The design and implementation of a package for sparse constrained least squares problems

Alan George
Esmond Ng

Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada

CS-85-39
August 1985

The Design and Implementation of a Package for Sparse Constrained Least Squares Problems*

Alan George

Esmond Ng

Department of Computer Science
University of Waterloo
Waterloo, Ontario, CANADA

August 22, 1985

ABSTRACT

This report describes the design and implementation of a mathematical software package for solving large sparse overdetermined systems of linear equations, possibly subject to sparse linear constraints. The algorithm used is due to Bjorck [1], which in turn is derived from basic algorithms described in [3] and [14]. The algorithm is very general in that rank deficiency is allowed in both the equations and the constraints, and the constraints need not be consistent. The package is implemented in a portable subset of ANSI-66 Fortran, and is distributed under license by the University of Waterloo.

* Research supported in part by the Canadian Natural Sciences and Engineering Research Council under grant A8111.

Table of Contents

1. Introduction 1

2. Reduction of sparse matrices to upper trapezoidal form using Givens rotations 4

   2.1. The basic algorithm 4

   2.2. Effect of column ordering 6

   2.3. Effect of row ordering 6

   2.4. Summary 7

3. Data structures in SPARSPAK-B 8

4. An algorithm for solving general sparse constrained linear least squares problems 9

   Stage 1: reduce sparse constraint equations 10

   Stage 2: process dense constraint equations and compute residual of the sparse constraint equations 13

   Stage 3: reduce sparse least squares equations 18

   Stage 4: process (modified) dense constraint equations and dense least squares equations 21

   Stage 5: compute the minimal norm solution 26

5. The design of SPARSPAK-B 31

   5.1. The interface 31

   5.2. The main interface subroutines 33

      Step 1: initialization 33

      Step 2: problem input 33

Step 3: column ordering 34
Step 4: row ordering 34
Step 5: solution 35
5.3. Other interface subroutines 35
Residual computation 35
Statistics 35
Save and restart 36
6. Concluding remarks 37
7. References 38

1. Introduction

In this report we consider the problem of solving the large sparse linear least squares problem

$$\min_{\beta} \quad \| X\beta - y \|_2 \quad , \tag{1.1}$$

where X is $m \times n$, with $m \geq n$, and y and β are vectors of appropriate dimensions. A basic technique for solving (1.1) is to transform X to upper trapezoidal form using orthogonal transformations, such as Givens rotations or Householder transformations [12]. That is, we first determine an $m \times m$ orthogonal matrix Q and an $n \times n$ upper triangular matrix R such that

$$QX = \begin{bmatrix} R \\ O \end{bmatrix} \quad .$$

We shall assume for the moment that X has linearly independent columns. Thus, R is nonsingular. Since

$$\| X\beta - y \|_2 = \| Q(X\beta - y) \|_2 = \| \begin{bmatrix} R \\ O \end{bmatrix} \beta - Qy \|_2 \quad ,$$

the least squares solution to (1.1) is then given by the solution to the triangular system

$$R\beta = c \quad ,$$

where c contains the first n elements of Qy.

In [3], George and Heath describe a scheme for transforming a sparse X to upper trapezoidal form by applying Givens rotations to the rows of X. As we shall see in the next section, efficient implementation of this algorithm relies heavily on the assumption that the symmetric positive definite matrix $X^T X$ and its Cholesky factor are sparse. This assumption usually holds when X is sparse, and in instances in which the assumption is false, it turns out that there are usually a few relatively dense rows in X that cause $X^T X$ and its Cholesky factor to be dense. A technique which has been found to be effective for handling such instances is to partition X into

$$X = \begin{bmatrix} A \\ B \end{bmatrix}$$

so that A and B contain respectively the sparse and dense rows of X. We assume A is $m_1 \times n$ and B is $m_2 \times n$. Thus $m = m_1 + m_2$. Let y be partitioned accordingly

$$y = \begin{bmatrix} y_A \\ y_B \end{bmatrix} ,$$

and assume A has full column rank. Suppose we first solve the sparse linear least squares problem

$$\min_{\beta} \| A\beta - y_A \|_2 \tag{1.2}$$

using the approach described in [3]. Then the solution to (1.1) can be obtained by modifying the solution to (1.2) using B and y_B. Algorithms for performing the modification are described in [3,14]. These algorithms assume that the sparse submatrix A has linearly independent columns, but of course this need not be true in general even if X has full rank.

In some applications, there may be some a-priori (linear) relationships among some of the unknown variables $\beta_1, \beta_2, \cdots, \beta_n$. Thus it is appropriate to consider the *constrained* linear least squares problem

$$\begin{aligned} &\min_{\beta} \| X\beta - y \|_2 \\ &\text{subject to} \quad Y\beta = z \quad , \end{aligned} \tag{1.3}$$

where Y is $p \times n$ and z is a p-vector. As in the previous case, some of the least squares equations may be sparse and some may be dense. Similarly, the constraint equations may also be partitioned into sparse and dense portions

$$Y = \begin{bmatrix} E \\ F \end{bmatrix} ,$$

where E is $p_1 \times n$ and F is $p_2 \times n$. That is, $p_1 + p_2 = p$. The p-vector z will be partitioned accordingly

$$z = \begin{bmatrix} z_E \\ z_F \end{bmatrix} .$$

In [14], Heath has proposed an algorithm for solving (1.3). The constraint equations are assumed to be consistent and are treated as dense, and the algorithm requires A to have full column rank.

In some cases, the constraint equations may be inconsistent. Thus, in general, it is more appropriate to consider the following general constrained linear least squares problem

$$\min_{\beta \in \Omega} \| X\beta - y \|_2$$
$$\text{where} \quad \Omega = \{ \beta \mid \beta \text{ minimizes } \| Y\beta - z \|_2 \} \quad . \tag{1.4}$$

Again, each of the least squares equations and the constraint equations may be either sparse or dense. In [1], Bjorck has proposed an algorithm for solving (1.4). This algorithm not only handles sparse and dense least squares equations and constraint equations, but also makes no assumptions about the ranks of the matrices X, Y, A, B, E, and F.

The purpose of this report is to describe the design and implementation of a FORTRAN package SPARSPAK-B for solving (1.4) using Bjorck's algorithm. The package SPARSPAK-B is efficient (in storage requirement and execution time) when the following assumptions hold. First, the number of dense least squares equations and constraint equations is small; that is, m_2 and p_2 are small. Second, the rank deficiency in $\begin{bmatrix} A \\ E \end{bmatrix}$ is small. If m_2 and p_2 are large, or if the rank deficiency in $\begin{bmatrix} A \\ E \end{bmatrix}$ is large, the overhead in both storage requirements and execution times may be so large that it may be more beneficial to treat the whole problem as dense and to solve (1.4) using a standard code for dense problems.

It should be noted that Bjorck's algorithm works even if $m < n$ or $p > n$. However, if $m \ll n$ and $p \leq n$, the storage and execution time required may also be large and it may be more advantageous to use other schemes to solve (1.4). In [4], alternate algorithms have been proposed to handle the case in which $m \ll n$ and $p = 0$. It has also been demonstrated that considerable improvements can be achieved by using these alternate algorithms.

An outline of the report is as follows. In Section 2 we review the basic approach used in reducing a sparse matrix to upper trapezoidal form, and in Section 3 we describe the data structures employed in SPARSPAK-B. In Section 4, Bjorck's algorithm and its implementation are described. In this section, specific details are provided concerning the way information is stored and managed in SPARSPAK-B. A set of interface subroutines which insulate the user from the internal structure of SPARSPAK-B is described in Section 5, and finally, some concluding remarks are provided in Section 6.

2. Reduction of sparse matrices to upper trapezoidal form using Givens rotations

2.1. The basic algorithm

Let X be an $m \times n$ sparse matrix with full column rank and consider the solution of the unconstrained linear least squares problem

$$\min_{\beta} \| X\beta - y \|_2 \quad . \tag{2.1}$$

A standard technique for solving (2.1) is to reduce X to upper trapezoidal form using orthogonal transformations

$$QX = \begin{bmatrix} R \\ O \end{bmatrix} \quad ,$$

where R is an $n \times n$ upper triangular matrix and Q is an $m \times m$ orthogonal matrix. One way of computing R is as follows. Let $R^{(0)} = O$ be the $n \times n$ zero matrix. A sequence of $n \times n$ upper triangular matrices

$$\{ R^{(1)}, R^{(2)}, \cdots, R^{(m)} \}$$

is then computed, where $R^{(k)}$ is obtained by annihilating the nonzeros of the k-th row of X using Givens rotations constructed from the diagonal elements of $R^{(k-1)}$ [3]. Note that these rotations can be applied simultaneously to the vector y and therefore need not be saved. Furthermore, the rows of X are accessed and processed sequentially. Thus it is not necessary to store X in main memory since the rows can be stored in secondary storage and read in one at a time when they are needed. This makes the approach described above attractive for large and sparse problems. Moreover, in practice, the matrix X is often generated row by row.

Another important aspect of the approach described above is the way in which sparsity is exploited for large problems. Note that

$$X^T X = X^T Q^T Q X = \begin{bmatrix} R^T & O \end{bmatrix} \begin{bmatrix} R \\ O \end{bmatrix} = R^T R \quad .$$

This shows that R is in fact the Cholesky factor of the symmetric positive definite matrix $X^T X$ (apart from possible sign differences in some rows). Assume that $X^T X$ and its Cholesky factor R are sparse. It is well known that the *structure* of R can be determined efficiently from the structure of $X^T X$ [6]. Such a process is commonly

referred to as the *symbolic factorization* of $\boldsymbol{X}^T\boldsymbol{X}$. Knowing the structure of $\boldsymbol{R}$ allows a data structure or storage scheme which exploits the sparsity of $\boldsymbol{R}$ to be set up prior to the numerical computation, which can then be performed using that static storage scheme.

The discussion above only shows that a storage scheme for the final upper triangular matrix $\boldsymbol{R}$ can be set up by symbolically factoring $\boldsymbol{X}^T\boldsymbol{X}$. We also need space to accommodate the nonzeros in the intermediate upper triangular matrices $\boldsymbol{R}^{(k)}$. However it can be shown that a data structure for the final upper triangular matrix $\boldsymbol{R}$ is also large enough to accommodate the nonzeros in each of the intermediate upper triangular matrices $\boldsymbol{R}^{(k)}$ [3,18]. Moreover, the computation can be organized in such a way that $\boldsymbol{R}^{(k-1)}$ can be overwritten by $\boldsymbol{R}^{(k)}$. Hence by using the (static) data structure for $\boldsymbol{R}$, we can then process the rows of $\boldsymbol{X}$ one at a time and compute the upper triangular matrices $\boldsymbol{R}^{(1)}$, $\boldsymbol{R}^{(2)}$, $\cdots$, and $\boldsymbol{R}^{(m)}=\boldsymbol{R}$.

At this point, it is appropriate to summarize the solution process which is proposed by George and Heath [3].

(1) Determine the structure (not the numerical values) of $\boldsymbol{X}^T\boldsymbol{X}$ from that of $\boldsymbol{X}$.

(2) Symbolically factorize $\boldsymbol{X}^T\boldsymbol{X}$, generating a row-oriented data structure for $\boldsymbol{R}$.

(3) Compute $\boldsymbol{R}$ by processing the rows of $\boldsymbol{X}$ one by one using Givens rotations. Apply the Givens rotations to the vector $\boldsymbol{y}$ simultaneously.

(4) Use the upper triangular matrix $\boldsymbol{R}$ and the first n elements of the modified vector $\boldsymbol{y}$ to compute the least squares solution.

A few remarks are in order. First, even though $\boldsymbol{R}$ is (mathematically) the Cholesky factor of $\boldsymbol{X}^T\boldsymbol{X}$, we *never* form $\boldsymbol{X}^T\boldsymbol{X}$ explicitly; $\boldsymbol{R}$ is computed from $\boldsymbol{X}$ using orthogonal transformations (Givens rotations). Second, the data structure is row-oriented because of the way in which the elements of $\boldsymbol{R}$ are accessed. Third, secondary storage can be used easily and in a natural manner. Fourth, since the first two steps in the solution process depend only on the *structure* of $\boldsymbol{X}$, it should be clear that the data structure obtained in step 2 is independent of whether the matrix $\boldsymbol{X}$ has full rank. That is, even if the columns of $\boldsymbol{X}$ are linearly dependent, we can still determine a storage scheme for $\boldsymbol{R}$ by treating $\boldsymbol{X}$ as if the columns were linearly independent. This is reasonable since we only make use of the structure of $\boldsymbol{X}$ at this stage. Then we can compute the upper triangular matrix $\boldsymbol{R}$ using the static data structure, although $\boldsymbol{R}$ is now singular.

Recently, Liu has proposed a new scheme for computing R using Givens rotations [16]. Preliminary experiments indicate that the new approach is comparable with the George-Heath approach in terms of storage requirements, and its execution times are often better than the latter approach. The incorporation of this new scheme into SPARSPAK-B is under investigation.

2.2. Effect of column ordering

Suppose P_c is an $n \times n$ permutation matrix and consider the matrix XP_c. We assume again that X has full column rank. Let $\overline{R}$ denote the upper triangular matrix obtained when XP_c is reduced to upper trapezoidal form using Givens rotations. From our discussion in Section 2.1, we have

$$\overline{R}^T\overline{R} \;=\; (XP_c)^T(XP_c) \;=\; P_c^T(X^TX)P_c \quad .$$

When X^TX is sparse, it is well known that the sparsity of $\overline{R}$ depends not only on the structure of X^TX, but also on the choice of P_c. Thus, in order to reduce the amount of space required to solve the problem, P_c should be chosen so that $\overline{R}$ is as sparse as possible. However, the problem of choosing the *best* P_c turns out to be very difficult [20]. On the other hand, there are many efficient heuristic algorithms for choosing P_c such that $\overline{R}$ will tend to be sparse [3]. In SPARSPAK-B, the column permutation P_c for X is determined using the minimum degree algorithm which is an effective algorithm for generating a P_c so that $\overline{R}$ is sparse. The technique of multiple elimination is employed to improve the execution time [17].

2.3. Effect of row ordering

Suppose P_r is an $m \times m$ permutation matrix and consider the reduction of P_rX to upper trapezoidal form using Givens rotations. Let $\overline{R}$ denote the upper triangular matrix obtained. Note that

$$\overline{R}^T\overline{R} \;=\; (P_rX)^T(P_rX) \;=\; X^T(P_r^TP_r)X \;=\; X^TX \;=\; R^TR \quad .$$

When X has full column rank, $\overline{R}=R$ (at least mathematically). This shows that row permutations P_r have no effect on the structure of the *final* upper triangular matrix R. However, P_r may affect the structures of the intermediate matrices. This may have a significant impact on the number of rotations and operations, and consequently the execution time, required in computing R [10,18].

For certain column orderings, it is possible to characterize row orderings that would tend to reduce the execution times [7,8,9,10]. However, the problem of finding a good P_r to reduce the execution time in general is not well understood. In SPARSPAK-B, several row ordering options are provided. See [11] for details.

2.4. Summary

In this subsection, we summarize the major steps involved in solving a sparse unconstrained linear least squares problem.

(1) Determine the structure of X^TX from that of X.

(2) Determine a column ordering P_c for X so that the Cholesky factor $\overline{R}$ of X^TX is (hopefully) sparse.

(3) Symbolically factorize $P_c^TX^TXP_c$, generating a row-oriented data structure for $\overline{R}$.

(4) Determine a row ordering P_r for XP_c so that the cost of computing $\overline{R}$ is hopefully small.

(5) Compute $\overline{R}$ by processing the rows of P_rXP_c one by one using Givens rotations. Apply the Givens rotations to the vector P_ry simultaneously.

(6) Use the upper triangular matrix $\overline{R}$ and the first n elements of the modified vector P_ry to compute the least squares solution. Then permute the elements in the computed solution back into the original column ordering of X.

It should be clear from our discussion in this section that it is important for $\overline{R}$ to be sparse for efficient solution of large sparse problems. Hence dense rows in X are not desirable and should be withheld from the sparse orthogonal decomposition.

The structure of SPARSPAK-B is almost identical to the structure of the solution process described above, with only a few differences. Steps (5) and (6) are replaced by a single step which implements Bjorck's algorithm for solving sparse constrained linear least squares problems. As we shall see in Section 4, instead of working with X in Steps (1)-(4), we have to work with a matrix containing the sparse least squares and constraint portions; that is, $\begin{bmatrix} E \\ A \end{bmatrix}$.

3. Data structures in SPARSPAK-B

As we have mentioned in the previous section, the rows of X and Y can be stored in secondary storage and read in one at a time when they are needed. For general constrained linear least squares problems, as we are going to see in the next section, very often we have to manipulate and modify the dense rows in B and F at various points in the algorithm. Thus it is more convenient to store these dense rows in main memory. In SPARSPAK-B, we store B^T and F^T (column by column) respectively in two two-dimensional arrays *DSEQNS* and *DSCONS*. The vectors y_B and z_F are stored respectively in two one-dimensional arrays *DSBEQN* and *DSBCON*. The reason for storing B^T and F^T by columns rather than B and F is that it turns out that in the solution process, we have to solve numerous triangular systems using the rows of B and F as right hand side vectors. Thus, for efficient access of the elements of the rows of B and F, it is more convenient to store B^T and F^T by columns.

The other quantity we have to store is the sequence of upper triangular matrices $R^{(k)}$. The discussion in Section 2.1 showed that all we need is a storage scheme for the final upper triangular matrix R which can be set up by symbolically factoring X^TX. Since the nonzeros of R are accessed row by row, it is appropriate to use a row-oriented data structure. That is, the nonzeros of R are stored row by row in a one-dimensional array called *RNZ*. (For convenience, we store the diagonal elements of R separately in a one-dimensional array *DIAG*. Thus, *RNZ* only contains the off-diagonal nonzeros of R.) A one-dimensional array *XRNZ* of size $n+1$ is used to point to the beginning positions of the off-diagonal nonzeros in the rows of R. More specifically, the off-diagonal nonzeros of row k of R are found in $RNZ(i)$, where $i = XRNZ(k)$, $XRNZ(k)+1$, $XRNZ(k)+2$, $\cdots$, $XRNZ(k+1)-1$. We have assumed that $XRNZ(n+1)=\rho+1$, where ρ denotes the total number of off-diagonal nonzeros that can be accommodated in the data structure. The column subscripts of the off-diagonal nonzeros in *RNZ* are stored in a *compressed* format [3,19] in a one-dimensional integer array *NZSUB*, and the beginning position of the column subscripts for row k is stored in $XNZSUB(k)$, where *XNZSUB* is another one-dimensional array of size n. That is, the column subscripts for row k of R can be found in $NZSUB(j)$, for $j = XNZSUB(k)$, $XNZSUB(k)+1$, $XNZSUB(k)+2$, $\cdots$. The storage scheme described here is common in the solution of sparse symmetric positive definite systems. Readers are referred to [3] for details.

The solution vector β will be stored in a one-dimensional array QB which is of size n. Both QB and RNZ are initialized to zero before the numerical computation begins.

The arrays $DIAG$, RNZ, $XRNZ$, $NZSUB$, $XNZSUB$, QB, $DSCONS$, $DSEQNS$, $DSBCON$, and $DSBEQN$, together with various working arrays, are allocated from a single one-dimensional floating-point array (see Section 5 for more discussion).

4. An algorithm for solving general sparse constrained linear least squares problems

We now describe our implementation of Bjorck's algorithm for solving a general sparse constrained linear least squares problem. We first re-state the problem we want to solve:

$$\begin{aligned} &\min_{\beta \in \Omega} \left\| \begin{bmatrix} A \\ B \end{bmatrix} \beta - \begin{bmatrix} y_A \\ y_B \end{bmatrix} \right\|_2 \\ &\text{where} \quad \Omega = \left\{ \beta \;\middle|\; \beta \text{ minimizes } \left\| \begin{bmatrix} E \\ F \end{bmatrix} \beta - \begin{bmatrix} z_E \\ z_F \end{bmatrix} \right\|_2 \right\} \quad . \end{aligned} \tag{4.1}$$

The matrices $\boldsymbol{A}$ and $\boldsymbol{E}$ are sparse, and $\boldsymbol{B}$ and $\boldsymbol{F}$ are dense. The row dimensions of $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{E}$ and $\boldsymbol{F}$ are respectively m_1, m_2, p_1 and p_2. We assume that $m_2 \ll m_1$ and $p_2 \ll p_1$. The size of β is n. It is assumed that the ordering of the unknown variables $\beta_1, \beta_2, \cdots, \beta_n$ has been fixed. That is, the column ordering of $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{E}$ and $\boldsymbol{F}$ is fixed. Furthermore we assume a static storage scheme for $\overline{\boldsymbol{R}}$ has been set up, where

$$\overline{\boldsymbol{R}}^T \overline{\boldsymbol{R}} = \begin{bmatrix} \boldsymbol{E}^T & \boldsymbol{A}^T \end{bmatrix} \begin{bmatrix} \boldsymbol{E} \\ \boldsymbol{A} \end{bmatrix} \quad .$$

There are no restrictions on m_1, m_2, p_1, p_2 and n. For example, m_2 and p_1 could be zero while m_1 and p_2 are nonzero, or m_1 and m_2 both could be zero, with p_1 and p_2 nonzero. Of course, in practice, the second example may not make any sense, but nevertheless, the algorithm and the package SPARSPAK-B are capable of handling this peculiar case. (The package will treat the constraint equations as least squares equations in this case.) The only restrictions we have in SPARSPAK-B, which we believe are reasonable, are that the number of sparse (least squares and constraint) equations m_1+p_1 and the number of unknowns n are nonzero.

Another important feature of Bjorck's algorithm is its ability to handle rank deficient problems. More specifically, the algorithm makes no assumptions about the ranks the various matrices $\boldsymbol{A}$, $\boldsymbol{B}$, $\boldsymbol{E}$ and $\boldsymbol{F}$. Hence there is a possibility that problem (4.1) may not have a unique solution, in which case the package will determine the solution that has the minimal Euclidean norm (the minimal norm solution). In SPARSPAK-B, we make the reasonable assumption that the rank deficiency in $\begin{bmatrix} \boldsymbol{E} \\ \boldsymbol{A} \end{bmatrix}$ is small.

The algorithm may be broken up into five stages which are described below. In the actual implementation, we have a FORTRAN subroutine for each stage (which may invoke other subroutines as well). These five major subroutines are *GENLS*1, *GENLS*2, *GENLS*3, *GENLS*4 and *GENLS*5. Note that some of the steps in a major stage will be omitted if some of the matrices are null.

Stage 1: reduce sparse constraint equations

The first thing we do is to reduce the sparse constraint matrix $\boldsymbol{E}$ to upper trapezoidal form using the approach described in [3]. Hence we obtain the following decomposition:

$$\boldsymbol{Q}_E\boldsymbol{E} = \begin{bmatrix} \boldsymbol{R}_E & \boldsymbol{S}_E \\ \boldsymbol{O} & \boldsymbol{O} \end{bmatrix} ,$$

where $\boldsymbol{Q}_E$ represents the sequence of Givens rotations applied to $\boldsymbol{E}$. We shall assume that the rank of $\boldsymbol{E}$ is p'_1, where $p'_1 \leq \min(p_1, n)$. Thus, $\boldsymbol{R}_E$ will be $p'_1 \times p'_1$, upper triangular and nonsingular, and $\boldsymbol{S}_E$ is $p'_1 \times (n - p'_1)$. For simplicity, we denote by $\boldsymbol{R}_1$ the $n \times n$ upper triangular matrix

$$\boldsymbol{R}_1 = \begin{bmatrix} \boldsymbol{R}_E & \boldsymbol{S}_E \\ \boldsymbol{O} & \boldsymbol{O} \end{bmatrix} . \tag{4.2}$$

(The nonzeros of $\boldsymbol{R}_1$ are stored in the data structure allocated for $\overline{\boldsymbol{R}}$.)

It is important to note that, in general, it is not possible to obtain the upper trapezoidal form as specified in (4.2) unless column interchanges are performed. Recall from our previous discussion that the column ordering for $\boldsymbol{E}$ and $\boldsymbol{A}$ is determined (and hence fixed) prior to any numerical computation so that a static storage scheme for the sparse upper triangular matrix $\overline{\boldsymbol{R}}$ can be used during the reduction. Thus, it is not feasible to change this column ordering during the numerical phase. Without column

interchanges, the *actual* structure of $\boldsymbol{R}_1$ will be a permuted form of the one shown in (4.2). This does not create difficulties since in subsequent stages all we need is the ability to identify the rows in $\boldsymbol{R}_1$ that belong to $\begin{bmatrix}\boldsymbol{R}_E & \boldsymbol{S}_E\end{bmatrix}$. It is easy to show that if a diagonal element of $\boldsymbol{R}_1$ is zero, then the entire row (at least in exact arithmetic) must be null [18]. Thus, it simply amounts to determining which of the rows of $\boldsymbol{R}_1$ have nonzero diagonal elements. In order to make the description clearer, we shall assume throughout this section that the resulting upper triangular matrix $\boldsymbol{R}_1$ has been permuted appropriately so that it has the form given in (4.2), although this is not the case in the actual implementation.

Furthermore, in subsequent computations, the null rows of $\boldsymbol{R}_1$ will be modified by the least squares equations. However, it may be necessary to identify which of the non-null rows of the subsequently modified upper triangular matrix belong to $\begin{bmatrix}\boldsymbol{R}_E & \boldsymbol{S}_E\end{bmatrix}$. This is done by keeping a mask vector *ROWMSK* so that after the sparse constraint equations are reduced, $ROWMSK(i)$ is equal to 2 if and only if row i belongs to $\begin{bmatrix}\boldsymbol{R}_E & \boldsymbol{S}_E\end{bmatrix}$; otherwise, $ROWMSK(i)$ has the value 1.

A subtle problem to consider is the identification of "null" rows. Assume $p'_1 < n$. Since only finite-precision arithmetic is used, we are unlikely to find exactly $n - p'_1$ null rows in $\boldsymbol{R}_1$. It is more probable to find rows whose elements are small in magnitude. A reliable way to determine the rank of $\boldsymbol{E}$ is to use a singular value decomposition [13], but this method causes unacceptable fill for sparse problems. A heuristic but usually reliable method is to compare the diagonal elements of $\boldsymbol{R}_1$ with some user-specified small tolerance *TOL*. Suppose there are η diagonal elements that are smaller than *TOL* in magnitude. Then the corresponding η rows will be annihilated and $(n - \eta)$ will be regarded as the numerical rank of $\boldsymbol{E}$. See [14] for more details on this technique. Of course, the choice of *TOL* is itself a delicate problem. See [11] for a detailed discussion. This approach is also used in stage 3 when we annihilate the sparse least squares equations.

We now turn to the reduction of the sparse constraint equations. Suppose we have applied the Givens rotations to the vector $\boldsymbol{z}_E$:

$$\boldsymbol{Q}_E \boldsymbol{z}_E = \begin{bmatrix} \boldsymbol{e} \\ \hat{\boldsymbol{e}} \end{bmatrix} ,$$

where $\boldsymbol{e}$ and $\hat{\boldsymbol{e}}$ are respectively vectors of size p'_1 and $(p_1 - p'_1)$. The portion $\boldsymbol{e}$ will be used in subsequent computations and the vector $\begin{bmatrix} \boldsymbol{e} \\ \boldsymbol{0} \end{bmatrix}$ is stored in the array QB, but $\hat{\boldsymbol{e}}$

will not be needed and can be discarded.

We shall assume that the equations are stored in secondary storage. Thus we have to read in the equations to process the sparse constraint equations. At the same time, we can identify the dense constraint equations and store them (in the appropriate format) in *DSCONS* (and *DSBCON*).

We denote by $\bar{r}_E$ the residual vector of the sparse constraint equations

$$\bar{r}_E = E\beta - z_E \quad .$$

At the end of stage 1, the sparse constraint equations have been transformed into

$$Q_E\bar{r}_E = Q_EE\beta - Q_Ez_E = \begin{bmatrix} R_E & S_E \\ O & O \end{bmatrix}\beta - \begin{bmatrix} e \\ \hat{e} \end{bmatrix} \quad .$$

If the vector β is partitioned into

$$\beta = \begin{bmatrix} u \\ x \end{bmatrix} \quad ,$$

where u and x are respectively p'_1- and $(n-p'_1)$-vectors, then

$$Q_E\bar{r}_E = \begin{bmatrix} R_E & S_E \\ O & O \end{bmatrix}\begin{bmatrix} u \\ x \end{bmatrix} - \begin{bmatrix} e \\ \hat{e} \end{bmatrix} = \begin{bmatrix} R_Eu + S_Ex - e \\ -\hat{e} \end{bmatrix} \quad .$$

We denote by r_E the quantity

$$r_E = R_Eu + S_Ex - e \tag{4.3}$$

and after rearranging, we have

$$u = R_E^{-1}e + R_E^{-1}r_E - R_E^{-1}S_Ex \quad . \tag{4.4}$$

We now summarize the computations we have to perform in this stage (and in *GENLS*1).

(1) Reduce E to upper trapezoidal form using Givens rotations,

$$E \rightarrow \begin{bmatrix} R_E & S_E \\ O & O \end{bmatrix}$$

saving R_E and S_E in the static data structure.

(2) Apply the same transformations to the vector z_E,

$$z_E \rightarrow \begin{bmatrix} e \\ \hat{e} \end{bmatrix}$$

saving e in the array QB.

Stage 2: process dense constraint equations and compute residual of the sparse constraint equations

We partition the dense constraint matrix F into

$$F = \begin{bmatrix} F_u & F_x \end{bmatrix} \quad ,$$

where F_u and F_x are respectively $p_2 \times p'_1$ and $p_2 \times (n - p'_1)$. The residual vector $\bar{r}_F$ of the dense constraint equations can be written as

$$\bar{r}_F = z_F - F\beta = z_F - \begin{bmatrix} F_u & F_x \end{bmatrix} \begin{bmatrix} u \\ x \end{bmatrix}$$

$$= z_F - F_u u - F_x x \quad .$$

Replacing u by the expression in (4.4), we obtain

$$\begin{aligned} \bar{r}_F &= z_F - F_u(R_E^{-1}e + R_E^{-1}r_E - R_E^{-1}S_E x) - F_x x \\ &= (z_F - F_u R_E^{-1} e) - F_u R_E^{-1} r_E - (F_x - F_u R_E^{-1} S_E)x \\ &= \hat{z}_F - \hat{F}_u r_E - \hat{F}_x x \\ &= \hat{z}_F - \begin{bmatrix} \hat{F}_u & \hat{F}_x \end{bmatrix} \begin{bmatrix} r_E \\ x \end{bmatrix} \quad , \end{aligned} \tag{4.5}$$

where we have denoted $(z_F - F_u R_E^{-1} e)$, $F_u R_E^{-1}$ and $(F_x - F_u R_E^{-1} S_E)$ respectively by $\hat{z}_F$, $\hat{F}_u$ and $\hat{F}_x$. Note that $\hat{z}_F$, $\hat{F}_u$ and $\hat{F}_x$ depend only on known quantities. In fact, they can be computed using a simple scheme. Construct

$$R_2 = \begin{bmatrix} R_E & S_E \\ O & I \end{bmatrix} \quad .$$

That is, we simply replace the diagonal elements of all the null rows of R_1 by 1. Then it is not hard to see that

$$R_2^{-1} = \begin{bmatrix} R_E^{-1} & -R_E^{-1}S_E \\ O & I \end{bmatrix} ,$$

and

$$\begin{bmatrix} \hat{F}_u^T \\ \hat{F}_x^T \end{bmatrix} = \begin{bmatrix} R_E^{-T} & O \\ -S_E^T R_E^{-T} & I \end{bmatrix} \begin{bmatrix} F_u^T \\ F_x^T \end{bmatrix} = R_2^{-T} F^T \quad .$$

Of course we never compute R_2^{-1}; $\hat{F}_u$ and $\hat{F}_x$ are obtained by solving a few triangular systems. In SPARSPAK-B, a subroutine *RTSOLV* is provided for solving a lower triangular system using the transpose of a sparse upper triangular matrix as the coefficient matrix. Finally,

$$\hat{z}_F = z_F - F_u R_E^{-1} e = z_F - \hat{F}_u e = z_F - \begin{bmatrix} \hat{F}_u & \hat{F}_x \end{bmatrix} \begin{bmatrix} e \\ 0 \end{bmatrix} \quad .$$

In the actual implementation, $\hat{F}_u$ and $\hat{F}_x$ will overwrite F_u and F_x respectively. Also z_F will be overwritten by $\hat{z}_F$.

After we have computed $\hat{F}_u$ and $\hat{F}_x$, we reduce $\hat{F}_x$ to upper trapezoidal form using orthogonal transformations, and these transformations are applied also to $\hat{F}_u$ and $\hat{z}_F$

$$Q_F \begin{bmatrix} \hat{F}_u & \hat{F}_x \end{bmatrix} = \begin{bmatrix} H_1 & G \\ H_2 & O \end{bmatrix} \quad .$$

In SPARSPAK-B, since the transpose of $\begin{bmatrix} \hat{F}_u & \hat{F}_x \end{bmatrix}$ is stored, we have to reduce a subset of rows in a rectangular array to lower trapezoidal form. This subset of rows ($\hat{F}_x$) is identified using the information in the mask vector *ROWMSK*. The subroutine *MSKQR* is designed for this purpose.

Suppose G is $p'_2 \times (n-p'_1)$, where $p'_2 \leq p_2$. Then H_1 is $p'_2 \times p'_1$ and H_2 is $(p_2-p'_2) \times p'_1$. (Note that G has full rank.) Suppose

$$Q_F \hat{z}_F = \begin{bmatrix} g_1 \\ g_2 \end{bmatrix} ,$$

where g_1 and g_2 are vectors of size p'_2 and $(p_2-p'_2)$ respectively. Then (4.5) becomes

$$Q_F\bar{r}_F = Q_F\left(\hat{z}_F - \begin{bmatrix}\hat{F}_u & \hat{F}_x\end{bmatrix}\begin{bmatrix}r_E \\ x\end{bmatrix}\right) = \begin{bmatrix}g_1 \\ g_2\end{bmatrix} - \begin{bmatrix}H_1 & G \\ H_2 & O\end{bmatrix}\begin{bmatrix}r_E \\ x\end{bmatrix}$$

$$= \begin{bmatrix}g_1 - H_1 r_E - Gx \\ g_2 - H_2 r_E\end{bmatrix} \quad . \tag{4.6}$$

Since G now has full rank, we can always find an x so that the residual in the first p'_2 (modified) dense constraint equations is zero for *any* r_E, and (4.6) becomes

$$Q_F\bar{r}_F = \begin{bmatrix}0 \\ g_2 - H_2 r_E\end{bmatrix} \quad .$$

Now the residual of the (sparse and dense) constraint equations is given by

$$\left\| \begin{bmatrix}\bar{r}_E \\ \bar{r}_F\end{bmatrix} \right\|_2^2 = \| \bar{r}_E \|_2^2 + \| \bar{r}_F \|_2^2$$

$$= \| Q_E\bar{r}_E \|_2^2 + \| Q_F\bar{r}_F \|_2^2$$

$$= \left\| \begin{bmatrix}r_E \\ -\hat{e}\end{bmatrix} \right\|_2^2 + \left\| \begin{bmatrix}0 \\ g_2 - H_2 r_E\end{bmatrix} \right\|_2^2$$

$$= \| r_E \|_2^2 + \| g_2 - H_2 r_E \|_2^2 + \| \hat{e} \|_2^2 \quad .$$

The goal is to minimize the residual of the constraint equations. That is, we have to choose r_E so that

$$\| r_E \|_2^2 + \| H_2 r_E - g_2 \|_2^2 = \left\| \begin{bmatrix}r_E \\ H_2 r_E - g_2\end{bmatrix} \right\|_2^2$$

$$= \left\| \begin{bmatrix}I \\ H_2\end{bmatrix} r_E - \begin{bmatrix}0 \\ g_2\end{bmatrix} \right\|_2^2$$

is minimized. (Note that $\hat{e}$ is a constant vector.) This is simply an unconstrained linear least squares problem, and it has a unique solution since the coefficient matrix always has linearly independent columns. The dimension of the coefficient matrix is

$(p'_1+p_2-p'_2)\times p'_1$. We assume that p_2 (and hence p'_2) is small, but p'_1, which is the rank of the sparse constraint matrix $\boldsymbol{E}$, would be large (in the extreme case, p'_1 could be equal to n). Thus, the unconstrained linear least squares problem

$$\min_{r_E} \left\| \begin{bmatrix} I \\ H_2 \end{bmatrix} r_E - \begin{bmatrix} 0 \\ g_2 \end{bmatrix} \right\|_2 \tag{4.7}$$

may be large (and sparse). Fortunately, because of the structure of the problem, we do not have to employ sparse techniques in solving (4.7). Suppose we have computed a singular value decomposition of $\boldsymbol{H}_2$ (which is small and dense)

$$H_2 = U_H \Sigma_H V_H^T \quad ,$$

where U_H is a $(p_2-p'_2)\times(p_2-p'_2)$ orthogonal matrix, V_H is a $p'_1\times p'_1$ orthogonal matrix, and Σ is a $(p_2-p'_2)\times p'_1$ diagonal matrix whose diagonal elements are the singular values of $\boldsymbol{H}_2$. Then the solution to (4.7) can be expressed in terms of the singular values, $\boldsymbol{g}_2$, U_H and the first $(p_2-p'_2)$ columns of V_H if $(p_2-p'_2)\leq p'_1$, or V_H and the first p'_1 columns of U_H if $(p_2-p'_2)>p'_1$. The computation should be organized in such a way that only the necessary columns of U_H and V_H are computed. In SPARSPAK-B, the subroutines *LSSVD*1 (for $(p_2-p'_2)\leq p'_1$) and *LSSVD*2 (for $(p_2-p'_2)>p'_1$) are designed to solve (4.7) using a singular value decomposition of $\boldsymbol{H}_2$. The singular value decomposition is obtained using the subroutine *DSSVDC* which is a slight modification of the one in LINPACK [2].

Note that when the rows of $\boldsymbol{F}$ are exactly or almost linearly dependent, the elements of $\boldsymbol{H}_2$ may be very small in magnitude. In this case, it may be necessary to use an absolute tolerance to determine the numerical rank of $\boldsymbol{H}_2$ (even if relative tolerances are used elsewhere in the algorithm) so that we will be able to treat $\boldsymbol{H}_2$ as zero if all the singular values are very small. (If the elements of $\boldsymbol{H}_2$ have fairly large magnitude, it does not matter if absolute or relative tolerance is used. It will be up to the user's discretion.) This idea is also employed elsewhere when we have to determine the numerical rank of some submatrices from (probably modified) $\boldsymbol{B}$ and $\boldsymbol{F}$.

Suppose we have computed the residual vector $\boldsymbol{r}_E$ of the (modified) sparse constraint equations. We can then write the first p'_2 equations in (4.6) as

$$\boldsymbol{G}\boldsymbol{x} = \boldsymbol{h} \quad , \tag{4.8}$$

where $\boldsymbol{h}=\boldsymbol{g}_1-H_1\boldsymbol{r}_E$. Knowing $\boldsymbol{r}_E$ also allows (4.3) to be written as

$$R_E u + S_E x = f \quad , \tag{4.9}$$

where $f = e + r_E$.

Following is a summary of the steps involved at this stage (and in *GENLS2*).

(1) Compute

$$\hat{F} = \begin{bmatrix} \hat{F}_u & \hat{F}_x \end{bmatrix} = F R_2^{-1} \quad .$$

(2) Compute

$$\hat{z}_F = z_F - \hat{F} \begin{bmatrix} e \\ 0 \end{bmatrix} \quad .$$

(3) Transform

$$\begin{bmatrix} \hat{F}_u & \hat{F}_x \end{bmatrix}$$

into

$$\begin{bmatrix} H_1 & G \\ H_2 & O \end{bmatrix}$$

using orthogonal transformations. The submatrix G is upper trapezoidal and has full rank. Apply the same transformations to $\hat{z}_F$ and obtain $\begin{bmatrix} g_1 \\ g_2 \end{bmatrix}$.

(4) Solve the unconstrained least squares problem

$$\min_{r_E} \left\| \begin{bmatrix} I \\ H_2 \end{bmatrix} r_E - \begin{bmatrix} 0 \\ g_2 \end{bmatrix} \right\|_2 \quad .$$

(5) Compute $h = g_1 - H_1 r_E$ and $f = e + r_E$.

In the actual implementation, at the end of stage 2, *DSBCON* will contain the vector $\begin{bmatrix} h \\ 0 \end{bmatrix}$ and e will be overwritten by f in *QB*. We have replaced g_2 by zero since g_2 is no longer needed. Similarly, the portion in *DSCONS* containing H_1 and H_2 will also be set to zero.

Stage 3: reduce sparse least squares equations

We begin our description by reviewing what we have computed so far. The storage scheme for $\overline{R}$ contains the upper triangular matrix R_1 and we also have the vector f in QB.

$$R_1 = \begin{bmatrix} R_E & S_E \\ O & O \end{bmatrix} \quad , \quad \begin{bmatrix} f \\ 0 \end{bmatrix}$$

The arrays *DSCONS* and *DSBCON* contain respectively G and h.

$$\begin{bmatrix} O & G \\ O & O \end{bmatrix} \quad , \quad \begin{bmatrix} h \\ 0 \end{bmatrix}$$

Now we read in the rows from secondary storage again and annihilate the sparse least squares equations. At the same time, the dense least squares equations are identified and stored in the arrays *DSEQNS* and *DSBEQN* (in an appropriate format).

It is important to note that at this stage we do not always use Givens rotations to annihilate the nonzeros in the sparse least squares equations. The reason is that we do not want to modify R_E and S_E. Thus if we have to use a diagonal element of R_E to annihilate a nonzero in a row of A, we will use a Gaussian transformation instead of a Givens rotation. Again, rows belonging to $\begin{bmatrix} R_E & S_E \end{bmatrix}$ are identified by the fact that the corresponding *ROWMSK* values are 2. We use a Givens rotation only if the diagonal element of $\overline{R}$ does not belong to R_E. Loosely speaking, we first use the upper trapezoidal matrix in R_1 to modify the sparse least squares equations before we annihilate them using orthogonal transformations.

Note that row interchanges are not allowed when we use Gaussian transformations. Thus there is potentially numerical instability in the reduction of the sparse least squares equations. Such instability may be detected by computing an estimate of the condition number of R_E. We are currently looking into the implementation of such an estimator.

Thus, at the end of stage 3, the sparse least squares equations are reduced to upper trapezoidal form. More specifically, we have

$$Q_A \begin{bmatrix} R_E & S_E \\ A_u & A_x \end{bmatrix} = \begin{bmatrix} R_E & S'_E & R''_E \\ O & R_A & S_A \\ O & O & O \end{bmatrix} \quad . \tag{4.10}$$

Here we have partitioned A according to the partitioning in $\begin{bmatrix} R_E & S_E \end{bmatrix}$. That is, A_u and A_x are respectively $m_1 \times p'_1$ and $m_1 \times (n-p'_1)$. Assume R_A has rank m'_1, where $m'_1 \leq (n-p'_1)$. Then R_A is $m'_1 \times m'_1$, upper triangular and nonsingular. The dimensions of S_A, S'_E and S''_E are respectively $m'_1 \times (n-p'_1-m'_1)$, $p'_1 \times m'_1$ and $p'_1 \times (n-p'_1-m'_1)$. Note that $p'_1+m'_1$ is the rank of the matrix $\begin{bmatrix} E \\ A \end{bmatrix}$.

The rows of the upper triangular matrix in (4.10) can be categorized into three classes:

(a) Those that belong to the sparse constraints; that is, they are in the submatrix

$$\begin{bmatrix} R_E & S'_E & S''_E \end{bmatrix} \quad .$$

The corresponding *ROWMSK* entries still have the value 2.

(b) Those that belong to the sparse equations; that is, they are in the submatrix

$$\begin{bmatrix} O & R_A & S_A \end{bmatrix} \quad .$$

The corresponding *ROWMSK* entries will be 1.

(c) Those that are null. Their corresponding *ROWMSK* entries will be 0.

The information in *ROWMSK* allows the rows to be identified in subsequent computations.

Suppose the Gaussian transformations and Givens rotations are also applied to the vector $\begin{bmatrix} f \\ y_A \end{bmatrix}$ in the same order:

$$Q_A \begin{bmatrix} f \\ y_A \end{bmatrix} = \begin{bmatrix} f \\ s \\ \hat{s} \end{bmatrix} \quad ,$$

where s and $\hat{s}$ are respectively vectors of size m'_1 and $(n-p'_1-m'_1)$. In SPARSPAK-B, R_A and S_A are stored in the static data structure for $\overline{R}$. The vector s is stored in QB and the portion $\hat{s}$ is ignored. For simplicity, define R and S to be

$$R = \begin{bmatrix} R_E & S'_E \\ O & R_A \end{bmatrix} \quad ; \quad S = \begin{bmatrix} S''_E \\ S_A \end{bmatrix} \quad .$$

Now partition x into

$$x = \begin{bmatrix} v \\ w \end{bmatrix}$$

so that v and w are m'_1- and $(n-p'_1-m'_1)$-vectors. That is, the unknown vector β has been partitioned into

$$\beta = \begin{bmatrix} u \\ v \\ w \end{bmatrix} \quad .$$

From (4.9) and (4.10), we have

$$\begin{bmatrix} 0 \\ r_A \\ \hat{r}_A \end{bmatrix} = \begin{bmatrix} R_E & S'_E & S''_E \\ O & R_A & S_A \\ O & O & O \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} - \begin{bmatrix} f \\ s \\ \hat{s} \end{bmatrix} \quad ,$$

where $\begin{bmatrix} r_A \\ \hat{r}_A \end{bmatrix}$ is the residual vector of the (transformed) sparse least equations (and $\hat{r}_A = -\hat{s}$). The first $p'_1+m'_1$ equations are equivalent to

$$R \begin{bmatrix} u \\ v \end{bmatrix} + Sw = \begin{bmatrix} f \\ s \end{bmatrix} + \begin{bmatrix} 0 \\ r_A \end{bmatrix} \quad ,$$

and after rearranging, we obtain

$$\begin{bmatrix} u \\ v \end{bmatrix} = R^{-1} \begin{bmatrix} f \\ s \end{bmatrix} + R^{-1} \begin{bmatrix} 0 \\ r_A \end{bmatrix} - R^{-1} Sw \quad . \tag{4.11}$$

We now summarize the computational steps required in this stage (and in *GENLS*3).

(1) Reduce the sparse least squares equations A to upper trapezoidal form using Gaussian transformations and Givens rotations.

$$\begin{bmatrix} R_E & S_E \\ A_u & A_x \end{bmatrix} \rightarrow \begin{bmatrix} R_E & S'_E & S''_E \\ O & R_A & S_A \\ O & O & O \end{bmatrix} = \begin{bmatrix} R & S \\ O & O \end{bmatrix}$$

(2) Apply the same transformations to the vector containing f and y_A.

$$\begin{bmatrix} f \\ y_A \end{bmatrix} \rightarrow \begin{bmatrix} f \\ s \\ \hat{s} \end{bmatrix}$$

Stage 4: process (modified) dense constraint equations and dense least squares equations

If we partition the $p'_2 \times (n - p'_1)$ matrix G according to x

$$G = \begin{bmatrix} G_v & G_w \end{bmatrix} ,$$

then (4.8) can be written as

$$\begin{bmatrix} G_v & G_w \end{bmatrix} \begin{bmatrix} v \\ w \end{bmatrix} = \begin{bmatrix} O & G_v & G_w \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} = h$$

or

$$\begin{bmatrix} O & G_v \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} + G_w w = h \quad .$$

Substituting $\begin{bmatrix} u \\ v \end{bmatrix}$ by (4.11), we obtain

$$\begin{bmatrix} O & G_v \end{bmatrix} \left(R^{-1} \begin{bmatrix} f \\ s \end{bmatrix} + R^{-1} \begin{bmatrix} 0 \\ r_A \end{bmatrix} - R^{-1} S w \right) + G_w w = h$$

or

$$\begin{bmatrix} O & G_v \end{bmatrix} R^{-1} \begin{bmatrix} 0 \\ r_A \end{bmatrix} + \left(G_w - \begin{bmatrix} O & G_v \end{bmatrix} R^{-1} S \right) w$$

$$= h - \begin{bmatrix} O & G_v \end{bmatrix} R^{-1} \begin{bmatrix} f \\ s \end{bmatrix} \quad .$$

Denote $\begin{bmatrix} O & G_v \end{bmatrix} R^{-1}$ by $\begin{bmatrix} \hat{G}_u & \hat{G}_v \end{bmatrix}$, $\left(G_w - \begin{bmatrix} O & G_v \end{bmatrix} R^{-1} S \right)$ by $\hat{G}_w$ and $\left(h - \begin{bmatrix} O & G_v \end{bmatrix} R^{-1} \begin{bmatrix} f \\ s \end{bmatrix} \right)$ by $\hat{h}$. Then, we have

$$\begin{bmatrix} \hat{G}_u & \hat{G}_v \end{bmatrix} \begin{bmatrix} 0 \\ r_A \end{bmatrix} + \hat{G}_w w = \begin{bmatrix} \hat{G}_v & \hat{G}_w \end{bmatrix} \begin{bmatrix} r_A \\ w \end{bmatrix} = \hat{h} \quad . \tag{4.12}$$

As in stage 2, $\hat{h}$, $\hat{G}_v$ and $\hat{G}_w$ depend only on known quantities, and can be computed easily. Let

$$R_3 = \begin{bmatrix} R & S \\ O & I \end{bmatrix} \quad .$$

Thus R_3 is obtained from the triangular matrix obtained in stage 3 by adding 1 to the diagonal elements of all null rows. Then it is not hard to see that

$$R_3^T \begin{bmatrix} \hat{G}_u^T \\ \hat{G}_v^T \\ \hat{G}_w^T \end{bmatrix} = \begin{bmatrix} O \\ \hat{G}_v^T \\ \hat{G}_w^T \end{bmatrix}$$

and

$$\hat{h} = h - \begin{bmatrix} \hat{G}_u & \hat{G}_v & \hat{G}_w \end{bmatrix} \begin{bmatrix} f \\ s \\ 0 \end{bmatrix} \quad .$$

Hence $\hat{G}_v$, $\hat{G}_w$ and $\hat{h}$ are obtained by solving a few triangular systems (using *RTSOLV* in SPARSPAK-B). Again, $\hat{G}_v$, $\hat{G}_w$ and $\hat{h}$ can overwrite G_v, G_w and h respectively.

Now reduce $\hat{G}_v$ to upper trapezoidal form using orthogonal transformations, and apply these transformations to $\hat{G}_w$ and $\hat{h}$ as well (using *MSKQR* in SPARSPAK-B)

$$Q_G \begin{bmatrix} O & \hat{G}_v & \hat{G}_w \end{bmatrix} = \begin{bmatrix} O & T_1 & K \\ O & O & \hat{T}_1 \end{bmatrix} \quad ,$$

$$Q_G \hat{h} = \begin{bmatrix} t_1 \\ \hat{t}_1 \end{bmatrix} .$$

Next reduce $\hat{T}_1$ to upper trapezoidal form using orthogonal transformations, and apply the same transformations to $\hat{t}_1$ (using *MSKQR* again)

$$Q_T \begin{bmatrix} O & T_1 & K \\ O & O & \hat{T}_1 \end{bmatrix} = \begin{bmatrix} O & T_1 & K \\ O & O & T_2 \end{bmatrix} ,$$

$$Q_T \begin{bmatrix} t_1 \\ \hat{t}_1 \end{bmatrix} = \begin{bmatrix} t_1 \\ t_2 \end{bmatrix} .$$

(The portions $\hat{G}_v$ and $\hat{G}_w$ are identified using the information in the mask vector *ROWMSK*.) Note that both T_1 and T_2 have full rank (if they are not null). Assume T_1 is $p''_2 \times m'_1$ and T_2 is $p'''_2 \times (n - m'_1 - p'_1)$. Since G originally has full rank, $p''_2 + p'''_2 = p'_2$, $p''_2 \leq m'_1$, and $p'''_2 \leq (n - p'_1 - m'_1)$. Hence (4.12) is transformed into

$$\begin{bmatrix} T_1 & K \\ O & T_2 \end{bmatrix} \begin{bmatrix} r_A \\ w \end{bmatrix} = \begin{bmatrix} t_1 \\ t_2 \end{bmatrix} . \tag{4.13}$$

Now we can partition the columns of the dense least squares equations according the partitioning in β

$$B = \begin{bmatrix} B_u & B_v & B_w \end{bmatrix} .$$

The residual vector r_B of the dense least squares equations is

$$r_B = B\beta - y_B = \begin{bmatrix} B_u & B_v & B_w \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} - y_B ,$$

or

$$\begin{bmatrix} B_u & B_v \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} + B_w w = y_B + r_B .$$

Replacing $\begin{bmatrix} u \\ v \end{bmatrix}$ by (4.11) again, we get

$$\begin{bmatrix} B_u & B_v \end{bmatrix} \left(R^{-1} \begin{bmatrix} f \\ s \end{bmatrix} + R^{-1} \begin{bmatrix} 0 \\ r_A \end{bmatrix} - R^{-1} S w \right) + B_w w = y_B + r_B \quad ,$$

and after rearranging,

$$\begin{bmatrix} B_u & B_v \end{bmatrix} R^{-1} \begin{bmatrix} 0 \\ r_A \end{bmatrix} + \left(B_w - \begin{bmatrix} B_u & B_v \end{bmatrix} R^{-1} S \right) w$$

$$= y_B - \begin{bmatrix} B_u & B_v \end{bmatrix} R^{-1} \begin{bmatrix} f \\ s \end{bmatrix} + r_B \quad .$$

We denote $\begin{bmatrix} B_u & B_v \end{bmatrix} R^{-1}$ by $\begin{bmatrix} \hat{B}_u & \hat{B}_v \end{bmatrix}$, $\left(B_w - \begin{bmatrix} B_u & B_v \end{bmatrix} R^{-1} S \right)$ by $\hat{B}_w$, and $\left(y_B - \begin{bmatrix} B_u & B_v \end{bmatrix} R^{-1} \begin{bmatrix} f \\ s \end{bmatrix} \right)$ by $\hat{y}_B$. Then the equation above can be written as

$$\hat{B}_v r_A + \hat{B}_w w = \hat{y}_B + r_B \quad . \tag{4.14}$$

As before, the quantities $\hat{B}_u$, $\hat{B}_v$, $\hat{B}_w$ and $\hat{y}_B$ can be obtained by solving

$$R_3^T \begin{bmatrix} \hat{B}_u^T \\ \hat{B}_v^T \\ \hat{B}_w^T \end{bmatrix} = \begin{bmatrix} B_u^T \\ B_v^T \\ B_w^T \end{bmatrix}$$

(using *RTSOLV* in SPARSPAK-B) and computing

$$\hat{y}_B = y_B - \begin{bmatrix} \hat{B}_u & \hat{B}_v & \hat{B}_w \end{bmatrix} \begin{bmatrix} f \\ s \\ 0 \end{bmatrix} \quad .$$

We overwrite B_u, B_v, B_w and y_B respectively by $\hat{B}_u$, $\hat{B}_v$, $\hat{B}_w$ and $\hat{y}_B$ in our implementation.

The required computational tasks for this stage (and for *GENLS4*) are summarized below.

(1) Compute

$$\hat{G} = \begin{bmatrix} \hat{G}_u & \hat{G}_v & \hat{G}_w \end{bmatrix} = \begin{bmatrix} O & G_v & G_w \end{bmatrix} R_3^{-1} \quad .$$

(2) Compute

$$\hat{h} = h - \hat{G} \begin{bmatrix} f \\ s \\ 0 \end{bmatrix} \quad .$$

(3) Transform $\hat{G}_v$ to upper trapezoidal form and apply the same transformations to $\hat{h}$.

$$\begin{bmatrix} O & \hat{G}_v & \hat{G}_w \end{bmatrix} \rightarrow \begin{bmatrix} O & T_1 & K \\ O & O & \hat{T}_1 \end{bmatrix}$$

$$\hat{h} \rightarrow \begin{bmatrix} t_1 \\ \hat{t}_1 \end{bmatrix}$$

(4) Transform $\hat{T}_1$ to upper trapezoidal form and apply the same transformations to $\hat{t}_1$.

$$\begin{bmatrix} O & T_1 & K \\ O & O & \hat{T}_1 \end{bmatrix} \rightarrow \begin{bmatrix} O & T_1 & K \\ O & O & T_2 \end{bmatrix}$$

$$\begin{bmatrix} t_1 \\ \hat{t}_1 \end{bmatrix} \rightarrow \begin{bmatrix} t_1 \\ t_2 \end{bmatrix}$$

(5) Compute

$$\hat{B} = \begin{bmatrix} \hat{B}_u & \hat{B}_v & \hat{B}_w \end{bmatrix} = \begin{bmatrix} B_u & B_v & B_w \end{bmatrix} R_3^{-1} \quad .$$

(6) Compute

$$\hat{y}_B = y_B - \hat{B} \begin{bmatrix} f \\ s \\ 0 \end{bmatrix} \quad .$$

Stage 5: compute the minimal norm solution

This is the last (and most complicated) stage in the numerical solution. The static storage scheme contains the upper triangular matrix $\begin{bmatrix} R & S \\ O & O \end{bmatrix}$. The arrays *DSCONS* and *DSEQNS* respectively contain $\begin{bmatrix} O & T_1 & K \\ O & O & T_2 \end{bmatrix}$ and $\begin{bmatrix} \hat{B}_u & \hat{B}_v & \hat{B}_w \end{bmatrix}$, while the arrays *QB*, *DSBCON* and *DSBEQN* contain respectively $\begin{bmatrix} f \\ s \\ 0 \end{bmatrix}$, $\begin{bmatrix} t_1 \\ t_2 \\ 0 \end{bmatrix}$, and $\hat{y}_B$. Consider the residual of the (modified) least squares equations. Our goal is to minimize

$$\left\| \begin{bmatrix} r_A \\ r_B \end{bmatrix} \right\|_2 \quad ,$$

where r_A and r_B satisfy (from (4.13) and (4.14))

$$T_1 r_A = t_1 - Kw$$

and

$$\hat{B}_v r_A - r_B = \hat{y}_B - \hat{B}_w w \quad ,$$

with the condition that w satisfies $T_2 w = t_2$ (from (4.13)). That is, we have to solve the following constrained linear least squares problem

$$\begin{aligned} &\min \left\| \begin{bmatrix} r_A \\ r_B \end{bmatrix} \right\|_2 \\ &\text{subject to} \quad \begin{bmatrix} \hat{B}_v & -I \\ T_1 & O \end{bmatrix} \begin{bmatrix} r_A \\ r_B \end{bmatrix} = \begin{bmatrix} \hat{y}_B \\ t_1 \end{bmatrix} - \begin{bmatrix} \hat{B}_w \\ K \end{bmatrix} w \quad . \end{aligned} \tag{4.15}$$

The coefficient matrix $\begin{bmatrix} \hat{B}_v & -I \\ T_1 & O \end{bmatrix}$ is $(m_2+p''_2)\times(m'_1+m_2)$. Since $p''_2 \leq m'_1$, (4.15) is equivalent to the problem of computing the minimal norm solution to an underdetermined system of linear equations. Also note that since the coefficient matrix *always* has full row rank (T_1 has full rank), (4.15) always has a unique minimal norm solution. To solve (4.15), we proceed as follows. Let L be an $(m_2+p''_2)\times(m_2+p''_2)$ lower triangular matrix and Q_L be an $(m'_1+m_2)\times(m'_1+m_2)$

orthogonal matrix such that

$$\begin{bmatrix} \hat{B}_v & -I \\ T_1 & O \end{bmatrix} = \begin{bmatrix} L & O \end{bmatrix} Q_L \quad .$$

Since m_2 and p''_2 are assumed to be small, this decomposition can be computed easily. In SPARSPAK-B, a subroutine *DSQRDC* is provided to compute the *QR*-decomposition of a small dense matrix using Householder transformations. Thus, in order to compute L and Q_L, it is necessary to extract $\hat{B}_v$ and T_1 from the arrays *DSEQNS* and *DSCONS* using the information in *ROWMSK* and store the transpose of $\begin{bmatrix} \hat{B}_v & -I \\ T_1 & O \end{bmatrix}$. Then the underdetermined system can be written as

$$\begin{bmatrix} L & O \end{bmatrix} Q_L \begin{bmatrix} r_A \\ r_B \end{bmatrix} = \begin{bmatrix} \hat{y}_B \\ t_1 \end{bmatrix} - \begin{bmatrix} \hat{B}_w \\ K \end{bmatrix} w$$

or

$$\begin{bmatrix} I & O \end{bmatrix} Q_L \begin{bmatrix} r_A \\ r_B \end{bmatrix} = L^{-1} \begin{bmatrix} \hat{y}_B \\ t_1 \end{bmatrix} - L^{-1} \begin{bmatrix} \hat{B}_w \\ K \end{bmatrix} w \quad .$$

(Note that L is nonsingular.) Denote $L^{-1} \begin{bmatrix} \hat{y}_B \\ t_1 \end{bmatrix}$ and $L^{-1} \begin{bmatrix} \hat{B}_w \\ K \end{bmatrix}$ by $\hat{t}$ and $\hat{M}$ respectively. Also let $\begin{bmatrix} \hat{r}_1 \\ \hat{r}_2 \end{bmatrix} = Q_L \begin{bmatrix} r_A \\ r_B \end{bmatrix}$. Thus we have

$$\begin{bmatrix} I & O \end{bmatrix} \begin{bmatrix} \hat{r}_1 \\ \hat{r}_2 \end{bmatrix} = \hat{t} - \hat{M}w \quad .$$

This implies that

$$\hat{r}_1 = \hat{t} - \hat{M}w$$

and $\hat{r}_2$ can be arbitrary. Since

$$\min \left\| \begin{bmatrix} r_A \\ r_B \end{bmatrix} \right\|_2^2 = \min \left\| \begin{bmatrix} \hat{r}_1 \\ \hat{r}_2 \end{bmatrix} \right\|_2^2 = \min \left\| \hat{r}_1 \right\|_2^2 + \left\| \hat{r}_2 \right\|_2^2 \quad ,$$

$\hat{r}_2$ must be zero and $\hat{r}_1$ is the minimal residual vector of the following constrained least squares problem

$$\begin{aligned} &\min_{w} \left\| \hat{t} - \hat{M}w \right\|_2 \\ &\text{subject to} \quad T_2 w = t_2 \quad . \end{aligned} \tag{4.16}$$

This problem is small since $\hat{M}$ is $(p''_2+m_2)\times(n-p'_1-m'_1)$ and T_2 is $p'''_2\times(n-p'_1-m'_1)$, and we have assumed p''_2, p'''_2, m_2 and $(n-p'_1-m'_1)$ (which is the rank-deficiency in the final upper triangular matrix $\overline{R}$) are small. Note that the constraints are always consistent since T_2 has full rank, and the set of constraints is an underdetermined system because $p'''_2 \leq (n-p'_1-m'_1)$. We can solve this using any standard technique for solving small dense constrained least squares problem [15]. In SPARSPAK-B, there is a subroutine *DSCLS* for solving (4.16) using singular value decompositions. Let $\overline{w}$ be a solution to (4.16). A solution to (4.15) is then given by

$$\begin{bmatrix} r_A \\ r_B \end{bmatrix} = Q_L^T \begin{bmatrix} \hat{t} - \hat{M}\overline{w} \\ 0 \end{bmatrix} \quad .$$

Apparently, knowing r_A and $\overline{w}$ allows the final solution $\begin{bmatrix} u \\ v \end{bmatrix}$ to be computed using (4.11). (Note that $\overline{w}$ is part of the final solution vector.) Unfortunately this is not the case since $\hat{M}$ may be rank-deficient and (4.16) may not have a unique solution. Hence (4.11) may not give a unique solution for $\begin{bmatrix} u \\ v \end{bmatrix}$. In this case, we decide to compute the minimal norm solution. However, even if $\overline{w}$ is the minimal norm solution to (4.16) and if we compute $\begin{bmatrix} u \\ v \end{bmatrix}$ using $\overline{w}$ in (4.11), the resulting solution, say $\begin{bmatrix} \overline{u} \\ \overline{v} \\ \overline{w} \end{bmatrix}$, need not be the minimal norm solution to the original problem. This can readily be seen from (4.11); the norm of $\begin{bmatrix} u \\ v \\ w \end{bmatrix}$ also depends on R and S.

To compute the minimal solution to (4.1), we proceed as follows. Denote the general solution to (4.16) by

$$w = \bar{w} + Z\theta \quad ,$$

where $\bar{w}$ is *any* solution to (4.16), Z is a matrix containing linearly independent columns that span the null space associated with (4.16), and θ is some vector of appropriate dimension. (Z can be obtained easily if singular value decompositions are used to solve (4.16).) From (4.11), we have

$$\begin{bmatrix} u \\ v \end{bmatrix} = R^{-1}\begin{bmatrix} f \\ s \end{bmatrix} + R^{-1}\begin{bmatrix} 0 \\ r_A \end{bmatrix} - R^{-1}S(\bar{w} + Z\theta) \quad .$$

Let $d = \begin{bmatrix} f \\ s \end{bmatrix} + \begin{bmatrix} 0 \\ r_A \end{bmatrix}$. Then

$$\begin{bmatrix} u \\ v \end{bmatrix} = R^{-1}d - R^{-1}S\bar{w} - R^{-1}SZ\theta \quad .$$

Our goal is to find θ so that the Euclidean norm of $\begin{bmatrix} u \\ v \\ w \end{bmatrix}$ is minimized; that is, we want to solve the following problem

$$\min \left\| \begin{bmatrix} u \\ v \\ w \end{bmatrix} \right\|_2 = \min \left\| \begin{bmatrix} R^{-1}d - R^{-1}S\bar{w} - R^{-1}SZ\theta \\ \bar{w} + Z\theta \end{bmatrix} \right\|_2$$

$$= \min_{\theta} \left\| \begin{bmatrix} R^{-1}d - R^{-1}S\bar{w} \\ \bar{w} \end{bmatrix} - \begin{bmatrix} R^{-1}SZ \\ -Z \end{bmatrix}\theta \right\|_2 \quad . \tag{4.17}$$

This is again a small dense unconstrained linear squares problem in which the coefficient matrix has full column rank (since Z has linearly independent columns). Thus one can use an orthogonal decomposition of the coefficient matrix to solve (4.17). (We can use *DSQRDC* again in SPARSPAK-B.) Note that all we need is the residual vector of (4.17). The vector $\begin{bmatrix} R^{-1}d - R^{-1}S\bar{w} \\ \bar{w} \end{bmatrix}$ and the coefficient matrix $\begin{bmatrix} R^{-1}SZ \\ -Z \end{bmatrix}$ can be obtained by solving

$$\begin{bmatrix} R & S \\ O & I \end{bmatrix} b = \begin{bmatrix} d \\ \overline{w} \end{bmatrix}$$

and

$$\begin{bmatrix} R & S \\ O & I \end{bmatrix} C = \begin{bmatrix} O \\ -Z \end{bmatrix} .$$

In SPARSPAK-B, the subroutine *RSOLV*, which solves a sparse upper triangular system, can be used.

The various computational steps involved in this final stage (and in *GENLS*5) are summarized below.

(1) Determine the following matrices and vector from the various arrays using the information in *ROWMSK*.

$$\begin{bmatrix} \hat{B}_v & -I \\ T_1 & O \end{bmatrix} , \quad \begin{bmatrix} \hat{B}_w \\ K \end{bmatrix} , \quad \begin{bmatrix} \hat{y}_B \\ t_1 \end{bmatrix}$$

(2) Compute the following orthogonal decomposition.

$$\begin{bmatrix} \hat{B}_v & -I \\ T_1 & O \end{bmatrix} = \begin{bmatrix} L & O \end{bmatrix} Q_L$$

(3) Use L to solve for $\hat{t}$ and $\hat{M}$.

$$L\hat{t} = \begin{bmatrix} \hat{y}_B \\ t_1 \end{bmatrix}$$

$$L\hat{M} = \begin{bmatrix} \hat{B}_w \\ K \end{bmatrix}$$

(4) Solve the following small dense constrained least squares problem.

$$\min_{w} \| \hat{t} - \hat{M}w \|_2$$

subject to $T_2 w = t_2$

Denote a solution by $\overline{w}$ and let Z denote a set of linearly independent vectors that span the null space.

(5) Compute

$$\begin{bmatrix} r_A \\ r_B \end{bmatrix} = Q_L^T \begin{bmatrix} \hat{t} - \hat{M}\overline{w} \\ 0 \end{bmatrix} \quad .$$

(6) Form

$$d = \begin{bmatrix} f \\ s \end{bmatrix} + \begin{bmatrix} 0 \\ r_A \end{bmatrix} \quad .$$

(7) Solve

$$\begin{bmatrix} R & S \\ O & I \end{bmatrix} b = \begin{bmatrix} d \\ \overline{w} \end{bmatrix}$$

and

$$\begin{bmatrix} R & S \\ O & I \end{bmatrix} C = \begin{bmatrix} O \\ -Z \end{bmatrix} \quad .$$

(8) Compute the residual vector of the following least squares problem.

$$\min_{\theta} \| b - C\theta \|_2$$

5. The design of SPARSPAK-B

5.1. The interface

The software package SPARSPAK-B is a collection of FORTRAN subroutines for solving large sparse least squares problems. The design and implementation are similar to those of SPARSPAK (sparse matrix package), which is a package for solving large sparse symmetric positive definite systems [5]. The subroutines of SPARSPAK-B can roughly be divided into two classes: the interface subroutines and the internal subroutines.

The internal subroutines form the core of the package. These subroutines are responsible for performing the various tasks, such as finding the column and row permutations, determining the structure of $\overline{R}$ and setting up an efficient storage scheme, and computing the minimal norm solution. Those mentioned in the previous section, such as *GENLSi*, i=1,2,3,4,5, *MSKQR*, *RTSOLV*, *RSOLV*, and *DSSVDC* are examples of internal subroutines. Because of the use of sophisticated data structure

and storage management, and the large number of working arrays, these subroutines tend to have long and complicated calling sequences.

The interface subroutines are designed to be easy to use and have very simple calling sequences. As their name implies, they form an interface between the user and the internal subroutines. Each interface subroutine initiates a major task and it is responsible for invoking various internal subroutines to carry out that task. Thus these subroutines effectively insulate the user from the complications of the internal subroutines.

A problem that arises immediately is the communication among the internal subroutines and the communication between the interface and internal subroutines. Our solution involves the use of a one-dimensional floating-point array which we denote by T, labelled common blocks, and secondary storage. As we have mentioned in previous sections, the equations (and other information that describes the problem, such as weights) are stored in secondary storage. These equations are accessible by any subroutine of the package. The logical unit number of the external sequential file is supplied by the user through a call to an interface subroutine *FILEB*. The data structure and the numerical values of $\overline{R}$ are stored in the floating-point array T. Other (integer and floating-pointing) arrays required by the internal subroutines are also allocated from T. Thus it is important that the user not modify the array T between calls to the interface subroutines. In order to keep track of where the individual arrays are, pointers to the beginning of the arrays are needed. These pointers are stored in labelled common blocks. They are accessible by all the interface subroutines and are used when the internal subroutines are invoked.

Certain control information is also stored in the labelled common blocks. This includes timing information, storage requirements, sequence control and some system parameters. Sequence control is necessary to make sure that the interface subroutines are invoked in the proper sequence. System parameters include logical unit numbers for output files and the ratio of the number of bits in a floating-point number to that in an integer number. The latter parameter is needed in the allocation of integer arrays from the floating-point array T. The length of T is also stored in a variable in one of the common blocks and it should be initialized by the user at the beginning of the program.

It should be noted that the user has the option of using the internal subroutines directly. This allows the user to modify or replace any modules if the user develops new algorithms. Such flexibility also allows the user to tailor the package to her/his

applications.

5.2. The main interface subroutines

As we have seen in Section 2, the solution process can conveniently be broken into several steps, and the corresponding interface subroutines are described below.

Step 1: initialization

Initialization is done by invoking two interface subroutines, *SPRSPK* and *FILEB*. The FORTRAN statements are as follow.

CALL SPRSPK
CALL FILEB (FILE)

The first subroutine *SPRSPK* needs to be called only once in each program. It is responsible for initializing the timing routine and setting the logical unit numbers for output files. The second subroutine is called for each problem to be solved. The integer variable *FILE* contains the logical unit number for an external sequential file which will be used by the package.

Step 2: problem input

After the initialization subroutines have been called, the user then invokes the interface subroutine *INXYWB* to input each equation. The calling sequence has the form shown below.

CALL INXYWB (ROWNUM, TYPE, NSUBS, SUBS, VALUES, RHS, WEIGHT, T)

The integer variable *ROWNUM* contains a positive integer which is usually less than $m+p+1$. It may be regarded as a label for each equation. To indicate that an input equation is a sparse least squares equation, the integer variable *TYPE* should contain 1. If the input equation is a dense least squares equation, a sparse constraint equation or a dense constraint equation, *TYPE* should then contain 2, 3 or 4 accordingly. The integer variable *NSUBS* contains the number of nonzeros in the input equation (excluding the entry in $\boldsymbol{y}$ or $\boldsymbol{z}$). The column indices and numerical values of the nonzeros are stored respectively in the integer array *SUBS* and floating-point array *VALUES*. The corresponding entry in $\boldsymbol{y}$ or $\boldsymbol{z}$ is stored in the floating-point variable *RHS*. If the problem is a weighted least squares problem, the floating-point variable *WEIGHT* contains the weight; otherwise, it should be set to 1.0.

Thus the problem can be provided to the package by calling *INXYWB* repeatedly. Note that the equations can be supplied to the package in any order.

Step 3: column ordering

This step is initiated by invoking the subroutine *ORCOLB*. The FORTRAN statement to be used is shown below.

CALL ORCOLB (T)

This subroutine serves several purposes. First, it indicates that Step 2 has finished; that is, that the user has supplied the entire problem to the package. Second, the package reorders the columns of $\begin{bmatrix} A \\ E \end{bmatrix}$. The minimum degree algorithm is used to generate a good P_c. Third, the package determines the structure of $\overline{R}$ from that of A^TA+E^TE and the permutation matrix P_c, and sets up an efficient storage scheme for $\overline{R}$.

Step 4: row ordering

The subroutine to be invoked is *ORROWB*, and its calling sequence is as follows.

CALL ORROWB (OPTION, T)

The integer variable *OPTION* indicates the order in which the rows should be processed. By setting $OPTION=0$, the user specifies that the rows should be processed in the order they were input. Other possibilities ($OPTION\neq 0$) include

(1) arranging the rows in the order of the parameter *ROWNUM*,

(2) arranging the rows in the order of increasing number of nonzeros,

(3) arranging the rows in the order of increasing weight (this is useful when the weights vary widely in value), and

(4) reordering the rows in order to attempt to reduce the cost of computing $\overline{R}$ (see the discussion in Section 2.3).

Step 5: solution

The numerical computation of the the least squares solution is initiated by calling the subroutine *LSQSLV*. The FORTRAN statement is as follows.

CALL LSQSLV (TOL, TYPTOL, T)

Here *TOL* is a tolerance used to determine when a numerical value should be regarded as numerically zero. The test can be done in absolute ($TYPTOL=0$) or relative ($TYPTOL=1$) terms.

After *LSQSLV* is called, the first n elements of T contain the least squares solution.

5.3. Other interface subroutines

The interface contains a few other subroutines which may be useful.

Residual computation

The subroutine *RESIDB* can be used after *LSQSLV* has been executed successfully to compute the residuals in the least squares equations and constraint equations. The FORTRAN statement to be used is as follows.

CALL RESIDB (RESEQN, RESCON, T)

After *RESIDB* is called, the floating-point variables *RESEQN* and *RESCON* will contain respectively the residuals of least squares and constraint equations.

Statistics

Another useful interface subroutine is *STATSB*, whose calling sequence is shown below.

CALL STATSB

The purpose of this subroutine is to display the statistics gathered by the package in the computation. The statistics include timing information and storage required by various interface subroutines mentioned above.

Save and restart

Sometimes it may be necessary to save the intermediate information provided by SPARSPAK-B so that the user can perform something else before returning to the solution of the least squares problem. For example, each major task in the solution process requires a different amount of space from the storage array T. There is a possibility that the computation cannot proceed at some stage because there is not enough space in T. In this case, it is desirable to be able to save what has been computed successfully so far before terminating the program. Then the user can change the size of the array T, and resume the computation by restoring the information that has been saved. This avoids the need to execute those tasks that were successfully carried out previously.

In order to make sure that everything is saved so that execution can be resumed at a later point, a subroutine *SAVEB* has been provided. The FORTRAN statement to be used is

CALL SAVEB (K, T)

where K is a logical unit number of an external sequential file. The contents of the floating-point array T and the values of the variables in the labelled common blocks are written onto the sequential file specified. It is important for the sequential file in the save operation to be different from that supplied to the package using the subroutine *FILEB*.

When the user is ready to resume execution, the following statement should be used to restore everything saved by *SAVEB*.

CALL RSTRTB (K, T)

Note that after *SAVEB* is called, the user can use the floating-point array T in other computations.

6. Concluding remarks

In this report we have described the implementation of a software package for solving large sparse constrained linear least squares problems. This package, which is written using a portable subset of ANSI-66 Standard FORTRAN, contains a set of interface subroutines that are easy to use and which insulate the user from the complicated data structures and storage management involved in the actual solution subroutines.

There are several reasons for designing the package as a "library" of interface subroutines for different major tasks in the solution process, rather than as a "stand-alone" program. First, it allows the package to be embedded in a "super-package" where different tasks may be carried out at different instances. Second, it provides flexibility that is not available in stand-alone programs, particularly in terms of storage management. Note that each major task may require a different amount of work space from the storage array T. In a stand-alone program, if the computation cannot proceed at some stage because of the lack of space in T, the user can only terminate the program, change the size of the work space and run the program again. There is often no effective way to save the output from those tasks that were executed successfully. This is not desirable in the solution of large problems. In the package, the design of the interface allows the user to insert checkpoints in the main program. Output from tasks that are successfully executed can be saved before the program is terminated and the program can be restarted by restoring the information that is saved. This avoids the need to "re-do" those successful tasks. Finally, the design provides an efficient way to experiment with new ideas. For example, if we want to compare two different algorithms for handling dense rows, we do not have to carry out the column ordering (and row ordering) twice. All we need to do is to save the ordering information when the first method is tested. Then we simply restore the ordering information before we use the second method. Such flexibility is much less easy to provide if the package is designed as a stand-alone program.

Finally, it should be pointed out that the implementation of the Bjorck's algorithm in the package is by no means perfect, though the algorithm is mathematically correct. Its behavior in the presence of roundoff errors is not well understood. Experience has shown that the numerical algorithm may be sensitive to roundoff errors for some ill-conditioned problems.

7. References

[1] A. BJORCK, "A general updating algorithm for constrained linear least squares problems", *SIAM J. Sci. Stat. Comput.* **5**, pp. 394-402 (1984).

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

[3] J.A. GEORGE AND M.T. HEATH, "Solution of sparse linear least squares problems using Givens rotations", *Linear Algebra and its Appl.* **34**, pp. 69-83 (1980).

[4] J.A. GEORGE, M.T. HEATH, AND E.G-Y. NG, "Solution of sparse underdetermined systems of linear equations", *SIAM J. Sci. Stat. Comput.* **5**, pp. 988-997 (1984).

[5] J.A. GEORGE AND J.W-H. LIU, "The design of a user interface for a sparse matrix package", *ACM Trans. on Math. Software* **5**, pp. 134-162 (1979).

[6] J.A. GEORGE AND J.W-H. LIU, *Computer solution of large sparse positive definite systems,* Prentice-Hall Inc., Englewood Cliffs, New Jersey (1981).

[7] J.A. GEORGE, J.W-H. LIU, AND E.G-Y. NG, "Row ordering schemes for sparse Givens transformations, I. Bipartite graph model", *Linear Algebra and its Appl.* **61**, pp. 55-81 (1984).

[8] J.A. GEORGE, J.W-H. LIU, AND E.G-Y. NG, "Row ordering schemes for sparse Givens transformations, II. Implicit graph model", *Linear Algebra and its Appl.*, (1985). (To appear.)

[9] J.A. GEORGE, J.W-H. LIU, AND E.G-Y. NG, "Row ordering schemes for sparse Givens transformations, III. Analysis for a model problem", *Linear Algebra and its Appl.*, (1985). (To appear.)

[10] J.A. GEORGE AND E.G-Y. NG, "On row and column orderings for sparse least squares problems", *SIAM J. Numer. Anal.* **20**, pp. 326-344 (1983).

[11] J.A. GEORGE AND E.G-Y. NG, "User's guide for SPARSPAK-B: Waterloo sparse constrained linear least squares package", Research Report CS-84-37, Department of Computer Science, University of Waterloo (1984).

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

[13] G.H. GOLUB AND C. REINSCH, "Singular value decomposition and least squares solutions", *Numer. Math.* **14**, pp. 403-420 (1970).

[14] M.T. HEATH, "Some extensions of an algorithm for sparse linear least squares problems", *SIAM J. Sci. Stat. Comput.* **3**, pp. 223-237 (1982).

[15] C.L. LAWSON AND R.J. HANSON, *Solving least squares problems,* Prentice-Hall Inc., Englewood Cliffs, N.J. (1974).

[16] J.W-H. LIU, "On general row merging schemes for sparse Givens transformations", Technical Report No. 83-04, Department of Computer Science, York University, Downsview, Ontario (1983).

[17] J.W-H. LIU, "Modification of the minimum degree algorithm by multiple elimination", *ACM Trans. on Math. Software* **11**, pp. 141-153 (1985).

[18] E.G.Y. NG, "Row elimination in sparse matrices using rotations", Research report CS-83-01, Department of Computer Science, University of Waterloo (1983). (Doctoral Dissertation)

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

[20] M. YANNAKAKIS, "Computing the minimum fill-in is NP-complete", *SIAM J. Alg. Disc. Meth.* **2**, pp. 77-79 (1981).