithm 2 in Section 2) must be allowed to proceed further, creating more blocks. The results of these experiments are summarized in Table 3.1. All tests were made on an IBM 3033 at the Oak Ridge National Laboratory. The times reported are in seconds and the storage in words.

The factor and solve time includes the orthogonal decomposition time in applying Algorithm 1, together with the final back substitution time in computing the least squares solution. The total elapsed time also includes I/O processing and represents the total amount of IBM 3033 time in solving the problem.

The maximum storage used includes all array storage, storage for permutations, bookkeeping, etc., in words on the IBM 3033.

Our second set of experiments was designed to demonstrate the influence of the block size limit μ on execution times and total storage requirements. We solved one moderately large-scale problem from each of the two classes a number of times, with different values of μ, leading to different values for the resulting number of blocks p. The results of these experiments are summarized in Table 3.2.

1. Geodetic Network Problems

Unknowns n	Equations m	Junction boxes q	Block size limit μ	Number blocks p	Maximum storage used	Factor and solve time in seconds	Total elapsed time in seconds
402	1,512	3	500	1	7,430	1.69	4.69
1,290	4,920	5	1,500	1	25,367	7.21	22.77
2,674	10,248	7	1,500	3	32,070	22.27	57.69
4,554	17,469	9	1,500	7	30,141	46.18	107.40

2. Finite Element Grid Problems

Unknowns n	Equations m	Nodes q	Block size limit μ	Number blocks p	Maximum storage used	Factor and solve time in seconds	Total elapsed time in seconds
400	1,444	20	500	1	10,029	3.10	5.97
1,225	4,624	35	1,000	3	30,547	20.78	32.23
2,500	9,604	50	1,000	7	28,462	72.98	96.10
4,225	16,384	65	500	23	28,987	142.81	208.20

Table 3.1. Summary of test results showing storage and execution times for a sequence of problems, where the maximum total amount of fast storage is fixed at about 30,000 words.

1. Geodetic Network

n = 2,674 unknowns

m = 10,248 equations

Block size limit μ	Number blocks p	Maximum storage used	Factor and solve time in seconds	Total elapsed time in seconds
3,000	1	53,858	17.73	71.40
1,500	3	32,070	22.27	57.69
800	7	17,966	23.66	52.71
500	18	13,699	27.56	61.80

2. Finite Element Grid

n = 2,500 unknowns

m = 9,604 equations

Block size limit μ	Number blocks p	Maximum storage used	Factor and solve time in seconds	Total elapsed time in seconds
2,000	3	73,002	72.58	106.80
1,500	5	46,178	68.73	96.00
1,000	7	28,462	72.98	96.10
500	15	23,332	69.11	100.80

Table 3.2 Summary of results for the solution of a single problem from each class, using different levels of blocking.

3.4 Observations

1. In Table 3.1, the factor and solve times, as well as the total elapsed times, increase in a roughly linear fashion as the problem sizes go up under the constraint of a fixed maximum amount of in-core storage.

2. For the two problems represented in Table 3.2, the factor and solve times are relatively insensitive to the block size limit μ. These times gradually increase for the geodetic network and oscillate for the finite element grid. Also, the maximum storage used drops considerably at first and then levels out as the block size limit decreases in each problem.

3. The fact that finite element problems generally result in a less sparse observation matrix than geodetic network problems has the obvious result. In each of our tables the storage and execution times are larger for the finite element problems.

4. One possible disadvantage of the use of our automatic blocking program (Algorithm 2) is that the entire structure of A^TA must be initially stored in-core in order to apply the nested dissection scheme. However, in practice this would not often be a serious limitation since some preliminary blocking is usually provided for the larger problems as a by-product of the modelling procedure (finite element analysis) or data acquisition (geodesy).

5. In each of our problems the time required for the automatic blocking provided by Algorithm 2 turns out to be small in comparison to the total elapsed time. These automatic blocking times are not reported separately in Tables 3.1 and 3.2; however, Algorithm 2

requires only .78 seconds out of a total elapsed time of 208.20 seconds for our largest problem.

6. Our scheme is storage effective. In each test problem the maximum storage used is a modest multiple of the number of variables n. In summary, our numerical experiments demonstrate that the approach taken in this paper can be used to solve large-scale least squares problems using a relatively small amount of in-core storage.

REFERENCES

J. H. Argyris and O. E. Brönlund [1975], "The natural factor formulation of the stiffness matrix displacement method," Computer Methods in Applied Mech. and Eng. 5, 97-119.

J. H. Argyris, T. L. Johnsen and H. P-Mlejnek [1978], "On the natural factor in nonlinear analysis," Computer Methods in Applied Mech. and Eng. 15, 389-406.

J. K. Avila and J. A. Tomlin [1979], "Solution of very large least squares problems by nested dissection on a parallel processor," Proc. Computer Science and Statistics: Twelfth Annual Symposium on the Interface, Waterloo, Canada.

A. Björck [1976], "Methods for sparse linear least squares problems," in Sparse Matrix Computations, J. R. Bunch and D. J. Rose, eds., Academic Press, NY, 177-194.

A. Björck and I. S. Duff [1980], "A direct method for the solution of sparse linear least squares problems," Linear Algebra and Its Applic., to appear.

I. S. Duff and J. K. Reid [1976], "A comparison of some methods for the solution of sparse overdetermined systems of linear equations," J. Inst. Math. Applic. 7, 267-280.

A. George and M. T. Heath [1980], "Solution of sparse least squares problems using Givens rotations," Linear Algebra and Its Applic., to appear.

A. George and J. Liu [1978], "An automatic nested dissection algorithm for irregular finite element problems," SIAM J. Numer. Anal. 15, 1053-1069.

A. George and J. Liu [1979], "The design of a user interface for a sparse matrix package," ACM Trans. on Math. Software 5, 134-162.

A. George and J. Liu [1980], Computer solution of large sparse symmetric positive definite systems of linear equations, Prentice Hall, Inc., Englewood Cliffs, NJ.

A. George, W. Poole and R. Voigt [1978], "Incomplete nested dissection for solving n by n grid problems," SIAM J. Numer. Anal. 15, 662-673.

P. Gill and W. Murray, [1976], "The orthogonal factorization of a large sparse matrix," in Sparse Matrix Computations, J. R. Bunch and D. J. Rose, eds., Academic Press, NY, 201-212.

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

G. H. Golub and R. J. Plemmons [1980], "Large scale geodetic least squares adjustments by dissection and orthogonal decomposition," Linear Algebra and Its Applic., to appear.

G. H. Golub, F. T. Luk and M. Pagano [1979], "A sparse least squares problem in photogrammetry," Proc. Computer Science and Statistics: Twelfth Annual Symposium on the Interface, Waterloo, Canada.

G. Hachtel [1976], "The sparse tableau approach to finite element assembly," in Sparse Matrix Computations, J. R. Bunch and D. J. Rose, eds., 344-364.

G. B. Kolata [1978], "Geodesy: Dealing with an enormous computer task," Science 200, 421-422, 466.

G. Peters and J. H. Wilkinson [1970], "The least squares problem and pseudo-inverses," The Computer Journal 12, 309-316.

P. Vanicek, M. R. Elliott and R. O. Castle [1979], "Four-dimensional modeling of recent vertical movements in the area of the Southern California uplift," Tectonophysics 52, 287-300.

R. L. Weil and P. C. Kettler [1971], "Rearranging matrices to block-angular form for decomposition (and other) algorithms," Management Science 18, 98-108.